Fine-Grained Complexity and Algorithm Engineering of Geometric Similarity Measures

A dissertation submitted towards the degree Doctor of Engineering of the Faculty of Mathematics and Computer Science of Saarland University

by André Nusser

Saarbrücken / 2021

Day of Colloquium:	13. May 2022
Dean of the Faculty:	Prof. Dr. Jürgen Steimle
Chair of the Committee:	Prof. Dr. Raimund Seidel
Reporters	
First reviewer:	Prof. Dr. Karl Bringmann
Second reviewer:	Prof. Dr. Kurt Mehlhorn
Third reviewer:	Prof. Dr. Timothy Chan
Fourth reviewer:	Prof. Dr. Mark de Berg
Academic Assistant:	Dr. Roohani Sharma

We know that there is no help for us but from one another, that no hand will save us if we do not reach out our hand. And the hand that you reach out is empty, as mine is. You have nothing. You possess nothing. You own nothing. You are free. All you have is what you are, and what you give.

The Dispossessed, Ursula K. Le Guin

Abstract

Abstract. Point sets and sequences are fundamental geometric objects that arise in any application that considers movement data, geometric shapes, and many more. A crucial task on these objects is to measure their similarity. Therefore, this thesis presents results on algorithms, complexity lower bounds, and algorithm engineering of the most important point set and sequence similarity measures like the Fréchet distance, the Fréchet distance under translation, and the Hausdorff distance under translation. As an extension to the mere computation of similarity, also the approximate near neighbor problem for the continuous Fréchet distance on time series is considered and matching upper and lower bounds are shown.

Zusammenfassung. Punktmengen und Sequenzen sind fundamentale geometrische Objekte, welche in vielen Anwendungen auftauchen, insbesondere in solchen die Bewegungsdaten, geometrische Formen, und ähnliche Daten verarbeiten. Ein wichtiger Bestandteil dieser Anwendungen ist die Berechnung der Ähnlichkeit von Objekten. Diese Dissertation präsentiert Resultate, genauer gesagt Algorithmen, untere Komplexitätsschranken und Algorithm Engineering der wichtigsten Ähnlichkeitsmaße für Punktmengen und Sequenzen, wie zum Beispiel Fréchetdistanz, Fréchetdistanz unter Translation und Hausdorffdistanz unter Translation. Als eine Erweiterung der bloßen Berechnung von Ähnlichkeit betrachten wir auch das Near Neighbor Problem für die kontinuierliche Fréchetdistanz auf Zeitfolgen und zeigen obere und untere Schranken dafür.

Acknowledgments

Obviously, the attempt to exhaustively thank the people that I want to acknowledge is futile and would probably double the size of this thesis. However, as you are reading a thesis and not my memoirs, I tried to create a good short version. Obviously, this did not work out, but I had to submit at *some* point. Consequently, let me first just thank *you* — yes, you! — the reader, who for some reason indeed opened this thesis — no matter if you are a dear friend, a stranger, a colleague, or an enemy (do I have any? If so, please contact me and let me know!).

A bit more seriously, I want to express my deepest gratitude towards Karl for being an incredible supervisor, and Marvin for being an incredible unofficial supervisor. Thank you Kurt for giving me the best advice at the start of my PhD. Thank you Bhaskar for being a wonderful office mate (or dude?), and thanks to all my fellow PhD students and all present and past group members of D1 who created such a wonderful atmosphere. On a more ancient note, thanks Sabine and Stefan for collaborating with me on our first papers and introducing me to research. I very much thank all my other wonderful collaborators and the people who were so generous to invite me for a research visit. I also very much want to thank the external reviewers of this thesis, Timothy Chan and Mark de Berg, for devoting their very valuable time.

Most importantly, I thank all my friends without whom life would just be utterly senseless, I thank my family for always being unbelievably supportive, and, from the bottom of my heart, I thank Terry (🐾), Linux (🐾), Kropotkin (🐾), Yoda (🐾), Tolstoi (🐾), Franzi, and Hana.

Preface

I worked on a multitude of topics during and also before my PhD, therefore, a significant part of my previous research is excluded from this thesis. To still give an overview of my previous work, I list all my publications below. At the time of writing I published 17 conference papers, 6 journal papers, and one work that I list is currently under submission (17 of these papers were mainly done during my PhD studies and 7 before I started my PhD; ordering of the names is always alphabetical):

[1] D. Bahrdt, M. Becher, S. Funke, F. Krumpe, A. Nusser, M. Seybold, and S. Storandt. Growing balls in $\mathbb{R}^d$. In *Proceedings of the Ninteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 247–258, 2017.

[2] M. Brankovic, K. Buchin, K. Klaren, A. Nusser, A. Popov, and S. Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In *28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 99–110, 2020.

[3] K. Bringmann, A. Driemel, A. Nusser, and I. 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. To appear.

[4] K. Bringmann, M. Künnemann, and A. Nusser. Discrete Fréchet distance under translation: conditional hardness and an improved algorithm. *ACM Trans. Algorithms*, 17(3):25:1–25:42, 2021.

[5] K. Bringmann, M. Künnemann, and A. 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)*, pages 2902–2921, 2019.

[6] K. Bringmann, M. Künnemann, and A. Nusser. Walking the dog fast in practice: algorithm engineering of the Fréchet distance. In *35th International Symposium on Computational Geometry (SoCG)*, volume 129 of *LIPIcs*, 17:1–17:21, 2019.

[7] K. Bringmann, M. Künnemann, and A. Nusser. When Lipschitz walks your dog: algorithm engineering of the discrete Fréchet distance under translation. In *28th Annual European Symposium on Algorithms (ESA)*, volume 173 of *LIPIcs*, 25:1–25:17, 2020.

[8] K. Bringmann, M. Künnemann, and A. Nusser. Walking the dog fast in practice: algorithm engineering of the Fréchet distance. *Journal of Computational Geometry*, 12(1), 2021.

[9] K. Bringmann and A. Nusser. Translating Hausdorff is hard: fine-grained lower bounds for Hausdorff distance under translation. In *37th International Symposium on Computational Geometry (SoCG)*, volume 189 of *LIPIcs*, 18:1–18:17, 2021.

[10] K. Buchin, A. Driemel, N. van de L'Isle, and A. Nusser. Klcluster: center-based clustering of trajectories. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 496–499, 2019.

[11] D. Coudert, A. Nusser, and L. Viennot. Computing graph hyperbolicity using dominating sets. In *Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX)*, 2022. To appear.

[12] D. Coudert, A. Nusser, and L. Viennot. Enumeration of far-apart pairs by decreasing distance for faster hyperbolicity computation. *CoRR*, abs/2104.12523, 2021. Under submission.

[13] E. Cruciani, E. Natale, A. Nusser, and G. Scornavacca. On the emergent behavior of the 2-choices dynamics. In *Proceedings of the 19th Italian Conference on Theoretical Computer Science*, volume 2243 of *CEUR Workshop Proceedings*, pages 60–64, 2018.

[14] E. Cruciani, E. Natale, A. Nusser, and G. Scornavacca. Phase transition of the 2-choices dynamics on core-periphery networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, pages 777–785, 2018.

[15] E. Cruciani, E. Natale, A. Nusser, and G. Scornavacca. Phase transition of the 2-choices dynamics on core-periphery networks. *Distributed Comput.*, 34(3):207–225, 2021.

[16] A. Driemel, A. Nusser, J. M. Phillips, and I. Psarros. The VC Dimension of metric balls under Fréchet and Hausdorff distances. *Discrete & Computational Geometry*, 2021. ISSN: 1432-0444. DOI: 10.1007/s00454-021-00318-z.

[17] S. Funke, A. Nusser, and S. Storandt. On k-path covers and their applications. *Proc. VLDB Endow.*, 7(10):893–902, 2014.

[18] S. Funke, A. Nusser, and S. Storandt. On k-path covers and their applications. *VLDB J.*, 25(1):103–123, 2016.

[19] S. Funke, A. Nusser, and S. Storandt. Placement of loading stations for electric vehicles: allowing small detours. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (ICAPS)*, pages 131–139, 2016.

[20] S. Funke, A. Nusser, and S. Storandt. Placement of loading stations for electric vehicles: no detours necessary! In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 417–423, 2014.

[21] S. Funke, A. Nusser, and S. Storandt. Placement of loading stations for electric vehicles: no detours necessary! *J. Artif. Intell. Res.*, 53:633–658, 2015.

[22] S. Funke, A. Nusser, and S. Storandt. The simultaneous maze solving problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 808–814, 2017.

[23] S. Funke, T. Rupp, A. Nusser, and S. Storandt. PATHFINDER: storage and indexing of massive trajectory sets. In *Proceedings of the 16th International Symposium on Spatial and Temporal Databases (SSTD)*, pages 90–99, 2019.

[24] M. Künnemann and A. Nusser. Polygon placement revisited: (Degree of freedom + 1)-SUM hardness and an algorithm via offline dynamic rectangle union. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2022. To appear.

The papers that I worked on before my PhD are [1, 17, 18, 19, 20, 21, 22]. Considering the remaining publications, a large part of my work is in the field of computational geometry. These works can again be subdivided into theoretical insights [16], algorithmics (algorithm design and fine-grained lower bounds) [3, 4, 5, 9, 24], and algorithm engineering [2, 6, 7, 8, 10]. Theoretical and practical results are roughly equally represented. All the (theoretical) algorithmic results in computational geometry are in combination with fine-grained lower bounds. The remainder of my work is located in the area of algorithm engineering for graph problems [11, 12, 23] and analysis of the behavior of opinion dynamics [13, 14, 15]. Papers whose results and content I include in this thesis are [3, 4, 5, 6, 7, 8, 9].

On collaboration. All of the works included in this thesis were created in close collaboration with my co-authors and the contribution is equally shared between the authors of each paper.

Contents

CHAPTER 1

Introduction

Computer science as a field aims to better understand the power and limitations of computation but also enables new results in fields that crucially rely on computation. While on first glance it seems that algorithmics in the form of algorithm design should be the central area of research in this regard, progress very much relies on complexity theory to understand the inherent hardness of certain problems, and an engineering approach, to transfer the theoretical knowledge into practical knowledge that can then be further exploited. In this spirit, I consider three central parts of algorithms research in this thesis: algorithm design, complexity lower bounds, and algorithm engineering. Algorithm design — also called *upper bounds*, from a complexity perspective — provides us with knowledge and tools for performant computation, lower bounds help to gain insight about the difficulty of problems, and algorithm engineering gives us the tools to make theoretical results available to those who most require them in the first place. While aesthetics of problems and their solutions by themselves are a great motivation for research — probably even *the* most beautiful motivation — in my opinion, to provide tools that can have wider impact on areas where performant computation is crucially needed should be the main justification and driver behind computer science research.

1.1 A Powerful Combination

Imagine being approached by a scientist of another field with a new computational problem whose resolution would significantly advance their field. Obviously, you do feel very motivated to resolve this problem. The first natural approach to help your fellow scientist is to do several iterations of theoretically designing algorithms that solve the problem with a running time and memory consumption that gets closer and closer to the desired performance. Now two things can happen:

(1) You might find an algorithm that matches or even exceeds the theoretically required bounds and you can continue with practically evaluating this algorithm using algorithm engineering.

(2) You are stuck and seemingly unable to find an algorithm with sufficiently good performance.

Let us first focus on the second case. Lower bounds can provide a way out of this situation by providing evidence that algorithms with the desired performance cannot exist. However, unconditional lower bounds are often still not sufficiently developed to make statements with the strength that we desire. Thus, reducing from seemingly hard problems to our problem can at least give us the insight, that our problem is at least as hard as another problem and thus we can bundle our hardness assumptions into a few selected core problems that seem difficult to improve. Not only does this allow us to find a (conditionally) negative answer to the existence of an algorithm with a certain performance, it can actually also guide us towards better algorithms by understanding difficult structures in the input that need to be considered by the algorithm.

Having arrived at a best possible algorithm like that, we can again — as in the first case from above — turn to algorithm engineering to guide us towards a practically fast solution. While algorithm design and fine-grained lower bounds are crucial components in finding *theoretically* best algorithms, unfortunately, the obtained algorithms are not necessarily the best ones *in practice* as we always work with simplified models in theory to work on a reasonable abstraction level. Reasons for this disparity are that there is structure in practical input, that worst-case instances might be highly artificial, that there might be large hidden constants, caching effects, and many more. Thus, for finding best practical algorithms, we need a different approach, namely algorithm engineering. While this approach still strongly relies on theoretical results as source of inspiration and for deep insights into the structure of the problem, other considerations like cache alignment and adaptivity of the algorithm to the difficulty of the input instance also play a crucial role.

1.2 Point Set and Point Sequence Similarity Measures

The main objects of study of this thesis are *point sets* and *point sequences*. In this thesis point sets are always a finite subset of some finite-dimensional $\mathbb{R}^d$. If we have points that have a specific order (and with potential multiplicities), we call it a point sequence. In this thesis we also use the term *sequence* to refer to point sequences as these are the only sequences that we focus on in the technical parts. Sequences can either be seen as a discrete set of points with a certain order, or as a continuous path through $\mathbb{R}^d$ by connecting all neighboring points by line segments. We call the number of points that define a sequence its *complexity*. One basic operation that one wants to perform on these kinds of objects is to quantify how similar a pair is. More precisely, given two sets of points or two sequences, we want to compute their distance with respect to some mathematical measure. However, while for a pair of points there arguably is a most natural distance to use — the Euclidean distance — for point sets and point sequences this is not the case. Instead there is a multitude of distance measures that can all be suitable depending on the specific requirements posed by the targeted application.

1.2.1 Motivation

Before we introduce the most important point set and sequence similarity measures, let us first elaborate why such measures are of crucial importance. Point sets are among the most fundamental discrete geometric objects arising in every scenario where we have a collection of multi-dimensional objects without any additional structure. Examples where point sets arise are collections of multi-dimensional measurements, simple descriptions of geometric shapes (point clouds), data bases consisting of numeric entries, a collection of geographic points of interest, and many more. In this sense, quantifying the similarity between point sets enables us to determine the similarity of each of the above types of data. More concretely, point set similarity measures are used for medical image analysis [92], audio steganalysis [126], synthetic aperture radar image matching [130], image search [114, 145], and many more.

While point sets do not have any additional structure inherent in their description, sequences merely additionally give an order of the points and allow for multiplicities. Sequences arise in many scenarios such as measurements that are taken in a certain temporal order. Example of such processes are stock price development, handwritten characters, audio recordings, GPS trajectories and movements in general including sports analysis, and many more. More concretely, sequence similarity measures are crucial for gaining insights into data in movement ecology [73], for map matching [35, 52, 61, 67, 163], signature verification [137, 164], analyzing movement patterns [50, 55], character classification [45], and sports/player analysis [58, 78]. Hence, any research that works towards a more performant computation of such measures or provides insights about their nature can directly impact, and even crucially enable, a vast amount of research in other fields.

1.2.2 Similarity Measures

In this section we give an overview of important point and sequence similarity measures. Note that we only give intuitive definitions and refer to Chapter 2 for formal definitions. Probably the most famous point set and sequence similarity measures are Hausdorff distance, dynamic time warping (DTW), and Fréchet distance. The Hausdorff distance is a point set similarity measure, while the other two measures are sequence similarity measures. Furthermore, DTW minimizes a sum of distances, while the other two minimize the maximal distance — therefore, they are also called *bottleneck measures*. More precisely, the Hausdorff distance of two point sets is defined as the maximum distance of any point to its closest neighbor in the other set. The Fréchet distance and DTW consider sequences and, intuitively, minimize an objective function over all ways to monotonously traverse two trajectories from start to end without skipping any part of the sequence. As these measures minimize over all ways to traverse two sequences in the way described above, we also call them *traversal-based* measures. As stated above, The difference between the Fréchet distance and DTW is then that for the former, the objective function to be minimized is the maximal distance (i.e., the bottleneck), while for the latter the objective function to be minimized is the sum (or integral) of all distances during the traversal.

The Fréchet distance as well as DTW can each be further subdivided into a discrete and a continuous version, ending up with four different measures: the discrete Fréchet distance, continuous Fréchet distance (often only referred to as the Fréchet distance), dynamic time warping, and continuous dynamic time warping (CDTW or also called the *average Fréchet distance*). In the discrete versions, the traversals are defined such that we jump from vertex to vertex, while in the continuous version we continuously traverse the line segments of the sequence. As the Fréchet distance is a bottleneck measure, the definition in the discrete as well as continuous setting is very natural: it simply evaluates to the maximal distance that occurs during the traversal. However, this is not the case for DTW. While evaluating the sum is straightforward, it is not clear what the best definition for the integral is as one can take the path length in parameter space into account in different ways. For a detailed discussion of the different possibilities, see Chapter 6 of [57].

Another variation that can be introduced to all of the distance measures mentioned above is to make them translation-invariant. The canonical way to achieve translational invariance is by minimizing these measures over all translations of one of the objects.[1] This notion enables a comparison that is merely based on the shape but not on the position of the objects, which is desirable in many applications. Distance measures of this variant are commonly called *under translation*, e.g., Fréchet distance under translation. Translation-invariant measures that are relevant for this thesis are *Hausdorff distance under translation*, *Fréchet distance under translation*, and *dynamic time warping under translation*. While there is considerable literature for the first two, dynamic time warping under translation has not been considered yet to the best of my knowledge. A possible reason is that dynamic time warping under translation in Euclidean space contains the geometric median problem as a special case for which there is no known exact algorithm [29].

String similarity measures like edit distance and longest common subsequence were also generalized to point sequences [68, 154], however, they tend to not perform as well as the Fréchet distance and dynamic time warping on geometric data as noted in [150]. For an experimental comparison of sequence similarity measures see, e.g., [150] and [73].

1.2.3 Near Neighbor Searching

While computing the similarity between two objects is an important task on its own, it also serves as an important building block in similarity search. A common problem in this setting is *nearest neighbor* searching where we are given a set of objects $\mathcal{P}$ that we preprocess into a data structure to subsequently query this data structure with an object Q of the same type and the data structure should return the object in $\mathcal{P}$ that is closest to Q under a prespecified distance measure. Often one does not only want to retrieve the nearest neighbor but all neighbors that are in a certain distance. This problem is called the *(fixed radius) near neighbors* problem and is usually formulated as follows: Given a set of objects $\mathcal{P}$ and a distance δ, preprocess $\mathcal{P}$ into a data structure such that it allows queries where an object Q is given and the data structure should return all objects in $\mathcal{P}$ that are in distance at most δ from Q. Unfortunately, such data structures often have to be approximate to allow for reasonably performant preprocessing and query times. This leads us to the so called *(fixed radius) approximate near neighbors* problem which can be described as follows: Given an input set $\mathcal{P}$, a radius δ, and an approximation factor c, we want to answer queries where a query object Q is given and we have to return all objects in $\mathcal{P}$ that are in distance at most δ from Q, while we must not return any objects that are in distance larger than $c\delta$ (note that for objects in the distance range $(\delta, c\delta]$ reporting as well as not reporting is acceptable). We drop the “fixed radius” part in the remainder as we only consider near neighbors problems of this sort. Furthermore, the above near neighbors problems can also be formulated as *near neighbor* problems, where just a single object in $\mathcal{P}$ that has distance at most $c\delta$ to Q must be reported if any object with distance at most δ exists. This is the version that we consider in this thesis and we simply refer to it by ANN. Note that the different versions of near(est) neighbor(s) searching are strongly related [24].

[1]As we only consider measures in spaces where only the relative position of the two objects that we want to measure the distance of matters, it is insignificant if we translate the first or the second object.

1.3 State of Research per Area

Let us sketch state of upper bounds, lower bounds, and algorithm engineering with respect to point sets and sequences. Instead of giving a complete overview of the results here for the respective distance measures, we draw a coarse picture to give a better general idea. In Section 1.4 we then give some more details. We additionally briefly mention the contributions of this thesis to the different areas, but defer the details of the contributions to Sections 1.4 and 3.

1.3.1 Upper Bounds

Upper bounds by far constitute the largest body of research compared to lower bounds and algorithm engineering when it comes to point and sequence similarity measures. We now give an overview of upper bounds for the measures introduced above, focusing on exact computation. Furthermore, we assume that the dimension is constant to simplify the overview. For Fréchet distance, dynamic time warping, Hausdorff distance, and Hausdorff distance under translation there are classic upper bounds. For the (discrete and continuous) Fréchet distance and (discrete) dynamic time warping, there is a dynamic programming approach on the parameter space that solves the problems in (near-)quadratic time in the curve complexity [19, 34, 89, 153]. For the Hausdorff distance, there is a classic near-linear time algorithm in the Euclidean plane [17]. Upper bounds for Hausdorff distance under translation in the plane were also given roughly 30 years ago [72, 113]. None of the problems that the above works solve had a polynomial time improvement for the general case after their publication. Thus, to resolve this line of research, lower bounds are necessary to show that no improvements are possible or to give a hint at where improvements might still be possible.

For other measures like the discrete Fréchet distance under translation and continuous dynamic time warping, the previously best (i.e., before the results in this thesis) upper bounds were only recently found [33, 120]. For the continuous Fréchet distance under translation the best known algorithms are from 2001 [22, 88]. For continuous dynamic time warping no exact algorithm is known [35], but only approximation algorithms [36, 127]. Thus, both upper and lower bounds are still waiting to be discovered.

In the ANN setting there are results for almost all of the distance measures that we introduced, see for example [85, 91, 93, 117, 125]. Let us focus on the Fréchet distance. While there is a series of results for the discrete Fréchet distance [78, 86, 90, 93, 117, 135], the continuous Fréchet distance was waiting for its first result with running times dependent on arc length until 2021 [85].

Contributions. Part I of this thesis is dedicated to upper bounds. There, we present two upper bounds that were first published in [43] and [39]. In Chapter 5, we give an upper bound for the discrete Fréchet distance under translation: a polynomial time improvement over [33]. In Chapter 4, we give multiple upper bounds for the continuous Fréchet ANN problem in one dimension, significantly expanding the results and techniques of [85].

1.3.2 Lower Bounds

Before giving an overview of lower bounds for point and sequence similarity measures, let us first consider lower bounds in general. While conditional lower bounds rely on hardness assumptions and thereby are only true *conditional* on these assumptions being true, unconditional lower bounds show fundamental running time or space barriers in general or for specific models of computation. Naturally, unconditional lower bounds are stronger in their statements as they do not rely on assumptions. However, this strength has the downside of them being significantly harder to prove. In this sense, unconditional lower bounds remain very low and for most problems we seem to be very far from proving unconditional lower bounds that match their upper bounds. Actually, no techniques for proving lower bounds beyond $\Theta(n \log n)$ are known [37]. Widgerson summarizes the current situation in [159] as follows:

> Concluding, I view the mystery of the difficulty of proving (even the slightest non-trivial) computational difficulty of natural problems to be one of the greatest mysteries of contemporary mathematics.

However, turning a back on lower bounds in general due to their difficulty would leave us completely in the dark regarding the tightness of upper bounds. While conditional lower bounds do not show fundamental barriers, they can structure the complexity landscape and by that at least reduce it to a small number of core problems that are difficult. More precisely, these lower bounds are based on popular hypotheses in complexity theory and intuitively state: "a significant improvement of the running time is only possible if there is a breakthrough for another well-studied problem". This is very similar to showing *NP*-hardness to rule out polynomial time algorithms, assuming $P \neq NP$, but in the polynomial time regime. In particular, instead of uncertainty about the best possible running time of many problems, we reduce them to a small set of *core problems* which then capture the hardness. Only if a core problem can be solved faster, we can also obtain better running times for the problems we reduced it to.

While the Strong Exponential Time Hypothesis (SETH) [115] is arguably the most popular conjecture used to show lower bounds (potentially via Orthogonal Vectors [160]), fine-grained complexity theory originated in computational geometry where the 3Sum problem was first used to rule out subquadratic running times based on its conjectured hardness [97]. Another popular hardness conjecture from fine-grained complexity theory — which we do not use in this thesis — is based on the All Pairs Shortest Paths problem [141, 161].

For point set and sequence similarity measures, for a long time there were no lower bounds that complemented the classical upper bounds. The first lower bounds were given by Bringmann in [38] for the discrete and continuous Fréchet distance. This result was then followed by lower bounds for dynamic time warping [2, 42]. All of these lower bounds are based on SETH. I am not aware of any lower bounds for the translation-invariant similarity measures that were published, apart from the works included in this thesis.

Contributions. Part II of this thesis is dedicated to lower bounds. There, we show multiple lower bounds for the Hausdorff distance under translation, the Fréchet distance under translation, and continuous Fréchet ANN in one dimension. A large part of these lower bounds are tight. These lower bounds were first presented in [49], [43], and [39]. All of the above lower bounds are based on SETH, except for one for Hausdorff distance under translation that is based on 3Sum. More precisely, the lower bounds that are based on SETH are all based on one of the k-OV Hypotheses, which are generalizations of the Orthogonal Vectors Hypothesis.

1.3.3 Algorithm Engineering

For a great overview of algorithm engineering — its history and also its scope — see [146]. Algorithm engineering can be seen as a reaction on the growing gap between algorithmics and practical applications. Already in the late 1980s this gap was recognized and first efforts to close it were conducted. In the late 1990s and early 2000s, the workshops and conferences which are now the main venues for publication in this area came into existence. Since then algorithm engineering had significant contributions and is still a very active community within theoretical computer science.

In computational geometry, the foundation for today's landscape of software libraries was laid at the end of the 1980s when multiple institutions started developing their own algorithms libraries and simultaneously joined forces in several EEC/EU projects (ALCOM [12], ALCOM II [14], ALCOM-IT [15], and ALCOM-FT [13]) between 1989 and 2003. This collaboration led to an EU project in 1996 called CGAL [60], establishing the CGAL library [151], which probably is the most powerful computational geometry library that currently exists. For a detailed overview of the history of theoretical computer science in Europe around that time, see [28].

Considering computing the similarity of point sets and sequences, there was only limited work in the area of algorithm engineering. While the Hausdorff distance between point sets has a very fast algorithm using standard tools from computational geometry (which are also included in these libraries), the other similarity measures do not have theoretical algorithms that satisfy the desired practical performance constraints. As DTW is heavily used in practice, there is a significant need for performant implementations, and thus there are multiple attempts towards it [121, 122, 140, 147, 148]. For the Fréchet distance, the GIS cup 2017 [158] inspired the work on algorithm engineering for computing the Fréchet distance exactly, which in particular resulted in the three winning submissions [30, 54, 87]. For the translation-invariant distance measures and CDTW there are no existing implementations that I am aware of that compute the exact value. For CDTW there only exists an implementation that computes the approximate value [36]. Thus, many problems still wait for a decently engineered solution.

Contributions. Part III of this thesis is dedicated to algorithm engineering. There, we present the engineering of the Fréchet distance and the Fréchet distance under translation that were already published in [44] and [45]. While in the presentation we focus on the continuous Fréchet distance and the discrete Fréchet distance under translation, the approaches can also be used to compute the discrete Fréchet distance and the continuous Fréchet distance under translation. In fact, the engineered implementation for the Fréchet distance also contains the code for the discrete version. No better engineered solution is known until now for both measures to the best of my knowledge.

1.4 State of Research per Measure

To give a comprehensive overview of the different measures, we now consider the state of research for each measure separately. We mainly focus on the measures and results that are relevant for this thesis. In particular, we do not consider the Hausdorff distance, as it has a classical near-linear time algorithm. Instead, we first consider the Hausdorff distance under translation.

1.4.1 Hausdorff Distance Under Translation

Given two point sets in the plane of size n and m, the Hausdorff distance under translation can be computed in time $\mathcal{O}(nm \log^2 nm)$ for the L_1 and L_∞ metric [72], and in time $\mathcal{O}(nm(n+m)\log nm)$ for the L_2 metric [113]. I am not aware of any lower bounds for this problem previous to the works in this thesis, not even conditional on a plausible hypothesis. The only results in this direction are $\Omega(n^3)$ lower bounds on the arrangement size [72] and on the number of connected components of the feasible translations [144] (for the decision problem on points in the plane with $n = m$). However, these bounds also hold for L_1 and L_∞, where they are "broken" by the $\mathcal{O}(nm \log^2 nm)$-time algorithm [72], so apparently these bounds do not give us information about the best possible running time complexity. Previously, an $n^{2-o(1)}$ lower bound was only known for the more general problem of computing the Hausdorff distance under translation of sets of *segments* in the case that both sets have size n — a problem for which the best known algorithm runs in time[2] $\tilde{\mathcal{O}}(n^4)$ [31].

Contributions. In this thesis we describe two lower bounds for the Hausdorff distance under translation in the plane. First, we show that no polynomial time improvement over the classical algorithm for L_1 and L_∞ is possible under the Strong Exponential Time Hypothesis. Second, for L_2 we show a lower bound of $n^{2-o(1)}$ under 3Sum for the imbalanced case where one point set has size n and the other $\mathcal{O}(1)$. Note that this also shows that no polynomial time improvement is possible over the classical algorithm for L_2 in this input size regime.

1.4.2 Fréchet Distance

The Fréchet distance was introduced more than 100 years ago by Maurice Fréchet [94]. The time complexity of the Fréchet distance is well understood. For the continuous Fréchet distance, Alt and Godau designed an $\mathcal{O}(n^2 \log n)$-time algorithm for polygonal curves consisting of n vertices [19]. Buchin et al. [51] improved on this result by giving an algorithm that runs in time $\mathcal{O}(n^2\sqrt{\log n}(\log\log n)^{3/2})$ on the Real RAM and $\mathcal{O}(n^2(\log\log n)^2)$ on the Word RAM. The first algorithm for the discrete Fréchet distance ran in time $\mathcal{O}(n^2)$ [89], which was later improved to $\mathcal{O}\big(n^2\frac{\log\log n}{\log n}\big)$ [11]. On the hardness side, an $n^{2-o(1)}$ lower bound conditional on the Strong Exponential Time Hypothesis was shown in 2014 by Bringmann [38] for the discrete and continuous Fréchet distance. This lower bound even holds for one-dimensional curves [48, 56]. Very recently matching bounds on the problem of subtrajectory clustering under the Fréchet distance were

[2]By $\tilde{\mathcal{O}}$-notation we ignore logarithmic factors in n and m.

shown [107]. Furthermore, Abboud and Bringmann [3] showed that any $\mathcal{O}(n^2/\log^{17+\varepsilon} n)$-time algorithm for the discrete Fréchet distance would prove novel circuit lower bounds. Many extensions and variants of the Fréchet distance have been studied, e.g., generalizing from curves to other types of objects, replacing the ground space $\mathbb{R}^d$ by more complex spaces, and many more (see, e.g., [18, 32, 52, 62, 75, 81, 128]).

Considering the amount of theoretical research about the Fréchet distance that was produced by the algorithms community, it is surprising that algorithm engineering of this distance measure was mostly initiated as late as 2017 by the GIS cup of that year [158]. Fortunately, the GIS cup was very successful and lead to the three performant winning submissions [30, 54, 87]. However, none of these implementations resolved the main performance bottleneck — the construction of the free-space diagram — in a satisfying manner.

Restricted input models. To avoid the quadratic complexity of the Fréchet distance, which even (conditionally) holds for approximations with a factor less than 3 in one dimension [56], several realistic input models were suggested that have faster approximation algorithms depending on the parameter of the model. I want to explicitly mention three of these realistic input models: κ-straight, κ-bounded, and c-packed. A curve is κ-straight if for all points p, q on the curve, it holds that the distance on the curve between p and q is at most κ times the Euclidean distance between p and q, see [20]. A curve is κ-bounded if for all points p, q on the curve, it holds that the subcurve between p and q is contained in the union of the balls around p and q with radius $\kappa/2$ times their Euclidean distance, see [21]. A curve is c-packed if for all $r > 0$, any ball of radius r intersects the curve at most on a total length of $c \cdot r$, see [82]. For constant ε, dimension d, and parameter c or κ, we can compute a $(1+\varepsilon)$-approximation of the Fréchet distance in near-linear time for these realistic input models. For c-packed curves, it was also shown that this cannot be improved for approximation factors close to 1 [38].

Approximate Near Neighbor. Recall that the ANN problem that we consider in this thesis is defined as follows: Given a set of curves $\mathcal{P}$, a radius δ, and an approximation factor c, we want to preprocess $\mathcal{P}$ to answer queries with a curve Q such that we either return a curve in $\mathcal{P}$ in distance at most $c\delta$ from Q or we report that no curve in distance at most δ from Q exists in $\mathcal{P}$. Let m be the maximal complexity of curves in $\mathcal{P}$ and k the maximal complexity of Q. For the discrete Fréchet distance, there is a significant amount of work on ANN data structures. Most notably, there is a $(1+\varepsilon)$-ANN data structure with preprocessing time $n \cdot \mathcal{O}(1/\varepsilon)^{dk} + \mathcal{O}(nm)$ and query time $\mathcal{O}(kd)$ [93]. For continuous Fréchet ANN, for a long time there were only results that provided a data structure whose preprocessing and query times were dependent on the arc length of the curves [77]. In a first result that is restricted to the one-dimensional case but independent of the arc length, a $(2+\varepsilon)$-ANN with preprocessing time $\mathcal{O}(nm) \cdot \mathcal{O}(m/k\varepsilon)^k$ and query time $\mathcal{O}(k \cdot 2^k)$, and a $(5+\varepsilon)$-ANN with preprocessing time $\mathcal{O}(nm) \cdot \mathcal{O}(1/\varepsilon)^k$ and query time $\mathcal{O}(k)$ were shown [85].

Contributions. In this thesis, we describe an engineered solution for the Fréchet distance which currently is by far the fastest practical algorithm to the best of my knowledge. Its main idea is to use a divide and conquer approach on the free-space diagram — the crucial technical contribution of the classical algorithm of [19]. Divide and conquer enables massive pruning on the running time intensive instances and thus leads to running time improvements of several orders of magnitude. On benchmarks, we improve the running time by two orders of magnitude compared to the winning submission of [158].

On the theoretical side, we describe upper bounds as well as lower bounds for the continuous Fréchet ANN problem. More precisely, we give a $(1+\varepsilon)$-ANN in one dimension with the same preprocessing and query times as the previous $(2+\varepsilon)$-ANN data structure, we give multiple $(2+\varepsilon)$-ANNs that provide a tradeoff between preprocessing and query time, and a $(3+\varepsilon)$-ANN with the same preprocessing and query times as the previous $(5+\varepsilon)$-ANN. While this significantly improves on the results of [85], we also show that many of our upper bounds are essentially tight under the Strong Exponential Time Hypothesis, reducing from a novel one-sided sparse Orthogonal Vectors problem which is of independent interest.

1.4.3 Fréchet Distance Under Translation

The results for the continuous and discrete Fréchet distance under translation differ significantly. The first two algorithm for the *continuous* Fréchet distance under translation were independently given by [88] and [22], who designed algorithms in the plane with running time $\tilde{\mathcal{O}}(n^{10})$ and $\tilde{\mathcal{O}}(n^8)$, respectively. Both also present approximation algorithms, for example, a $(1+\varepsilon)$-approximation with running time $\mathcal{O}(n^2/\varepsilon^2)$ in the plane is presented in [22] and a similar result in [88]. In three dimensions there exists an algorithm with running time $\tilde{\mathcal{O}}(n^{11})$ [157]. Exactly computing the *discrete* Fréchet distance under translation was first studied by Jiang et al. [120] who designed an $\tilde{\mathcal{O}}(n^6)$-time algorithm in the plane. The previously (before the results of this thesis) best known algorithm for the discrete Fréchet distance under translation in the plane is due to Ben Avraham et al. [33]. It is an improvement of the algorithm by Jiang et al. [120] and runs in time $\tilde{\mathcal{O}}(n^5)$. A different translation-invariant variant of the Fréchet distance is presented in [76] and fast algorithms are provided.

Contributions. In this thesis, we describe the currently best upper bound, the lower bound, and an engineered implementation for the discrete Fréchet distance under translation in the plane [43, 45]. Our upper bound improves the best running time from $\tilde{\mathcal{O}}(n^5)$ to $\tilde{\mathcal{O}}(n^{4.667})$ and we show a lower bound of $n^{4-o(1)}$ conditional on SETH. Interestingly, this lower bound matches the size of the arrangement that all algorithms for this problem build and it thus suggests that this arrangement is necessary.

Despite this discouraging lower bound, we engineered an implementation that is the first for this problem to the best of my knowledge. Despite the high theoretical running times, we manage to give an implementation that is very fast on practical data. In fact, the running time for a distance comparison between two practical curves is in the milliseconds range, even making such computations usable in real-time scenarios.

1.4.4 (Continuous) Dynamic Time Warping

Dynamic time warping was, to the best of my knowledge, introduced in a slightly different form by Vintsyuk in [153], who also discovered the classic quadratic time dynamic programming algorithm. Later DTW was stated in today's form by [34]. Unless OV and

SETH fail, no polynomial improvements to the running time are possible [2, 42], but polylogarithmic improvements were achieved by [103]. Due to its practical significance, DTW has seen a lot of algorithm engineering work [121, 122, 140, 147, 148]. Dynamic time warping under translation in the Euclidean space is not known to have an exact algorithm as it contains the geometric median problem for which it is not known whether an exact algorithm exists [29]. Research for this measure is still largely open, e.g., approximation algorithms or results for other metrics. The complexity of CDTW is still not well understood and published results are quite scarce. A nice overview of CDTW is given in [57]. Currently there are only three approximation algorithms [35, 36, 127] but no exact algorithm is known. However, also no hardness result is known and thus the complexity still remains open.

Contributions. While there are no contributions for these measures in this thesis, I am a co-author of one of the few works on CDTW [36] and there is more work in progress.

CHAPTER 2

Notation and Preliminaries

We now formally define the most important objects and notation that we use in this thesis. To denote index sets, we often use $[n] \coloneqq \{1, \dots, n\}$. In this thesis we mostly focus on geometric objects that consist of points in $\mathbb{R}^d$ — often considering the special case $d = 1$ or $d = 2$ — and we measure the distance between points in some p-norm. Given a point $x \in \mathbb{R}^d$, its p-norm is defined as

$$\|x\|_p \coloneqq \left(\sum_{i \in [d]} |x_i|^p \right)^{\frac{1}{p}},$$

and the distance between two points $x, y \in \mathbb{R}^d$ is then simply $\|x - y\|_p$. If the p-norm is clear from the context, we may omit the index and just write $\|\cdot\|$. For a point $p \in \mathbb{R}^d$, we use $p + \tau$ to denote the point p translated by $\tau \in \mathbb{R}^d$. For a point set $A \subset \mathbb{R}^d$ and a translation $\tau \in \mathbb{R}^d$, we define the translated point set as $A + \tau \coloneqq \{a + \tau \mid a \in A\}$. Furthermore, most distance notions that we work with are Lipschitz when one of the objects is translated. We say that a distance measure d is L-Lipschitz if for all x, y, τ, we have that

$$|d(x, y) - d(x, y + \tau)| \leq L \cdot \|\tau\|,$$

where the norm $\|\cdot\|$ depends on the context.

2.1 Trajectories and Distance Measures

In this section we define two different notions of distance measures: the Hausdorff distance and its variants as a similarity measure between point sets, and the Fréchet distance and its variants as a similarity measure between trajectories.

Hausdorff distance. Let $A, B \subset \mathbb{R}^d$ be two point sets. The probably most natural distance measure between two such sets is the Hausdorff distance. Intuitively, the Hausdorff distance results from associating each point with the closest point in the other set and then taking the maximum distance over all such associations. The *directed Hausdorff distance* is defined as

$$d_{\vec{H}}(A, B) \coloneqq \max_{a \in A} \min_{b \in B} \|a - b\|.$$

Note that the directed Hausdorff distance measures the distance from A to B but not from B to A. In particular, note that it is not symmetric. A symmetric variant of the Hausdorff distance, the *undirected Hausdorff distance*, is defined as

$$d_H(A, B) \coloneqq \max\{d_{\vec{H}}(A, B), d_{\vec{H}}(B, A)\}.$$

Note that the directed Hausdorff distance is at most the undirected Hausdorff distance by definition, i.e., $d_{\vec{H}}(A,B) \leq d_H(A,B)$.

Both of the above distance measures can be modified to a version which is invariant under translation by minimizing the distance between A and any translation of B. Therefore, the *directed Hausdorff distance under translation* is defined as

$$d^T_{\vec{H}}(A,B) := \min_{\tau \in \mathbb{R}^d} d_{\vec{H}}(A, B+\tau),$$

and the *undirected Hausdorff distance under translation* is defined as

$$d^T_{H}(A,B) := \min_{\tau \in \mathbb{R}^d} d_{H}(A, B+\tau).$$

Again, it holds that $d^T_{\vec{H}}(A,B) \leq d^T_H(A,B)$.

For the Hausdorff distance (without translation) the undirected distance is at most as hard to compute as the directed distance, as the undirected distance can be calculated using two calls to an algorithm computing the directed distance.[1] However, note that for the Hausdorff distance under translation, we cannot just compute the directed distance twice and then obtain the undirected distance as we have to take the maximum over the directed distance for the same translation. The directed Hausdorff distance under translation is still at least as hard as the undirected Hausdorff distance under translation by another argument. The same idea was already used for $d = 1$ in [142].

Observation 2.1. *Let $f(n,m)$ be the running time to compute the directed Hausdorff distance under translation of two points sets of size n and m. There is an algorithm for the undirected Hausdorff distance under translation with running time $\mathcal{O}(f(n+m, n+m))$.*

Proof. Let $A, B \subset \mathbb{R}^d$ be the input sets and let D be the diameter of $A \cup B$, i.e., the largest distance between any pair of points. Let A', B' be copies of A, B reflected at the same arbitrary point and jointly translated such that the closest points of $A \cup B$ and $A' \cup B'$ are at least $3D$ apart. Note that $d_{\vec{H}}(B+\tau, A) = d_{\vec{H}}(B'-\tau, A') = d_{\vec{H}}(B', A'+\tau)$ for any $\tau \in \mathbb{R}^d$. Due to $A \cup B$ and $A' \cup B'$ being far apart, it thus holds that

$$d^T_H(A,B) = \min_{\tau \in \mathbb{R}^d} \max\{d_{\vec{H}}(A, B+\tau), d_{\vec{H}}(B+\tau, A)\} = d^T_{\vec{H}}(A \cup B', B \cup A').$$

Observing that $|A \cup B'| = |B \cup A'| = n + m$ completes the proof. □

Trajectories. A *trajectory* π is defined by an ordered sequence of points $(\pi_1, \dots, \pi_n)$ with $\pi_i \in \mathbb{R}^d, i \in [n]$, which are also called the *vertices* of the trajectory. If we consider the trajectory as a continuous object, then consecutive vertices are connected by line segments and we call it a *continuous trajectory*. If we only consider the vertices, then we call it a *discrete trajectory*. We define the *complexity* of a trajectory π as the number of vertices and we denote it by $|\pi|$. Depending on what is more convenient, we either use π, σ to denote trajectories and use π_i with $i \in [|\pi|]$ and σ_j with $j \in [|\sigma|]$ to refer to their vertices. Or, if we have multiple trajectories, we often use $P_1, \dots, P_n$ to refer to them

[1] Actually, for point sets of similar size the directed Hausdorff distance is also at most as hard to compute as the undirected Hausdorff distance (thus, they are equally hard to compute), as $d_{\vec{H}}(A,B) = d_H(A \cup B, B)$.

and use $p_1, \dots, p_{|P|}$ to refer to the vertices of a specified trajectory P. In data structure settings, we often denote query trajectories by Q.

To refer to specific points (not necessarily vertices) on a continuous trajectory, it is helpful to parametrize it using a one-dimensional interval. A *parametrization* of a trajectory π is a monotone bijection between a one-dimensional range $\mathcal{I} = [a, b]$ and the curve. Let ℓ be the length of the trajectory, i.e., the sum of the lengths of the line segments. In this thesis, we assume that we parametrize the trajectory by mapping $a + x \cdot (b - a)$ for each $x \in [0, 1]$ to the point on π after traversing it for a distance $x \cdot \ell$. Depending on what is more convenient, we either parametrize π using $\mathcal{I} = [0, 1]$ or $\mathcal{I} = [1, |\pi|]$. To this end, for an $x \in \mathcal{I}$, we use $\pi(x)$ to refer to points on the trajectory. To denote the subcurve of π between $x \in \mathcal{I}$ and $x' \in \mathcal{I}$ — where $\pi(x)$ and $\pi(x')$ are not necessarily vertices of π — we use the notation $\pi[x, x']$.

We say that a trajectory π is *degenerate* if there are three consecutive vertices $\pi_{i-1}, \pi_i, \pi_{i+1}$, such that π_i lies on the segment between π_{i-1} and π_{i+1}. In this case, we call π_i a degenerate vertex of this curve. Given a sequence of points $\pi_1, \dots, \pi_n$, we can define a non-degenerate curve by omitting degenerate vertices. We denote the resulting curve by $\langle \pi_1, \dots, \pi_n \rangle$. For any trajectories $\pi = (\pi_1, \dots, \pi_n)$ and $\sigma = (\sigma_1, \dots, \sigma_m)$ we let $\pi \circ \sigma$ denote the polygonal curve $\langle \pi_1, \dots, \pi_n, \sigma_1, \dots \sigma_m \rangle$, that is the *concatenation* of π and σ. For n polygonal curves $P_1, \dots, P_n$, we denote by $\bigcirc_{i=1}^{n} P_i$ the concatenation $P_1 \circ P_2 \circ \dots \circ P_n$. Furthermore, given any translation vector $\tau \in \mathbb{R}^d$, we denote by $\pi + \tau$ the trajectory defined by the sequence of points $(\pi_1 + \tau, \dots, \pi_{|\pi|} + \tau)$. Finally, we use the term *curve* and *trajectory* interchangeably in this thesis.

Traversals. To define the Fréchet distance and its variants, we first have to define the notion of traversal. Intuitively, a traversal is a continuous (meaning, we do not jump), simultaneous, non-backwards movement over two trajectories where the respective speed can vary arbitrarily, starting in the pair of start vertices and ending in the pair of end vertices. More formally, let $\pi : \mathcal{I} \to \mathbb{R}^d$ and $\sigma : \mathcal{I}' \to \mathbb{R}^d$ be two trajectories. A *continuous traversal* is simply a function $\phi : [0, 1] \to \mathcal{I} \times \mathcal{I}'$ with

$$\phi(t) \mapsto (f(t), g(t)),$$

where the re-parametrizations $f : [0, 1] \to \mathcal{I}$ and $g : [0, 1] \to \mathcal{I}'$ are surjective and (not necessarily strictly) monotone functions. Note that we can also give a traversal by just providing the pair of functions (f, g). Note that, for each point on one of the two trajectories, this induces a correspondence to a subcurve (which might just be a single point) on the other curve. For discrete trajectories, on the other hand, during a traversal we can either jump in one of the trajectories to the next vertex or in both. More precisely, defining $n = |\pi|$ and $m = |\sigma|$, a *discrete traversal* is a sequence $\phi = (\,(s_1, t_1), \dots, (s_k, t_k)\,)$ of pairs $(s_i, t_i) \in [n] \times [m]$ such that $(s_1, t_1) = (1, 1)$, $(s_k, t_k) = (n, m)$ and $(s_{i+1}, t_{i+1}) \in \{(s_i + 1, t_i), (s_i, t_i + 1), (s_i + 1, t_i + 1)\}$ for all $1 \leq i < k$. Having defined traversals, we can now define the Fréchet distance.

Fréchet distance. For any curves $\pi = (\pi_1, \dots, \pi_n), \sigma = (\sigma_1, \dots, \sigma_m)$, we define their *discrete Fréchet distance* as

$$\mathrm{d_{dF}}(\pi, \sigma) := \min_{\phi = ((s_1, t_1), \dots, (s_k, t_k))} \max_{i \in [k]} \left\| \pi_{s_i} - \sigma_{t_i} \right\|,$$

where the minimum ranges over all discrete traversals ϕ of π and σ. The *continuous Fréchet distance* is defined as

$$\mathrm{d_F}(\pi,\sigma) \coloneqq \min_{\phi=(f,g)} \max_{t\in[0,1]} \|\pi(f(t)) - \sigma(g(t))\|,$$

where the minimum ranges over all continuous traversals ϕ of π and σ. As is common, we often use *Fréchet distance* for the continuous version and *discrete Fréchet distance* for the discrete one. It is well-known that $\mathrm{d_F}(\pi,\sigma) \le \mathrm{d_{dF}}(\pi,\sigma)$ [89]. The classical algorithm to compute the Fréchet distance on polygonal curves is by Alt and Godau [19]. We state their result here for the decision version.

Theorem 2.2 ([19])**.** *There is an algorithm which, given polygonal curves π, σ and a threshold parameter $\delta > 0$, decides in $\mathcal{O}(|\pi|\cdot|\sigma|)$ time whether $\mathrm{d_F}(\pi,\sigma) \le \delta$.*

The same result also holds for the discrete Fréchet distance [89]. There we can even compute the distance, not just the decision, in the time stated in Theorem 2.2.

Fréchet distance under translation. Both of the above distance measures can be modified to a version which is invariant under translation by minimizing the distance between π and any translation of σ. More precisely, we define the *discrete Fréchet distance under translation* as

$$\mathrm{d^T_{dF}}(\pi,\sigma) \coloneqq \min_{\tau\in\mathbb{R}^d} \mathrm{d_{dF}}(\pi,\sigma+\tau),$$

and the *continuous Fréchet distance under translation* as

$$\mathrm{d^T_F}(\pi,\sigma) \coloneqq \min_{\tau\in\mathbb{R}^d} \mathrm{d_F}(\pi,\sigma+\tau).$$

As for the Fréchet distance, we also have that $\mathrm{d^T_F}(\pi,\sigma) \le \mathrm{d^T_{dF}}(\pi,\sigma)$ for the Fréchet distance under translation. Furthermore, the discrete and continuous Fréchet distance under translation is 1-Lipschitz in the translation.

Observation 2.3 (Lipschitz property)**.** *Given two trajectories $\pi,\sigma : [0,1] \to \mathbb{R}^d$, the continuous (respectively, discrete) Fréchet distance under translation $f(\tau) = \mathrm{d_F}(\pi,\sigma+\tau)$ (respectively, $f(\tau) = \mathrm{d_{dF}}(\pi,\sigma+\tau)$) is 1-Lipschitz, i.e., $|f(\tau) - f(\tau+\tau')| \le \|\tau'\|$ for all $\tau,\tau' \in \mathbb{R}^d$.*

Proof. Note that for any $\pi(x),\sigma(y),\tau,\tau' \in \mathbb{R}^d$ with $x,y \in [0,1]$, we have

$$\big|\|\pi(x) - (\sigma(y)+\tau)\| - \|\pi(x) - (\sigma(y)+\tau+\tau')\|\big| \le \|\tau'\|$$

by triangle inequality. Thus, the maximal distances in any traversal ϕ for $\pi,\sigma+\tau$ and $\pi,\sigma+\tau+\tau'$ differ by at most $\|\tau'\|$, which immediately yields the observation. □

Free-space diagram. The free-space diagram is the basis of the algorithm of [19], where it was also first defined. Given two trajectories π and σ and a distance δ, the free-space diagram is defined as the set of all pairs of indices of points from π and σ that are in distance at most δ, i.e.,

$$F \coloneqq \{(p,q) \in [1,n]\times[1,m] \mid \|\pi(p) - \sigma(q)\| \le \delta\}.$$

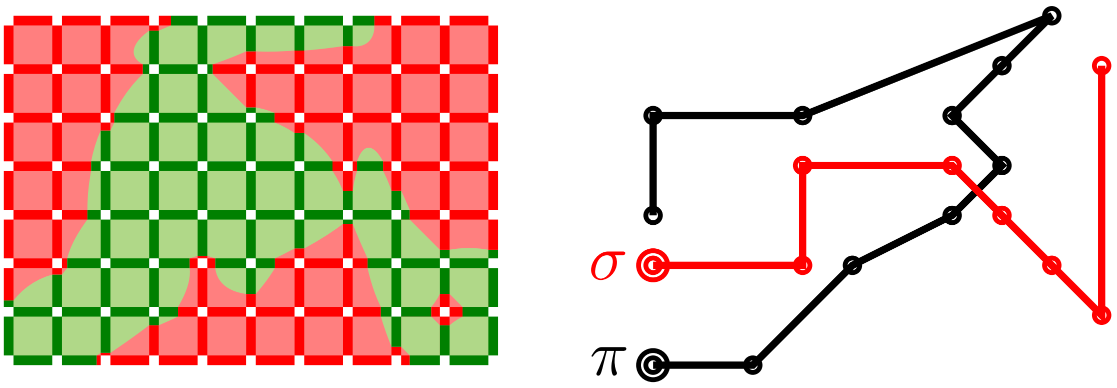


Figure 2.1: Example of a free-space diagram for curves π (black) and σ (red). Curve π is on the horizontal axis of the free-space diagram, while σ is on the vertical axis. The doubly-circled vertices mark the start. The free-space, i.e., the pairs of indices of points which are close, is colored green. The non-free areas are colored red. The threshold distance δ is roughly the distance between the first vertex of σ and the third vertex of π.

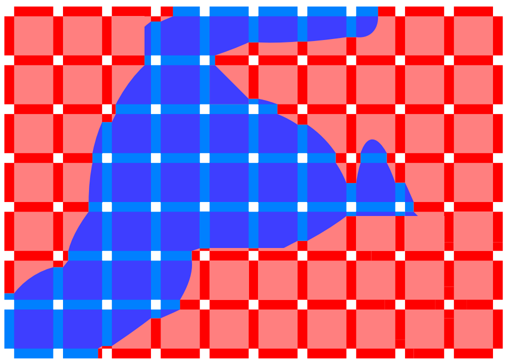


Figure 2.2: Reachable space of the free-space diagram in Figure 2.1. The reachable part is blue and the non-reachable part is red. Note that the reachable part is a subset of the free-space.

For an example see Figure 2.1. A *path* from a to b in the free-space diagram F is defined as a continuous mapping $P : [0, 1] \to F$ with $P(0) = a$ and $P(1) = b$. A path P in the free-space diagram is *monotone* if $P(x)$ is component-wise at most $P(y)$ for any $0 \leq x \leq y \leq 1$. The *reachable space* is then defined as

$$R := \{(p, q) \in F \mid \text{there exists a monotone path from } (1, 1) \text{ to } (p, q) \text{ in } F\}.$$

Figure 2.2 shows the reachable space for the free-space diagram of Figure 2.1. It is well known that $\mathrm{d_F}(\pi, \sigma) \leq \delta$ if and only if $(n, m) \in R$.

Note that if we transfer this concept to the discrete Fréchet distance, we end up with a binary matrix of size $n \times m$. More precisely, we define a matrix $M \in \{0, 1\}^{n \times m}$ where $M_{i,j} = 1$ if $\|\pi_i - \sigma_j\| \leq \delta$, and $M_{i,j} = 0$ otherwise. We now analogously have a monotone path from $M_{1,1}$ to $M_{n,m}$ that only uses the 1-entries and can only go to neighboring fields (including diagonally) if and only if the discrete Fréchet distance between π and σ is at most δ. Note that this matrix can also be interpreted as a directed acyclic grid graph.

2.2 Lower Bounds

We now define the problems and hypotheses that are relevant for the lower bounds in this thesis, starting with the probably most popular one: the Strong Exponential Time Hypothesis.

Strong Exponential Time Hypothesis. The *Strong Exponential Time Hypothesis (SETH)* was introduced by Impagliazzo and Paturi [115] and essentially postulates that there is no exponential-time improvement over exhaustive search for the Satisfiability Problem. Note that SETH implies $P \neq NP$ and it is thus a stronger assumption.

Hypothesis 2.4 (Strong Exponential Time Hypothesis (SETH)). *For any $\varepsilon > 0$ there exists $k \geq 3$ such that k-SAT has no $\mathcal{O}((2-\varepsilon)^n)$-time algorithm.*

Orthogonal Vectors. While the Strong Exponential Time Hypothesis is a powerful tool, encoding arbitrary SAT instances can make reductions somewhat complicated. The Orthogonal Vectors Problem is a problem with a simpler structure that cannot have a polynomial time improvement over its naive algorithm if SETH holds.

Definition 2.5 (Orthogonal Vectors Problem (OV)). *Given two sets $U, V \subset \{0,1\}^D$ with $|U| = n, |V| = m$, decide whether there exist $u \in U$ and $v \in V$ that are orthogonal, i.e., $u \cdot v = 0$.*

As we can reduce from SAT to OV, it is natural to formulate a corresponding hypothesis, which also is implied by SETH [160].

Hypothesis 2.6 (Orthogonal Vectors Hypothesis (OVH)). *The Orthogonal Vectors problem cannot be solved in time $\mathcal{O}((nm)^{1-\varepsilon} poly(D))$ for any $\varepsilon > 0$.*

In fact, one can even reduce from SAT to a more general problem, which is called the k-Orthogonal Vectors Problem.

Definition 2.7 (k-Orthogonal Vectors Problem (k-OV)). *Given $V_1, \dots, V_k \subset \{0,1\}^D$ with $N := |V_1| = \dots = |V_k|$, decide whether*

$$\exists v_1 \in V_1, \dots, v_k \in V_k \; \forall j \in [D] \; \bigvee_{i \in [k]} v_i[j] = 0.$$

Again, we formulate the corresponding hypothesis.

Hypothesis 2.8 (k-OV Hypothesis). *For any $k \geq 2$ and $\varepsilon > 0$, there is no algorithm that runs in time $\mathcal{O}(N^{k-\varepsilon}\text{poly}(D))$ for k-OV.*

The well-known split-and-list technique due to Williams [160] shows that SETH implies the k-OV Hypothesis. Thus, any conditional lower bound that holds under the k-OV hypothesis also holds under SETH. Often we need a more precise hypothesis to show tighter hardness results. Therefore, we also consider an OV variant which, first, is hard for comparably low-dimensional inputs and, second, is hard for a large range of unbalanced cases.

Lemma 2.9. *Assume OVH holds true. For every $\alpha \in (0,1)$ and $\varepsilon > 0$ there exists a constant $c > 0$ such that there is no algorithm solving OV instances $U, V \subset \{0,1\}^D$ with $|V| = |U|^\alpha$ and $D = c \log |U|$ in time $\mathcal{O}(|U|^{1+\alpha-\varepsilon})$.*

The above statement (commonly stated in the balanced case) is also sometimes called the Low-Dimensional Orthogonal Vectors Hypothesis and it is implied by the Strong Exponential Time Hypothesis [160]. Furthermore, it is well known that *balanced* OV with sets of the same size is equally hard as *unbalanced* OV [6, 41].

3Sum. The historically first problem that was used to show fine-grained lower bounds is the 3Sum problem [97], for which there exist multiple equivalent formulations. In this thesis we use the following variant.

Definition 2.10 (3Sum)**.** *Given three sets of positive integers X, Y, Z all of size n, do there exist $x \in X, y \in Y, z \in Z$ such that $x + y = z$?*

The corresponding hypothesis conjectures a quadratic hardness of this problem.

Hypothesis 2.11 (3Sum Hypothesis)**.** *There is no $\mathcal{O}(n^{2-\varepsilon})$ algorithm for* 3Sum *for any $\varepsilon > 0$.*

Note that the naive algorithm for 3Sum is cubic and not quadratic. Thus, as opposed to OVH, the 3Sum Hypothesis does not conjecture that the naive brute-force algorithm for the problem is optimal. However, there is a problem that is simpler and equivalent to 3Sum: the convolution 3Sum problem (Conv3Sum) [63, 138].

Definition 2.12 (Conv3Sum)**.** *Given a sequence of positive integers $X = (x_0, \ldots, x_{n-1})$ of size n, do there exist i, j such that $x_i + x_j = x_{i+j}$?*

Note that for this problem the naive brute-force algorithm is conjectured to be optimal as we decrease the number of indices by one. For completeness, we note that there is a generalized family of problems called the kSum problems, however, we do not utilize them in this thesis and thus do not introduce them.

CHAPTER 3

Technical Overviews

In this chapter we give a technical overview of the different results in this thesis. For upper and lower bounds for the same problem we give a joint technical overview to show their complementary nature. We start in Section 3.1 with our upper and lower bounds of the continuous Fréchet approximate near neighbor (ANN) search for time series, which first appeared in [39], and in Section 3.2 continue with the upper and lower bound for the Fréchet distance under translation, which first appeared in [43]. Then we give a brief overview of the lower bounds for the Hausdorff distance under translation in Section 3.3, first published in [49]. Finally, we turn to algorithm engineering and in Section 3.4 give an overview of the engineered solution of the Fréchet distance from [44] and then the Fréchet distance under translation from [45] in Section 3.5.

3.1 Continuous Fréchet Approximate Near Neighbor Search

We now give an overview of the upper and lower bounds for continuous Fréchet ANN. The details of the upper bounds are then presented in Chapter 4 and the details of the lower bounds in Chapter 7. This work focuses on the special case of one-dimensional curves, which we also refer to as time series. We aim to resolve approximate near neighbor searching for this special case of the continuous Fréchet distance. We obtain strong lower bounds based on the Orthogonal Vectors Hypothesis in the regime of small approximation factors. More specifically, we differentiate a range of lower bounds for different approximation factors and preprocessing/query time. We show that our bounds are tight by devising data structures that asymptotically match the lower bounds in all cases considered. The new data structures improve upon the state of the art in several ways. For the same preprocessing and query time, we can improve the approximation factor from $(2+\varepsilon)$ to $(1+\varepsilon)$. For the same approximation factor $(2+\varepsilon)$, we get a better time complexity—in some cases we can even achieve linear preprocessing time and space.

3.1.1 Problem Definition

The central problem of this work is defined as follows.

Definition 3.1 (c-Approximate Near Neighbors problem (c-ANN))**.** *The input consists of a set $\mathcal{P}$ of n curves in $\mathbb{R}^d$, each of complexity m, and a number $2 \leq k \leq m$. Given a distance threshold $\delta > 0$ and an approximation factor $c > 1$, preprocess $\mathcal{P}$ into a data structure such that for any query curve Q of complexity k, the data structure reports as follows:*

- *if $\exists P \in \mathcal{P}$ such that $\mathrm{d_F}(P,Q) \leq \delta$, then it returns $P' \in \mathcal{P}$ such that $\mathrm{d_F}(P',Q) \leq c\delta$,*
- *if $\forall P \in \mathcal{P}$, $\mathrm{d_F}(P,Q) \geq c\delta$ then it returns "no",*
- *otherwise, it either returns a curve $P \in \mathcal{P}$ such that $\mathrm{d_F}(P,Q) \leq c\delta$, or "no".*

The assumption that all input curves have the same number of vertices m and that the queries have k vertices is mostly to simplify presentation; all our data structures are easily generalized to allow input curves of complexity *at most* m and query curves of complexity *at most* k. Note, however, that we assume the input has size in $\Omega(nm)$ and that $2 \leq k \leq m$. The case $k = 1$ is a boundary case that is easier to solve; we ignore it throughout this work.

3.1.2 State of the Art

We start by reviewing the state of the art for the *discrete* variant of the Fréchet distance. In the discrete Fréchet distance, the continuous traversal ϕ is replaced by a discrete traversal of the two point sequences, see Chapter 2 for the formal definition. The currently best known data structure for $(1+\varepsilon)$-ANN under the discrete Fréchet distance is by Filtser et al. [93]. Their data structure uses space in $n \cdot \mathcal{O}(1/\varepsilon)^{kd} + \mathcal{O}(nm)$ and query time in $\mathcal{O}(kd)$, where k denotes the complexity of the query (measured in the number of vertices), m denotes the complexity of an input curve and n denotes the number of input curves. It is an interesting question whether the same bounds can be obtained for the continuous Fréchet distance. At first glance, the discrete and continuous variants of the Fréchet distance seem very similar, but there is an important difference: while the metric space of bounded complexity curves under the discrete Fréchet distance has bounded doubling dimension, this does not hold in the continuous case, even when restricted to polygonal curves of constant complexity [83]. (A metric space has doubling dimension at most d if any ball of any radius r can be covered by 2^d balls of radius $\frac{r}{2}$.) This immediately shows that the technique employed by Filtser et al., which effectively applies a doubling oracle to the metric balls centered at input curves (more specifically, simplifications thereof), does not directly extend to the continuous Fréchet distance, since such a doubling oracle cannot exist in this case.

So the *discrete* Fréchet distance has a simple ANN that seems optimal, but there is indication that for the *continuous* Fréchet distance resolving the time complexity of ANN is more challenging. Note that it is possible to reduce the ANN problem for the continuous Fréchet distance to the ANN problem for the discrete Fréchet distance by subsampling along the continuous curves. However, it seems that this approach introduces an (otherwise avoidable) dependency on the arclength. In 2018, Driemel and Afshani [10] described data structures based on multi-level partition trees (using semi-algebraic range searching techniques) which can also be used for exact near neighbor searching under the continuous Fréchet distance. For n curves of complexity m in $\mathbb{R}^2$, their data structure uses space bounded by $n \cdot (\log\log n)^{\mathcal{O}(m^2)}$ and the query time is bounded by $\sqrt{n} \cdot (\log n)^{\mathcal{O}(m^2)}$. (If the input is restricted to curves in $\mathbb{R}$, these bounds can be slightly improved.) Recently, Driemel and Psarros [85] obtained bounds for the continuous Fréchet distance that are similar to the bounds of Filtser et al., albeit at the expense of a higher approximation factor and only for curves in $\mathbb{R}$. They present a $(5+\varepsilon)$-ANN data structure which uses space in $n \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k + \mathcal{O}(nm)$ and has query time in $\mathcal{O}(k)$, and a $(2+\varepsilon)$-ANN data structure, which uses space in $n \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k + \mathcal{O}(nm)$ and has query time in $\mathcal{O}\left(k \cdot 2^k\right)$. Even more efficient data structures can be obtained at the expense of an even larger approximation factor, see the work of Driemel, Silvestri, and Psarros [86] and [85] which uses locality-sensitive hashing. In these results neither the space nor the query time is exponential in the complexity of the curves (neither input nor query), but the approximation factor is linear in the query complexity k.

(Unconditional) lower bounds. Given these results, one may ask whether the cited bounds are optimal for the respective approximation factor that they guarantee. We review some efforts in answering this question and discuss the limitations of the current techniques. Driemel and Psarros [84, 85] approach this question using a technique by Miltersen [134] for proving cell-probe lower bounds. Their results indicate that any data structure answering a query for a near neighbor under the continuous Fréchet distance by using only a constant number of probes to memory cells cannot have a space usage that is independent of the arclength of the input curves (assuming a query radius of 1). In addition, their bounds indicate that, in some cases, space exponential in the complexity of the query k is necessary. However, these bounds hold only for data structures that use a constant number of probes to memory cells for answering a query, while we would also be interested in data structures that use higher query time, such as $\mathcal{O}(k)$ or $\mathcal{O}(\log n)$. A different lower bound technique was used by Driemel and Afshani [10]. They show a lower bound in the pointer model of computation on the space-time tradeoff for *range reporting* under the Fréchet distance. In this problem, all curves contained inside the query radius need to be output by the query. The resulting lower bound matches the above cited upper bounds even up to the asymptotic number of factors of $\log(n)$. The proof uses a construction of input curves in $\mathbb{R}^2$ and a set of queries, such that the intersection of any two query results has small volume while the queries themselves have large volume. The main drawback of this technique is that, being a volume argument, it inherently uses the fact that all curves inside the query need to be returned and therefore it cannot easily be applied in the near neighbor setting.

Conditional lower bounds. The recent rise of fine-grained complexity has also lead to a renewed interest in conditional lower bounds for nearest neighbor data structure problems, see, e.g. [5, 16, 69, 70, 143]. These lower bounds are for the offline version of the data structure problem, by considering the total time needed for preprocessing and performing a number of queries. They are obtained in a similar way as NP-hardness, specifically via reductions from some fine-grained hypothesis such as the *Strong Exponential Time Hypothesis (SETH)* [115] or the *Orthogonal Vectors Hypothesis (OVH)* [160]. In the Orthogonal Vectors problem we are given two sets of vectors $U, V \subseteq \{0,1\}^D$ of size n and ask whether there exist two vectors $u \in U, v \in V$ such that $\langle u, v \rangle = 0$. The hypothesis postulates that for any constant $\varepsilon > 0$ there exists a constant $c > 0$ such that there is no algorithm solving the Orthogonal Vectors problem in time $\mathcal{O}(n^{2-\varepsilon})$ in dimension $D = c \log n$. It should be noted that OVH is at least as believable as SETH, because SETH implies OVH [160]. As an example, based on the OV-hardness of bichromatic Euclidean closest pair [16] and reducing via a variant of OV with unbalanced size $|U| \ll |V|$ [6], one can show that for any $\varepsilon, \beta > 0$ there is no data structure for Euclidean nearest neighbors on n points in $\mathbb{R}^d$ with preprocessing time $\mathcal{O}(n^\beta)$ and query time $\mathcal{O}(n^{1-\varepsilon})$, in some dimension $d = c \log n$. This rules out any sublinear query time for any data structure with polynomial preprocessing time, unless OVH fails.

For computing the Fréchet distance of two polygonal curves there is a tight conditional lower bound [38], also for the one-dimensional case [48, 56]. However, thus far, there seems

Fréchet dist.	Approx.	Preprocessing Time	Query Time	Reference
disc., dD	$1+\varepsilon$	$nm \cdot \big(\mathcal{O}(\frac{1}{\varepsilon})^{dk} + \mathcal{O}(d \log m)\big)$	$\mathcal{O}(dk)$	[93]
cont., 1D	$2+\varepsilon$	$n \cdot \mathcal{O}(\frac{m}{k\varepsilon})^k$	$\mathcal{O}(1)^k$	[85]
	$5+\varepsilon$	$n \cdot \mathcal{O}(\frac{1}{\varepsilon})^k + \mathcal{O}(nm)$	$\mathcal{O}(k)$	[85]
cont., 1D	$1+\varepsilon$	$n \cdot \mathcal{O}(\frac{m}{k\varepsilon})^k$	$\mathcal{O}(1)^k$	Thm. 4.24
	$2+\varepsilon$	$n \cdot \mathcal{O}(\frac{m}{k\varepsilon})^k$	$\mathcal{O}(k)$	Thm. 4.26
	$2+\varepsilon$	$n \cdot \mathcal{O}(\frac{1}{\varepsilon})^k + \mathcal{O}(nm)$	$\mathcal{O}(1)^k$	Thm. 4.28
	$2+\varepsilon$	$\mathcal{O}(nm)$	$\mathcal{O}(\frac{1}{\varepsilon})^{k+2}$	Thm. 4.30
	$3+\varepsilon$	$n \cdot \mathcal{O}(\frac{1}{\varepsilon})^k + \mathcal{O}(nm)$	$\mathcal{O}(k)$	Thm. 4.32

Table 3.1: Known upper bounds and our results. For the discrete case we only cite the best known result. The space complexity is implicitly bounded by the preprocessing time in each case. Our preprocessing time is randomized; the bounds can be derandomized at the cost of a factor $\log n$ in preprocessing and query time (by using search trees instead of perfect hashing).

to be no comprehensive study of conditional lower bounds for the corresponding data structure problem. We want to close this gap and show tight bounds for the case of one-dimensional curves. These are similar in spirit to the Euclidean nearest neighbor lower bounds discussed above.

3.1.3 Our Results

For the discrete Fréchet distance the ANN problem is by now well understood, but the continuous Fréchet distance remains very challenging. Therefore, in this work we focus on the important special case of one-dimensional curves, which arise in various domains such as finance and signal processing, where they are typically called "time series". We give several new data structure bounds for the problem of approximate near neighbor searching for one-dimensional curves under the continuous Fréchet distance. Table 3.1 provides an overview of our upper bounds, compared to known results. In the second part of this work, we show that most of these upper bounds are tight under the Orthogonal Vectors Hypothesis, when viewed as offline problems where the input and the set of queries are given in advance. To obtain these lower bounds, we introduce a novel OV-hard variant of Orthogonal Vectors in which one set contains sparse vectors, i.e., vectors that only contain few 1s; this problem may be of independent interest. Table 3.2 gives an overview of our lower bound results. To argue that most of our upper bounds are tight, we consider the following general scenario:

> *Suppose we have an α-ANN for some fixed constant α, we run its preprocessing on a data set of n curves, and then we run n queries.*

In particular, consider this scenario for the following three ranges of α.

Fréchet dist.	Approx.	Preproc.	Query	Parameter Setting	Reference
cont., 1D	$2-\varepsilon$	poly(n)	$\mathcal{O}(n^{1-\varepsilon'})$	$1 \ll k \ll \log n$ and $m = k \cdot n^{c/k}$	Thm. 7.3
	$3-\varepsilon$	poly(n)	$\mathcal{O}(n^{1-\varepsilon'})$	$m = k = c\log n$	Thm. 7.4
cont., 2D	$3-\varepsilon$	poly(n)	$\mathcal{O}(n^{1-\varepsilon'})$	$1 \ll k \ll \log n$ and $m = k \cdot n^{c/k}$	Thm. 7.5

Table 3.2: Our conditional lower bounds. Each row gives an approximation ratio and a setting of k and m where any poly(n) preprocessing time and $\mathcal{O}(n^{1-\varepsilon'})$ query time cannot be achieved simultaneously. The constants $\varepsilon, \varepsilon', c$ are quantified as $\forall \varepsilon, \varepsilon' > 0\colon \exists c > 0$. By $f(n) \ll g(n)$ we mean $f(n) = o(g(n))$. We refer to the respective theorems in Chapter 7 for the exact statements.

- $1 < \alpha < 2$: Using our $(1+\varepsilon)$-ANN, this scenario takes total time $n \cdot \mathcal{O}(\frac{m}{k\varepsilon})^k$, which simplifies to $n \cdot \mathcal{O}(\frac{m}{k})^k$ since $\varepsilon = \alpha - 1$ is fixed. Assuming OVH, our first lower bound shows that this running time cannot be improved to $n \cdot f(k) \cdot (\frac{m}{k})^{o(k)}$ for any function f, for the following reason. Pick $k = k(n)$ sufficiently small such that $f(k) = n^{o(1)}$. Pick $m = k \cdot n^{c/k}$, so that $(\frac{m}{k})^{o(k)} = (\frac{k \cdot n^{c/k}}{k})^{o(k)} = n^{o(1)}$. Then the total running time would be $n \cdot f(k) \cdot (\frac{m}{k})^{o(k)} = n^{1+o(1)}$, which contradicts that either the preprocessing time is superpolynomial or the query time near-linear, as stated in Theorem 7.3. This shows that the factor $(\frac{m}{k})^{\Theta(k)}$ in our running time is necessary. Our second lower bound shows that the running time cannot be improved to $n \cdot (\frac{m}{k})^{f(k)} \cdot 2^{o(k)}$ for any function f, as for $m = k = c\log n$ the total time would become $n \cdot (\frac{m}{k})^{f(k)} \cdot 2^{o(k)} = n \cdot 1^{f(k)} \cdot n^{o(1)} = n^{1+o(1)}$, which contradicts that either the preprocessing time is superpolynomial or the query time near-linear, as stated in Theorem 7.4. This shows that the factor $\mathcal{O}(1)^k$ in our query time is necessary. In this sense, the running time of our $(1+\varepsilon)$-ANN is tight.

- $2 < \alpha < 3$: By using our second or third $(2+\varepsilon)$-ANN (Theorem 4.28 or 4.30) we solve this scenario in total time $\mathcal{O}(nm) + n \cdot \mathcal{O}(\frac{1}{\varepsilon})^{k+2}$, which simplifies to $\mathcal{O}(nm) + n \cdot \mathcal{O}(1)^k$ since $\varepsilon = \alpha - 2$ is fixed. Assuming OVH, our second lower bound shows that this cannot be improved to time $n \cdot (\frac{m}{k})^{f(k)} \cdot 2^{o(k)}$ for any function f, as for $m = k = c\log n$ we would obtain a total time of $n \cdot (\frac{m}{k})^{f(k)} \cdot 2^{o(k)} = n \cdot 1^{f(k)} \cdot n^{o(1)} = n^{1+o(1)}$, which contradicts that either the preprocessing time is superpolynomial or the query time near-linear, as stated in Theorem 7.4. This shows that the factor $\mathcal{O}(1)^k$ in our running time is necessary. In this sense, the running time of our $(2+\varepsilon)$-ANNs from Theorems 4.28 and 4.30 are tight. (Our $(2+\varepsilon)$-ANN from Theorem 4.26 is not tight in this sense, but it realizes a different tradeoff between preprocessing and query time.)

- $\alpha > 3$: In this range, our ANNs still require exponential time in terms of k, but we cannot hope for a tight lower bound using the current techniques. This is due to a fundamental limitation of proving inapproximability factor > 3 for a metric problem, cf. e.g. [143, Open Question 3]. For this reason, we have no tight lower bounds in this range.

3.1.4 Technical Overview

The high-level view of our data structures employs a well-known technique: exhaustively enumerate a strategic subset of the query space with a set of “candidate” query curves during preprocessing, and store the answers to these candidate queries in a dictionary. During query time, we apply a simple transformation to the query curve (such as rounding vertices to a scaled integer grid) and look up the answer in the dictionary. Filtser et al. [93] used this technique for the discrete Fréchet distance and Driemel and Psarros [85] showed that it can also be applied for the continuous Fréchet distance of one-dimensional curves. A particular challenge that appears in the continuous case is that the doubling dimension can be unbounded, even if the complexity of the curves is small. Intuitively, what can happen is that the query contains some small noise that appears in the middle of a long edge. The continuous Fréchet distance—being robust to this noise—may match these short edges to the interior of a long edge on the near neighbor input curve. However, we cannot afford to generate all possible noisy query curves of this type, since this would introduce a dependency on the arclength in our time and space bounds. Driemel and Psarros overcome this challenge with the use of signatures, which allow to “guess” the approximate shape of a query curve within some approximation factor. The idea is that the signature acts as a “low-pass” filter that eliminates the noisy short edges. However, this is a delicate process as the signature may eliminate too many edges on one of the curves (either on the near neighbor or on the query curve) leading to the near neighbor being missed during query time. In addition, the process may introduce false-positives, hence the high approximation factor of $(5+\varepsilon)$ in the result of [85].

We see our contributions as three-fold:

(1) Our first contribution is to improve the approximation factors of Driemel and Psarros [85] while staying within the same time bounds, cf. Table 3.1 for a comparison.

  (a) For Theorem 4.32, we use almost the same algorithm as Driemel and Psarros, but combine this with a more careful analysis based on new observations on the Fréchet distance of approximately monotone curves. As a result, we can achieve a $(3+\varepsilon)$-approximation within the same time bounds as the previous $(5+\varepsilon)$-ANN.

  (b) In Theorem 4.24 we even achieve an approximation factor of $(1+\varepsilon)$ within the same time bounds as the previous $(2+\varepsilon)$-ANN. To achieve this result, we introduce the concept of straightenings in Section 4.2. Straightenings share some properties of signatures, but they provide a more refined approximation, leading to fewer false positives. They allow us to “guess” the shape of a query curve up to approximation factor $(1+\varepsilon)$.

We derive useful properties of both signatures and straightenings. Central to our analysis is the concept of δ-visiting orders, which we introduce in Section 4.2 and analyze in Section 4.6.

(2) Our second contribution is a range of data structures for the $(2+\varepsilon)$-ANN which together provide a tradeoff between preprocessing time and query time (see Theorems 4.26, 4.28, and 4.30). In each case, the preprocessing time implicitly bounds the number of candidates that are generated and therefore the size of the dictionary used by the data structure. Thus, these data structures also achieve a tradeoff between space and query time. An important observation that leads to this result is that the enumeration of candidates can be "dualized" and then be shifted from the preprocessing time to the query time. In the extreme case, this allows us to design a data structure that has linear preprocessing time and space, by performing most of the candidate generation during query time, see Theorem 4.30 for the exact result.

(3) Given the diverse range of upper bounds, it is natural to ask if these bounds can be improved. Our third main contribution is to show that most of our upper bounds are tight under the Orthogonal Vectors Hypothesis. All known OV-based hardness results for the Fréchet distance encode each of the dimensions using at least one vertex, thus transforming D-dimensional vectors into curves of length $k = \Omega(D)$. Since OVH postulates a lower bound in dimension $D = c \log n$, it is thus natural to prove OV-based lower bounds for curves of length $k = c \log n$. Our lower bound in Theorem 7.4 handles this setting, cf. Table 3.2.

However, for some of our lower bounds we require $k = o(\log n)$, as this is necessary to rule out time $(m/k)^{o(k)}$. Surprisingly, we overcome the barrier of using at least one vertex per dimension. Specifically, we prove OV-based lower bounds for any $1 \ll k \ll \log n$, see Theorem 7.3. For this, we use two crucial observations: (i) it is possible to only encode the 1s of one vector set, while the 0s do not require any additional vertices on the curve, and (ii) we can show hardness of a variant of OV where one set contains only sparse vectors, i.e., vectors with a very small number of 1s. See Theorem 7.3 for the hardness result we obtain in this case. Interestingly, a similar construction is also possible for $(3-\varepsilon)$-ANN for two-dimensional curves, see Theorem 7.5.

3.1.5 Conclusions and Open Problems

To summarize, in this work we largely resolve the α-ANN problem under the continuous Fréchet distance for one-dimensional curves from a fine-grained perspective for $1 < \alpha < 3$. We show that, in general, most of the running times presented in this work cannot be improved significantly, however, other tradeoffs between preprocessing time and query time are still possible, and other parameter regimes might be shown hard or more tractable, e.g., for $k \in \mathcal{O}(1)$. Indeed, there is a line of work on related data structure problems using the continuous Fréchet distance for the specific value of $k = 2$, which corresponds to queries with line segments, see [78, 106]. It also remains a fundamental problem to show fine-grained lower bounds for approximation factors larger than 3 for a metric problem, which seems to require fundamentally different techniques, cf. [143].

As for the continuous Fréchet distance, our new upper and lower bounds show that the case of one-dimensional curves provides a kaleidoscopic view into the computational complexity and the underlying challenges posed by the general problem for polygonal curves in $\mathbb{R}^d$. The obvious way forward in this line of research is to show upper and lower bounds for dimension 2 and higher. Some of our ideas might translate directly, such as the idea to generate candidate curves at query time in order to achieve a tradeoff between preprocessing and query time. While our lower bounds also hold in higher dimension, it is conceivable that higher lower bounds can be shown already in the plane. In fact, we already initiate this line of work by showing an equally high lower bound for $(3-\varepsilon)$-ANN in the plane as we have for $(2-\varepsilon)$-ANN for one-dimensional curves. This lower bound already hints at techniques that can potentially achieve a matching upper bound. We leave this as an open problem. Our notions of straightenings and signatures, which capture the approximate shape of one-dimensional curves in a best-possible way, currently do not exist in dimension 2 or higher. Extending these notions to the plane by itself would be very interesting.

3.2 Fréchet Distance Under Translation

We now give an overview of the upper and lower bound for the discrete Fréchet distance under translation in the Euclidean plane. The details of the upper bound are then presented in Chapter 5 and the details of the lower bound in Chapter 8. Let n be the complexity of both curves that we want to compute the Fréchet distance under translation of. On the upper bound side, we improve the running time from $\tilde{\mathcal{O}}(n^5)$ to $\tilde{\mathcal{O}}(n^{4.66\dots})$. This is achieved by designing an improved algorithm for a subroutine of the previously best algorithm, namely offline dynamic s-t-reachability in directed grid graphs.

Theorem 3.2. *The discrete Fréchet distance under translation on curves of length n in the plane can be computed in time $\tilde{\mathcal{O}}(n^{14/3}) = \tilde{\mathcal{O}}(n^{4.66..})$.*

Our second main result is a lower bound of $n^{4-o(1)}$, conditional on the standard Strong Exponential Time Hypothesis. The Strong Exponential Time Hypothesis essentially asserts that Satisfiability requires time $2^{n-o(n)}$; see Chapter 2 for a definition. This (conditionally) separates the discrete Fréchet distance under translation from the classic Fréchet distance, which can be computed in time $\tilde{\mathcal{O}}(n^2)$. Moreover, the first step of all known algorithms for the discrete Fréchet distance under translation is to construct an arrangement of disks of size $\mathcal{O}(n^4)$. Our conditional lower bound shows that this is essentially unavoidable.

Theorem 3.3. *The discrete Fréchet distance under translation of curves of length n in the plane requires time $n^{4-o(1)}$, unless the Strong Exponential Time Hypothesis fails.*

We leave closing the gap between $\tilde{\mathcal{O}}(n^{4.66..})$ and $n^{4-o(1)}$ as an open problem.

3.2.1 Technical Overview

Previous algorithms for the discrete Fréchet distance under translation. Let us sketch the algorithms by Jiang et al. [120] and Ben Avraham et al. [33]. Given sequences $\pi = (\pi_1, \dots, \pi_n)$ and $\sigma = (\sigma_1, \dots, \sigma_n)$ in $\mathbb{R}^2$ and a number $\delta \geq 0$, we want to decide whether the discrete Fréchet distance under translation of π and σ is at most δ. From this decision procedure one can obtain an algorithm to compute the actual distance via standard techniques (i.e., parametric search).

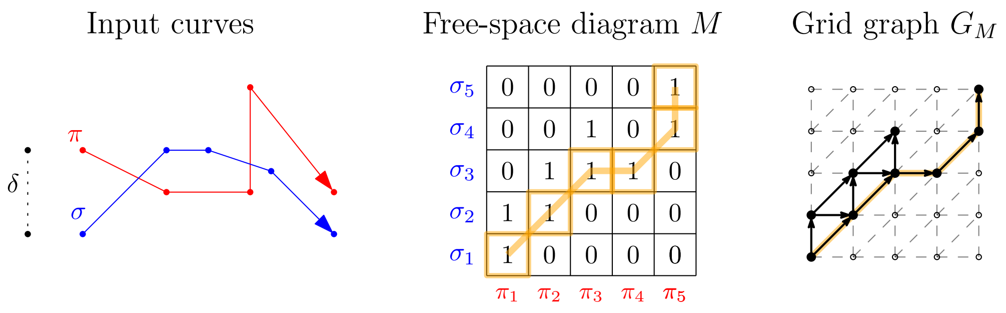
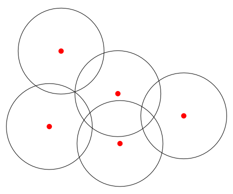


Figure 3.1: Two input curves π, σ and a distance δ, the corresponding free-space diagram M, and the grid graph G_M corresponding to M. A monotone traversal of M and G_M is marked in orange.



The translations τ for which the distance of π_i and $\sigma_j + \tau$ is at most δ form a disk in $\mathbb{R}^2$. Over all pairs (π_i, σ_j) this yields $\mathcal{O}(n^2)$ disks, all of them having radius δ. Construct their arrangement $\mathcal{A}$ (see an illustration to the right), which is guaranteed to have $\mathcal{O}(n^4)$ faces. Within each face of $\mathcal{A}$, any two translations are equivalent, in the sense that they leave the same pairs (π_i, σ_j) in distance at most δ. Thus, whether the discrete Fréchet distance is at most δ is constant in each face. Hence, it suffices to compute the discrete Fréchet distance between π and σ translated by τ over $\mathcal{O}(n^4)$ choices for τ, one for each face of $\mathcal{A}$. Since the discrete Fréchet distance can be computed in time $\mathcal{O}(n^2)$, this yields an $\mathcal{O}(n^6)$-time algorithm, which is essentially the algorithm by Jiang et al. [120].

Ben Avraham et al. [33] improve this algorithm as follows. Denote by M the $n \times n$ matrix with $M_{i,j} = 1$ if the points π_i, σ_j are in distance at most δ, and $M_{i,j} = 0$ otherwise (M is called the "free-space diagram"). It is well-known that the discrete Fréchet distance of π, σ is at most δ if and only if there exists a monotone path from the lower left to the upper right corner of M using only 1-entries. Equivalently, consider a *directed grid graph* G_M on $n \times n$ vertices, where each node (i, j) has directed edges to $(i+1, j), (i, j+1)$, and $(i+1, j+1)$, and the nodes (i, j) of G_M with $M_{i,j} = 0$ are "deactivated" (i.e., removed). Then the discrete Fréchet distance of π, σ is at most δ if and only if node (n, n) is reachable from node $(1, 1)$ in G_M. See Figure 3.1 for an example of a pair of curves, its corresponding free-space diagram M, and directed grid graph G_M.

Ben Avraham et al. observe that it is easy to construct a sequence of $\mathcal{O}(n^4)$ faces $f_1, \ldots, f_L$ of the arrangement $\mathcal{A}$ such that (1) each face of $\mathcal{A}$ is visited at least once and (2) f_ℓ and $f_{\ell+1}$ are neighboring in $\mathcal{A}$ for all ℓ. Since consecutive faces in this sequence are neighbors, only one pair (π_i, σ_j) changes its distance, i.e., either π_i, σ_j are in distance at most δ in f_ℓ and in distance larger than δ in $f_{\ell+1}$, or vice versa. This corresponds to one activation or deactivation of a node in G_M. After this update, we want to again check whether node (n, n) is reachable from node $(1, 1)$ in G_M. That is, using a dynamic algorithm for s-t-reachability in directed grid graphs, we can maintain whether the discrete Fréchet distance is at most δ. The best known solution to dynamic reachability in directed

$n \times n$ grids runs in time $\tilde{\mathcal{O}}(n)$ [80].[1] Over all $\mathcal{O}(n^4)$ faces, this yields time $\tilde{\mathcal{O}}(n^5)$ for the discrete Fréchet distance under translation in the plane [33].

Intuition. There are two parts to the above algorithm: (1) Constructing the arrangement $\mathcal{A}$ and iterating over its faces, and (2) maintaining reachability in the grid graph G_M. Both parts could potentially be improved.

The natural first attempt is to attack the arrangement enumeration, i.e., part (1). The size of the arrangement is $\mathcal{O}(n^4)$, and for no other computational problem it is known – to the best of our knowledge – that any optimal algorithm must construct such a large arrangement, so this part seems intuitively wasteful. Surprisingly, our conditional lower bound of Theorem 3.3 shows that constructing the arrangement is essentially unavoidable.

The remaining part (2) at first sight seems much less likely to be improvable, since it is a well-known open problem to find a faster dynamic algorithm for reachability in directed grid graphs. Nevertheless, we show how to improve the running time of this part of the algorithm.

Our algorithm. We observe that we do not need the full power of dynamic reachability, since we can precompute all $\mathcal{O}(n^4)$ updates. This leaves us with the following problem.

Offline Dynamic Grid Reachability: We start from the directed $n \times n$-grid graph G in which all nodes are deactivated.[2] We are given a sequence of updates $u_1, \dots, u_U$, where each u_ℓ is of the form "activate node (i,j)" or "deactivate node (i,j)". The goal is to compute for each $1 \leq \ell \leq U$ whether node $(1,1)$ can reach node (n,n) in G after performing the updates $u_1, \dots, u_\ell$.

Our main algorithmic contribution is an algorithm for Offline Dynamic Grid Reachability in amortized time $\tilde{\mathcal{O}}(n^{2/3})$ per update. This is faster than the update time $\tilde{\mathcal{O}}(n)$ obtained by using a dynamic algorithm for reachability in directed planar graphs [80].

Theorem 3.4. *Offline Dynamic Grid Reachability can be solved in time* $\tilde{\mathcal{O}}(n^2 + U \cdot n^{2/3})$.

The high-level approach of this algorithm is to consider all U updates in batches of size at most k, which we call *chunks*. Roughly speaking, we design a grid reachability data structure that given a chunk of k updates $u_1, \dots, u_k$, enables us to (1) for any $1 \leq j \leq k$, answer a grid reachability query in the matrix updated by $u_1, \dots, u_j$ in time $\tilde{\mathcal{O}}(k)$ and (2) obtain the data structure for the matrix updated by the complete chunk $u_1, \dots, u_k$ in time $\tilde{\mathcal{O}}(n\sqrt{k}+k)$. This way, for each of the $\mathcal{O}(U/k)$ chunks, we only need time $\tilde{\mathcal{O}}(k^2)$ to answer all k reachability queries for this chunk and time $\tilde{\mathcal{O}}(n\sqrt{k}+k)$ to update the data structure for the next chunk, leading to a total time of $\tilde{\mathcal{O}}((U/k)(k^2 + n\sqrt{k})) = \tilde{\mathcal{O}}(U(k + n/\sqrt{k}))$. By setting $k \approx n^{2/3}$, we obtain the desired algorithm running in time $\tilde{\mathcal{O}}(Un^{2/3})$ after $\tilde{\mathcal{O}}(n^2)$ preprocessing.

To obtain our data structure, we build on the reachability data structure of Ben Avraham et al. [33], augmented by two crucial insights: How to incorporate a chunk of k updates faster than k single updates, and how to succinctly store reachability information for k distinguished nodes in the grid (coined *terminals*, which correspond to the updates

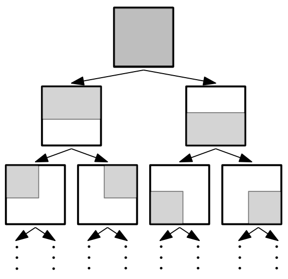
[1]This algorithm even works more generally for dynamic reachability in directed planar graphs.

[2]In fact, our algorithm also works in the general case in which nodes can be arbitrarily activated/deactivated in the beginning.

of the next chunk) in the data structure. The latter is given by a surprisingly succinct characterization of reachability of terminals in a grid graph (see Corollary 5.13), which is the key technical contribution for the algorithm.

Let us give a more detailed overview of our algorithm and its main ingredients. Start with a *block* $[n] \times [n]$ corresponding to the matrix M. Repeatedly split every block horizontally in the middle, and then split every block vertically in the middle, until we end up with constant-size blocks.

We call all the blocks considered during this process (not just the constant-size blocks!) the "canonical" blocks, see the figure to the right. Ben Avraham et al. [33] showed that one can store for each canonical block of sidelength s reachability information for each pair of boundary nodes, succinctly represented using only $\tilde{\mathcal{O}}(s)$ bits of space, and efficiently computable in time $\tilde{\mathcal{O}}(s)$ from the information of the two canonical child-blocks. In particular, over all blocks this information can be maintained in time $\tilde{\mathcal{O}}(n)$ per update u_i.



Ingredient 1: Batched updates. The first insight is that we can compute the reachability after a given chunk of k updates $u_1, \dots, u_k$ faster than $\tilde{\mathcal{O}}(nk)$: Intuitively, each update "touches" roughly $2 \log n$ blocks – all those that contain the node which is activated or deactivated. Our approach now uses that among the canonical blocks containing an update, the large blocks must be shared by many updates. Specifically, instead of recomputing the reachability information of the large blocks at the top of the hierarchy k times, we perform those updates jointly and thus avoid the runtime of k explicit updates of large blocks. A careful tradeoff yields an update time of $\tilde{\mathcal{O}}(n\sqrt{k} + k)$.

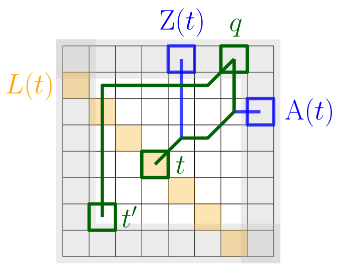
Ingredient 2: Reachability among terminals. Now fix a chunk $C = u_{\ell+1}, \dots, u_{\ell+k}$ and let M denote the matrix at the beginning of C, i.e., after incorporating all updates prior to $u_{\ell+1}$. Denote by $\mathcal{T}$ ("terminals") the entries that get activated or deactivated during this chunk C, and also add $(1,1)$ and (n,n) to the set of terminals. We first deactivate all terminals, obtaining a matrix $M^{\mathbf{0}}$ and a corresponding grid graph $G_{M^{\mathbf{0}}}$. The basic idea now is to determine for each pair of terminals $t, t' \in \mathcal{T}$ whether t' is reachable from t in $G_{M^{\mathbf{0}}}$.

Let us sketch a simplified algorithm that assumes we have built a graph H with vertex set $\mathcal{T}$, containing a directed edge (t, t') if and only if t' is reachable from t in $G_{M^{\mathbf{0}}}$. To answer the reachability query whether (n,n) is reachable from $(1,1)$ after updating M by $u_{\ell+1}, \dots, u_{\ell+j}$, we proceed as follows: For each terminal t, activate t in H if and only if t is activated in M updated by $u_{\ell+1}, ..., u_{\ell+j}$. Check whether (n,n) is reachable from $(1,1)$ in H. Since H has $\mathcal{O}(k)$ nodes and $\mathcal{O}(k^2)$ edges, this reachability check can be performed in time $\mathcal{O}(k^2)$. (By choosing a chunk size of $k \approx n^{2/5}$ this would result in an $\tilde{\mathcal{O}}(Un^{4/5})$ algorithm for Offline Dynamic Grid Reachability, ignoring the preprocessing time.) We will later show how to improve the reachability query time from $\mathcal{O}(k^2)$ to $\tilde{\mathcal{O}}(k)$ by working directly on the graph $G_{M^{\mathbf{0}}}$ instead of constructing the graph H. These details are given in Chapter 5.

It remains to describe how to determine reachability information among terminals. To this end, we design a surprisingly succinct representation of reachability from terminals to block boundaries. Consider a canonical block B and let $\mathcal{T}_B$ be the terminals in B. For each terminal $t \in \mathcal{T}_B$ let $A(t)$ be the lowest/rightmost point on the right/upper boundary of B that is reachable from t, and similarly let $Z(t)$ be the highest/leftmost reachable point, see the illustration to the right. We label any terminal $t = (x, y)$ by $L(t) := x + y$, i.e., the anti-diagonal that t is contained in. For any right/upper boundary point q of B, let $\ell(q)$ be the minimal label of any terminal in $\mathcal{T}_B$ from which q is reachable. We prove the following succinct representation of reachability (see Corollary 5.13) that significantly generalizes a previous characterization for reachability among the *boundaries* of blocks [19, 33].

For any right/upper boundary point q of B and any terminal $t \in \mathcal{T}_B$, q is reachable from t if and only if $q \in [A(t), Z(t)]$ and $\ell(q) \le L(t)$.

Here, $q \in [A(t), Z(t)]$ is to be understood as "q lies between $A(t)$ and $Z(t)$ in counterclockwise direction along the boundary of B", which can be expressed using a constant number of inequalities. The "only if" part is immediate, since t can only reach boundary vertices in $[A(t), Z(t)]$, and $\ell(q)$ is the minimal label of any terminal reaching q; the "if" part is surprising.



Assume we can maintain the information $A(t), Z(t), \ell(q)$. Then using this characterization we can determine all terminals reaching a boundary point q by a single call to *orthogonal range searching*, since we can express the characterization using a constant number of inequalities. A complex extension of this trick allows us to determine reachability among terminals (indeed, this technical overview is missing many details of Section 5.3). This yields our algorithm, see Sections 5.2 and 5.3 for details.

Conditional lower bound. Our reduction starts from the k-OV problem, which asks for k vectors from k given sets such that in no dimension all vectors are 1. More formally:

k-Orthogonal Vectors (k-OV): Given sets $V_1, \dots, V_k$ of N vectors in $\{0,1\}^D$, are there $v_1 \in V_1, \dots, v_k \in V_k$ such that for any $j \in [D]$ there exists an $i \in [k]$ with $v_i[j] = 0$?

A naive algorithm solves k-OV in time $\mathcal{O}(N^k D)$. It is well-known that the Strong Exponential Time Hypothesis implies that k-OV has no $\mathcal{O}(N^{k-\varepsilon}\text{poly}(D))$-time algorithm for all $\varepsilon > 0$ and $k \ge 2$ [160].

In our reduction we set $k = 4$. We consider *canonical translations* of the form $\tau = (\varepsilon \cdot h_1, \varepsilon \cdot h_2) \in \mathbb{R}^2$ with $h_1, h_2 \in \{0, \dots, N^2 - 1\}$. By a simple gadget, we ensure that any translation resulting in a discrete Fréchet distance of at most 1 must be close to a canonical translation. For simplicity, here we restrict our attention to exactly the canonical translations. Note that there are N^4 canonical translations, and thus they are in one-to-one correspondence to choices of vectors $(v_1, \dots, v_4) \in V_1 \times \ldots \times V_4$. In other words, the outermost existential quantifier in the definition of 4-OV corresponds to the existential quantifier over the translation τ in the discrete Fréchet distance under translation.

The next part in the definition of 4-OV is the universal quantifier over all dimensions $j \in [D]$. For this, our constructed curves π, σ are split into $\pi = \pi^{(1)} \dots \pi^{(D)}, \sigma = \sigma^{(1)} \dots \sigma^{(D)}$ such that $\pi^{(i)}, \sigma^{(j)}$ are very far for $i \ne j$. This ensures that the discrete

Fréchet distance of π, σ is the maximum over all discrete Fréchet distances of $\pi^{(i)}, \sigma^{(i)}$, and thus simulates a universal quantifier.

The next part is an existential quantifier over $i \in [k]$. Here we need an OR-gadget for the discrete Fréchet distance. Such a construction in principle exists in previous work [3, 38], however, no previous construction would work with translations, in the sense that a translation in y-direction could only decrease the discrete Fréchet distance. By constructing a more complex OR-gadget, we avoid this monotonicity.

Finally, we need to implement a check whether the translation τ corresponds to a particular choice of vectors. We exemplify this with the first dimension of the translation, which we call τ_1, explaining how it corresponds to choosing (v_1, v_2). Let $\mathrm{ind}(v_1), \mathrm{ind}(v_2) \in \{0, \ldots, N-1\}$ be the indices of these vectors in their sets V_1, V_2, respectively. We want to test whether $\tau_1 = \varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N)$. We split this equality into two inequalities. For the inequality $\tau_1 \geq \varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N)$, in one curve we place a point at $\pi_1 = (1 + \varepsilon \cdot \mathrm{ind}(v_1), -1 - \eta)$, and in the other we place a point at $\sigma_1 = (-1 - \varepsilon \cdot \mathrm{ind}(v_2) \cdot N, -1 - \eta)$, for some $\eta > 0$ which we specify later in this work. Then the distance of π_1 to the translated σ_1 is essentially their difference in x-coordinates, which is $(1 + \varepsilon \cdot \mathrm{ind}(v_1)) - (-1 - \varepsilon \cdot \mathrm{ind}(v_2) \cdot N + \tau_1) = 2 + \varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) - \tau_1$. This is at most 2 if and only if the inequality for τ_1 holds. We handle the opposite inequality similarly, and we concatenate the constructed points for both inequalities in order to test equality.

In total, our construction yields curves π, σ such that their discrete Fréchet distance under translation is at most 1 if and only if $V_1, \ldots, V_4$ contain orthogonal vectors. The curves π, σ consist of $n = \mathcal{O}(D \cdot N)$ vertices. Hence, an algorithm for the discrete Fréchet distance under translation in time $\mathcal{O}(n^{4-\varepsilon})$ would yield an algorithm for 4-OV in time $\mathcal{O}(N^{4-\varepsilon}\mathrm{poly}(D))$, and thus violate the Strong Exponential Time Hypothesis. See Chapter 8 for details.

3.2.2 Further Related Work

On directed planar/grid graphs. In this work we improve offline dynamic s-t-reachability in directed grid graphs. The previously best algorithm for this problem came from a more general solution to dynamic reachability in directed planar graphs. For this problem, a solution with $\tilde{\mathcal{O}}(N^{2/3})$ update time was given by Subramanian [149], which was later improved to update time $\tilde{\mathcal{O}}(\sqrt{N})$ by Diks and Sankowski [80]. In particular, our work yields additional motivation to study offline variants of classic dynamic graph problems.

Related work on dynamic directed planar or grid graphs includes, e.g., shortest path computation [8, 119, 124], reachability in the decremental setting [118], or computing the transitive closure [80]. Recently, the first conditional lower bounds for dynamic problems on planar graphs were shown by Abboud and Dahlgaard [4], however, they did not cover dynamic reachability in directed planar graphs.

Other work on directed planar and grid graphs studies, e.g., the minimum amount of space necessary to determine reachability between two nodes in polynomial time [25, 26]. For grid graphs this was recently improved from $\tilde{\mathcal{O}}(\sqrt{N})$ to $\tilde{\mathcal{O}}(N^{1/3})$ [26], but with very different techniques compared to ours.

On related reachability data structures. In [19], a reachability data structure on the free-space diagram is given to compute the Fréchet distance between closed curves and also to compute the best matching to any subcurve under the Fréchet distance. In [129], a data structure called the free-space map is presented, which improves the reachability data structure of [19] and as a consequence shaves off logarithmic factors in the running time of the above mentioned problems as well as improving the running time for other problems.

3.2.3 Conclusions and Open Problems

To conclude, in this work, we design an improved algorithm for the discrete Fréchet distance under translation running in time $\tilde{\mathcal{O}}(n^{14/3}) = \tilde{\mathcal{O}}(n^{4.66\ldots})$. As a crucial subroutine, we develop an improved algorithm for offline dynamic grid reachability. Additionally, we present a conditional lower bound of $n^{4-o(1)}$ based on the Strong Exponential Time Hypothesis, which, despite not yet matching our upper bound, strongly separates the discrete Fréchet distance under translation from the standard discrete Fréchet distance.

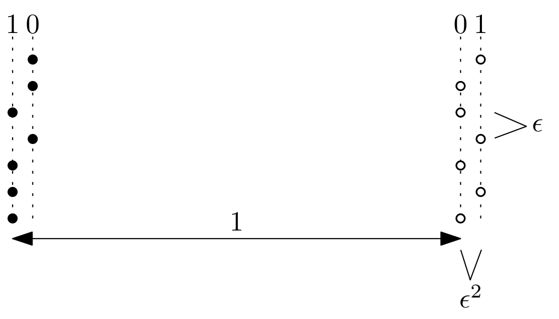
Our use of offline dynamic grid reachability yields further motivation for studying the offline setting of dynamic algorithms, for potential use as subroutines in static algorithms. Problems left open by this work include: (1) Closing the gap between our upper and conditional lower bound. This might require a solution to offline dynamic grid reachability with polylogarithmic amortized update time. (2) Generalizing our bounds to $d = 1$ or higher dimensions $d \geq 3$, as in this work we only consider curves in the plane. While generalizing our algorithm to $d = 1$ or $d \geq 3$ seems rather straight-forward but technical, obtaining strong conditional lower bounds for these cases is more interesting. (3) Considering different transformations such as scaling, rotation, or affine transformations in general; here we only treat translations. Significantly new ideas seem necessary to obtain meaningful lower bounds for other transformations. (4) Determine whether the time complexity of variants of the discrete Fréchet distance, such as the continuous or weak Fréchet distance, have similar or different relationships to their translation-invariant analogues.

3.3 Hausdorff Distance Under Translation

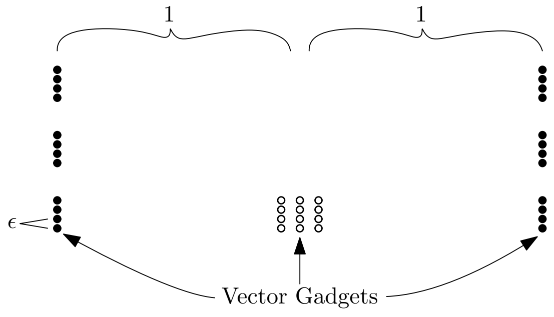
We now give an overview of our lower bounds for the Hausdorff distance under translation in the plane. The details are given in Chapter 6. In this work, we approach the Hausdorff distance under translation from the viewpoint of fine-grained complexity theory [152]. Let n and m be the sizes of the point sets that we want to compute the Hausdorff distance under translation of. For two problem settings, we show that the known algorithms are optimal up to lower order factors assuming standard hypotheses:

(1) We show an $(nm)^{1-o(1)}$ lower bound for all L_p norms — and in particular L_1 and L_∞, matching the $\mathcal{O}(nm \log^2 nm)$-time algorithm from [72] up to lower order factors, see Section 6.1.

This result holds conditional on the Orthogonal Vectors Hypothesis, which states that finding two orthogonal vectors among two given sets of n binary vectors in D dimensions cannot be done in time $\mathcal{O}(n^{2-\varepsilon}\mathrm{poly}(D))$ for any $\varepsilon > 0$. It is well-known



(a) Encoding vectors in gadgets.



(b) Arranging the vector gadgets.

Figure 3.2: Overview of reduction from Orthogonal Vectors to Hausdorff distance under translation. The filled dots represent the first point set while the hollow dots represent the other point set. Some details are omitted.

that the Orthogonal Vectors Hypothesis is implied by the Strong Exponential Time Hypothesis [160], and thus our lower bound also holds assuming the latter [116]. These two hypotheses are the most standard assumptions used in fine-grained complexity theory in the last decade [152].

(2) We show an $n^{2-o(1)}$ lower bound for L_2 in the imbalanced case $m = \mathcal{O}(1)$, matching the $\mathcal{O}(nm(n+m)\log nm)$-time algorithm from [72] up to lower order factors, see Section 6.2. Previously, an $n^{2-o(1)}$ lower bound was only known for the more general problem of computing the Hausdorff distance under translation of sets of *segments* in the case that both sets have size n (a problem for which the best known algorithm runs in time $\tilde{\mathcal{O}}(n^4)$) [31].

Our result holds conditional on the 3SUM Hypothesis, which states that deciding whether, among n given integers, there are three that sum up to 0 requires time $n^{2-o(1)}$. This hypothesis was introduced by Gajentaan and Overmars [97], is a standard assumption in computational geometry [123], and has also found a wealth of applications beyond geometry (see, e.g., [1, 7, 23, 138]).

Our lower bounds close gaps that have not seen any progress over 25 years. Furthermore, note that our second lower bound shows a separation between the L_2 norm and the L_1 and L_∞ norms, as in the imbalanced case $m = \mathcal{O}(1)$ the former admits a $\tilde{\mathcal{O}}(n)$-time algorithm [72] while the latter requires time $n^{2-o(1)}$ assuming the 3SUM Hypothesis. We leave it as an open problem whether for L_2 the balanced case $n = m$ requires time $n^{3-o(1)}$.

3.3.1 Technical Overview

We now give a rough outline of the two reductions, focusing on explaining the intuition behind the constructions.

Reduction from Orthogonal Vectors. For a reduction from Orthogonal Vectors to the Hausdorff distance under translation, we have to encode two structures into our

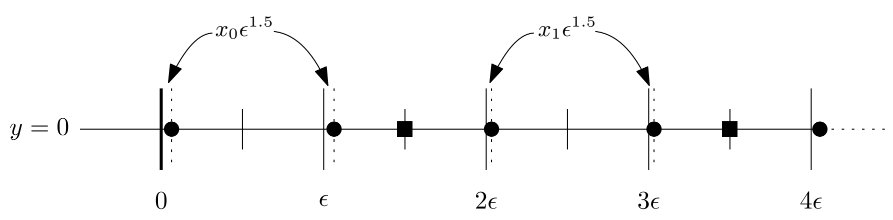


Figure 3.3: The points of the basic gadget to encode the sequence $x_1, \dots, x_n$. The other point set of the Hausdorff under translation instance then just consists of two points that have a horizontal spacing of $2 + \varepsilon$, concretely, the points $(-1, 0)$ and $(1 + \varepsilon, 0)$. This restricts the horizontal translations to $\tau_x = i \cdot \varepsilon + x_i \cdot \varepsilon^{1.5}$ for $i \in [n]$.

Hausdorff distance under translation instance, see Figure 3.2 for a sketch. First, we have to encode vectors such that their Hausdorff distance indicates whether they are orthogonal or not. Second, we have to place these encoded vectors such that we can check for orthogonality between all combinations of vectors. To encode the vectors, we can place a point for each dimension such that all these points are almost vertically aligned, except that they exhibit small horizontal perturbations depending on the entry in the respective dimension. If we align these vector gadgets vertically in a certain horizontal distance, then they only have a small Hausdorff distance if the encoded vectors are orthogonal. See Figure 3.2(a) for an example of such vector gadgets. Thus, by arranging the vector gadgets in the plane such that by translations only above mentioned alignments are possible for all vector combinations, we can check for orthogonality between all vectors using the Hausdorff distance under translation. See Figure 3.2(b) for a sketch of the placement of the vector gadgets.

Reduction from 3SUM. Instead of reducing from 3SUM to Hausdorff distance under translation, we reduce from an equivalent problem called convolution 3SUM (Conv3Sum). As opposed to the 3SUM problem where the input is a *set* of integers, for Conv3Sum we are given a *sequence* of integers $X = (x_1, \dots, x_n)$. We then ask whether there exist $i, j \in [n]$ such that $x_i + x_j = x_{i+j}$. Note that Conv3Sum also has a straightforward quadratic time algorithm: simply iterate over all choices of $i, j \in [n]$ and check if it fulfills the constraint. To encode the sequence of numbers $(x_1, \dots, x_n)$ into a Hausdorff distance under translation instance, we add a linear number of points to the first set and a constant number of points to the second set to encode the constraint $\tau_x = i \cdot \varepsilon + x_i \cdot \varepsilon^{1.5}$ on the translation $\tau = (\tau_x, \tau_y)$ for some small $\varepsilon > 0$, see Figure 3.3 for a sketch of this construction. By rotating and scaling this gadget, we can additionally obtain the constraints

$$\tau_y = j \cdot \varepsilon + x_j \cdot \varepsilon^{1.5} \quad \text{and} \quad \tau_x + \tau_y = k \cdot \varepsilon + x_k \cdot \varepsilon^{1.5}.$$

By plugging the first two constraints into the third constraint and choosing $\varepsilon > 0$ sufficiently small, we can see that all of the above constraints can only be fulfilled by a translation $\tau = (\tau_x, \tau_y)$ with $\tau_x + \tau_y = k \cdot \varepsilon + x_k \cdot \varepsilon^{1.5}$ such that $k = i + j$ and $x_i + x_j = x_{i+j}$ for some $i, j \in [n]$. Thus, we encode Conv3Sum into a Hausdorff distance under translation instance.

3.3.2 Related Work

Our work continues a line of research on fine-grained lower bounds in computational geometry, which had early success with the 3SUM Hypothesis [97] and recently got a new impulse with the Orthogonal Vectors Hypothesis (or Strong Exponential Time Hypothesis) and resulting lower bounds for the Fréchet distance [38], see also [48, 56]. Continuing this line of research is getting increasingly difficult, although there are still many classic problems from computational geometry without matching lower bounds. In this work we obtain such bounds for two settings of the classic Hausdorff distance under translation. Further work on the Hausdorff distance under translation includes an $\mathcal{O}((n+m)\log nm)$-time algorithm for point sets in one dimension [142]. For generalizations to dimensions $d > 2$ see [71, 72].

3.4 Engineering of the Fréchet Distance

We now give an overview of our engineered solution for the Fréchet distance. The details are given in Chapter 9. Initially defined more than one hundred years ago [94], the Fréchet distance quickly gained popularity in computer science after the first algorithm to compute it was presented by Alt and Godau [19]. In particular, they showed how to decide whether two length-n curves have Fréchet distance at most δ in time $\mathcal{O}(n^2)$ by full exploration of a quadratic-sized search space, the so-called *free-space* (we refer to Chapter 2 for a definition). Almost twenty years later, it was shown that, conditional on the Strong Exponential Time Hypothesis (SETH), there cannot exist an algorithm with running time $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$ [38]. Even for realistic models of input curves, such as c-packed curves [82], exact computation of the Fréchet distance requires time $n^{2-o(1)}$ under SETH [38]. Only if we relax the goal to finding a $(1+\varepsilon)$-approximation of the Fréchet distance, algorithms with near-linear running times in n and c on c-packed curves are known to exist [40, 82].

It is a natural question whether these hardness results are mere theoretical worst-case results or whether computing the Fréchet distance is also hard in practice. This line of research was particularly fostered by the research community in form of the GIS Cup 2017 [158]. In this competition, the 28 contesting teams were challenged to give a fast implementation for the following problem: Given a data set of two-dimensional trajectories $\mathcal{D}$, answer queries that ask to return, given a curve π and query distance δ, all $\sigma \in \mathcal{D}$ with Fréchet distance at most δ to π. We call this the *near neighbors problem*.

The three top implementations [30, 54, 87] use multiple layers of heuristic filters and spatial hashing to decide as early as possible whether a curve belongs to the output set or not, and finally use an essentially exhaustive Fréchet distance computation for the remaining cases. Specifically, these implementations perform the following steps:

(0) Preprocess $\mathcal{D}$.

On receiving a query with curve π and query distance δ:

(1) Use spatial hashing to identify candidate curves $\sigma \in \mathcal{D}$.

(2) For each candidate σ, decide whether π, σ have Fréchet distance $\leq \delta$:

a) Use heuristics (*filters*) for a quick resolution in simple cases.

b) If unsuccessful, use a *complete decision procedure via free-space exploration.*

Let us highlight the *Fréchet decider* outlined in steps 2a and 2b: Here, *filters* refer to sound, but incomplete Fréchet distance decision procedures, i.e., whenever they succeed to find an answer, they are correct, but they may return that the answer remains unknown. In contrast, a *complete decision procedure via free-space exploration* explores a sufficient part of the free space (the search space) to always determine the correct answer. As it turns out, the bottleneck in all three implementations is precisely Step 2b, the complete decision procedure via free-space exploration. Especially [30] improved upon the naive implementation of the free-space exploration by designing very basic pruning rules, which might be the advantage because of which they won the competition. There are two directions for further substantial improvements over the cup implementations: (1) increasing the range of instances covered by fast filters and (2) algorithmic improvements of the exploration of the reachable free-space.

3.4.1 Our Contribution

We develop a fast, practical Fréchet distance implementation. To this end, we give a complete decision procedure via free-space exploration that uses a divide-and-conquer interpretation of the Alt-Godau algorithm for the Fréchet distance and optimize it using sophisticated pruning rules. These pruning rules greatly reduce the search space for the realistic benchmark sets we consider – this is surprising given that simple constructions generate hard instances which require the exploration of essentially the full quadratic-sized search space [38, 48]. Furthermore, we present improved filters that are sufficiently fast compared to the complete decider. Here, the idea is to use adaptive step sizes (combined with useful heuristic tests) to achieve essentially "sublinear" time behavior for testing if an instance can be resolved quickly. Additionally, our implementation is certifying (see [131] for a survey on certifying algorithms), meaning that for every decision of curves being close/far, we provide a short proof (certificate) that can be checked easily; we also implemented a computational check of these certificates.

An additional contribution of this work is the creation of benchmarks to make future implementations more easily comparable. We compile benchmarks both for the near neighbors problem (Steps 0 to 2) and for the decision problem (Step 2). For this, we used publicly available curve data and created queries in a way that should be representative for the performance analysis of an implementation. As data sets we use the GIS Cup trajectories [9], a set of handwritten characters called the Character Trajectories Data Set [65] from [79], and the GeoLife data set [102] of Microsoft Research [165, 166, 167]. Our benchmarks cover different distances and also curves of different similarity, giving a broad overview of different settings. The source code as well as the benchmarks are publicly available to enable independent comparisons with our approach [46]. Additionally, we particularly focus on making our implementation easily readable to enable and encourage others to reuse the code.

3.4.2 Evaluation

The GIS Cup 2017 had 28 submissions, with the top three submissions[3] (in decreasing order) due to Bringmann and Baldus [30], Buchin et al. [54], and Dütsch and Vahrenhold [87]. We compare our implementation with all of them by running their implementations on our new benchmark set for the near neighbors problem and also comparing to the improved decider of [30]. The comparison shows significant speed-ups up to almost a factor of 30 for the near neighbors problem and up to more than two orders of magnitude for the decider.

3.4.3 Related Work

After the GIS Cup 2017, several practical papers studying aspects of the Fréchet distance appeared [27, 59, 155]. Some of this work [27, 59] addressed how to improve upon the spatial hashing step (Step 1) if we relax the requirement of exactness. Since this is orthogonal to our approach of improving the complete decider, these improvements could possibly be combined with our algorithm. The other work [155] neither compared itself with the GIS Cup implementations, nor provided their source code publicly to allow for a comparison, which is why we have to ignore it here.

3.5 Engineering of the Fréchet Distance Under Translation

We now give an overview of our engineered solution for the Fréchet distance under translation in the Euclidean plane. The details are given in Chapter 10. The question that we want to answer is: can we compute the Fréchet distance under translation quickly? The existing theoretical work yields a rather pessimistic outlook: For the discrete Fréchet distance under translation in the plane, the currently fastest algorithm runs in time $\mathcal{O}(n^{4.667})$, and any algorithm requires time $n^{4-o(1)}$ under the Strong Exponential Time Hypothesis, see Chapters 5 and 8. These high polynomial bounds appear prohibitive in practice, and have likely impeded algorithmic uses of this similarity measure. (For the continuous analogue, the situation appears even worse, as the fastest algorithm has a significantly higher worst-case bound of $\mathcal{O}(n^8 \log n)$; we thus solely consider the discrete version in this work.) Given the surprising performance of recent Fréchet distance implementations on realistic curves, see [158] and Chapter 9, can we still hope for faster algorithms on realistic inputs also for its translation-invariant version?

Towards making the Fréchet distance under translation applicable for practical applications, we engineer a fast implementation and analyze it empirically on realistic inputs. Perhaps surprisingly, our fastest solution for the problem combines inexact *continuous optimization* techniques with an exact, but expensive problem-specific approach from computational geometry to obtain an *exact decision* algorithm. The source code as well as the benchmarks are publicly available to enable independent comparisons with our approach [47].

[3]The submissions were evaluated "for their correctness and average performance on a[sic!] various large trajectory databases and queries". Additional criteria were the following: "We will use the total elapsed wall clock time as a measure of performance. For breaking ties, we will first look into the scalability behavior for more and more queries on larger and larger datasets. Finally, we break ties on code stability, quality, and readability and by using different datasets."

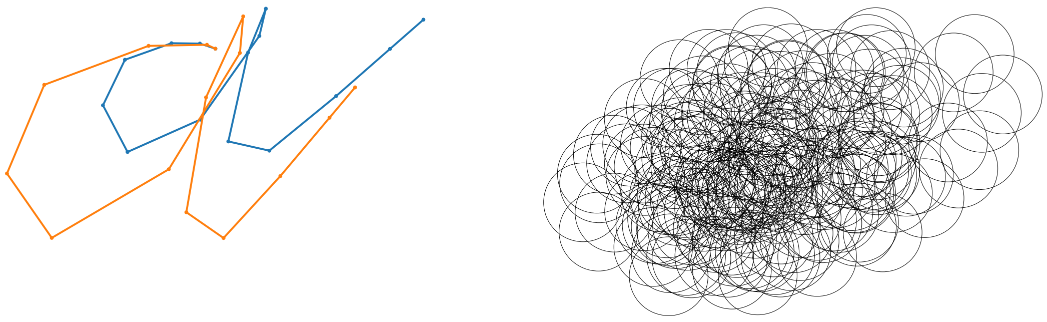


Figure 3.4: Example curves (left) together with their arrangement (right).

In the remainder of this section, we give a technical overview of our result. There are two possible approaches to computing the Fréchet distance under translation in the literature. First, to compute it exactly, we can use the arrangement-based approach that was first presented in [120]. Second, observing that the Fréchet distance is 1-Lipschitz in the translation, we can compute an approximation by using techniques from Lipschitz optimization. Let us briefly consider both approaches.

3.5.1 Arrangement-Based Algorithms

We intuitively explain the arrangement-based approach of [120] for deciding whether the Fréchet distance under translation of two trajectories π and σ is at most δ. Let n be the complexity of both π and σ. For all pairs p, q of vertices of π, σ, we want to partition the translation space into translations that translate q into distance δ of p and the remaining translations. Note that this partition is induced by the δ-circle around $p - q$. Thus, inserting all such $\mathcal{O}(n^2)$ circles into an arrangement gives us a partition of the plane of size $\mathcal{O}(n^4)$ of the translation space. See Figure 3.4 for an example of two curves and their arrangement. The important observation now is that all translations in a cell of this arrangement have the same decision for whether the Fréchet distance under translation of π and σ is at most δ. This is the case because, for all pairs of vertices, whether they are in distance at most δ or not is uniform for all translations in a cell and thus a traversal is either in distance at most δ for all such translations or none. Consequently, checking a representative from each cell of the partition and only returning "yes" if any of these representatives returns "yes" is a correct decision algorithm. For this check to be as fast as possible, we need a very fast Fréchet distance decider. We engineered such a decider in previous work and present it in Chapter 9 of this thesis.

3.5.2 Global Lipschitz Optimization

Consider how the Fréchet distance behaves when translating one of the curves. In Figure 3.5 we plot the discrete Fréchet distance $d_{dF}(\pi, \sigma + \tau)$ for a range of values $\tau \in \mathbb{R}^2$ for two curves from a handwritten characters data set. On the left side, we can see that globally the function looks smooth and almost convex, however, on the right side, we can

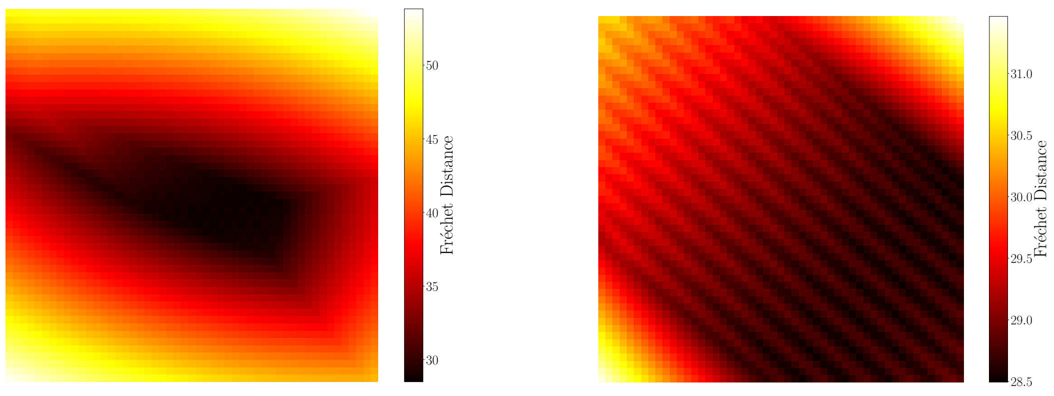


Figure 3.5: An example of the Fréchet distance plotted over the translations in a zoomed out view (left) and a zoomed in view (right). Globally the function is almost convex, but locally it shows highly non-convex artifacts.

see that the function has highly non-convex artifacts locally. Regarding smoothness, one can ask how much the Fréchet distance can change when translating one of the curves by some τ. As the maximal distance of the best traversal of the curves without translation can change at most by $\|\tau\|$, we have that

$$|\, \mathrm{d}_{\mathrm{dF}}(\pi, \sigma + \tau) - \mathrm{d}_{\mathrm{dF}}(\pi, \sigma)| \leq \|\tau\| \,.$$

Thus, the Fréchet distance is 1-Lipschitz in the translation. Due to this property, we can use techniques from Lipschitz optimization on the Fréchet distance under translation, which we describe briefly in the following.

We start with an initial region of translations from which we know that it contains the translation that realizes the Fréchet distance under translation, and then use divide and conquer with pruning to narrow down the region in which the minimum lies. More precisely, due to the Lipschitz property, evaluating a single translation inside a region gives us a lower and upper bound on all the costs in this region (as they cannot deviate more than the diameter of this region from the sample). If we already found a better global solution than the lower bound of the current region, we can stop searching in this region. If we find a better upper bound, then we update our global upper bound.

3.5.3 Our Solution: A Combination

Both approaches have major drawbacks when used by themselves. The arrangement-based approach already has a very high complexity for very short curves: see Figure 3.4 for how complicated an arrangement of two practical, low-complexity curves can become. On the other hand, the Lipschitz optimization approach is only approximative and it becomes significantly slower when requiring close approximations. In particular, the local non-convex features shown in Figure 3.5 cannot easily be pruned as they all contain local minima that are close to the global minimum. Our main idea consists of combining both approaches in the following way. We run the Lipschitz optimization described above and then upper bound the arrangement size for each region in the divide and conquer approach before processing it. If this arrangement size upper bound is below a certain threshold, we use the arrangement-based approach to decide the Fréchet distance for all translations in this region, avoiding deep recursions in practice. As is common in algorithm engineering, there are many details that need to be engineered in order to achieve a performant implementation. See Chapter 10 for these details.

PART I

Upper Bounds

CHAPTER 4

Continuous Fréchet Approximate Near Neighbor Search

For a technical overview of this chapter see Section 3.1. The structure of this chapter is as follows. In Section 4.1 we define the notation and state some known facts and observations. In Section 4.2 we define key concepts, and we present their properties and our main technical lemmas. Our data structures are described and analyzed in Sections 4.3, 4.4, and 4.5. In Section 4.6 we prove our main technical lemmas.

4.1 Preliminaries

Recall that we are in the ANN setting and, unless mentioned otherwise, n denotes the number of input curves, m denotes the complexity of the input curves, and k denotes the complexity of the query curves. For any two points $p, q \in \mathbb{R}^d$, $\overline{pq}$ denotes the directed line segment connecting p with q in the direction from p to q. In this chapter, we always parametrize the polygonal curves via the interval $[0, 1]$, i.e., a polygonal curve is defined as $P : [0, 1] \mapsto \mathbb{R}^d$. For $d = 1$, we may refer to the curve as a *one-dimensional curve* or as a *time series*. For a point $x \in \mathbb{R}^d$ and a polygonal curve P, we use the notation $x \in P$ to indicate that there exists a $t \in [0, 1]$ such that $P(t) = x$. Furthermore, let $\mathcal{G}_\varepsilon := \{i \cdot \varepsilon \mid i \in \mathbb{Z}\}$ be the regular grid with side-length $\varepsilon > 0$. We will use the following known observations (see also [53] and [81]).

Observation 4.1. *For any two line segments $X = \overline{ab}$, $Y = \overline{cd}$ it holds that $\mathrm{d_F}(X, Y) = \max\{\|a - c\|, \|b - d\|\}$.*

Observation 4.2. *Let two polygonal curves $Q : [0, 1] \mapsto \mathbb{R}^d$ and $P : [0, 1] \mapsto \mathbb{R}^d$ be the concatenations of two subcurves each, $Q = Q_1 \circ Q_2$ and $P = P_1 \circ P_2$. Then it holds that $\mathrm{d_F}(P, Q) \leq \max\{\mathrm{d_F}(Q_1, P_1), \mathrm{d_F}(Q_2, P_2)\}$.*

Observation 4.3. *Let Q be a line segment and let P be a curve with $\mathrm{d_F}(P, Q) \leq \delta$. Let P' be a curve that is formed from a subsequence of the vertex sequence of P including the first and last vertex of P. Then, $\mathrm{d_F}(P', Q) \leq \delta$.*

Our data structures can be implemented to work on the Word-RAM and under certain assumptions on the Real-RAM, as discussed next. Central to our approach is the use of a dictionary, which we define as follows.

Definition 4.4 (Dictionary)**.** *A* dictionary *is a data structure which stores a set of (key, value) pairs and when presented with a key, either returns the corresponding value, or returns that the key is not stored in the dictionary.*

In the Word-RAM model, such a dictionary can be implemented using perfect hashing. For storing a set of n (key,value) pairs, where the keys come from a universe U^k, perfect hashing provides us with a dictionary using $\mathcal{O}(nk)$ space and $\mathcal{O}(k)$ query time which can be constructed in $\mathcal{O}(nk)$ expected time [95]. During look-up, we compute the hash function in $\mathcal{O}(k)$ time, we access the corresponding bucket in the hashtable in $\mathcal{O}(1)$ time and check if the key stored there is equal to the query in $\mathcal{O}(k)$ time. This gives an efficient randomized implementation of dictionaries. Alternatively, we can use balanced binary search trees and pay an additional $\log n$ factor in preprocessing and query time of the dictionary. This deterministic algorithm also works in the Real-RAM model, if we assume that the floor function can be computed in constant time—a model which is often used in the literature [109]. In the Word-RAM model, we use the standard assumption that the word size is logarithmic in the size of the input, and we ensure that all numbers (vertices of the time series, results of intermediate computations, etc.) are restricted to be of the form a/b where a is an integer in $[-(nm)^{\mathcal{O}(1)}, (nm)^{\mathcal{O}(1)}]$ and $b = (nm)^{\mathcal{O}(1)}$.

4.2 Simplifications, Signatures, and Straightenings

In this section we state the main definitions and lemmas that we use to describe our algorithms and prove their correctness. To allow for an easier understanding of our results, we then already describe our algorithms and prove correctness using these lemmas. In Section 4.6 we then give the proofs of the lemmas presented in the current section.

4.2.1 Definitions

Let us start with two basic definitions.

Definition 4.5. *We say a curve $P : [0,1] \to \mathbb{R}$ is* δ-monotone *if one of the following statements holds:*
(i) $\forall\ t < t' \in [0,1] : P(t') \geq P(t) - \delta$,
(ii) $\forall\ t < t' \in [0,1] : P(t') \leq P(t) + \delta$.
More specifically, we say the curve is δ-monotone increasing *in case (i) and* δ-monotone decreasing *in case (ii). Note that a curve can be both δ-monotone increasing and decreasing at the same time. In addition, we may say P is δ-monotone with respect to a directed segment $\overline{ab}$, if $a \leq b$ in case (i) and if $b \leq a$ in case (ii).*

Definition 4.6. *The δ-range of a point $p \in \mathbb{R}$ is the interval $B(p, \delta) = [p - \delta, p + \delta]$. The δ-range of a curve P is the interval $B(P, \delta) = \bigcup_{x \in P} B(x, \delta)$.*

We now define the notion of *simplification* that we use in this chapter.

Definition 4.7 (δ-simplification)**.** *Given a curve $P : [0,1] \mapsto \mathbb{R}^d$, a δ-simplification is a curve $P' : [0,1] \mapsto \mathbb{R}^d$ that is given as $P' = \langle P(t_1), \ldots, P(t_\ell) \rangle$ for a sequence of values $0 = t_1 < \cdots < t_\ell = 1$, such that each $P(t_i)$ is a vertex of P, P' is non-degenerate, and*

$$\mathrm{d_F}(\overline{P(t_i)P(t_{i+1})}, P[t_i, t_{i+1}]) \leq \delta, \text{ for all } 1 \leq i < \ell. \tag{4.1}$$

We also refer to (4.1) as the *locality property*. Furthermore, note that if P' is a δ-simplification of P, then $\mathrm{d_F}(P, P') \leq \delta$ and the complexity of P' is at most the complexity of P. Note that the vertices of a δ-simplification P' give us a natural partition of P. Furthermore, we want to highlight that our definition of a simplification is one out of many definitions that are used in literature. In particular, in other work curves which are degenerate or non vertex-restricted are also called simplifications. Now we define some properties that a simplification can or must have.

Observation 4.8 (direction-preserving property). *For any i and any δ-simplification $P' = \langle P(t_1), \ldots, P(t_\ell)\rangle$ of a curve $P : [0,1] \mapsto \mathbb{R}$, the subcurve $P[t_i, t_{i+1}]$ is 2δ-monotone with respect to $\overline{P(t_i)P(t_{i+1})}$.*

Definition 4.9 (vertex-range-preserving property). *Let $P' = \langle P(t_1), \ldots, P(t_\ell)\rangle$ be a δ-simplification of a curve $P : [0,1] \mapsto \mathbb{R}$. We say P' is range-preserving on the vertex $P(t_i)$ if the following holds:*

(i) if $P(t_i)$ is a local maximum on P', then $P(t) \leq P(t_i)$ for all t in $[t_{i-1}, t_{i+1}]$, and

(ii) if $P(t_i)$ is a local minimum on P', then $P(t) \geq P(t_i)$ for all t in $[t_{i-1}, t_{i+1}]$.

We say P' is vertex-range-preserving, if it is vertex-range-preserving on all interior vertices.

Definition 4.10 (edge-range-preserving property). *Let $P' = \langle P(t_1), \ldots, P(t_\ell)\rangle$ be a δ-simplification of $P : [0,1] \mapsto \mathbb{R}$. We say that P' is edge-range-preserving on edge $\overline{P(t_i)P(t_{i+1})}$ if for any $t \in [t_i, t_{i+1}]$ it holds that $P(t) \in \overline{P(t_i)P(t_{i+1})}$. We say P' is edge-range-preserving if this condition holds for all edges of P'.*

Note that the vertex-range-preserving property is implied by the edge-range-preserving property, but not the other way around. However, the vertex-range preserving property implies the edge-range-preserving property on all edges except the first and the last edge.

Definition 4.11 (δ-edge-length property). *We say that a one-dimensional curve $P = \langle p_1, \ldots, p_m\rangle$ has the δ-edge-length property if*

- *$|p_1 - p_2| > \delta$ and $|p_{m-1} - p_m| > \delta$, and*
- *$|p_i - p_{i+1}| > 2\delta$ for all $i \in \{2, \ldots, m-2\}$.*

Finally, we can define two of the main concepts that we use in our algorithms: δ-signatures and δ-straightenings. These two definitions help us to preprocess the input set of one-dimensional curves and the query curve in ways such that an efficient retrieval is possible.

Definition 4.12 (δ-signature). *A δ-simplification P' of a one-dimensional curve P is a δ-signature if it has the δ-edge length property and is vertex-range-preserving.*

Definition 4.13 (δ-straightening). *A δ-simplification P' of a one-dimensional curve P is a δ-straightening if it is edge-range-preserving.*

The above definition of a δ-signature is equivalent to the definition given in [83]. For any $\delta > 0$ and any curve $P : [0,1] \mapsto \mathbb{R}$ of complexity m, a δ-signature of P can be computed in $\mathcal{O}(m)$ time [83]. The δ-signature of a curve is unique under certain general-position assumptions, however we do not explicitly use this property in our proofs. Note

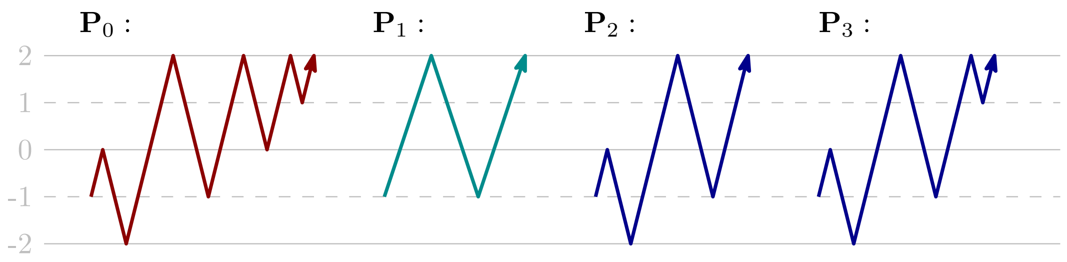


Figure 4.1: P_1 is a 1-signature of P_0, whereas P_2 and P_3 are 1-straightenings of P_0.

that δ-straightenings are *not* unique. In fact, there can be many different δ-straightenings of the same curve, e.g., P itself is a δ-straightening of P for any $\delta > 0$. We give an example of a signature and different straightenings of the same curve in Figure 4.1.

We introduce the notion of visiting orders, which we will use to prove correctness of our data structures.

Definition 4.14. *Let $P : [0,1] \to \mathbb{R}$ and $Q : [0,1] \to \mathbb{R}$ be curves. Let $u_1, \ldots, u_\ell$ denote the ordered vertices of Q and let $v_1, \ldots, v_m$ denote the ordered vertices of P. A (partial) δ-**visiting order** of Q on P is a sequence of indices $i_1 \leq \cdots \leq i_\ell$, such that $|u_j - v_{i_j}| \leq \delta$ for each vertex u_j of Q.*

In particular, if we know that there exists a δ-visiting order of Q on P, then we can approximately "guess" Q from the vertex sequence of P, by enumerating all possible visiting orders of the vertices of P and for any fixed visiting order, enumerating all eligible sequences over a grid within the δ-ranges of these vertices.

Driemel, Krivosija and Sohler proved the following lemma (rephrased using δ-visiting orders).

Lemma 4.15 (Lemma 3.2 [83])**.** *Let $P : [0,1] \to \mathbb{R}$ and $Q : [0,1] \to \mathbb{R}$ be curves and let P' be a δ-signature of P. If $\mathrm{d_F}(P,Q) \leq \delta$, then there exists a δ-visiting order of P' on Q.*

4.2.2 Main Lemmas

In this section we present the main lemmas for signatures and straightenings that we will use in Sections 4.3 to 4.5. Their proofs are deferred to Section 4.6.

Most of our lemmas improve the basic triangle inequality $\mathrm{d_F}(P,Q) \leq \mathrm{d_F}(P,X) + \mathrm{d_F}(X,Q)$ in some situations involving signatures and straightenings.

Lemma 4.16. *Let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be two curves and let Q' be any δ-straightening of Q. If $\mathrm{d_F}(P,Q') \leq \delta$ then $\mathrm{d_F}(P,Q) \leq \delta$.*

We would like to show the equivalent statement of Lemma 4.16 for signatures. However, as the example in Figure 4.2 shows, this is not possible. Instead, we show a slightly weaker bound in the following lemma.

Lemma 4.17. *Let $\delta = \delta' + \delta''$ for $\delta, \delta', \delta'' \geq 0$ and let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be two curves. Let Q' be any δ'-signature of Q. If $\mathrm{d_F}(Q',P) \leq \delta$, $|Q(0) - P(0)| \leq \delta''$, and $|Q(1) - P(1)| \leq \delta''$, then $\mathrm{d_F}(P,Q) \leq \delta$.*

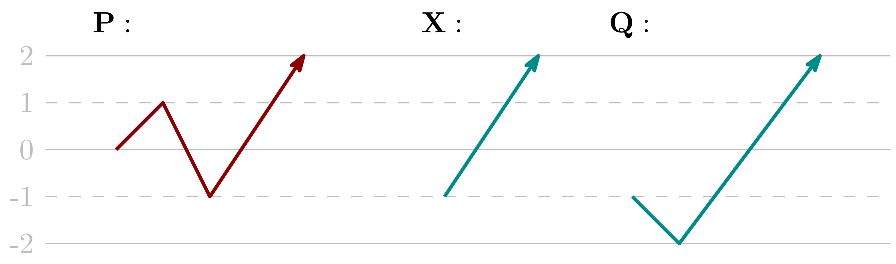


Figure 4.2: This example shows that an equivalent statement of Lemma 4.16 for signatures is not true. The curve $X = \langle -1, 2\rangle$ is a 1-signature of $Q = \langle -1, -2, 2\rangle$ and the curve $P = \langle 0, 1, -1, 2\rangle$ has Fréchet distance 1 to X, but the Fréchet distance of P to Q is 2.

Note that Lemma 4.16 is much stronger than what we would get by merely applying the triangle inequality on the Fréchet distances on the curves P, Q and Q'. Lemma 4.17, although weaker, is still stronger than the bound we would get from the triangle inequality. To illustrate this we include the following corollary. Note that merely using triangle inequality would yield $\mathrm{d_F}(P, Q) \leq 6\delta$, instead of $\mathrm{d_F}(P, Q) \leq 3\delta$.

Corollary 4.18. *For one-dimensional curves P, Q let P' be a δ-signature of P, and let Q' be the 2δ-signature of Q. If $\mathrm{d_F}(P', Q') \leq 3\delta$ and $|P'(0) - Q'(0)| \leq \delta, |P'(1) - Q'(1)| \leq \delta$, then $\mathrm{d_F}(P, Q) \leq 3\delta$.*

Proof. Follows from applying of Lemma 4.17 twice. We first apply the lemma to P', Q' and P and obtain $\mathrm{d_F}(P, Q') \leq 3\delta$. In the second step, we apply the lemma to P, Q' and Q and obtain $\mathrm{d_F}(P, Q) \leq 3\delta$. □

The following lemma is used to show correctness for our $(1 + \varepsilon)$ and $(2 + \varepsilon)$-ANN.

Lemma 4.19. *Let $P : [0, 1] \mapsto \mathbb{R}$ and $Q : [0, 1] \mapsto \mathbb{R}$ be curves such that $\mathrm{d_F}(Q, P) \leq \delta$, there exists a δ-straightening Q' of Q which satisfies the following properties:*
(i) there exists a (11δ)-visiting order of Q' on P, and
(ii) $\mathrm{d_F}(Q', P) \leq \delta$.

We use the following lemma to show correctness for our $(3+\varepsilon)$-ANN. One part of the lemma statement, the existence of a 2δ-visiting orders, was already used in [84]. However, the resulting approximation factor of the ANN obtained there was $(5 + \varepsilon)$. In order to show correctness of our $(3 + \varepsilon)$-ANN, it is necessary to prove the bound of 3δ on the resulting Fréchet distance of the two signature curves. Note that the triangle inequality implies a bound of 4δ—which would not be sufficient for us.

Lemma 4.20. *For one-dimensional curves P, Q let P' be a δ-signature of P, and let Q' be a 2δ-signature of Q. If $\mathrm{d_F}(P, Q) \leq \delta$ then $\mathrm{d_F}(P', Q') \leq 3\delta$ and there exists a 2δ-visiting order of Q' on P'.*

4.3 $(1 + \varepsilon)$-Approximation

In this section, we show that there exists a $(1+\varepsilon)$-ANN data structure for one-dimensional curves under the Fréchet distance, with space in $n \cdot \mathcal{O}(\frac{m}{k\varepsilon})^k$, expected preprocessing time in $nm \cdot \mathcal{O}(\frac{m}{k\varepsilon})^k$ and query time in $\mathcal{O}(k \cdot 2^k)$. We describe the data structure in Section 4.3.1 and we analyze its performance in Section 4.3.2.

4.3.1 The Data Structure

Data structure. We are given as input a set of one-dimensional curves $\mathcal{P}$, as sequences of vertices, the distance threshold $\delta > 0$, the approximation error $\varepsilon > 0$, and the complexity of the supported queries k. To discretize the query space, we use the grid $\mathcal{G}_{\varepsilon\delta/2}$ (recall that $\mathcal{G}_\varepsilon := \{i \cdot \varepsilon \mid i \in \mathbb{Z}\}$ is the regular grid with side-length ε). Let $\mathcal{H}$ be a dictionary which is initially empty. For each input one-dimensional curve $P \in \mathcal{P}$ we compute a set $\mathcal{C}' := \mathcal{C}'(P)$ which contains all curves Q such that: i) Q has complexity at most k, ii) all vertices of Q belong to $\mathcal{G}_{\varepsilon\delta/2}$, and iii) there is an $((11+\varepsilon/2)\delta)$-visiting order of Q on P. Formally,

$$\begin{aligned}\mathcal{C}' = \{\langle u_1, \ldots, u_\ell\rangle \mid \ell \le k \text{ and } \exists(i_1, \ldots, i_\ell)(i_1 \le \cdots \le i_\ell \text{ and}\\ (\forall j \in [\ell])(u_j \in B(p_{i_j}, (11+\varepsilon/2)\delta) \cap \mathcal{G}_{\varepsilon\delta/2}))\}.\end{aligned}$$

Next, we filter $\mathcal{C}'$ to obtain the set $\mathcal{C}(P) = \{Q \in \mathcal{C}' \mid \mathrm{d_F}(Q, P) \le (1+\varepsilon/2)\delta\}$. We store $\mathcal{C}(P)$ in $\mathcal{H}$ as follows: for each $Q \in \mathcal{C}(P)$, if Q is not already stored in $\mathcal{H}$, then we insert Q into $\mathcal{H}$, associated with a pointer to P.

The complete pseudocode for the preprocessing algorithm can be found in Algorithms 1 and 2. To achieve approximation factor $(1+\varepsilon)$, we run `preprocess`$(P, \delta, \varepsilon/2, k)$.

Query algorithm. Let Q be the query curve with vertices $q_1, \ldots, q_k$ and let $\varepsilon > 0$ be the approximation error. The query algorithm first enumerates all curves Q' such that $Q' \in \{\langle q_1, S, q_k\rangle \mid S \text{ is a subsequence of } q_2, \ldots, q_{k-1}\}$. For each such Q' we test whether it is a δ-straightening of Q. To this end, we first test if each shortcut taken in Q' is within distance δ from the corresponding subcurve of Q. Then we check for each shortcut if the corresponding subcurve of Q stays within range by testing all vertices of the subcurve one by one. If Q' is a δ-straightening of Q, then we snap the vertices of Q' to $\mathcal{G}_{\varepsilon\delta/2}$, to obtain a new curve Q'' and we probe $\mathcal{H}$: if Q'' is stored in $\mathcal{H}$, then we return its associated input curve $P \in \mathcal{P}$. If Q'' is not stored in $\mathcal{H}$, then we return "no".

The complete pseudocode for the query algorithm can be found in Algorithm 3. To achieve approximation factor $(1+\varepsilon)$, we run `query`$(Q, \delta, \varepsilon/2)$.

4.3.2 Analysis

In this section, we analyze the performance of our data structure.

Lemma 4.21. *For any curve P with vertices $p_1, \ldots, p_m$, $\delta > 0$, $\varepsilon > 0$, $r \ge \varepsilon$, $k \in \mathbb{N}$, the procedure* `generate_candidates`$(P, \delta, r, \varepsilon, k)$ *has running time in*

$$\binom{m+k-2}{k-2} \cdot \mathcal{O}\left(\frac{r}{\varepsilon}\right)^k.$$

Proof. The set $\mathcal{I}$ contains all sequences of indices $(i_1, \ldots, i_\ell) \in [m]^\ell$ such that $\ell \le k$, and $1 = i_1 \le \cdots \le i_\ell = m$. Let $\mathcal{I}_\ell$ be the subset of $\mathcal{I}$ containing the sequences of length ℓ as denoted in `generate_orders`. We first claim that `generate_orders`(m, k) runs in time

```
1: procedure generate_orders($m \in \mathbb{N}$, $k \in \mathbb{N}$)
2:     $\mathcal{I}_2 \leftarrow \{(1, m)\}$
3:     for each $\ell = 3, \ldots, k$ do
4:         $\mathcal{I}_\ell \leftarrow \emptyset$
5:         for each $(i_1, \ldots, i_{\ell-1}) \in \mathcal{I}_{\ell-1}$ do                    $\triangleright$ $i_{\ell-1} = m$
6:             for each $j = i_{\ell-2}, \ldots, m$ do
7:                 $\mathcal{I}_\ell \leftarrow \mathcal{I}_\ell \cup \{(i_1, \ldots, i_{\ell-2}, j, m)\}$
8:     return $\bigcup_{2 \leq \ell \leq k} \mathcal{I}_\ell$
```

Algorithm 1: A call to `generate_orders`$(m,\ k)$ returns all $(i_1, \ldots, i_\ell) \in [m]^\ell$, where $\ell \in [k]$ and such that $1 = i_1 \leq \cdots \leq i_\ell = m$. We assume $k \geq 2$.

```
1: procedure preprocess(input set $\mathcal{P}$, $\delta > 0$, $\varepsilon > 0$, $k \in \mathbb{N}$)
2:     Initialize empty dictionary $\mathcal{H}$
3:     for each $P \in \mathcal{P}$ do
4:         $\mathcal{C}(P) \leftarrow$ generate_keys$(P, \delta, \varepsilon, k)$
5:         for each $Q \in \mathcal{C}(P)$ do
6:             if $Q$ not in $\mathcal{H}$ then
7:                 insert key $Q$ in $\mathcal{H}$, associated with a pointer to $P$
8: procedure generate_keys(curve $P$, $\delta > 0$, $\varepsilon > 0$, $k \in \mathbb{N}$)
9:     $\mathcal{C}' \leftarrow$generate_candidates$(P, \delta, (11+\varepsilon), \varepsilon, k)$
10:    $\mathcal{C} \leftarrow \emptyset$
11:    for each $Q \in \mathcal{C}'$ do
12:        if $\mathrm{d_F}(P, Q) \leq (1+\varepsilon)\delta$ then
13:            $\mathcal{C} \leftarrow \mathcal{C} \cup \{Q\}$
14:    return $\mathcal{C}$
15: procedure generate_candidates(curve $P$ with vertices $p_1, \ldots, p_m$, $\delta > 0$, $r > 0$,
    $\varepsilon > 0$, $k \in \mathbb{N}$)
16:    $\mathcal{S} \leftarrow \emptyset$, $\mathcal{C}' \leftarrow \emptyset$
17:    $\mathcal{I} \leftarrow$generate_orders$(m, k)$
18:    for each $(i_1, \ldots, i_\ell) \in \mathcal{I}$ do
19:        $\mathcal{S} \leftarrow \mathcal{S} \cup \prod_{j=1}^{\ell} B(p_{i_j}, r\delta) \cap \mathcal{G}_{\varepsilon\delta}$
20:    for each $\sigma \in \mathcal{S}$ do
21:        $\mathcal{C}' \leftarrow \mathcal{C}' \cup \{\langle\sigma\rangle\}$
22:    return $\mathcal{C}'$
```

Algorithm 2: Preprocessing algorithm. We call `preprocess` to build the data structure.

```
1: procedure query(curve $Q$ with vertices $q_1, \dots, q_k$, $\delta > 0$, $\varepsilon > 0$)
2:     $\mathcal{I} \leftarrow$generate_orders$(k, k)$
3:     for each $(i_1, \dots, i_\ell) \in \mathcal{I}$ do
4:         $flag \leftarrow 1$
5:         for $j = 1, \dots, \ell - 1$ do
6:             if $\mathrm{d_F}(\overline{q_{i_j} q_{i_{j+1}}}, \langle q_{i_j}, \dots, q_{i_{j+1}} \rangle) > \delta$ then    ▷ test $\delta$-simplification property
7:                 $flag \leftarrow 0$
8:             for each $t = i_j, \dots, i_{j+1}$ do    ▷ test edge-range-preserving property
9:                 if $q_t \notin \overline{q_{i_j} q_{i_{j+1}}}$ then
10:                    $flag \leftarrow 0$
11:        if $flag = 1$ then
12:            $Q' \leftarrow \langle q_{i_1}, \dots, q_{i_\ell} \rangle$    ▷ a $\delta$-straightening of $Q$
13:            $Q'' \leftarrow \left\langle \left\lfloor \frac{q_{i_1}}{\varepsilon\delta} \right\rfloor \cdot (\varepsilon\delta), \dots, \left\lfloor \frac{q_{i_\ell}}{\varepsilon\delta} \right\rfloor \cdot (\varepsilon\delta) \right\rangle$    ▷ snap $Q'$ to $\mathcal{G}_{\varepsilon\delta}$
14:            if $Q''$ in $\mathcal{H}$ then
15:                return input curve $P$ associated with $Q''$ in $\mathcal{H}$
16:    return "no"
```

Algorithm 3: Query algorithm

$\mathcal{O}(|\mathcal{I}| \cdot k)$. To see that, consider any sequence of indices $s \in \mathcal{I}$. During the execution of `generate_orders`, s is added to the sets of indices (Line 7) only once. This step costs $\mathcal{O}(k)$, therefore the running time of `generate_orders`(m, k) is in $\mathcal{O}(|\mathcal{I}| \cdot k)$. Now, let $\mathcal{S}'$ be a multiset which contains all sequences (including duplicates) which are generated and inserted to $\mathcal{S}$ in all executions of Line 19 of `generate_candidates`. The running time of `generate_candidates`$(P, \delta, r, \varepsilon, k)$ is upper bounded by $\mathcal{O}(|\mathcal{S}'| \cdot k)$, because $|\mathcal{S}'| \geq |\mathcal{I}|$ and computing $\mathcal{C}'$ costs $\mathcal{O}(|\mathcal{S}'| \cdot k)$ time. We proceed by showing an upper bound on $|\mathcal{S}'|$.

Any sequence $(x_1, \dots, x_\ell) \in \mathcal{G}^{\ell}_{\varepsilon\delta}$, which is included in $\mathcal{S}'$, may appear in the computation taking place in Line 19 multiple times: once for each sequence of indices $(i_1, \dots, i_\ell) \in \mathcal{I}$ such that for each $j \in [\ell]$, $x_j \in B(p_{i_j}, r\delta)$. Notice that $|\mathcal{I}_\ell|$ is equal to the number of combinations of $\ell - 2$ objects taken (with repetition) from a set of size m, i.e. $|\mathcal{I}_\ell| = \binom{m+\ell-3}{\ell-2}$. Hence, by the Hockey-stick identity,

$$|\mathcal{I}| = \sum_{\ell=2}^{k} |\mathcal{I}_\ell| = \sum_{\ell=2}^{k} \binom{m+\ell-3}{\ell-2} = \sum_{\ell=0}^{k-2} \binom{m+\ell-1}{\ell} = \binom{m+k-2}{k-2}. \tag{4.2}$$

Using (4.2), we can bound $|\mathcal{S}'|$ as follows:

$$\begin{aligned} |\mathcal{S}'| &\leq \sum_{\ell=2}^{k} \sum_{(i_1,\dots i_\ell) \in \mathcal{I}_\ell} \left| \prod_{j=1}^{\ell} B(p_{i_j}, r\delta) \cap \mathcal{G}_{\varepsilon\delta} \right| \\ &\leq \sum_{\ell=2}^{k} |\mathcal{I}_\ell| \cdot \mathcal{O}\left(\frac{r}{\varepsilon}\right)^{\ell} \leq |\mathcal{I}| \cdot \mathcal{O}\left(\frac{r}{\varepsilon}\right)^{k} \leq \binom{m+k-2}{k-2} \cdot \mathcal{O}\left(\frac{r}{\varepsilon}\right)^{k}. \end{aligned}$$

Hence, the running time is $\mathcal{O}(|\mathcal{S}'| \cdot k) = \binom{m+k-2}{k-2} \cdot \mathcal{O}\left(\frac{r}{\varepsilon}\right)^{k}$. □

Lemma 4.22. *If* `query`$(Q, \delta, \varepsilon/2)$ *returns an input curve $P \in \mathcal{P}$, then* $\mathrm{d_F}(Q, P) \leq (1+\varepsilon)\delta$. *If* `query`$(Q, \delta, \varepsilon/2)$ *returns "no" then there is no $P \in \mathcal{P}$ such that* $\mathrm{d_F}(Q, P) \leq \delta$.

Proof. When `query`$(Q, \delta, \varepsilon/2)$ returns an input curve $P \in \mathcal{P}$, it must be that there is a δ-straightening Q' of Q such that P is associated with Q'' in $\mathcal{H}$. This implies that $\mathrm{d_F}(Q'', P) \leq (1+\varepsilon/2)\delta$. By the triangle inequality,

$$\mathrm{d_F}(Q', P) \leq \mathrm{d_F}(Q'', Q') + \mathrm{d_F}(Q'', P) \leq (1+\varepsilon)\delta.$$

Since Q' is a δ-straightening of Q, we have that $\mathrm{d_F}(Q', Q) \leq \delta$. Hence, by Lemma 4.16 applied on P, Q, Q' for distance threshold $(1+\varepsilon)\delta$, we obtain $\mathrm{d_F}(Q, P) \leq (1+\varepsilon)\delta$.

If `query`$(Q, \delta, \varepsilon/2)$ returns "no" then there is no δ-straightening Q' of Q such that Q'' is associated with an input curve in $\mathcal{H}$. Suppose, for the sake of contradiction, that there exists a curve $P \in \mathcal{P}$ such that $\mathrm{d_F}(Q, P) \leq \delta$. By Lemma 4.19, there exists a δ-straightening Q' of Q such that i) there exists an 11δ-visiting order of Q' on P and ii) $\mathrm{d_F}(Q', P) \leq \delta$. Let Q'' be the curve obtained by snapping vertices of Q' to the grid $\mathcal{G}_{\varepsilon\delta/2}$. By the triangle inequality, there exists a $((11+\varepsilon/2)\delta)$-visiting order of Q'' on P and

$$\mathrm{d_F}(Q'', P) \leq \mathrm{d_F}(Q'', Q') + \mathrm{d_F}(Q', P) \leq (1+\varepsilon/2)\delta.$$

Hence, $Q'' \in \mathcal{C}(P)$ and Q'' is associated with some input curve P' in $\mathcal{H}$. This leads to contradiction and we conclude that if `query`$(Q, \delta, \varepsilon/2)$ returns "no" then there is no curve $P \in \mathcal{P}$ such that $\mathrm{d_F}(P, Q) \leq \delta$. □

Lemma 4.23. *For any query curve Q of complexity k, $\delta > 0$, $\varepsilon > 0$,* `query`(Q, δ, ε) *runs in time $\mathcal{O}(k \cdot 2^k)$.*

Proof. Let $q_1, \ldots, q_k$ be the vertices of Q. We enumerate all sequences starting with q_1, followed by any possible subsequence of $q_2, \ldots, q_{k-1}$ and ending with q_k. There are at most 2^{k-2} such sequences, and for each one of them we test whether it defines a δ-straightening of Q. This is done in two steps: we first test if each shortcut is within distance δ from the corresponding subcurve, and then we decide if the edge-range-preserving property is satisfied. Computing the Fréchet distance between a shortcut and the original subcurve costs linear time in the complexity of the subcurve by Theorem 2.2. Hence, we can decide in $\mathcal{O}(k)$ time if the sequence in question defines a δ-simplification of Q. To decide if the edge-range-preserving property is satisfied, we check for each shortcut if the corresponding subcurve stays within range by testing all of its vertices one by one. Therefore, this step also costs $\mathcal{O}(k)$ time. Since we employ perfect hashing, each probe to $\mathcal{H}$ costs $\mathcal{O}(k)$ time. We can also check in $\mathcal{O}(k)$ time if the answer returned by $\mathcal{H}$ is the one we are searching for. Hence, the overall query time is in $\mathcal{O}(k \cdot 2^k)$.

□

Theorem 4.24. *Let $\varepsilon \in (0, 1]$. There is a data structure for the $(1+\varepsilon)$-ANN problem, which stores n one-dimensional curves of complexity m and supports query curves of complexity k, uses space in $n \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$, needs $\mathcal{O}(nm) \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$ expected preprocessing time and answers a query in $\mathcal{O}(k \cdot 2^k)$ time.*

Proof. The data structure is described in Section 4.3.1. Lemma 4.22 shows correctness. The bound on the query time follows from Lemma 4.23. It remains to analyze the running time of `preprocess`$(\mathcal{P}, \delta, \varepsilon/2, k)$ and the space complexity of the data structure.

By Lemma 4.21, for any $P \in \mathcal{P}$, the running time needed to compute $\mathcal{C}'$ is upper bounded by $\binom{m+k-2}{k-2} \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k = \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$. Hence, for each $P \in \mathcal{P}$, $|\mathcal{C}(P)| = \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$. Therefore, the space required for each input curve $P \in \mathcal{P}$ is upper bounded by $\mathcal{O}(|\mathcal{C}(P)| \cdot k)$. Computing $\mathcal{C}(P)$ costs $\mathcal{O}(|\mathcal{C}'| \cdot mk) = \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k \cdot \mathcal{O}(m)$ time, because we need to decide for each curve $Q \in \mathcal{C}'$, whether its Fréchet distance from P is at most $(1+\varepsilon/2)\delta$, which can be done in $\mathcal{O}(|Q| \cdot |P|)$ time using Theorem 2.2. Assuming perfect hashing for $\mathcal{H}$, the overall expected preprocessing time is in $\mathcal{O}(nm) \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$ and the space usage is in $\mathcal{O}(n) \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$. □

4.4 $(2+\varepsilon)$-Approximation

In this section we present three $(2+\varepsilon)$-ANN data structures with different tradeoffs between preprocessing and query time.

4.4.1 Fast Query Algorithm

In this section, we propose a data structure for the $(2+\varepsilon)$-ANN problem, with query time in $\mathcal{O}(k)$. The space complexity and the preprocessing time are the same as in the $(1+\varepsilon)$-ANN data structure of Theorem 4.24.

Data structure. We are given as input a set of one-dimensional curves $\mathcal{P}$, as sequences of vertices, the distance threshold $\delta > 0$, the approximation error $\varepsilon > 0$ and the complexity of the supported queries k. The data structure is exactly the same as in Section 4.3. To build it, we call `preprocess`$(P, \delta, \varepsilon/2, k)$, as defined in Algorithm 2, in Section 4.3.1. Let $\mathcal{H}$ be the resulting dictionary, constructed by `preprocess`$(P, \delta, \varepsilon/2, k)$.

Query algorithm. Let Q be the query curve with vertices $q_1, \ldots, q_k$ and let $\varepsilon > 0$ be the approximation error. The query algorithm first computes a δ-signature Q' of Q, and then it snaps the vertices of Q' to the grid $\mathcal{G}_{\varepsilon\delta/2}$, to obtain a curve Q''. If Q'' is stored in $\mathcal{H}$, then we return its associated input curve $P \in \mathcal{P}$, otherwise we return "no". The query algorithm is implemented in `query2`, which can be found in Algorithm 4. To achieve approximation factor $2+\varepsilon$, we run `query2`$(Q, \delta, \varepsilon/2)$.

```
1: procedure query2(curve $Q$ with vertices $q_1, \ldots, q_k$, $\delta > 0$, $\varepsilon > 0$)
2:     $Q' \leftarrow \delta$-signature of $Q$
3:     $q'_1, \ldots, q'_\ell \leftarrow$ vertices of $Q'$
4:     $Q'' \leftarrow \left\langle \left\lfloor \frac{q'_1}{\varepsilon\delta} \right\rfloor \cdot (\varepsilon\delta), \ldots, \left\lfloor \frac{q'_\ell}{\varepsilon\delta} \right\rfloor \cdot (\varepsilon\delta) \right\rangle$        ▷ snap $Q'$ to $\mathcal{G}_{\varepsilon\delta}$
5:     if $Q''$ in $\mathcal{H}$ then
6:         return input curve $P$ associated with $Q''$ in $\mathcal{H}$
7:     return "no"
```

Algorithm 4: Query algorithm

Lemma 4.25. *If* `query2`$(Q, \delta, \varepsilon/2)$ *returns an input curve* $P \in \mathcal{P}$, *then* $\mathrm{d_F}(Q, P) \leq (2+\varepsilon)\delta$. *If* `query2`$(Q, \delta, \varepsilon/2)$ *returns "no" then there is no* $P \in \mathcal{P}$ *such that* $\mathrm{d_F}(Q, P) \leq \delta$.

Proof. If `query2`$(Q, \delta, \varepsilon/2)$ returns an input curve $P \in \mathcal{P}$, then it must be that Q'' is stored in $\mathcal{H}$, and P is its associated input curve. By the construction of $\mathcal{H}$, it must be that $\mathrm{d_F}(P, Q'') \leq (1+\varepsilon/2)\delta$. By the definition of signatures we know that $\mathrm{d_F}(Q, Q') \leq \delta$, and by the triangle inequality we obtain

$$\mathrm{d_F}(Q, Q'') \leq \mathrm{d_F}(Q, Q') + \mathrm{d_F}(Q'', Q') \leq (1+\varepsilon/2)\delta.$$

Hence, by the triangle inequality we obtain

$$\mathrm{d_F}(P, Q) \leq \mathrm{d_F}(P, Q'') + \mathrm{d_F}(Q, Q'') \leq (2+\varepsilon)\delta.$$

Now suppose that `query2`$(Q, \delta, \varepsilon/2)$ returns "no". This means that Q'' is not stored in $\mathcal{H}$. Suppose that there exists a $P \in \mathcal{P}$ such that $\mathrm{d_F}(P, Q) \leq \delta$. Then by Lemma 4.15 there exists a δ-visiting order of Q' on P. Therefore, by the triangle inequality, there exists a $((1+\varepsilon/2)\delta)$-visiting order of Q'' on P, which implies that $Q'' \in \mathcal{C}(P)$, and hence Q'' is stored in $\mathcal{H}$. This leads to a contradiction, since we have assumed that Q'' is not stored in $\mathcal{H}$. Hence, if `query2`$(Q, \delta, \varepsilon/2)$ returns "no" then there is no $P \in \mathcal{P}$ such that $\mathrm{d_F}(P, Q) \leq \delta$. ☐

Theorem 4.26. *Let $\varepsilon \in (0, 1]$. There is a data structure for the $(2+\varepsilon)$-ANN problem, which stores n one-dimensional curves of complexity m and supports query curves of complexity k, uses space in $n \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$, needs $\mathcal{O}(nm) \cdot \mathcal{O}\left(\frac{m}{k\varepsilon}\right)^k$ expected preprocessing time and answers a query in $\mathcal{O}(k)$ time.*

Proof. Correctness of the data structure follows from Lemma 4.25. The space complexity and the preprocessing time are analyzed in the proof of Theorem 4.24. It remains to show that `query2`$(Q, \delta, \varepsilon/2)$ runs in $\mathcal{O}(k)$ time.

To compute a δ-signature of Q, we use the algorithm of Driemel, Krivosija and Sohler [83], which runs in $\mathcal{O}(k)$ time. Since we employ perfect hashing and we assume that the floor function can be computed in constant time, each probe to $\mathcal{H}$ costs $\mathcal{O}(k)$ time, and we can also check at the same time if the answer returned by $\mathcal{H}$ is the one we are searching for. We conclude that `query2`$(Q, \delta, \varepsilon/2)$ runs in $\mathcal{O}(k)$ time. ☐

4.4.2 Improved Preprocessing Time

In this section, we show that there exists a data structure for the $(2+\varepsilon)$-ANN problem, with space complexity and preprocessing time in $n \cdot \mathcal{O}(1/\varepsilon)^k + \mathcal{O}(nm)$. The query time is in $\mathcal{O}(k \cdot 2^k)$. This avoids the factor $(m/k)^k$ of our previous data structures.

Data structure. We are given as input a set of one-dimensional curves $\mathcal{P}$, as sequences of vertices, the distance threshold $\delta > 0$, the approximation error $\varepsilon > 0$, and the complexity of the supported queries k. To build the data structure, we use a modified version of the preprocessing algorithm in Section 4.3. For each input curve $P \in \mathcal{P}$, we compute a δ-signature P' of P. If the complexity of P' is at most $k+2$ then we compute a set $\mathcal{C}' := \mathcal{C}'(P')$ which contains all curves Q such that: i) Q has complexity at most k, ii) all vertices of Q belong to $\mathcal{G}_{\varepsilon\delta/2}$, and iii) there is a $((16+\varepsilon/4)\delta)$-visiting order of Q on P'. This step is similar to the one in the preprocessing algorithm in Section 4.3, although here we consider signatures of the input curves instead of the original curves.

The filtering process is also slightly different. We filter $\mathcal{C}'$ to obtain a set $\mathcal{C}(P)$ which contains only those curves of $\mathcal{C}'$ with: i) Fréchet distance at most $(2+\varepsilon/4)\delta$ from P, ii) their first point within distance $(1+\varepsilon/4)\delta$ from $P(0)$, and iii) their last point within distance $(1+\varepsilon/4)\delta$ from $P(1)$. Let $\mathcal{H}$ be a dictionary which is initially empty. For each $P \in \mathcal{P}$, we store $\mathcal{C}(P)$ in $\mathcal{H}$ as follows: for each $Q \in \mathcal{C}(P)$, if Q is not already stored in $\mathcal{H}$, then we insert Q into $\mathcal{H}$, associated with a pointer to P. The preprocessing algorithm is implemented in `preprocess2`, which can be found in Algorithm 5. We also make use of the subroutine `generate_candidates` described in Algorithm 2, in Section 4.3.1. To achieve approximation factor $(2+\varepsilon)$, we run `preprocess2`$(\mathcal{P}, \delta, 22, 2, \varepsilon/4, k)$.

Query algorithm. Let Q be the query curve with vertices $q_1, \ldots, q_k$ and let $\varepsilon > 0$ be the approximation error. The query algorithm is the same as in the data structure of Section 4.3, but we run it with different input parameters. In particular, we run `query`$(Q, 2\delta, \varepsilon/4)$ (see Algorithm 3) on the dictionary $\mathcal{H}$ which is constructed by `preprocess2`$(\mathcal{P}, \delta, 22, 2, \varepsilon/4, k)$.

```
1: procedure preprocess2(input set $\mathcal{P}$, $\delta > 0$, $r > 0$, $t > 0$, $\varepsilon > 0$, $k \in \mathbb{N}$)
2:     Initialize empty dictionary $\mathcal{H}$
3:     for each $P \in \mathcal{P}$ do
4:         $P' \leftarrow \delta$-signature of $P$
5:         if $|P'| \le k+2$ then
6:             $\mathcal{C}(P) \leftarrow$ generate_keys2($P', \delta, r, t, \varepsilon, k$)
7:             for each $Q \in \mathcal{C}(P)$ do
8:                 if $Q$ not in $\mathcal{H}$ then
9:                     insert key $Q$ in $\mathcal{H}$, associated with a pointer to $P$
10: procedure generate_keys2(curve $P$, $\delta > 0$, $r > 0$, $t > 0$, $\varepsilon > 0$, $k$)
11:     $\mathcal{C}' \leftarrow$generate_candidates($P, \delta, r+\varepsilon, \varepsilon, k$)
12:     $\mathcal{C} \leftarrow \emptyset$
13:     for each $Q \in \mathcal{C}'$ do
14:         if $d_F(P,Q) \le (t+\varepsilon)\delta$ and $|P(0)-Q(0)| \le (1+\varepsilon)\delta$ and $|P(1)-Q(1)| \le (1+\varepsilon)\delta$
    then
15:             $\mathcal{C} \leftarrow \mathcal{C} \cup \{Q\}$
16:     return $\mathcal{C}$
```

Algorithm 5: Preprocessing algorithm. We call `preprocess2` to build the data structure.

Lemma 4.27. *If* ***query***$(Q, 2\delta, \varepsilon/4)$ *returns an input curve* $P \in \mathcal{P}$*, then* $d_F(Q,P) \le (2+\varepsilon)\delta$*. If* ***query***$(Q, 2\delta, \varepsilon/4)$ *returns "no" then there is no* $P \in \mathcal{P}$ *such that* $d_F(Q,P) \le \delta$*.*

Proof. When `query`$(Q, 2\delta, \varepsilon/4)$ returns an input curve $P \in \mathcal{P}$, it must be that there is a δ-straightening Q' of Q such that P is associated with Q'' in $\mathcal{H}$, where Q'' denotes the curve produced by snapping vertices of Q' to $\mathcal{G}_{\varepsilon\delta/4}$. This implies that $Q'' \in \mathcal{C}(P)$, and therefore $d_F(P', Q'') \le (2+\varepsilon/4)\delta$, $|P'(0) - Q''(0)| \le (1+\varepsilon/4)\delta$, $|P'(1) - Q''(1)| \le (1+\varepsilon/4)\delta$, where P' is the δ-signature of P computed by `preprocess`. By the triangle inequality,

$$d_F(P', Q') \le d_F(P', Q'') + d_F(Q', Q'') \le (2+\varepsilon/2)\delta.$$

Similarly, by the triangle inequality, $|P'(0) - Q'(0)| \leq (1+\varepsilon/2)\delta$, $|P'(1) - Q'(1)| \leq (1+\varepsilon/2)\delta$. Lemma 4.17 implies that $\mathrm{d_F}(P, Q') \leq (2+\varepsilon)\delta$, because P' is a δ-signature of P, $\mathrm{d_F}(P', Q') \leq (2+\varepsilon/2)\delta$, $|P(0) - Q'(0)| \leq (1+\varepsilon/2)\delta$ and $|P(1) - Q'(1)| \leq (1+\varepsilon/2)\delta$. Then, by Lemma 4.16, we conclude that $\mathrm{d_F}(P, Q) \leq (2+\varepsilon)\delta$.

If `query`$(Q, 2\delta, \varepsilon/4)$ returns "no", then there is no input curve $P \in \mathcal{P}$ such that $|P'| \leq k+2$, where P' is the δ-signature computed by `preprocess2` and such that there exists a δ-straightening Q' of Q with $Q' \in \mathcal{C}(P)$. Suppose for the sake of contradiction that there is an input curve $P \in \mathcal{P}$ such that $\mathrm{d_F}(Q, P) \leq \delta$. Then by the triangle inequality and the fact that $\mathrm{d_F}(P, P') \leq \delta$, we obtain $\mathrm{d_F}(Q, P') \leq 2\delta$. In addition, by Lemma 4.15 there is a δ-visiting order of P' on Q. Since P' satisfies the δ-edge-length property, any two consecutive interior vertices lie at distance at least 2δ to each other. Thus, no two consecutive interior vertices can belong to the same δ-range. Hence, $|P'| \leq |Q|+2 \leq k+2$. By Lemma 4.19, there exists a 2δ-straightening Q' of Q which satisfies

i) there exists a 22δ-visiting order of Q' on P',

ii) $\mathrm{d_F}(Q', P') \leq 2\delta$.

By the definition of signatures, we have $P(0) = P'(0)$ and $P(1) = P'(1)$, and since $\mathrm{d_F}(P, Q) \leq \delta$, we have $|P'(0) - Q(0)| \leq \delta$ and $|P'(1) - Q(1)| \leq \delta$. By the definition of straightenings, we have $Q'(0) = Q(0)$ and $Q'(1) = Q(1)$ and therefore $|P'(0) - Q'(0)| \leq \delta$ and $|P'(1) - Q'(1)| \leq \delta$. Hence, by the triangle inequality there exists a $((22+\varepsilon/4)\delta)$-visiting order of Q'' on P', $\mathrm{d_F}(Q'', P') \leq (2+\varepsilon/4)\delta$, $|P'(0) - Q''(0)| \leq (1+\varepsilon/4)\delta$ and $|P'(1) - Q''(1)| \leq (1+\varepsilon/4)\delta$. This implies that $Q'' \in \mathcal{C}(P)$ which leads to a contradiction. □

Theorem 4.28. *Let $\varepsilon \in (0,1]$. There is a data structure for the $(2+\varepsilon)$-ANN problem, which stores n one-dimensional curves of complexity m and supports query curves of complexity k, uses space in $n \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k + \mathcal{O}(nm)$, needs $n \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k + \mathcal{O}(nm)$ expected preprocessing time and answers a query in $\mathcal{O}(k \cdot 2^k)$ time.*

Proof. Correctness follows from Lemma 4.27. The bound on the query time follows from Lemma 4.23. It remains to bound the space complexity and the preprocessing time of the data structure.

Computing one δ-signature for each $P \in \mathcal{P}$ takes linear time $\mathcal{O}(mn)$ in total, using the algorithm of Driemel, Krivosija and Sohler [83]. Let P' be the δ-signature of some curve $P \in \mathcal{P}$ as computed during preprocessing. If $|P'| > k$ we ignore P. By Lemma 4.21, for any $P' \in \mathcal{P}'$, the running time needed to compute $\mathcal{C}'$, is upper bounded by $\binom{|P'|+k-2}{k-2} \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k = \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k$. The space required for P' is upper bounded by $\mathcal{O}(|\mathcal{C}(P')| \cdot k + m) = \mathcal{O}(|\mathcal{C}'| \cdot k + m) = \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k + \mathcal{O}(m)$. Computing $\mathcal{C}(P')$ costs $\mathcal{O}(|\mathcal{C}'| \cdot k) = \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k$ time, since we take a decision on the Fréchet distance between each curve in $\mathcal{C}'$, and P', by making use of Theorem 2.2 . Assuming perfect hashing for $\mathcal{H}$, the overall expected preprocessing time is in $\mathcal{O}(n) \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k$ and the space usage is in $\mathcal{O}(n) \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^k$. □

4.4.3 Linear Preprocessing Time

In this section we present a data structure for the $(2+\varepsilon)$-ANN problem with linear space and preprocessing time $\mathcal{O}(nm)$ and with query time in $\mathcal{O}(1/\varepsilon)^k$.

```
1: procedure preprocess3(input set $\mathcal{P}$, $\delta > 0$, $\varepsilon > 0$, $k$)
2:     Initialize empty dictionary $\mathcal{H}$
3:     for each $P \in \mathcal{P}$ do
4:         $P' \leftarrow \delta$-signature of $P$
5:         if $|P'| \leq k + 2$ then
6:             $p_1, \ldots, p_\ell \leftarrow$ vertices of $P'$
7:             $P'' \leftarrow \langle \lfloor \frac{p_1}{\varepsilon\delta} \rfloor \cdot (\varepsilon\delta), \ldots, \lfloor \frac{p_\ell}{\varepsilon\delta} \rfloor \cdot (\varepsilon\delta) \rangle$
8:             if $P''$ not in $\mathcal{H}$ then
9:                 insert key $P''$ in $\mathcal{H}$, associated with a pointer to $P$
```

Algorithm 6: Preprocessing algorithm

```
1: procedure query3(curve $Q$ with vertices $q_1, \ldots, q_k$, $\delta > 0$, $\varepsilon > 0$)
2:     $\mathcal{C}(Q) \leftarrow$ generate_keys2($Q, \delta, 1, 2, \varepsilon, k + 2$)
3:     for each $P'' \in \mathcal{C}(Q)$ do
4:         if $P''$ in $\mathcal{H}$ then
5:             return input curve $P$ associated with $P''$ in $\mathcal{H}$
6:     return “no”
```

Algorithm 7: Query algorithm

Data structure. We are given as input a set of one-dimensional curves $\mathcal{P}$, as sequences of vertices, a distance threshold $\delta > 0$, the approximation error $\varepsilon > 0$ and the complexity of the supported queries k. For each input curve $P \in \mathcal{P}$, we compute a δ-signature P' of P. If $|P'| > k + 2$ then we ignore P, otherwise we snap it to $\mathcal{G}_{\varepsilon\delta/2}$ to obtain a curve P''. Let $\mathcal{H}$ be a dictionary which is initially empty. For each $P \in \mathcal{P}$, we store P'' in $\mathcal{H}$ as follows: if P'' is not already stored in $\mathcal{H}$, then we insert P'' into $\mathcal{H}$, associated with a pointer to P. To achieve approximation factor $2 + \varepsilon$, we run `preprocess3`$(\mathcal{P}, \delta, \varepsilon/2, k)$, as defined in Algorithm 6.

Query algorithm. Let Q be a query curve of complexity k. We compute a set $\mathcal{C}' := \mathcal{C}'(Q)$ which contains all curves P such that: i) P has complexity at most k, ii) all vertices of P belong to $\mathcal{G}_{\varepsilon\delta/2}$, and iii) there is a $((1+\varepsilon/2)\delta)$-visiting order of P on Q. We filter $\mathcal{C}'$ to obtain a set $\mathcal{C}(Q)$ which contains only those curves of $\mathcal{C}'$ with: i) Fréchet distance at most $(2+\varepsilon/2)\delta$ from Q, ii) their first point within distance $(1+\varepsilon/2)\delta$ from $Q(0)$, and iii) their last point within distance $(1+\varepsilon/2)\delta$ from $Q(1)$. We probe $\mathcal{H}$ for each key $P \in \mathcal{C}(Q)$: if we find a $P \in \mathcal{C}(Q)$ stored in $\mathcal{H}$ then we return the associated input curve. If there is no $P \in \mathcal{C}(Q)$ stored in $\mathcal{H}$ then we return “no”. To achieve the desired approximation, we run `query3`$(Q, \delta, \varepsilon/2)$, as defined in Algorithm 7.

Lemma 4.29. *If `query3`$(Q, \delta, \varepsilon/2)$ returns an input curve $P \in \mathcal{P}$, then* $\mathrm{d_F}(Q, P) \leq (2+\varepsilon)\delta$. *If `query3`$(Q, \delta, \varepsilon/2)$ returns “no” then there is no $P \in \mathcal{P}$ such that* $\mathrm{d_F}(Q, P) \leq \delta$.

Proof. If `query3`$(Q, \delta, \varepsilon/2)$ returns an input curve, then it must be that there is a curve $P'' \in \mathcal{C}(Q)$ which is stored in $\mathcal{H}$, associated with a pointer to P. Since P'' is stored in $\mathcal{H}$, there is a curve $P \in \mathcal{P}$ with a δ-signature P' such that $\mathrm{d_F}(P', P'') \leq \varepsilon\delta/2$. Moreover, since $P'' \in \mathcal{C}(Q)$, we have that $\mathrm{d_F}(Q, P'') \leq (2 + \varepsilon/2)\delta$, $|Q(0) - P''(0)| \leq (1 + \varepsilon/2)\delta$,

$|Q(1) - P''(1)| \leq (1+\varepsilon/2)\delta$. By the triangle inequality we obtain, $\mathrm{d_F}(Q, P') \leq (2+\varepsilon)\delta$, $|Q(0) - P'(0)| \leq (1+\varepsilon)\delta$, $|Q(1) - P'(1)| \leq (1+\varepsilon)\delta$. By Lemma 4.17, since $\mathrm{d_F}(Q, P') \leq (2+\varepsilon)\delta$, $|Q(0) - P'(0)| \leq (1+\varepsilon)\delta$, $|Q(1) - P'(1)| \leq (1+\varepsilon)\delta$ and P' is a δ-signature of P, we conclude $\mathrm{d_F}(Q, P) \leq (2+\varepsilon)\delta$.

If `query3`$(Q, \delta, \varepsilon/2)$ returns "no", then it must be that there is no curve $P'' \in \mathcal{C}(Q)$ which is stored in $\mathcal{H}$. Suppose for the sake of contradiction that there is an input curve $P \in \mathcal{P}$ such that $\mathrm{d_F}(Q, P) \leq \delta$. Let P' be the δ-signature of P, as computed during preprocessing. By Lemma 4.15, there is a δ-visiting order of P' on Q and therefore $|P'| \leq k+2$. Let P'' be the curve produced by snapping the vertices of P' to the grid $\mathcal{G}_{\varepsilon\delta/2}$. By the triangle inequality there is a $((1+\varepsilon/2)\delta)$-visiting order of P'' on Q. Therefore, P'' must be included in $\mathcal{C}(Q)$, which leads to contradiction. □

Theorem 4.30. *Let $\varepsilon \in (0, 1]$. There is a data structure for the $(2+\varepsilon)$-ANN problem, which stores n one-dimensional curves of complexity m and supports query curves of complexity k, uses space in $\mathcal{O}(nm)$, needs $\mathcal{O}(nm)$ expected preprocessing time and answers a query in $\mathcal{O}(1/\varepsilon)^{k+2}$ time.*

Proof. Correctness follows from Lemma 4.29. It remains to bound the space complexity, the preprocessing time and the query time.

Using the algorithm of Driemel, Krivosija and Sohler [83], we can compute a signature in linear time. Since we assume that the floor function can be computed in $\mathcal{O}(1)$, and that $\mathcal{H}$ is implemented using perfect hashing, `preprocess3`$(\mathcal{P}, 1, \varepsilon/2, k)$ has running time $\mathcal{O}(nm)$. Therefore, the space usage is also in $\mathcal{O}(nm)$.

We now show the upper bound on the query time. To this end, we first bound the running time of `generate_keys2`$(Q, \delta, 1, 2, \varepsilon/2, k+2)$, because the last part of `query3` is an enumeration over all curves returned by `generate_keys2` and probing $\mathcal{H}$ for each one of them. To bound the running time of `generate_keys2`$(Q, \delta, 1, 2, \varepsilon/2, k+2)$, it suffices to bound the running time of `generate_candidates`$(Q, \delta, (1+\varepsilon/2), \varepsilon/2, k+2)$. By Lemma 4.21, this running time is upper bounded by $\binom{2k}{k-2} \cdot \mathcal{O}\left(\frac{1}{\varepsilon}\right)^{k+2} = \mathcal{O}\left(\frac{1}{\varepsilon}\right)^{k+2}$. Recall that we employ perfect hashing and we assume that the floor function can be computed in constant time. Hence each probe to $\mathcal{H}$ costs $\mathcal{O}(k)$ time, and we can also check in $\mathcal{O}(k)$ if $\mathcal{H}$ returns the correct answer. We conclude that `query2`$(Q, \delta, \varepsilon/2)$ runs in time $\mathcal{O}\left(\frac{1}{\varepsilon}\right)^{k+2}$. □

4.5 $(3+\varepsilon)$-Approximation

In this section, we present a data structure for the $(3+\varepsilon)$-ANN problem with preprocessing time and space complexity in $n \cdot \mathcal{O}(1/\varepsilon)^k + \mathcal{O}(nm)$ and query time in $\mathcal{O}(k)$.

Data structure. We are given as input a set of one-dimensional curves $\mathcal{P}$, as sequences of vertices, a distance threshold $\delta > 0$, the approximation error $\varepsilon > 0$ and the complexity of the supported queries k. To build the data structure, we use the preprocessing algorithm of the data structure in Section 4.4.2. Let $\mathcal{H}$ be the dictionary, constructed by `preprocess2`$(\mathcal{P}, \delta, 2, 3, \varepsilon/2, k)$.

Query algorithm. Let Q be a query curve. We run the query algorithm of the data structure in Section 4.4.1. In particular, we run `query2`$(Q, 2\delta, \varepsilon/2)$ on $\mathcal{H}$.

Lemma 4.31. *If `query2`$(Q, 2\delta, \varepsilon/2)$ returns an input curve $P \in \mathcal{P}$, then $\mathrm{d_F}(Q,P) \leq (3+\varepsilon)\delta$. If `query2`$(Q, 2\delta, \varepsilon/2)$ returns "no" then there is no $P \in \mathcal{P}$ such that $\mathrm{d_F}(Q,P) \leq \delta$.*

Proof. Let Q' be the 2δ-signature of Q and let Q'' be the curve obtained by snapping vertices of Q' to $\mathcal{G}_{\varepsilon\delta/2}$, as computed in `query2`.

If `query2`$(Q, 2\delta, \varepsilon/2)$ returns an input curve $P \in \mathcal{P}$, then it must be that $Q'' \in \mathcal{C}(P)$, where $\mathcal{C}(P)$ is the result of `generate_keys2`$(P', \delta, 2, 3, \varepsilon/2, k)$ and P' is a δ-signature of P, as computed by `preprocess2`. By the construction of $\mathcal{C}(P)$, it must be that $\mathrm{d_F}(P', Q'') \leq (3+\varepsilon/2)\delta$, $|P'(0) - Q''(0)| \leq (1+\varepsilon/2)\delta$ and $|P'(1) - Q''(1)| \leq (1+\varepsilon/2)\delta$. Hence, by the triangle inequality $\mathrm{d_F}(Q', P') \leq (3+\varepsilon)\delta$, $|P'(0) - Q'(0)| \leq (1+\varepsilon)\delta$ and $|P'(1) - Q'(1)| \leq (1+\varepsilon)\delta$. We now apply Lemma 4.17 twice. We first apply it on P', Q', Q. Since $\mathrm{d_F}(P', Q') \leq (3+\varepsilon)\delta$, $|P'(0) - Q'(0)| \leq (1+\varepsilon)\delta$, $|P'(1) - Q'(1)| \leq (1+\varepsilon)\delta$ and Q' is a 2δ-signature of Q, we obtain $\mathrm{d_F}(P', Q) \leq (3+\varepsilon)\delta$. Then, we apply it on P', P, Q. Since $\mathrm{d_F}(P', Q) \leq (3+\varepsilon)\delta$, $|P'(0) - Q(0)| = |P'(0) - Q'(0)| \leq (1+\varepsilon)\delta \leq (2+\varepsilon)\delta$, $|P'(1) - Q(1)| = |P'(1) - Q'(1)| \leq (1+\varepsilon)\delta \leq (2+\varepsilon)\delta$, and P' is a δ-signature of P, we obtain $\mathrm{d_F}(P, Q) \leq (3+\varepsilon)\delta$.

If `query2`$(Q, 2\delta, \varepsilon/2)$ returns "no" then Q'' is not stored in $\mathcal{H}$ as a key. For the sake of contradiction, we assume that there exists an input curve $P \in \mathcal{P}$ such that $\mathrm{d_F}(P, Q) \leq \delta$. Then by definition, $|P'(0) - Q'(0)| \leq \delta$ and $|P'(1) - Q'(1)| \leq \delta$. In addition, by Lemma 4.20, $\mathrm{d_F}(P', Q') \leq 3\delta$ and there is a 2δ-visiting order of Q' on P', By the triangle inequality we obtain $\mathrm{d_F}(P', Q'') \leq (3+\varepsilon/2)\delta$, $|P'(0) - Q''(0)| \leq (1+\varepsilon/2)\delta$, $|P'(1) - Q''(1)| \leq (1+\varepsilon/2)\delta$, and that there is a $((2+\varepsilon/2)\delta)$-visiting order of Q'' on P'. Hence, by the construction of $\mathcal{C}(P)$, it must be that $Q'' \in \mathcal{C}(P)$ which implies that Q'' is stored as a key in $\mathcal{H}$. This is a contradiction. ☐

Theorem 4.32. *Let $\varepsilon \in (0, 1]$. There is a data structure for the $(3+\varepsilon)$-ANN problem, which stores n one-dimensional curves of complexity m and supports query curves of complexity k, uses space in $n \cdot \mathcal{O}(1/\varepsilon)^k + \mathcal{O}(nm)$, needs $n \cdot \mathcal{O}(1/\varepsilon)^k + \mathcal{O}(nm)$ expected preprocessing time and answers a query in $\mathcal{O}(k)$ time. where k is the complexity of the query curve.*

Proof. Correctness follows from Lemma 4.31. The bounds on the preprocessing time and space complexity follow from Theorem 4.28. The bound on the query time follows from Theorem 4.26. ☐

4.6 Proofs of Main Lemmas

In this section we give full proofs of the lemmas stated in Section 4.2. We start by proving a fundamental observation and lemma on the Fréchet distance of approximately monotone one-dimensional curves.

Observation 4.33. *Let Q be a directed line segment and let $P : [0, 1] \mapsto \mathbb{R}$ be a curve. It holds that $\mathrm{d_F}(P, Q) \leq \delta$ if and only if the following conditions are satisfied:*

(i) P is 2δ-monotone with respect to Q, and

(ii) $|P(0) - Q(0)| \leq \delta$, $|P(1) - Q(1)| \leq \delta$, and

(iii) $P \subseteq B(Q, \delta)$.

Proof. We assume that $Q(0) \leq Q(1)$ as the other case is symmetric. Now, assume first that $\mathrm{d_F}(P, Q) \leq \delta$, then (ii) holds because start and end points are matched in any traversal and (iii) holds as the Hausdorff distance is a lower bound for the Fréchet distance. Finally, (i) holds as otherwise there exist two indices $s, t \in [0, 1]$ with $s < t$ and $P(t) < P(s) - 2\delta$. As Q is increasing, no traversal can match $P(s)$ and $P(t)$ in distance at most δ.

Second, assume that (i), (ii), and (iii) hold. Then $\mathrm{d_F}(P, Q) \leq \delta$ is implied by Lemma 4.34, below, but to provide some intuition we give a simpler proof here. The following traversal with position s on P and position t on Q stays within distance δ. We start in $P(0), Q(0)$, then we continue on P until $P(s) = Q(0) + \delta$. Then we always choose t such that $Q(t) = \min_{s' \geq s} P(s') + \delta$ while traversing P, i.e., continuously increasing s. When we reach the end of Q, we can traverse P until the end while staying in $Q(1)$. It is easy to check that properties (i), (ii), and (iii) ensure distance δ during the described traversal. □

The following lemma statement is similar to the above observation with the important difference that the line segment Q is replaced by a 2δ-monotone curve. The proof works by constructing a traversal greedily and showing correctness of the greedy algorithm.

Lemma 4.34. *Let P and Q be 2δ-monotone curves with*
(i) P is 2δ-monotone with respect to $\overline{Q(0)Q(1)}$, and
(ii) $|P(0) - Q(0)| \leq \delta$, $|P(1) - Q(1)| \leq \delta$, and
(iii) $P \subseteq B(Q, \delta)$, and
(iv) $Q \subseteq \overline{Q(0)Q(1)}$.
It holds that $\mathrm{d_F}(P, Q) \leq \delta$.

Proof. We assume that $Q(0) \leq Q(1)$ as the other case is symmetric. If Q is not 2δ-monotone increasing, then it also cannot be 2δ-monotone decreasing: if there are two points $s, t \in [0, 1]$ with $s < t$ such that $Q(t) < Q(s) - 2\delta$, then, as $Q(t) \geq Q(0)$ by condition (iv), we have that $Q(s) > Q(t) + 2\delta \geq Q(0) + 2\delta$ and thus Q is not 2δ-monotone decreasing. However, as Q is 2δ-monotone, it has to be 2δ-monotone *increasing*. Due to condition (i), P is also 2δ-monotone *increasing*. We give a traversal of P, Q with distance at most δ — denoting the position during the traversal with $(s, t) \in [0, 1]^2$ — that tries to maintain two invariants:

(1) P and Q are in a position $(s, t) \in [0, 1]^2$ such that $P(s) = Q(t) + \delta$.

(2) The suffix of Q is strictly greater than the current value $Q(t)$, i.e., $\forall t' > t : Q(t') > Q(t)$.

In general, both invariants may be violated at the very beginning of the traversal, that is, for $s = t = 0$. Let us first describe how we traverse from the beginning of P, Q to a position $(s, t) \in [0, 1]^2$ such that these invariants are fulfilled. We first traverse P until it first reaches $Q(0) + \delta$, while in Q we stay in $Q(0)$. Note that by condition (ii), we cannot have $P(0) > Q(0) + \delta$. Furthermore, this traversal is feasible as the traversed prefix of P is in the range $[Q(0) - \delta, Q(0) + \delta]$, by condition (iii), and thus within distance δ to $Q(0)$. If we reach $P(1)$ before reaching $Q(0) + \delta$, then we know that $Q \subseteq [P(1) - \delta, P(1) + \delta]$ and we can thus traverse complete Q and $\mathrm{d_F}(P, Q) \leq \delta$. If we did not reach $P(1)$, we now traverse Q until its last point with value $Q(0)$, which is possible as the traversed prefix of Q lies in $[Q(0), Q(0) + 2\delta]$, due to condition (iv) and as Q is 2δ-monotone increasing, and the position on P is currently $Q(0) + \delta$.

From now on, we traverse P and Q with the same speed in image space, unless one of the two invariants would be violated by continuing the traversal. If both invariants would be violated at the same time, we break ties by restoring Invariant ((1)) before Invariant ((2)). Now, let s be the position on P and t the position on Q when an invariant would be violated. When Invariant ((1)) would be violated, we continue traversing P while staying in $Q(t)$ on Q until the next time we reach a position s' on P with value $P(s') = P(s)$. Note that we might not reach such a position s' because we reached the end of P. However, if we did not reach the end of P, the invariant is restored. This traversal keeps the two positions at distance δ as $P(s) = Q(t) + \delta$ and as P is 2δ-monotone increasing. In case Invariant ((2)) would be violated, we continue traversing Q until we reach the largest position $t' > t$ such that $Q(t') = Q(t)$. Note that afterwards, both invariants hold (as we restore Invariant ((1)) before Invariant ((2))), and, in particular, we cannot reach the end of Q due to the existence of $Q(t')$ which we reach at the end of restoring Invariant ((2)). This traversal also keeps the two positions at distance δ as initially $Q(t) = P(s) - \delta$ and Q is 2δ-monotone increasing and there is no position t'' on Q with $Q(t'') < Q(t)$, i.e., all the points before reaching position t' on Q have to be in the range $[Q(t), Q(t) + 2\delta]$.

In all of the above cases we are guaranteed to make progress in our traversal. Furthermore, we will reach the end of P before or at the same time as we reach the end of Q because, first, while restoring invariants we can only reach the end of P but not of Q as argued above and, second, if we reach the end of Q while both invariants would continue to hold, we also have to reach the end of P at the same time as otherwise we would violate condition (iii) of the lemma. When we reach the end of P, we know that $P(1) \in [Q(1) - \delta, Q(1) + \delta]$ due to condition (ii), and the remaining Q is in $[P(1) - \delta, Q(1)]$. Thus, the remaining Q is in $[P(1) - \delta, P(1) + \delta]$ and consequently Q can be traversed until the end.

It follows from the traversal constructed thereby that $\mathrm{d_F}(P, Q) \leq \delta$. $\square$

4.6.1 Proofs of Lemmas for Straightenings

Next, we want to prove Lemma 4.16 from Section 4.2. We first prove a simpler statement, which can be thought of as a special case where the straightening consists of only one edge.

Lemma 4.35. *Let $X = \overline{ab} \subset \mathbb{R}$ be a line segment and let $Q: [0,1] \mapsto \mathbb{R}$ be a curve such that: $Q(0) = X(0)$, $Q(1) = X(1)$, for all $t \in [0,1] : Q(t) \in \overline{ab}$ and $\mathrm{d_F}(Q, X) \leq \delta$. For any curve $P: [0,1] \mapsto \mathbb{R}$ with $\mathrm{d_F}(P, X) \leq \delta$, it holds that $\mathrm{d_F}(P, Q) \leq \delta$.*

Proof. To show the lemma statement, we want to apply Lemma 4.34 to P and Q. For this, we need to show that the conditions on Q and P from the lemma statement are met. By Observation 4.33 applied to Q and the line segment X, it follows that Q must be 2δ-monotone with respect to X, and by our assumptions, Q is range-preserving (condition (iv)). By Observation 4.33 applied to P and X, it also follows that P is 2δ-monotone, and conditions (ii), (iii) and (i) are satisfied. Therefore, Lemma 4.34 can be applied to P and Q and the claim is implied. $\square$

Lemma 4.16. *Let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be two curves and let Q' be any δ-straightening of Q. If $\mathrm{d_F}(P, Q') \leq \delta$ then $\mathrm{d_F}(P, Q) \leq \delta$.*

Proof. Let $q_1, \ldots, q_\ell$ be the parameters corresponding to the vertices of Q' in Q, i.e., the vertices of Q' are $Q(q_1), \ldots, Q(q_\ell)$. Let $\phi : [0,1] \to [0,1]^2$ be a δ-traversal between P and Q'. Let $0 = t_1 \leq \cdots \leq t_\ell = 1$ be a partition of the parameter space of P such that for any $1 \leq i \leq \ell - 1$, the edge $\overline{Q(q_i)Q(q_{i+1})}$ is mapped to $P[t_i, t_{i+1}]$ under ϕ. As such, we have

$$\mathrm{d_F}(P[t_i, t_{i+1}], \overline{Q(q_i)Q(q_{i+1})}) \leq \delta$$

By the locality property of δ-simplifications, we also have that

$$\mathrm{d_F}(Q[q_i, q_{i+1}], \overline{Q(q_i)Q(q_{i+1})}) \leq \delta$$

Now, Lemma 4.35 implies that

$$\mathrm{d_F}(P[t_i, t_{i+1}], Q[q_i, q_{i+1}]) \leq \delta.$$

Finally, we apply Observation 4.2 on $P = \bigcirc_{i=1}^{\ell} P[t_i, t_{i+1}]$ and $Q = \bigcirc_{i=1}^{\ell} Q[q_i, q_{i+1}]$, and we obtain

$$\mathrm{d_F}(P, Q) \leq \max_{i \in [\ell]} \mathrm{d_F}\left(P[t_i, t_{i+1}], Q[q_i, q_{i+1}]\right) \leq \delta.$$

□

4.6.2 Proofs of Lemmas for Signatures

Next, we want to prove Lemma 4.17 from Section 4.2. We first prove an auxiliary statement for signature edges in Lemma 4.36. In particular, we need to take care of the first and last edge of the signature. For the other edges we can use Lemma 4.35. Technically, we will also need the symmetric statement of this lemma for $a > b$; this follows by mirroring at the origin. The proof of this lemma turns out be technically involved. For the proof of Lemma 4.17 we can then use the same approach as for Lemma 4.16 above.

Lemma 4.36. *Let $\delta = \delta' + \delta''$ for $\delta, \delta', \delta'' \geq 0$. Let $X = \overline{ab} \subset \mathbb{R}$ be a line segment with $a \leq b$ and let $Q : [0,1] \mapsto \mathbb{R}$ be a curve such that: $Q(0) = X(0)$, $Q(1) = X(1)$ and $\mathrm{d_F}(Q, X) \leq \delta'$. Let $P : [0,1] \mapsto \mathbb{R}$ be a curve with $\mathrm{d_F}(P, X) \leq \delta$.*

If either

(i) $|Q(0) - P(0)| \leq \delta''$ and $|Q(1) - P(1)| \leq \delta''$, or

(ii) $|Q(0) - P(0)| \leq \delta''$ and $\max_{t \in [0,1]}(Q(t)) \leq Q(1)$, or

(iii) $\min_{t \in [0,1]}(Q(t)) \geq Q(0)$ and $|Q(1) - P(1)| \leq \delta''$,

then it holds that $\mathrm{d_F}(P, Q) \leq \delta$.

Proof. Let $t_{\min} = \arg\min\{Q(t)\}$ and $t_{\max} = \arg\max\{Q(t)\}$. In case the minimum (resp. maximum) is not unique, we choose any of them. By Observation 4.33, we have that $\forall t \in [0,1]$ $Q(t) \in [Q(0) - \delta', Q(1) + \delta']$ and by assumption of case (i) $|P(0) - Q(0)| \leq \delta''$ and $|P(1) - Q(1)| \leq \delta''$. Therefore, by triangle inequality, we have in case (i), that

$$|P(0) - Q(t_{\min})| \leq \delta \quad \text{and} \quad |P(1) - Q(t_{\max})| \leq \delta$$

It is easy to see that this holds in the cases (ii) and (iii), as well, since $|P(0) - Q(0)| \leq \delta$ and $|P(1) - Q(1)| \leq \delta$ holds in any case as we assume $\mathrm{d_F}(P, X) \leq \delta$.

Now, define

$$t_1 = \min\{t \in [0,1] \mid P(t) \geq Q(t_{\min}) + \delta\}$$

$$t_2 = \max\{t \in [0,1] \mid P(t) \leq Q(t_{\max}) - \delta\}$$

If such a t_1 does not exist, then we set $t_1 = 1$. If t_2 does not exist, then we set $t_2 = 0$.

Note that by construction and Observation 4.33 we have

$$\mathrm{d_F}(P[0,t_1], Q(0)) \leq \delta \quad \text{and} \quad \mathrm{d_F}(P[t_2,1], Q(1)) \leq \delta \tag{4.3}$$

Indeed, (4.3) holds true since $Q(t_{\min}) \leq Q(0) \leq Q(t_{\min}) + \delta'$ and, likewise, $Q(t_{\max}) \geq Q(1) \geq Q(t_{\max}) - \delta'$, and, moreover, the image of the subcurve $P[0,t_1]$ is contained in the interval $[Q(0) - \delta, Q(t_{\min}) + \delta]$ and the image of the subcurve $P[t_2,1]$ is contained in the interval $[Q(t_{\max}) - \delta, Q(1) + \delta]$.

In addition, we have

$$\mathrm{d_F}(P(t_1), Q[0,t_{\min}]) \leq \delta \quad \text{and} \quad \mathrm{d_F}(P(t_2), Q[t_{\max},1]) \leq \delta \tag{4.4}$$

Indeed, (4.4) holds true, since $\delta' \leq \delta$ and by Observation 4.33, Q is $2\delta'$-monotone increasing, and therefore the image of the subcurve $Q[0,t_{\min}]$ is contained in the interval $[Q(t_{\min}), Q(t_{\min}) + 2\delta']$ which by construction is equal to $[P(t_1) - \delta', P(t_1) + \delta']$ and the image of the subcurve $Q[t_{\max},1]$ is contained in the interval $[Q(t_{\max}) - 2\delta', Q(t_{\max})]$, which by construction is equal to $[P(t_2) - \delta', P(t_2) + \delta']$.

Now, assume that $t_1 \leq t_2$ and $t_{\min} \leq t_{\max}$. In this case, the subcurves $P[t_1,t_2]$ and $Q[t_{\min}, t_{\max}]$ are well-defined. By construction, $|P(t_1) - Q(t_{\min})| \leq \delta$, $|P(t_2) - Q(t_{\max})| \leq \delta$ and $Q[t_{\min}, t_{\max}] \subseteq \overline{Q(t_{\min})Q(t_{\max})}$. By Observation 4.33, P and Q are both 2δ-monotone with respect to X, and, by definition, $X = \overline{Q(0)Q(1)}$. Moreover, by the definition of t_1, t_2, we have $P[t_1,t_2] \subseteq B(Q[t_{\min}, t_{\max}], \delta)$. Therefore all conditions of Lemma 4.34 are satisfied, which implies that

$$\mathrm{d_F}(P[t_1,t_2], Q[t_{\min}, t_{\max}]) \leq \delta \tag{4.5}$$

In summary, we have by (4.3),(4.4), and (4.5) that

$$\max \begin{pmatrix} \mathrm{d_F}(P[0,t_1], Q(0)) \\ \mathrm{d_F}(P(t_1), Q[0,t_{\min}]) \\ \mathrm{d_F}(P[t_1,t_2], Q[t_{\min}, t_{\max}]) \\ \mathrm{d_F}(P(t_2), Q[t_{\max},1]) \\ \mathrm{d_F}(P[t_2,1], Q(1)) \end{pmatrix} \leq \delta$$

Now, by Observation 4.2 we can concatenate these subcurves and $\mathrm{d_F}(P,Q) \leq \delta$ is implied.

If the assumption $t_1 \leq t_2$ fails, then, in fact, a simpler decomposition works. Indeed, if $t_1 > t_2$, then it holds by (4.3) and (4.4) that

$$\max \begin{pmatrix} \mathrm{d_F}(P[0,t_1], Q(0)) \\ \mathrm{d_F}(P(t_1), Q) \\ \mathrm{d_F}(P[t_1,1], Q(1)) \end{pmatrix} \leq \delta$$

Therefore, also in this case, $\mathrm{d_F}(P,Q) \leq \delta$ holds true.

Finally, we need to consider the case that the assumption $t_{\min} \leq t_{\max}$ fails. We may assume that $t_1 \leq t_2$, as we covered the case $t_1 > t_2$ above. We will consider the different cases from the lemma statement separately. First, note that if $t_{\min} > t_{\max}$, then $|Q(t_{\max}) - Q(t_{\min})| \leq 2\delta$, since Q is 2δ-monotone, and therefore, Q is contained in the interval $[P(t_1) - \delta, P(t_1) + \delta]$. By a similar argument, Q is contained in the interval $[P(t_2) - \delta, P(t_2) + \delta]$.

Now, assume case (ii) from the lemma statement. In this case, we have by the above and by Lemma 4.34

$$\max \begin{pmatrix} \mathrm{d_F}(P[0,t_1], Q(0)) \\ \mathrm{d_F}(P(t_1), Q[0,t_{\min}]) \\ \mathrm{d_F}(P[t_1,1], Q[t_{\min},1])) \end{pmatrix} \leq \delta$$

Assume case (iii) from the lemma statement. In this case, we have symmetrically

$$\max \begin{pmatrix} \mathrm{d_F}(P[0,t_2], Q[0,t_{\max}]) \\ \mathrm{d_F}(P(t_2), Q[t_{\max},1]) \\ \mathrm{d_F}(P[t_2,1], Q(1)) \end{pmatrix} \leq \delta$$

Now, for case (i), we claim that there exist $0 \leq q_1 \leq q_2 \leq 1$, such that

$$\max \begin{pmatrix} \mathrm{d_F}(P[0,t_1], Q(0)) \\ \mathrm{d_F}(P(t_1), Q[0,q_1]) \\ \mathrm{d_F}(P[t_1,t_2], Q[q_1,q_2]) \\ \mathrm{d_F}(P(t_2), Q[q_2,1]) \\ \mathrm{d_F}(P[t_2,1], Q(1)) \end{pmatrix} \leq \delta$$

Indeed, from what we derived, $\mathrm{d_F}(P(t_1), Q[0,q_1]) \leq \delta$ and $\mathrm{d_F}(P(t_2), Q[q_2,1]) \leq \delta$ holds for any choice of $q_1, q_2 \in [0,1]$. The first and last line hold by (4.3). It remains to show that we can choose q_1, q_2 so that $\mathrm{d_F}(P[t_1,t_2], Q[q_1,q_2]) \leq \delta$ holds. Since $\mathrm{d_F}(P,X) \leq \delta$, there must be a subsegment $X[x_1,x_2]$ of X, such that $\mathrm{d_F}(P[t_1,t_2], X[x_1,x_2]) \leq \delta$. Recall that $\overline{Q(0)Q(1)} = X$ and by the intermediate value theorem we can define suitable q_1, q_2 as follows

$$q_1 = \max\{q \in [0,1] \mid Q(q) = X(x_1)\}$$

$$q_2 = \min\{q \in [q_1,1] \mid Q(q) = X(x_2)\}$$

Now, we can apply Lemma 4.34 and conclude that $\mathrm{d_F}(P[t_1,t_2], Q[q_1,q_2]) \leq \delta$. Therefore, also in case (i), we have $\mathrm{d_F}(P,Q) \leq \delta$. □

Now we are ready to prove Lemma 4.17.

Lemma 4.17. *Let $\delta = \delta' + \delta''$ for $\delta, \delta', \delta'' \geq 0$ and let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be two curves. Let Q' be any δ'-signature of Q. If $\mathrm{d_F}(Q', P) \leq \delta$, $|Q(0) - P(0)| \leq \delta''$, and $|Q(1) - P(1)| \leq \delta''$, then $\mathrm{d_F}(P,Q) \leq \delta$.*

Proof. This follows by a modification of the proof of Lemma 4.16. Although the two proofs are very similar, the differences are subtle. Therefore, we give the full proof for the sake of completeness. Let $q_1, \ldots, q_\ell$ be the parameters corresponding to the vertices of Q' in Q, i.e., the vertices of Q' are $Q(q_1), \ldots, Q(q_\ell)$. Let $\phi : [0,1] \to [0,1]^2$ be a δ-traversal between P and Q'. Let $0 = t_1 \leq \cdots \leq t_\ell = 1$ be a partition of the parameter space of P

such that for any $1 \leq i \leq \ell - 1$, the edge $\overline{Q(q_i)Q(q_{i+1})}$ is mapped to $P[t_i, t_{i+1}]$ under ϕ. As such, we have

$$d_F(P[t_i, t_{i+1}], \overline{Q(q_i)Q(q_{i+1})}) \leq \delta \tag{4.6}$$

By the definition of δ-simplifications, we also have that

$$d_F(Q[q_i, q_{i+1}], \overline{Q(q_i)Q(q_{i+1})}) \leq \delta' \leq \delta \tag{4.7}$$

Now, if the edge $\overline{Q(q_i)Q(q_{i+1})}$ of Q' is range-preserving, then Lemma 4.35 implies that

$$d_F(P[t_i, t_{i+1}], Q[q_i, q_{i+1}]) \leq \delta. \tag{4.8}$$

Otherwise, it must be (by the definition of signatures) that either $i = 1$ or $i + 1 = \ell$ or both (the edge is the first or last edge of the signature Q' or Q' consists of just one edge). In any of those cases, Lemma 4.36 implies $d_F(P[t_i, t_{i+1}], Q[q_i, q_{i+1}]) \leq \delta$.

Finally, we apply Observation 4.2 on $P = \bigcirc_{i=1}^{\ell} P[t_i, t_{i+1}]$ and $Q = \bigcirc_{i=1}^{\ell} Q[q_i, q_{i+1}]$, and we obtain

$$d_F(P, Q) \leq \max_{i \in [\ell]} d_F\left(P[t_i, t_{i+1}], Q[q_i, q_{i+1}]\right) \leq \delta.$$

□

4.6.3 Proofs of Lemmas for Visiting Orders

In order to prove the existence of δ'-visiting orders for some $\delta' \in O(\delta)$ as claimed in Lemma 4.19, we introduce the concept of a visiting sequence. A visiting sequence is not necessarily monotonically increasing, while visiting orders according to Definition 4.14 are. Nonetheless, this definition of visiting sequence will turn out to be useful. It is important that a δ-visiting sequence is derived from a monotone traversal. We will show (Lemma 4.39 and 4.40) that any non-monotonic visiting sequence can be turned into a monotonic one at the expense of a constant factor in the radius of the visiting relationship.

Definition 4.37. *Let $P : [0,1] \to \mathbb{R}$ and $Q : [0,1] \to \mathbb{R}$ be curves, let $\delta > 0$, and let $\phi : [0,1] \to [0,1]^2$ be a monotone traversal. We say a vertex w of Q δ-**visits** a vertex v of P **under** ϕ if the following holds:*
(i) $|w - v| \leq \delta$ and
(ii) at least one of the following holds:
(a) ϕ associates w with v, or
(b) ϕ associates w with the interior of an edge of P that is incident to v, or
(c) ϕ associates v with the interior of an edge of Q that is incident to w.
Note that the induced relation on the vertices is symmetric for any fixed δ and ϕ.

Definition 4.38. *Let $P : [0,1] \to \mathbb{R}$ and $Q : [0,1] \to \mathbb{R}$ be curves and let $\phi : [0,1] \to [0,1]^2$ be a monotone traversal. Let S be a subsequence of the vertices of Q of length ℓ. Let $u_1, \ldots, u_\ell$ denote the ordered vertices of S and let $v_1, \ldots, v_m$ denote the ordered vertices of P. A δ-**visiting sequence** of S on P **under** ϕ is a sequence of indices $i_1, \ldots, i_\ell$, such that each u_j of S δ-visits the vertex v_{i_j} of P under ϕ.*

Lemma 4.39. *Let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be curves such that $d_F(Q, P) \leq \delta$ and let ϕ be a monotone traversal realizing this distance. Let v_i, v_j be two vertices of Q*

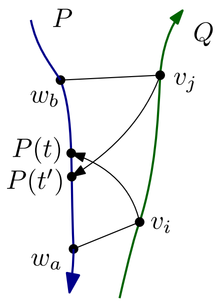


Figure 4.3: Illustration to the proof of Lemma 4.39. Assuming $w_a < w_b$ as in the proof, v_i visits w_a and v_j visits w_b, but $i < j$ and $a > b$, so the visiting sequence is not monotone.

with $i < j$ in the ordering along Q. Assume v_i δ-visits a vertex w_a of P under ϕ and v_j δ-visits a vertex w_b of P under ϕ such that $a > b$ in the ordering along P. Then, it must be that v_i 3δ-visits w_b under ϕ and that v_j 3δ-visits w_a under ϕ.

Proof. As $a > b$, however, in ϕ a point on an adjacent edge of w_a is matched earlier than a point on an adjacent edge of w_b, we conclude due to the monotonicity of ϕ that $\overline{w_b w_a}$ is an edge in P. Let $P(t)$ and $P(t')$ be the points that v_i and v_j are mapped to on $\overline{w_b w_a}$ under ϕ, respectively. By the monotonicity of ϕ we have $t \leq t'$. See Figure 4.3 for an illustration.

Assume that $w_a < w_b$, as the case $w_a > w_b$ is symmetric. Since $P(t)$ and $P(t')$ are both on the edge $\overline{w_b w_a}$, the fact that $t \leq t'$ implies that $P(t') \leq P(t)$. Using the facts that $|v_i - w_a| \leq \delta$ and $|v_j - w_b| \leq \delta$, we obtain

$$v_i - \delta \leq w_a \leq P(t') \leq P(t) \leq w_b \leq v_j + \delta.$$

At the same time we have

$$v_j - \delta \leq P(t') \leq P(t) \leq v_i + \delta.$$

It follows that $|v_i - v_j| \leq 2\delta$.

Thus, the claim that v_i is contained in the 3δ-range of w_b is then implied by triangle inequality, as well as the symmetric claim that v_j is contained in the 3δ-range of w_a. As v_i and v_j are both matched to the edge $\overline{w_b w_a}$, we also have that v_i and v_j visit the 3δ-ranges of w_b and w_a, respectively. □

Lemma 4.40. *Let $P : [0, 1] \to \mathbb{R}$ and $Q : [0, 1] \to \mathbb{R}$ be curves and let $\phi : [0, 1] \to [0, 1]^2$ be a monotone traversal that maps them within distance δ. Let S be a subsequence of the vertices of Q. Any δ-visiting sequence of S on P under ϕ implies a 3δ-visiting order of S on P.*

Proof. Let $u_1, \ldots, u_\ell$ denote the vertices of S and let $i_1, \ldots, i_\ell$ denote the visiting sequence. We generate a monotonically increasing sequence as follows. For every u_j, we set i_j to the minimum of the suffix sequence $i_j, \ldots, i_\ell$. If i_j was already a minimum, then nothing changes. Otherwise, let i_k be an index, where this minimum was attained. By Lemma 4.39 the vertex u_j is contained in the 3δ-range of the vertex v_{i_k}. After applying this to all elements of the sequence, starting with $j = 1$ and ending with $j = \ell$, the sequence $i_1, \ldots, i_\ell$ is monotonically increasing. □

The next two lemmas are used in the proof of Lemma 4.19.

Lemma 4.41. *Let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be curves such that $\mathrm{d_F}(Q,P) \leq \delta$ and let ϕ be a monotone traversal realizing this distance. If none of the inner vertices of P and Q δ-visit each other under ϕ, then P and Q are 2δ-monotone.*

Proof. We prove the lemma by induction. We reconstruct the matching ϕ and use "matched" as shorthand for "matched under ϕ". Recall that we denote the ordered vertices of P and Q by $p_1, p_2, \ldots$ and $q_1, q_2, \ldots$, respectively. Note that if either P or Q consist of a single vertex or single segment, then the claim immediately follows from Observation 4.33. Otherwise, either p_2 is matched to a point on $\overline{q_1q_2}$ or q_2 is matched to a point on $\overline{p_1p_2}$ and p_2, q_2 are inner vertices. As the lemma statement is symmetric with respect to P and Q, we assume without loss of generality that p_2 is matched to $\overline{q_1q_2}$. As p_2 and q_2 are inner vertices, they cannot δ-visit each other, and thus either $p_2 < q_2 - \delta$ or $p_2 > q_2 + \delta$. By mirroring the curves P and Q at the origin, these two cases are symmetric, and we thus assume $p_2 < q_2 - \delta$ without loss of generality. As p_2 is matched to $\overline{q_1q_2}$, it follows that $q_1 < q_2$. Thus, $\overline{q_1q_2}$ is increasing and $\overline{p_1p_2}$ has to be 2δ-monotone increasing as otherwise the matching would have distance larger than δ. Now, for the inductive step, assume that $\langle p_1, \ldots, p_i \rangle$ and $\langle q_1, \ldots, q_j \rangle$ are 2δ-monotone increasing curves, p_i, q_j are inner vertices, and p_i is matched to a point on $\overline{q_{j-1}q_j}$ with $p_i < q_j - \delta$. Note that this again implies $q_{j-1} < q_j$.

Let us now prove the inductive step. If p_{i+1} is an inner vertex, then either (i) p_{i+1} is also matched to a point on $\overline{q_{j-1}q_j}$ or (ii) q_j is matched to a point on $\overline{p_ip_{i+1}}$.

In case (i), p_{i+1} extends a subcurve $\langle p_{i'}, \ldots, p_i \rangle$ with $i' \geq 1$ that is completely matched to a part of the increasing segment $\overline{q_{j-1}q_j}$. The subcurve $\langle p_{i'}, \ldots, p_i \rangle$ has to be 2δ-monotone increasing according to Observation 4.33. Either $p_{i'}$ is the start of P (i.e, $i' = 1$) and thus $\langle p_1, \ldots, p_{i+1} \rangle$ is 2δ-monotone increasing, or $p_{i'-1}$ has to be matched to a part of Q before q_{j-1} and thus q_{j-1} is an inner vertex. As q_{j-1} was already matched, it follows that either $p_{i'-1}$ is the start of P (i.e., $i' - 1 = 1$) and $p_{i'-1} \leq q_{j-1} + \delta$, or $p_{i'-1}$ is an inner vertex and $p_{i'-1} < q_{j-1} - \delta$ as they do not δ-visit each other. In both cases $\langle p_1, \ldots, p_{i'-1} \rangle$ is contained in $[-\infty, q_{j-1} + \delta)$; for the first case this holds as $\langle p_1, \ldots, p_{i'-1} \rangle$ is 2δ-monotone increasing by induction. Consequently, the concatenation of $\langle p_1, \ldots, p_{i'-1} \rangle$ and $\langle p_{i'}, \ldots, p_{i+1} \rangle$ is also 2δ-monotone increasing.

Now consider case (ii), i.e., q_j is matched to a point on $\overline{p_ip_{i+1}}$. In this case $\overline{p_ip_{i+1}}$ is increasing as $p_i < q_j$ and $q_j < p_{i+1}$, which is the case because q_j is matched to $\overline{p_ip_{i+1}}$ and $p_i < q_j - \delta$. Therefore, also in this case it holds that $\langle p_1, \ldots, p_{i+1} \rangle$ is 2δ-monotone increasing. Note that after exchanging P and Q, we again fulfill the inductive hypothesis. In particular, since q_j is matched to $\overline{p_ip_{i+1}}$ but p_{i+1} and q_j do not δ-visit each other as both are inner vertices, we must have $q_j < p_{i+1} - \delta$.

Now consider the case that p_{i+1} is not an inner vertex, i.e., it is the last vertex of P. In this case, part of $\overline{p_ip_{i+1}}$ has to be matched to q_j as no previous part of P was matched to q_j. This implies that $\overline{p_ip_{i+1}}$ again is increasing as $p_i < q_j - \delta$ and $p_{i+1} \geq q_j - \delta$. Hence $\langle p_1 \ldots p_{i+1} \rangle$ is 2δ-monotone increasing. As the remainder of Q, starting from q_j, has to be matched to part of $\overline{p_ip_{i+1}}$ and therefore this part is 2δ-monotone increasing by Observation 4.33, and $\langle q_1, \ldots, q_{j-1} \rangle$ is 2δ-monotone by induction and also contained in $[-\infty, p_i - \delta)$ as p_i is matched to the increasing $\overline{q_{j-1}q_j}$, it follows that the whole curve Q is 2δ-monotone increasing. □

Lemma 4.42. *Let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be curves such that $\mathrm{d_F}(Q,P) \leq \delta$ and let ϕ be a monotone traversal realizing this distance. Further assume that for all $t \in [0,1]$ we have $Q(t) \in \overline{Q(0)Q(1)}$. If none of the inner vertices of Q δ-visit an inner vertex of P under ϕ, then the line segment $Q' = \overline{Q(0)Q(1)}$ is a range-preserving δ-simplification of Q with $\mathrm{d_F}(Q',P) \leq \delta$.*

Proof. By Lemma 4.41, Q and P must be 2δ-monotone. Moreover, Q' is range-preserving by assumption. Therefore, Q' is a range-preserving δ-simplification of Q. It remains to show the bound on the Fréchet distance of P and Q'. To this end, we want to invoke Observation 4.33. Indeed, it must be that

$$\forall t \in [0,1] : P(t) \in \bigcup_{s\in[0,1]} B(Q(s),\delta),$$

since $\mathrm{d_F}(P,Q) \leq \delta$ and since Q' is range-preserving. Therefore, the conditions of Observation 4.33 are satisfied and the bound is implied. □

We are now ready to prove Lemma 4.19 from Section 4.2.

Lemma 4.19. *Let $P : [0,1] \mapsto \mathbb{R}$ and $Q : [0,1] \mapsto \mathbb{R}$ be curves such that $\mathrm{d_F}(Q,P) \leq \delta$, there exists a δ-straightening Q' of Q which satisfies the following properties:*
(i) there exists a (11δ)-visiting order of Q' on P, and
(ii) $\mathrm{d_F}(Q',P) \leq \delta$.

Proof. Let ϕ be a monotone traversal that realizes the Fréchet distance between P and Q. We will construct a δ-straightening Q' together with a $\mathcal{O}(\delta)$-visiting order of Q' on P. To this end, consider the subset of vertices of Q that each δ-visit *some* vertex of P under ϕ (Definition 4.37). Denote this subset by S. Lemma 4.40 implies that there exists a 3δ-visiting order of S on P. We denote this visiting order by the function $\kappa : S \to [m]$ that assigns every vertex of S the index of a vertex of P (where m denotes the number of vertices of P).

It is quite possible that S is not a δ-simplification of Q with the desired properties. In a second phase of the construction we will therefore add more vertices of Q to S. Consider any maximal subcurve $Q[s,s']$ of Q, such that none of the inner vertices of $Q[s,s']$ δ-visit a vertex of P under ϕ. It must be that $Q(s)$ corresponds to some vertex w of S and $Q(s')$ corresponds to some vertex w' of S. Moreover, w' comes directly after w along Q among the vertices included in S. Assume that $Q[s,s']$ has at least one inner vertex. We distinguish two cases:

(C1) $B(v_{\kappa(w)},3\delta) \cap B(v_{\kappa(w')},3\delta) \neq \emptyset$,

(C2) otherwise

In the first case (C1), we will add all inner vertices $Q[s,s']$ to S and assign them the index $\kappa(w)$ in the constructed visiting order κ. In the second case (C2), we will only add a specific subset of vertices, which we define as follows. Define α and β as follows:

$$\alpha = \max\{t \mid t \in [s,s'] \text{ and } Q(t) \in B(v_{\kappa(w)},3\delta)\}$$

$$\beta = \min\{t \mid t \in [\alpha,s'] \text{ and } Q(t) \in B(v_{\kappa(w')},3\delta)\}$$

Since the 3δ-ranges of $v_{\kappa(w)}$ and $v_{\kappa(w')}$ are disjoint, α and β are well-defined and it follows by definition that $s \leq \alpha \leq \beta \leq s'$. Therefore, the subcurves $Q[s, \alpha]$, $Q[\alpha, \beta]$, and $Q[\beta, s']$ are well-defined. Now, we proceed as follows, we add the inner vertices of $Q[s, \alpha]$ to S and assign them the index $\kappa(w)$ in the constructed visiting order κ. Secondly, we add the inner vertices of $Q[\beta, s]$ to S and assign them the index $\kappa(w')$ in the constructed visiting order κ.

We apply this to all such maximal subcurves $Q[s, s']$ (note that these are pairwise disjoint), thereby constructing the sequence S along with the visiting order κ. Let $u_1, \ldots, u_\ell$ be the sequence of vertices of the resulting S in their order along Q. Denote with Q' the curve that results from linearly interpolating $u_1, \ldots, u_\ell$. Note that it is different from Q only in the sections where we omitted the vertices of the subcurve $Q[\alpha, \beta]$ in case (C2). We claim that Q' is an edge-range-preserving δ-simplification of Q. To see this, consider a subcurve $Q[s, s']$, assume we are in case (C2). By construction, the subcurve $Q[\alpha, \beta]$ is range-preserving (for all $x \in [\alpha, \beta]$ we have $Q(x) \in \overline{Q(\alpha)Q(\beta)}$). Let $P[t, t']$ be a subcurve of P mapped to $Q[\alpha, \beta]$ under ϕ. Now, Lemma 4.42 applied to the subcurves $P[t, t']$ and $Q[\alpha, \beta]$ implies that $\overline{Q(\alpha)Q(\beta)}$ is an edge-range-preserving δ-simplification of $Q[\alpha, \beta]$ with $\mathrm{d_F}(\overline{Q(\alpha)Q(\beta)}, P[t, t']) \leq \delta$. Therefore, by Observation 4.2, when removing all vertices of Q in the parameter range (α, β) for each such maximal subcurve $Q[s, s']$, we obtain a δ-straightening Q' of Q with $\mathrm{d_F}(Q', P) \leq \delta$.

Finally, we argue that the constructed visiting order $\kappa(u_1), \ldots, \kappa(u_\ell)$ is an 11δ-visiting order of Q' on P. Clearly it is monotonically increasing by construction. Also, it is clear that any vertex added in the first phase is contained in the 3δ-range of its assigned vertex of P. It remains to argue for any vertex added to S in the second phase, that it is contained in the 11δ-range of its assigned vertex in P. Consider a subcurve $Q[s, s']$ from above and assume we are in case (C1). We have that $Q(s) \in B(v_{\kappa(w)}, 3\delta)$ and $Q(s') \in B(v_{\kappa(w')}, 3\delta)$. By the case distinction, these two ranges are not disjoint. Therefore, the subcurve starts and ends in the 9δ-range of the assigned vertex $v_{\kappa(w)}$. Moreover, by Lemma 4.41, $Q[s, s']$ has to be 2δ-monotone. This implies that the entire subcurve lies in the 11δ-range of $v_{\kappa(w)}$ and this is also the vertex that we assigned to all of its inner vertices. A similar argument can be applied in case (C2). By the way we chose α, we have that $Q(\alpha)$ is contained in the 3δ-range of $v_{\kappa(w)}$, which is also the vertex assigned to the entire subcurve. Since also the subcurve $Q[s, \alpha]$ is 2δ-monotone, all remaining vertices in the range $[s, \alpha]$ are contained in the 5δ-range of the same vertex. A symmetric argument can be applied to show that all remaining vertices in the range $[\beta, s']$ are contained in the 5δ-range of their assigned vertex. □

Finally, we also prove Lemma 4.20 from Section 4.2.

Lemma 4.20. *For one-dimensional curves P, Q let P' be a δ-signature of P, and let Q' be a 2δ-signature of Q. If $\mathrm{d_F}(P, Q) \leq \delta$ then $\mathrm{d_F}(P', Q') \leq 3\delta$ and there exists a 2δ-visiting order of Q' on P'.*

Proof. By the triangle inequality we have that $\mathrm{d_F}(P', Q) \leq \mathrm{d_F}(P', P) + \mathrm{d_F}(P, Q) \leq 2\delta$. Now Lemma 4.15 applied to P' and the 2δ-signature of Q implies that there exists a 2δ-visiting order of Q' on P'.

It remains to argue that $\mathrm{d_F}(P', Q') \leq 3\delta$. Let $\phi : [0, 1] \to [0, 1]^2$ be a δ-traversal of P and Q. Consider an edge X of Q' and let $Q[\alpha, \beta]$ be the subcurve of Q that corresponds to X. Let $P[\alpha', \beta']$ be a subcurve of P that is mapped to $Q[\alpha, \beta]$ under ϕ. By the triangle inequality

$$d_F(P[\alpha', \beta'], X) \leq d_F(P[\alpha', \beta'], Q[\alpha, \beta]) + d_F(Q[\alpha, \beta]), X) \leq 3\delta$$

Assume that P' is range-preserving for now (we will treat the general case below) and let $P'[\alpha'', \beta'']$ be the corresponding subcurve of P' starting at $P(\alpha')$, ending at $P(\beta')$, and with inner vertices being the δ-signature vertices of P in the parametrization interval $[\alpha', \beta']$. Note that $P'[\alpha'', \beta'']$ is well-defined since P' is a range-preserving as assumed above. By Observation 4.3 it follows that $d_F(P'[\alpha'', \beta''], X) \leq 3\delta$. To show the claim for the case of range-preserving P', we now want to use Observation 4.2 to concatenate the corresponding subcurves of P' and Q' and obtain that $d_F(P', Q') \leq 3\delta$. For this, we can choose the values of α' and β' in the above argument such that we obtain a decomposition of P into subcurves. Concretely, let $X_1, \ldots, X_s$ be the edges of Q' in their order along Q', with $X_i = \overline{Q(\alpha_i)Q(\beta_i)}$. Then, we can choose the corresponding subcurves of P as $P[\alpha'_i, \beta'_i]$, with

$$\alpha'_{i-1} \leq \beta'_{i-1} = \alpha'_i \leq \beta'_i$$

for any $1 < i \leq s$, with $\alpha'_1 = 0$ and $\beta'_s = 1$. Thus, we obtain a decomposition of P. Now, if P' is a range-preserving simplification of P, then the above construction induces a decomposition of P' into subcurves $P'[\alpha''_i, \beta''_i]$ and we can apply Observation 4.2.

As noted above, P' is not necessarily range-preserving on all edges since it is a signature. In particular, it may not be range-preserving on the first edge (or the last edge, or neither). This could lead to $P(\alpha'_2)$ (resp. $P(\alpha'_s)$ for the last edge) not being included in the image of the signature edge of P' that corresponds to the subcurve of P containing α'_2 (resp., α'_s). Note that if $P(\alpha'_2)$ is not contained in the image of the first signature edge, then it must be that $|P(\alpha'_2) - P(0)| \leq \delta$, and in fact, it must be that this holds for the entire subcurve, that is $|P(t) - P(0)| \leq \delta$ for any $t \in [0, \alpha'_2]$. We claim that in this case we can simply set α''_2, and β''_1 to 0 (resp., we can set β''_{s-1}, and α''_s to 1). We argue that this way of choosing the decomposition leads to $d_F(P'[\alpha''_1, \beta''_1], X_1) \leq 3\delta$ and $d_F(P'[\alpha''_2, \beta''_2], X_2) \leq 3\delta$ so that the above arguments can be applied (for the last two edges of Q' a symmetric argument can be applied and we will omit the explicit analysis).

By the triangle inequality, we have that

$$|P'(0) - Q(\alpha_2)| \leq |P(0) - P(\alpha'_2)| + |P(\alpha'_2) - Q(\alpha_2)| \leq 2\delta.$$

Together with

$$|P'(0) - Q(\alpha_1)| = |P(0) - Q(0)| \leq \delta$$

this implies by Observation 4.1 that $d_F(P'(0), X_1) \leq 3\delta$ since X_1 is a line segment and $X_1 = \overline{Q(0)Q(\alpha_2)}$. Applying the triangle inequality again, we obtain for any $t \in [0, \alpha'_2]$ that

$$|P(t) - Q(\alpha_2)| \leq |P(t) - P(0)| + |P(0) - Q(\alpha_2)| \leq 3\delta$$

By Observation 4.2 and since $X_2 = \overline{Q(\alpha_2)Q(\beta_2)}$, this implies that

$$d_F(P[0, \beta'_2], X_2) \leq \max\Big(d_F(P[0, \alpha'_2], Q(\alpha_2)) \ , \ d_F(P[\alpha'_2, \beta'_2], \overline{Q(\alpha_2)Q(\beta_2)}) \Big) \leq 3\delta$$

By Observation 4.3 it follows that $d_F(P'[0, \beta''_2], X_2) \leq 3\delta$. □

CHAPTER 5

Fréchet Distance Under Translation

For a technical overview of this chapter see Section 3.2. The structure of this chapter is as follows. We start off with introducing basic definitions, notational conventions, and algorithmic tools in Section 5.1. Afterwards, in Section 5.2, we give an overview of our algorithmic result and we reduce the problem to designing a certain data structure for Offline Dynamic Grid Reachability. This data structure, our main technical contribution, is developed in Section 5.3.

5.1 Preliminaries

For convenience, we use as convention that $\min \emptyset = \infty$ and $\max \emptyset = -\infty$.

5.1.1 Curves, Traversals, Fréchet Distances, and More

Throughout this chapter, we only consider polygonal curves in the Euclidean plane, i.e., $d = 2$. Let us define the concatenation of traversals. Given two traversals $T = (t_1, \dots, t_\ell)$ and $T' = (t'_1, \dots, t'_{\ell'})$ with $t_\ell = t'_1$, we define the concatenated traversal as $T \circ T' := (t_1, \dots, t_\ell = t'_1, t'_2, \dots, t'_{\ell'})$. Note that we obtain a traversal from t_1 to $t'_{\ell'}$. We call any pair $(i, j) \in [n] \times [m]$ a *position.* We obtain a well-known alternative discrete Fréchet distance definition as follows: Fix some distance $\delta \geq 0$. We call a position (i, j) *free* if $\|\pi_i - \sigma_j\| \leq \delta$. We say that a traversal $T = (t_1, \dots, t_\ell)$ of π, σ is a *valid traversal* for δ if $t_1, \dots, t_\ell$ are all free positions. The discrete Fréchet distance of π, σ is then the smallest δ such that there is a valid traversal of π, σ for δ.

Analogously, consider the $n \times n$ matrix M with $M_{i,j} = 1$ if (i, j) is free, and $M_{i,j} = 0$ otherwise. We call any traversal $T = (t_1, \dots, t_\ell)$ a *monotone path from t_1 to t_ℓ*. If all positions (i, j) visited by T satisfy $M_{i,j} = 1$, we call T a *monotone 1-path* from t_1 to t_ℓ in M. As yet another formulation, consider the $n \times n$ grid graph G_M where vertex (i, j) has directed edges to all of $(i, j+1), (i+1, j)$, and $(i+1, j+1)$ (in case they exist). Deactivate (i.e., remove) all non-free vertices (i, j) from G_M. Then a monotone 1-path in M corresponds to a (directed) path in G_M. Hence, $\mathrm{d}_{\mathrm{dF}}(\pi, \sigma) \leq \delta$ is equivalent to the existence of a valid traversal of π, σ for δ, which in turn is equivalent to the existence of a monotone 1-path from $(1, 1)$ to (n, n) in the matrix M, and to vertex (n, n) being reachable from $(1, 1)$ in G_M.

5.1.2 Orthogonal Range Searching Data Structures

We will use a tool from geometric data structures, namely *(dynamic) orthogonal range data structures.* Let S be a set of key-value pairs $s = (k_s, v_s) \in \mathbb{Z}^d \times \mathbb{Z}$; in our applications, we will have $d = 2$ or $d = 3$. An orthogonal range data structure on S enables us to query the maximal value of any pair in S whose key lies in a given orthogonal range. Formally, we say $\mathcal{OR}$ *stores* v_s *under the key* k_s *for* $s \in S$ *for minimization queries*, if $\mathcal{OR}$ supports, for any $\ell_1, u_1, \ell_2, u_2, \ldots, \ell_d, u_d \in \mathbb{Z} \cup \{-\infty, \infty\}$, queries of the form

$$\mathcal{OR}.\min([\ell_1, u_1] \times \cdots \times [\ell_d, u_d]) : \\ \text{return } \min\{v_s \mid s = (k_s, v_s) \in S, k_s \in [\ell_1, u_1] \times \cdots \times [\ell_d, u_d]\}.$$

We will also consider analogous maximization queries.

Classic results [66, 96] show that for any set S of size n and $d = 2$, we can construct such a data structure $\mathcal{OR}$ in time and space $\mathcal{O}(n \log n)$, supporting minimization (or maximization) queries in time $\mathcal{O}(\log n)$.

In Section 5.3.6, we will also use an orthogonal range searching data structure that allows (1) to *report* all values of pairs in S whose keys lie in a given orthogonal range, and (2) to *remove* a key-value pair from S. Formally, we say that $\mathcal{OR}$ *stores* v_s *under the key* k_s *for* $s \in S$ *for decremental range reporting queries*, if $\mathcal{OR}$ supports, for any $\ell_1, u_1, \ell_2, u_2, \ldots, \ell_d, u_d \in \mathbb{Z} \cup \{-\infty, \infty\}$, queries of the form

$$\mathcal{OR}.\text{report}([\ell_1, u_1] \times \cdots \times [\ell_d, u_d]) : \\ \text{return } \{v_s \mid s = (k_s, v_s) \in S, k_s \in [\ell_1, u_1] \times \cdots \times [\ell_d, u_d]\},$$

as well as deletions from the set S.

Mortensen [136] and Chan and Tsakalidis [64] showed how to construct such a data structure $\mathcal{OR}$ for any set S of size n in time and space $\mathcal{O}(n \log^{d-1} n)$, deletion time $\mathcal{O}(\log^{d-1} n)$ and query time $\mathcal{O}(\log^{d-1} n + k)$, where k denotes the output size of the query. (These works obtain even stronger results, however, we use simplified bounds for ease of presentation.)

5.2 Algorithm: Reduction to Grid Reachability

In this section, we prove our algorithmic result by showing how a certain grid reachability data structure (that we give in Section 5.3) yields an $\tilde{\mathcal{O}}(n^{4+2/3})$-time algorithm for computing the discrete Fréchet distance under translation.

We start with a formal overview of the algorithm. First, we reduce the decision problem (i.e., is the discrete Fréchet distance under translation of π, σ at most δ?) to the problem of determining reachability in a dynamic grid graph, as shown by Ben Avraham et al. [33]. However, noting that all updates and queries are known in advance, we observe that the following *offline* version suffices.

Problem 5.1 (Offline Dynamic Grid Reachability). *Let M be an $n \times n$ matrix over $\{0, 1\}$. We call $u = (p, b)$ with $p \in [n] \times [n]$ and $b \in \{0, 1\}$ an* update *and define $M[\![u]\!]$ as the matrix obtained by setting the bit at position p to b, i.e.,*

$$M[\![(p, b)]\!]_{i,j} = \begin{cases} b & \text{if } p = (i, j), \\ M_{i,j} & \text{otherwise.} \end{cases}$$

For any sequence of updates $u_1, \ldots, u_k \in ([n] \times [n]) \times \{0, 1\}$ with $k \geq 2$, we define $M[\![u_1, \ldots, u_k]\!] := (M[\![u_1]\!])[\![u_2, \ldots, u_k]\!]$.

The Offline Dynamic Grid Reachability *problem asks to determine, given M and any sequence of updates $u_1, \dots, u_U \in ([n] \times [n]) \times \{0,1\}$, whether there is a monotone 1-path from $(1,1)$ to (n,n) in $M[\![u_1, \dots, u_k]\!]$ for any $1 \leq k \leq U$.*

We show the following reduction in Section 5.2.1.

Lemma 5.2. *Assume there is an algorithm solving Offline Dynamic Grid Reachability in time $T(n, U)$. Then there is an algorithm that, given $\delta > 0$ and polygonal curves π, σ of length n over $\mathbb{R}^2$, determines whether $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ for some $\tau \in \mathbb{R}^2$ in time $\mathcal{O}(T(n, n^4))$.*

Our speed-up is achieved by solving Offline Dynamic Grid Reachability in time $T(n, U) = \tilde{\mathcal{O}}(n^2 + Un^{2/3})$ (Ben Avraham et al. [33] achieved $T(n, U) = \mathcal{O}(n^2 + Un)$). To this end, we devise a *grid reachability data structure*, which is our central technical contribution.

Lemma 5.3 (Grid reachability data structure)**.** *Given an $n \times n$ matrix M over $\{0,1\}$ and a set of* terminals *$\mathcal{T} \subseteq [n] \times [n]$ of size $k > 0$, there is a data structure $\mathcal{D}_{M,\mathcal{T}}$ with the following properties.*

i) (Construction:) We can construct $\mathcal{D}_{M,\mathcal{T}}$ in time $\mathcal{O}(n^2 + k \log^2 n)$.

ii) (Reachability Query:) Given $F \subseteq \mathcal{T}$, we can determine in time $\mathcal{O}(k \log^3 n)$ whether there is a monotone path from $(1,1)$ to (n,n) using only positions (i,j) with $M_{i,j} = 1$ or $(i,j) \in F$.

iii) (Update:) Given $\mathcal{T}' \subseteq [n] \times [n]$ of size k and an $n \times n$ matrix M' over $\{0,1\}$ differing from M in at most k positions, we can update $\mathcal{D}_{M,\mathcal{T}}$ to $\mathcal{D}_{M',\mathcal{T}'}$ in time $\mathcal{O}(n\sqrt{k} \log n + k \log^2 n)$. Here, we assume M' to be represented by the set Δ of positions in which M and M' differ.

Section 5.3 is dedicated to devising this data structure, i.e., proving Lemma 5.3. Equipped with this data structure, we can efficiently *batch* updates and queries to the data structure. Specifically, we obtain the following theorem.

Theorem 5.4. *Offline Dynamic Grid Reachability on an $n \times n$ grid with U updates can be solved in time $\mathcal{O}(n^2 + Un^{2/3} \log^2 n)$.*

We prove this theorem in Section 5.2.2. Finally, it remains to use standard techniques of parametric search to transform the decision algorithm to an algorithm computing the discrete Fréchet distance under translation. This has already been shown by Ben Avraham et al. [33]; we sketch the details in Section 5.2.3.

Lemma 5.5. *Let $T_{\mathrm{dec}}(n)$ be the running time to decide, given $\delta > 0$ and polygonal curves π, σ of length n over $\mathbb{R}^2$, whether $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ for some $\tau \in \mathbb{R}^2$. Then there is an algorithm computing the discrete Fréchet distance under translation for any curves π, σ of length n over $\mathbb{R}^2$ in time $\mathcal{O}((n^4 + T_{\mathrm{dec}}(n)) \log n)$.*

Combining Lemma 5.5, Lemma 5.2 and Theorem 5.4, we obtain an algorithm computing the discrete Fréchet distance under translation in time

$$\mathcal{O}((n^4 + T(n, n^4)) \log n) = \mathcal{O}(n^{4+2/3} \log^3 n),$$

as desired. In the remainder of this section, we provide the details of all steps mentioned above, except for Lemma 5.3 (which we prove in Section 5.3).

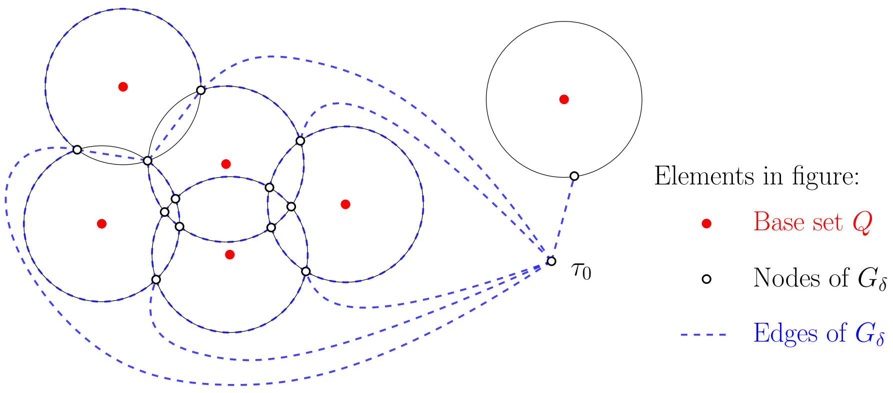


Figure 5.1: Arrangement $\mathcal{A}_\delta$ and construction of G_δ.

5.2.1 Reduction to Offline Dynamic Grid Reachability

In this section we prove Lemma 5.2. Given polygonal curves π, σ of length n over $\mathbb{R}^2$ and $\delta > 0$, we determine whether $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ for some $\tau \in \mathbb{R}^2$ as follows.

For any radius r and point $p \in \mathbb{R}^2$, we let $D_r(p)$ denote the disk of radius r with center p.

Observation 5.6. *Let $\tau \in \mathbb{R}^2$ and define the $n \times n$ matrix M^τ over $\{0,1\}$ by*

$$M^\tau_{i,j} = 1 \qquad \iff \qquad \tau \in D_\delta(\pi_i - \sigma_j).$$

We have $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ if and only if there is a monotone 1-path from $(1,1)$ to (n,n) in M^τ.

By the above observation, it suffices to check for the existence of monotone 1-paths from $(1,1)$ to (n,n) in a bounded number of matrices. To this end, let $Q := \{\pi_i - \sigma_j \mid i,j \in [n]\}$. We construct the arrangement $\mathcal{A}_\delta$ of the disks $D_\delta(q)$ for $q \in Q$, in the sense that we construct the following plane graph G_δ (cf. Figure 5.1). First, we include the vertices of $\mathcal{A}_\delta$ in its node set (i.e., intersections of disks $D_\delta(q), D_\delta(q')$ with $q, q' \in Q$). Second, for each $q \in Q$ for which $D_\delta(q)$ intersects no $D_\delta(q')$ for $q' \in Q \setminus \{q\}$, we include an arbitrary τ_q on the boundary of $D_\delta(q)$. Finally, we add an arbitrary vertex $\tau_0 \in \mathbb{R}^2$ lying in the outer face of $\mathcal{A}_\delta$ to the node set. Any nodes τ, τ' of G_δ are connected by an edge if they are neighboring vertices on the boundary of some face of $\mathcal{A}_\delta$; additionally, we connect τ_0 to all nodes which lie on the boundary separating the outer face from some other face. Note that G_δ can only be disconnected now if there is an inner face of $\mathcal{A}_\delta$ that completely encloses a non-empty component of $\mathcal{A}_\delta$. In this case, we can simply insert any edge from the part connected to the outer face, to the part that is enclosed. Performing this recursively, we turn G_δ into a connected plane graph. Note that it has $\mathcal{O}(|Q|^2) = \mathcal{O}(n^4)$ nodes and edges, and can be constructed in time $\mathcal{O}(n^4)$.

Note that by Observation 5.6, it suffices to check whether $\mathrm{d_{dF}}(\pi, \sigma + \tau_v) \leq \delta$ for any node τ_v in[1] G_δ: for any (bounded) face f of $\mathcal{A}_\delta$, there is at least one point τ_v in G_δ

[1]Note that our approach here deviates somewhat from the description in the introduction. This is due to the fact that for adversarial δ, we might need to consider degenerate faces consisting of a single point only; due to the parametric search that we describe in Section 5.2.3, we may not assume δ to avoid such degenerate cases. Traversing vertices of the arrangement instead of the faces takes care of such border cases in a natural manner.

that lies on the boundary of f. The corresponding matrix M^{τ_v} has at least the same 1-positions as the matrix M^{τ} for any $\tau \in f$ (and might have more).

To obtain a walk visiting all nodes in G_δ, we simply compute a spanning tree T of G_δ, double all edges of T, and find an Eulerian cycle starting and ending in τ_0. Denote this cycle by $\tau_0, \dots, \tau_L$ and observe that $L = \mathcal{O}(n^4)$. Let $M_0 = M^{\tau_0}$ be the $n \times n$ all-zeroes matrix. For any $0 \leq i < L$, we construct an update sequence $\bar{u}_i$ that first sets all positions (i,j) with $M^{\tau_i} = 1$ and $M^{\tau_{i+1}} = 0$ to zero, and then sets all positions (i,j) with $M^{\tau_i} = 0$ and $M^{\tau_{i+1}} = 1$ to 1. Thus, if we start with M^{τ_i} and perform the updates in $\bar{u}_i$, then at any point in time, the current matrix is dominated by either M^{τ_i} or $M^{\tau_{i+1}}$ (that is, the free positions of the current matrix are always a subset of M^{τ_i}'s free positions or a subset of $M^{\tau_{i+1}}$'s free positions), and at the end we obtain $M^{\tau_{i+1}}$. Thus, by concatenating all updates to $\bar{u}_0, \dots, \bar{u}_{L-1}$, we obtain an instance of the Offline Dynamic Grid Reachability problem with initial matrix M_0 and update sequence $u_1, \dots, u_{L'}$ with the following property: There is some $i \in \{0, \dots, L\}$ with $\mathrm{d_{dF}}(\pi, \sigma + \tau_i) \leq \delta$ if and only if there is some $i' \in [L']$ such that (n,n) is reachable from $(1,1)$ via a monotone 1-path in $M_0[\![u_1, \dots, u_{i'}]\!]$. Since $\tau_0, \dots, \tau_L$ visits all nodes in G_δ, this is equivalent to testing whether $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ for any $\tau \in \mathbb{R}^2$.

It remains to bound L'. We assume general position of the input points $P \cup S$ where $P = \{\pi_i \mid i \in [n]\}$ and $S = \{\sigma_j \mid j \in [n]\}$. Observe that there is some universal constant C such that no C points in Q lie on a common circle.[2] Thus, if we move from vertex τ_i to τ_{i+1} along an edge in G_δ, e.g., from one vertex of the boundary of some face to a neighboring vertex on that boundary, there are at most $2C$ entries that change from M^{τ_i} to $M^{\tau_{i+1}}$, since for both τ_i and τ_{i+1}, there are at most C disks intersecting this vertex and no other entries change when moving along this edge (by construction of G_δ). Thus, $L' \leq 2CL = \mathcal{O}(n^4)$. Consequently, given an algorithm solving Offline Dynamic Grid Reachability in time $T(n,U)$, we can determine whether $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ for some $\tau \in \mathbb{R}^2$ in time $\mathcal{O}(T(n, L')) = \mathcal{O}(T(n, n^4))$.

5.2.2 Solving Offline Dynamic Grid Reachability

We prove Theorem 5.4 using the grid reachability data structure given in Lemma 5.3. Specifically, we claim that the following algorithm (formalized as Algorithm 8) solves Offline Dynamic Grid Reachability in time $\mathcal{O}(n^2 + Un^{2/3}\log^2 n)$. We partition our updates $u_1, \dots, u_U$ into groups $\bar{u}_1, \dots, \bar{u}_{\mathcal{O}(U/k)}$ containing k updates each. For any group $\bar{u}_i$, let M_i be obtained from M by performing all updates *prior* to $\bar{u}_i$. Note that $\bar{u}_i$ will update a set of at most k positions; denote this set by $\mathcal{T}_i$. We build the grid reachability data structure $\mathcal{D}_i = \mathcal{D}_{M_i^{\mathbf{0}}, \mathcal{T}_i}$ with terminal set $\mathcal{T}_i$ and matrix $M_i^{\mathbf{0}}$ obtained from M_i by setting

[2]To be more precise, we sketch how to argue that the general position assumption for $P \cup S$ "transfers" to Q. Assume that there exist points $q_1, \dots, q_\ell \in Q$ lying on a common circle. For all i, we must have $q_i = p_i - s_i$ for some $p_i \in P, s_i \in S$. First assume that $\ell = 4$ and that there is some s such that $s_i = s$ for all $i \in \{1,2,3,4\}$. Then already $p_1, \dots, p_4$ lie on a common circle (it has the same radius as the original circle, and its center is translated by s), which violates the general position assumption of points in P. Otherwise, let the points $q_1, \dots, q_\ell$ be arbitrary with $\ell \geq 36$. By the first case, any s_i appears at most 3 times among $s_1, \dots, s_\ell$. After removing copies, we may assume without loss of generality that $q_1, \dots, q_{\ell'}$ with $\ell' \geq \ell/3$ have distinct s_i's. Similarly, we may also assume that $q_1, \dots, q_{\ell''}$ with $\ell'' \geq \ell/9 \geq 4$ have distinct p_i's as well. The fact that q_4 lies on the circle defined by q_1, q_2, q_3 can be expressed by a nonzero degree-2 polynomial $P_{q_1,q_2,q_3}(x,y)$ vanishing on q_4. Since $q_4 = p_4 - s_4$, we obtain a nonzero degree-2 polynomial $P'_{q_1,q_2,q_3}(p,s)$ vanishing on (p_4, s_4). This contradicts general position of $P \cup S$.

```
1: function $\textsc{OfflineDynamicGridReachability}(M,\, u_1,\dots,u_U)$
2:     parameter: $k$
3:     Divide $u_1,\dots,u_U$ into $s = \lceil \frac{U}{k} \rceil$ subsequences $\bar{u}_1,\dots,\bar{u}_s$ of length $k$.[3]
4:     Initialize $M_1 \leftarrow M$
5:     Set $\mathcal{T}_1$ to the set of positions updated in $\bar{u}_1$.
6:     Let $M_1^{\mathbf{0}}$ be obtained from $M_1$ by updating all positions in $\mathcal{T}_1$ to 0
7:     Build $\mathcal{D}_{M_1^{\mathbf{0}},\mathcal{T}_1}$
8:     for $i \leftarrow 1$ to $s$ do
9:         for $j \leftarrow 1$ to $k$ do
10:            Let $F \subseteq \mathcal{T}_i$ be the free terminals in $M_i[\![\bar{u}_i[1],\dots,\bar{u}_i[j]]\!]$
11:            if reachability query in $\mathcal{D}_{M_i^{\mathbf{0}},\mathcal{T}_i}$ with free terminals $F$ is successful then
12:                return true
13:        Set $M_{i+1} \leftarrow M_i[\![\bar{u}_i]\!]$
14:        Set $\mathcal{T}_{i+1}$ to the set of positions updated in $\bar{u}_{i+1}$.[4]
15:        Let $M_{i+1}^{\mathbf{0}}$ be obtained from $M_{i+1}$ by updating all positions in $\mathcal{T}_{i+1}$ to 0
16:        update $\mathcal{D}_{M_i^{\mathbf{0}},\mathcal{T}_i}$ to $\mathcal{D}_{M_{i+1}^{\mathbf{0}},\mathcal{T}_{i+1}}$
17:    return false
```

Algorithm 8: Solving Offline Dynamic Grid Reachability on matrix M and update sequence $u_1,\dots,u_U$.

the positions of all terminals $\mathcal{T}_i$ to 0. Observe that the state after any update within $\bar{u}_i$ corresponds to $M_i^{\mathbf{0}}$ with some additional positions in $\mathcal{T}_i$ set to 1 (the *free terminals*). Thus, for each update within $\bar{u}_i$, we can determine whether it creates a monotone 1-path from $(1,1)$ to (n,n) by simply determining the set $F \subseteq \mathcal{T}_i$ of free terminals at the point of this update and performing the corresponding reachability query in $\mathcal{D}_i$. It is straightforward to argue that the resulting algorithm correctly solves Offline Dynamic Grid Reachability.

To analyze the running time of Algorithm 8, note that each data structure $\mathcal{D}_{M_i^{\mathbf{0}},\mathcal{T}_i}$ has a terminal set of size at most k and each $M_{i+1}^{\mathbf{0}}$ differs from $M_i^{\mathbf{0}}$ in at most $2k$ entries. Thus by Lemma 5.3, we need time $\mathcal{O}(n^2 + k\log^2 n)$ to build $\mathcal{D}_1 = \mathcal{D}_{M_1^{\mathbf{0}},\mathcal{T}_1}$ in Line 7. The time spent for handling a single group $\bar{u}_i$ is bounded by the time to perform k queries in $\mathcal{D}_i = \mathcal{D}_{M_i^{\mathbf{0}},\mathcal{T}_i}$ plus the time to update $\mathcal{D}_i = \mathcal{D}_{M_i^{\mathbf{0}},\mathcal{T}_i}$ to $\mathcal{D}_{i+1} = \mathcal{D}_{M_{i+1}^{\mathbf{0}},\mathcal{T}_{i+1}}$, which amounts to $\mathcal{O}(k^2\log^3 n + n\sqrt{k}\log n + k\log^2 n) = \mathcal{O}(k^2\log^3 n + n\sqrt{k}\log n)$ by Lemma 5.3. Thus, in total, we obtain a running time of

$$\mathcal{O}\left(n^2 + k\log^2 n + \frac{U}{k}\left(k^2\log^3 n + n\sqrt{k}\log n\right)\right) = \mathcal{O}\left(n^2 + U\left(k\log^3 n + \frac{n}{\sqrt{k}}\log n\right)\right).$$

This expression is minimized by setting $k := n^{2/3}/\log^{4/3} n$, resulting in a total running time of $\mathcal{O}(n^2 + Un^{2/3}\log^{1+2/3} n) = \mathcal{O}(n^2 + Un^{2/3}\log^2 n)$, as desired.

5.2.3 Parametric Search

In this section, we sketch how to use parametric search techniques (due to Megiddo [133] and Cole [74]) to reduce the optimization problem to the decision problem with small overhead, i.e., we prove Lemma 5.5. Specifically, for the readers' convenience, we describe the arguments made by Ben Avraham et al. [33] in slightly more detail.

Our aim in this section is to compute the discrete Fréchet distance under translation of polygonal curves π, σ of length n over $\mathbb{R}^2$, i.e., to determine

$$\delta^* := \min_{\tau \in \mathbb{R}^2} \mathrm{d_{dF}}(\pi, \sigma + \tau).$$

Using the decision algorithm, we can determine, for any $\delta > 0$, whether $\delta^* \leq \delta$ in time $T_{\mathrm{dec}}(n)$. As we shall see below, there is a range of $\mathcal{O}(n^6)$ possible values (defined by the point set of π, σ) that δ^* might attain (called *critical values*). Naively computing all critical values and performing a binary search would result in an $\mathcal{O}((n^6 + T_{\mathrm{dec}}(n)) \log n)$-time algorithm, which is too slow for our purposes. Instead, we use the parametric search technique to perform an implicit search over these critical values.

Conceptually, we aim to determine the combinatorial structure of the arrangement $\mathcal{A}_{\delta^*}$ defined in Section 5.2.1 (captured by the graph G_{δ^*}) without knowing δ^* in advance. To specify this combinatorial structure, define for every $q \in Q$ the set

$$I_\delta(q) := \{q' \in Q \setminus \{q\} \mid D_\delta(q), D_\delta(q') \text{ intersect}\}.$$

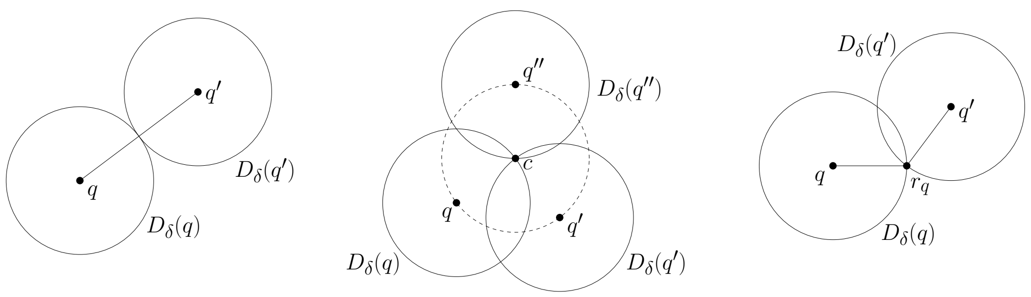
Note that for every $q' \in I_\delta(q)$, there are one or two intersection points of $D_\delta(q)$, $D_\delta(q')$, which we denote by $C^1_\delta(q, q')$ and $C^2_\delta(q, q')$ (note that we allow these points to coincide if $D_\delta(q), D_\delta(q')$ intersect in a single point only) – we assume this notation to be chosen consistently in the sense that $C^1_\delta(q, q')$ and $C^1_\delta(q', q)$ refer to the same point (likewise for $C^2_\delta(q, q')$ and $C^2_\delta(q', q)$). We denote by $\mathcal{C}_\delta(q)$ the set of all intersection points on the boundary of $D_\delta(q)$, i.e., $C^1_\delta(q, q')$, $C^2_\delta(q, q')$ for all $q' \in I_\delta(q)$. We obtain a list $L_\delta(q)$ by starting with the rightmost point on the boundary of $D_\delta(q)$, say r_q, and listing all intersection points $C \in \mathcal{C}_\delta(q)$ in counter-clockwise order. Observe that the combinatorial structure of $\mathcal{A}_\delta$ is completely specified by the lists $L_\delta(q)$ for $q \in Q$.

We wish to construct $L_{\delta^*}(q)$ for all $q \in Q$ using calls to our decision algorithm, i.e., queries of the form "Is $\delta^* \leq \delta$?". Along the way, we maintain a shrinking interval $(\alpha, \beta]$ such that $\delta^* \in (\alpha, \beta]$ – our aim is that in the end $(\alpha, \beta]$ no longer contains critical values except for β, and thus $\delta^* = \beta$ can be derived. We proceed in two steps.

Step 1: Determining $I_{\delta^*}(q)$. The critical values for this step are the half-distances of all pairs $q, q' \in Q$ (cf. Figure 5.2(a)). We list all these values and perform a binary search over them, using our decision algorithm. Since there are at most $\mathcal{O}(|Q|^2) = \mathcal{O}(n^4)$ such values, we obtain an algorithm running in time $\mathcal{O}((n^4 + T_{\mathrm{dec}}(n)) \log n)$ returning an interval $(\alpha_1, \beta_1]$ such that $\delta^* \in (\alpha_1, \beta_1]$ and no half-distance of a pair $q, q' \in Q$ is contained in (α_1, β_1). Thus, from this point on we know $I_{\delta^*}(q)$ for all $q \in Q$ (without knowing the exact value of δ^* yet).

Step 2: Sorting $L_{\delta^*}(q)$. We use the following well-known variant of Meggido's parametric search that is due to Cole [74].

Lemma 5.7 (implicit in [74])**.** *Let parametric values $f_1(\delta), \ldots, f_N(\delta)$ be given. Assume there is an unknown value $\delta^* > 0$ and a decision algorithm determining, given $\delta > 0$, whether $\delta^* \leq \delta$ in time $T(N)$. If we can determine $f_i(\delta^*) \leq f_j(\delta^*)$ for any $i, j \in [N]$ using only a constant number of queries to the decision algorithm, then in time $\mathcal{O}((N + T(N)) \log N)$, we can sort $f_1(\delta^*), \ldots, f_N(\delta^*)$ and obtain an interval $(\alpha, \beta]$ such that*



(a) The elements in $L_\delta(q)$ change at radius δ that is a half-distance to some other $q' \in Q$.

(b) The ordering of $L_\delta(q)$ might change at radius δ such that $q, q', q'' \in Q$ lie on a common circle around some center c.

(c) The ordering of $L_\delta(q)$ might change at radius δ such that $D_\delta(q')$ intersects $D_\delta(q)$ in r_q.

Figure 5.2: Critical values for the lists $L_\delta(q), q \in Q$.

$\delta^* \in (\alpha, \beta]$ *and no critical value for the sorted order of* $f_1(\delta), \ldots, f_N(\delta)$ *is contained in* (α, β).

Consider first the problem of sorting $L_{\delta^*}(q)$ for some $q \in Q$. By the above technique, we only need to argue that we can determine whether some $C^a_{\delta^*}(q, q')$ with $q' \in Q, a \in \{1, 2\}$ precedes some $C^b_{\delta^*}(q, q'')$ with $q'' \in Q, b \in \{1, 2\}$ in $L_{\delta^*}(q)$. Note that $C^a_\delta(q, q'), C^b_\delta(q, q'')$ move continuously on the boundary of $D_\delta(q)$ (while δ varies) and there are only constantly many choices of δ for which any of the points $C^a_\delta(q, q'), C^b_\delta(q, q''), r_q$ coincide (and thus the order might possibly change).[5] By testing for these $\mathcal{O}(1)$ critical values of δ, we can determine the order of $C^a_{\delta^*}(q, q'), C^b_{\delta^*}(q, q''), r_q$ on the boundary of $D_\delta(q)$, and thus resolve a comparison of $C^a_{\delta^*}(q, q')$ and $C^b_{\delta^*}(q, q'')$ in the order of $L_{\delta^*}(q)$ using only a constant number of calls to the decision algorithm.

Note that by arbitrarily choosing an order of Q, we may use Cole's sorting procedure (Lemma 5.7) to construct all lists $L_{\delta^*}(q), q \in Q$ simultaneously (we simply need to adapt the comparison function to compare $C^a_{\delta^*}(q, q'), C^b_{\delta^*}(\tilde{q}, q'')$ according to the order of Q if $q \neq \tilde{q}$). Note that in this application of Lemma 5.7, we have $N = \sum_{q \in Q} |\mathcal{C}_{\delta^*}(q)| = \mathcal{O}(|Q|^2) = \mathcal{O}(n^4)$.

It follows that in time $\mathcal{O}((n^4 + T_{\text{dec}}(n)) \log n)$, we can obtain an interval $(\alpha_2, \beta_2]$ such that $\delta^* \in (\alpha_2, \beta_2]$, while β_2 is the only value for δ for which the combinatorial structure of $\mathcal{A}_\delta$ changes in $(\alpha_2, \beta_2]$. Thus, $\delta^* = \beta_2$, as desired.

The overall running time of the above procedure amounts to $\mathcal{O}((n^4 + T_{\text{dec}}(n)) \log n)$, which concludes the proof of Lemma 5.5.

5.3 Grid Reachability Data Structure

In this section, we prove Lemma 5.3, which we restate here for convenience.

[5]The important critical values for this step are the $\mathcal{O}(|Q|^3) = \mathcal{O}(n^6)$ radii of points with three (or more) points of Q on their boundary. See Figure 5.2 for an illustration of all types of critical values.

Lemma 5.3 (Grid reachability data structure)**.** *Given an $n \times n$ matrix M over $\{0,1\}$ and a set of* terminals *$\mathcal{T} \subseteq [n] \times [n]$ of size $k > 0$, there is a data structure $\mathcal{D}_{M,\mathcal{T}}$ with the following properties.*

i) (Construction:) We can construct $\mathcal{D}_{M,\mathcal{T}}$ in time $\mathcal{O}(n^2 + k \log^2 n)$.

ii) (Reachability Query:) Given $F \subseteq \mathcal{T}$, we can determine in time $\mathcal{O}(k \log^3 n)$ whether there is a monotone path from $(1,1)$ to (n,n) using only positions (i,j) with $M_{i,j} = 1$ or $(i,j) \in F$.

iii) (Update:) Given $\mathcal{T}' \subseteq [n] \times [n]$ of size k and an $n \times n$ matrix M' over $\{0,1\}$ differing from M in at most k positions, we can update $\mathcal{D}_{M,\mathcal{T}}$ to $\mathcal{D}_{M',\mathcal{T}'}$ in time $\mathcal{O}(n\sqrt{k} \log n + k \log^2 n)$. Here, we assume M' to be represented by the set Δ of positions in which M and M' differ.

The rough outline is as follows: We obtain the data structure by repeatedly splitting the free-space diagram into smaller blocks. This yields $\mathcal{O}(\log n)$ levels of blocks, where in each block we store reachability information from all “inputs” to the block (i.e., the lower-left boundary) to all “outputs” of the block (i.e., the upper-right boundary). Any change in the matrix M is reflected only in $\mathcal{O}(\log n)$ blocks containing this position, thus, we can quickly update the information. This approach was pursued already by Ben Avraham et al. [33].

In addition, however, we need to maintain reachability of all terminals $\mathcal{T}$ to the inputs and from the outputs of each block. Surprisingly, we only need an additional storage of $\mathcal{O}(|\mathcal{T}|)$ per block. We show how to maintain this information also under updates and how it can be used by a divide and conquer approach to answer any reachability queries.

To this end, we start with some basic definitions (block structure, identifiers for each position, etc.) in Section 5.3.1. We can then prove the succinct characterization of terminal reachability in Section 5.3.2, which is the key aspect of our data structure. Given this information, we can define exactly what information we store for each block in Section 5.3.3. We give algorithms computing the information for some block given the information for its children in Section 5.3.4, which allows us to prove the initialization and update statements (i.e., i) and iii) of Lemma 5.3) in Section 5.3.5. Finally, Section 5.3.6 is devoted to the reachability queries, i.e., proving ii) of Lemma 5.3.

5.3.1 Basic Structures and Definitions

Without loss of generality, we may assume that $n = 2^\kappa + 1$ for some integer $\kappa \in \mathbb{N}$. Otherwise, for any $n \times n$ matrix M over $\{0,1\}$, we could define an $n' \times n'$ matrix M' with (1) $n' = 2^\kappa + 1$ for some $\kappa \in \mathbb{N}$ with $n < n' \leq 2n$ and (2) setting $M'_{i,j} = M_{i,j}$ for all $(i,j) \in [n] \times [n]$ and setting $M'_{i,j} = 1$ if and only if $i = j$ for all $(i,j) \in [n'] \times [n'] \setminus [n] \times [n]$. Clearly, existence of a monotone 1-path from $(1,1)$ to (n,n) in M is equivalent to existence of a monotone 1-path from $(1,1)$ to (n',n') in M'.

Canonical blocks. Let I, J be intervals in $[n]$ with $n = 2^\kappa + 1$. We call $I \times J \subseteq [n] \times [n]$ a *block*. In particular, we only consider blocks obtained by splitting the square $[n] \times [n]$ alternately horizontally and vertically until we are left with 2×2 blocks. Formally, we

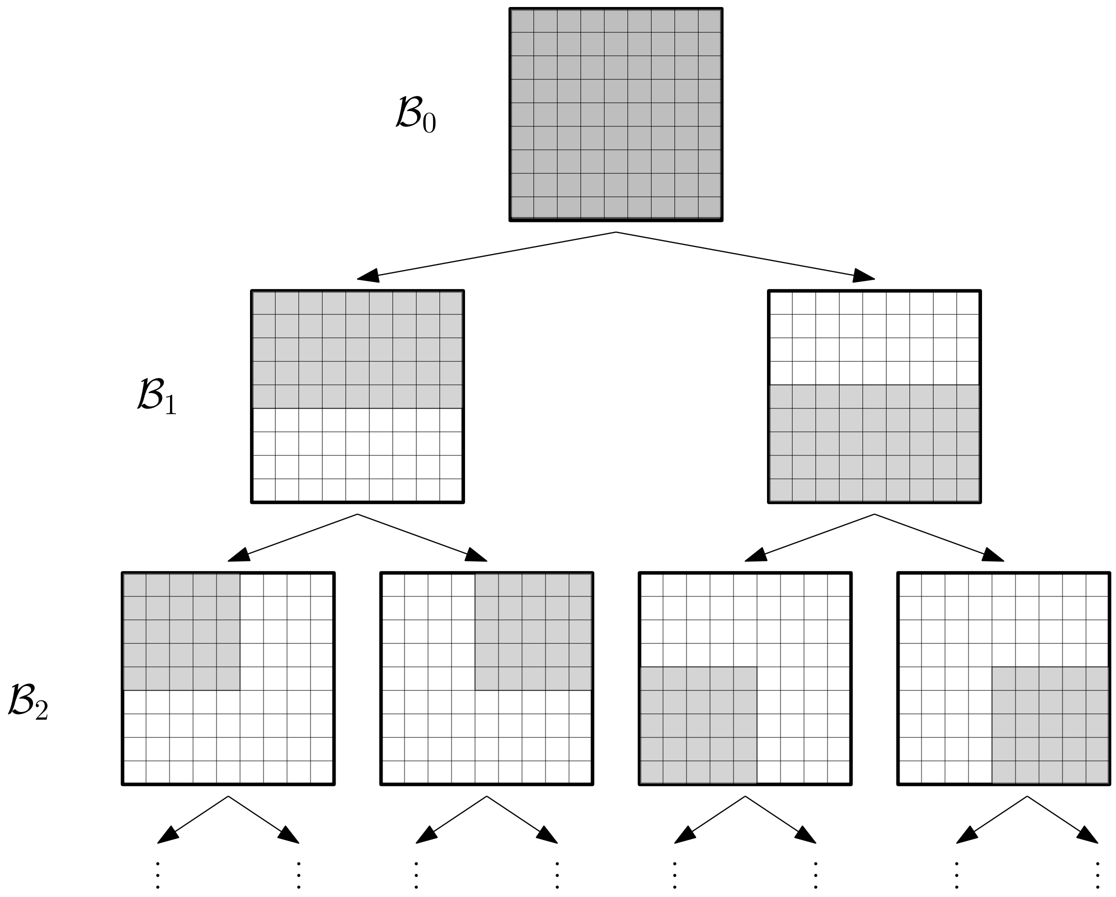


Figure 5.3: The sets of canonical blocks $\mathcal{B}_0, \mathcal{B}_1, \mathcal{B}_2, \dots, \mathcal{B}_{2\kappa}$. We alternate between horizontal and vertical splits. Note that child blocks overlap at their boundary.

define $\mathcal{B}_0 \coloneqq \{([n],[n])\}$ and construct $\mathcal{B}_{\ell+1}$ inductively by splitting each block $B \in \mathcal{B}_\ell$ as follows:

- For $\ell = 2i$ with $0 \le i < \kappa$, we have $B = (I, J)$ with $|I| = |J| = 2^{\kappa-i}+1$. We split J into intervals J_1, J_2, where J_1 contains the first $(2^{\kappa-i-1}+1)$ elements in J and J_2 contains the last $(2^{\kappa-i-1}+1)$ elements in J (thus J_1 and J_2 intersect in the middle element of J). Add (I, J_1) and (I, J_2) to $\mathcal{B}_{\ell+1}$.

- For $\ell = 2i+1$ with $0 \le i < \kappa$, we have $B = (I, J)$ with $|I| = 2^{\kappa-i}+1$ and $|J| = 2^{\kappa-i-1}+1$. Analogous to above, we split I into two equal-sized intervals I_1, I_2, where I_1 contains the first $(2^{\kappa-i-1}+1)$ elements in I and I_2 contains the last $(2^{\kappa-i-1}+1)$ elements in I. Add (I_1, J) and (I_2, J) to $\mathcal{B}_{\ell+1}$.

We let $\mathcal{B} \coloneqq \bigcup_{\ell=0}^{2\kappa} \mathcal{B}_\ell$ be the set of *canonical blocks*, and call each block $B \in \mathcal{B}_\ell$ a *canonical block on level ℓ*. The blocks $B_1 = (I_1, J), B_2 = (I_2, J) \in \mathcal{B}_{\ell+1}$ (or $B_1 = (I, J_1), B_2 = (I, J_2) \in \mathcal{B}_{\ell+1}$, respectively) obtained from $B = (I, J) \in \mathcal{B}_\ell$ are called the *children* of B. See Figure 5.3.

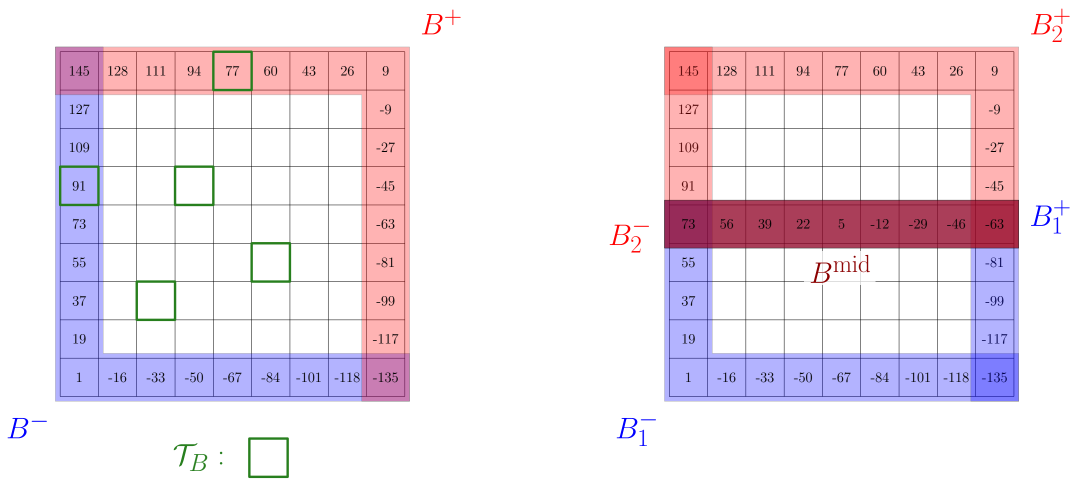


Figure 5.4: Structure of a block B

Boundaries. For any $B = (I, J) \in \mathcal{B}$, we denote the lower left boundary of B as $B^- = \{\min I\} \times J \cup I \times \{\min J\}$, and call each $p \in B^-$ an *input* of B. Analogously, we denote the upper right boundary of B as $B^+ = \{\max I\} \times J \cup I \times \{\max J\}$, and call each $q \in B^+$ an *output* of B. By slight abuse of notation, we define $|\partial B| = |B^- \cup B^+|$ as the size of the boundary of B, i.e., the number of inputs and outputs of B.

If B splits into children B_1, B_2, we call $B^{\mathrm{mid}} = B_1^+ \cap B_2^-$ the *splitting boundary* of B.

Indices. To prepare the description of this information, we first define, for technical reasons, *indices* for all positions in $[n] \times [n]$. It allows us to give each position a unique identifier with the property that for any canonical block B, the indices yield a local ordering of the boundaries.

Observation 5.8. *Let* $\mathrm{ind} : [n] \times [n] \to \mathbb{N}$, *where for any point* $p = (x, y) \in [n] \times [n]$, *we set* $\mathrm{ind}(p) := (y - x)(2n) + x$. *We call* $\mathrm{ind}(p)$ *the* index *of* p. *This function satisfies the following properties:*

(1) The function ind *is injective, can be computed in constant time, and given* $i = \mathrm{ind}(p)$, *we can determine* $\mathrm{ind}^{-1}(i) := p$ *in constant time.*

(2) For any $B \in \mathcal{B}$, ind *induces an ordering of* B^+ *in counter-clockwise order and an ordering of* B^- *in clockwise order.*

We refer to Figure 5.4 for an illustration of a block B, its boundaries, and the indices of all positions.

5.3.2 Reachability Characterization

Our aim is to construct a data structure $\mathcal{D}_{M,\mathcal{T}} = (\mathcal{D}_{M,\mathcal{T}}(B))_{B \in \mathcal{B}}$, where $\mathcal{D}_{M,\mathcal{T}}(B)$ succinctly describes reachability (via monotone 1-paths) between the boundaries B^-, B^+

and the terminals $\mathcal{T}_B := \mathcal{T} \cap B$ inside B. In particular, we show that we only require space $\mathcal{O}(|\partial B| + |\mathcal{T}_B|)$ to represent this information.

To prepare this, we start with a few simple observations that yield a surprisingly simple characterization of reachability from any terminal to the boundary.

Compositions of crossing paths. We say that we reach q from p, written $p \rightsquigarrow q$, if there is a traversal $T = (t_1, \dots, t_\ell)$ with $t_1 = p$, $t_\ell = q$, and t_i is free for all $1 < i < \ell$ (note that we do not require t_1 and t_ℓ to be free). We call such a slightly adapted notion of traversal a *reach traversal*. By connecting the points of T by straight lines, we may view T also as a polygonal curve in $\mathbb{R}^2$. The following property is a standard observation for problems related to the Fréchet distance.

Observation 5.9. *Let T_1, T_2 be reach traversals from p_1 to q_1 and from p_2 to q_2, respectively. Then if T_1 and T_2 intersect, we have $p_1 \rightsquigarrow q_2$ (and, symmetrically, $p_2 \rightsquigarrow q_1$).*

Proof. Let $t \in [n] \times [n]$ be a free position in which T_1, T_2 intersect (observe that such a point with integral coordinates must exist unless $p_1 = p_2$ or $q_1 = q_2$; in the latter case, the claim is trivial). Note that t splits T_1, T_2 into $T_1 = T_1^a \circ T_1^b$ and $T_2 = T_2^a \circ T_2^b$ such that T_1^a, T_2^a are reach traversals ending in t and T_1^b, T_2^b are reach traversal starting in t. By concatenating T_1^a and T_2^b, we obtain a reach traversal from p_1 to q_2. Symmetrically, $T_2^a \circ T_1^b$ proves $p_2 \rightsquigarrow q_1$. □

Let $B \in \mathcal{B}$ and recall that $\operatorname{ind}(\cdot)$ orders B^+ counter-clockwise. For any $p \in B$, we define $\mathrm{A}(p) := \min\{\operatorname{ind}(q) \mid q \in B^+, p \rightsquigarrow q\}$, and analogously $\mathrm{Z}(p) := \max\{\operatorname{ind}(q) \mid q \in B^+, p \rightsquigarrow q\}$ (note that $\mathrm{A}(p)$ and $\mathrm{Z}(p)$ correspond to the lowest/rightmost and highest/leftmost pointer, respectively, in [19, Section 3.2]). These two values define a corresponding *reachability interval* $\mathcal{I}(p) := [\mathrm{A}(p), \mathrm{Z}(p)]$ that contains all $q \in B^+$ with $p \rightsquigarrow q$. In the following analysis, we slightly abuse notation by also using $\operatorname{ind}(p)$ to denote the corresponding (unique) position $p \in [n] \times [n]$.

Definition 5.10. *Let $p \in B$ with $\infty > \mathrm{A}(p), \mathrm{Z}(p) > -\infty$ and fix any reach traversals T_A, T_Z from p to $\mathrm{A}(p)$ and $\mathrm{Z}(p)$ such that we can write*

$$T_A = P_{\mathrm{com}} \circ P'_A,$$
$$T_Z = P_{\mathrm{com}} \circ P'_Z,$$

for some polygonal curves $P_{\mathrm{com}}, P'_A, P'_Z$ with P'_A, P'_Z non-intersecting. Let $\mathcal{F}$ be the face enclosed by P'_A, P'_Z and the path from $\mathrm{A}(p)$ to $\mathrm{Z}(p)$ on B^+ (if $\mathrm{A}(p) = \mathrm{Z}(p)$, we let $\mathcal{F}$ be the empty set). We define the reach region *of p as*

$$\mathcal{R}(p) := \mathcal{F} \cup P_{\mathrm{com}}.$$

We refer to Figure 5.5 for an illustration. Observe that the desired traversals T_A, T_Z for defining $\mathcal{R}(p)$ always exist: For any reach traversals T'_A, T'_Z from p to $\mathrm{A}(p)$ and $\mathrm{Z}(p)$, respectively, consider the latest point in which T'_A, T'_B intersect, say t. We can define reach traversals T_A and T_Z by following T'_A until t and then following the remainder of T'_A or T'_Z to reach $\mathrm{A}(p)$ or $\mathrm{Z}(p)$, respectively. These traversals satisfy the conditions by construction. (Strictly speaking, any feasible choice for T_A, T_Z gives a potentially

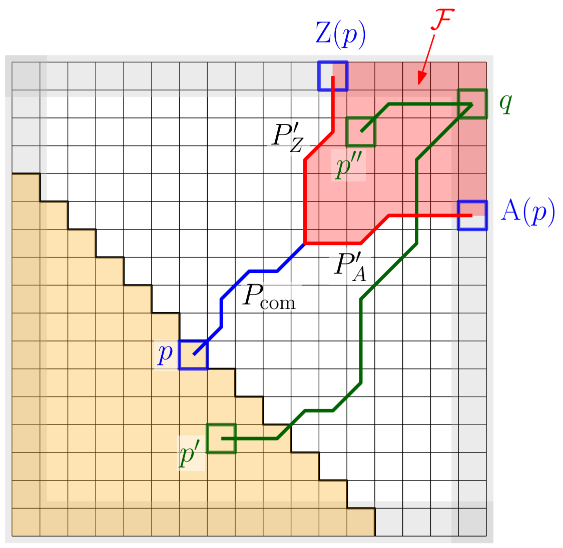


Figure 5.5: Illustration of $\mathcal{R}(p)$, Proposition 5.11 and Lemma 5.12: Any reach traversal from $p' \notin \mathcal{R}(p)$ must cross P'_A or P'_Z to reach q. However, if $p'' \rightsquigarrow q$ but $p'' \in \mathcal{R}(p)$, then q might not be reachable from p. A sufficient condition for $p' \notin \mathcal{R}(p)$ is that $p' \neq p$ and $L(p') \leq L(p)$ (indicated by the orange triangular area).

different reach region $\mathcal{R}(p)$. However, any fixed choice will be sufficient for our proofs, e.g., choosing lexicographically smallest/largest traversals.)

The following property generalizes an insightful property of reachability from the inputs to the outputs (cf. [19, Lemma 10] and [33, Corollary 4.2]) to a similar property for reachability from arbitrary block positions to the outputs, using the same argument of crossing traversals.

Proposition 5.11. *Let $p, p' \in B$, $q \in B^+$ with $\mathrm{ind}(q) \in \mathcal{I}(p)$ and $p' \notin \mathcal{R}(p)$. Then $p' \rightsquigarrow q$ implies $p \rightsquigarrow q$.*

Proof. The claim holds trivially if $\mathrm{ind}(q) = \mathrm{A}(p)$ or $\mathrm{ind}(q) = \mathrm{Z}(p)$. Thus, we may assume that $\mathrm{A}(p) < \mathrm{Z}(p)$, which implies that the face $\mathcal{F}$ in $\mathcal{R}(p)$ is nonempty with $q \in \mathcal{F}$ and $p' \notin \mathcal{F}$. Hence any reach traversal T from p' to q must cross the boundary of $\mathcal{F}$, in particular, the path $P(T_A)$ or $P(T_Z)$, where T_A, T_Z both originate in p. By Observation 5.9, this yields $p \rightsquigarrow q$. □

Reachability labelling. We define a total order on nodes in B that allows us to succinctly represent reachability towards B^+ for any subset $S \subseteq B$ in space $\tilde{\mathcal{O}}(|S|+|B^+|)$. The key is a labelling $L : [n] \times [n] \to \mathbb{N}$, defined by $L((x, y)) = x + y$, that we call the *reachability labelling*. For an illustration of the following lemma, we refer to Figure 5.5.

Lemma 5.12. *Let $p = (x, y), p' = (x', y') \in B$ with $L(p') \leq L(p)$ and $q \in B^+$ with $\mathrm{ind}(q) \in \mathcal{I}(p)$. Then $p' \rightsquigarrow q$ implies $p \rightsquigarrow q$.*

Proof. The proof idea is to show that $L(p') \leq L(p)$ implies that $p' \notin \mathcal{R}(p)$, and hence Proposition 5.11 shows the claim. Note that by monotonicity of reach traversals, any point $r = (r_x, r_y) \in \mathcal{R}(p)$ satisfies $r_x \geq x$ and $r_y \geq y$. Thus, $p' \in \mathcal{R}(p)$ only if $x' \geq x$,

$y' \geq y$, but this together with $x' + y' = L(p') \leq L(p) = x + y$ implies $(x', y') = (x, y)$. Summarizing, we either have $p = p'$, which trivially satisfies the claim, or $p' \notin \mathcal{R}(p)$, which yields the claim by Proposition 5.11. □

For any $S \subseteq B$, this labelling enables a surprisingly succinct characterization of which terminals in S have reach traversals to which outputs in B^+ by the following lemma (greatly generalizing a simpler characterization[6] for the special case of $S = B^-$, cf. [19, Lemma 10] and [33, implicit in Lemma 4.4]). This is one of our key insights.

Corollary 5.13. *Let $q \in B^+$ and define $\ell(q) := \min\{L(p) \mid p \in B, p \rightsquigarrow q\}$. Then for any $p \in B$, we have*

$$p \rightsquigarrow q \qquad \text{if and only if} \qquad \mathrm{ind}(q) \in \mathcal{I}(p) \text{ and } \ell(q) \leq L(p).$$

Proof. Clearly, $p \rightsquigarrow q$ implies, by definition of $\mathrm{A}(p), \mathrm{Z}(p)$, and $\ell(q)$, that $\mathrm{A}(p) \leq \mathrm{ind}(q) \leq \mathrm{Z}(p)$ and $\ell(q) \leq L(p)$.

Conversely, assume that $\mathrm{ind}(q) \in \mathcal{I}(p)$ and $\ell(q) \leq L(p)$. Take any $p' \in B$ with $p' \rightsquigarrow q$ and $\ell(q) = L(p')$. Thus we have $L(p') = \ell(q) \leq L(p)$, $\mathrm{ind}(q) \in \mathcal{I}(p)$ and $p' \rightsquigarrow q$, which satisfies the requirements of Lemma 5.12, yielding $p \rightsquigarrow q$. □

Given this characterization, we obtain a highly succinct representation of reachability information. Specifically, to represent the information which terminals in S have reach traversals to which outputs in B^+, we simply need to store $\ell(q)$ for all $q \in B^+$ as well as the interval $\mathcal{I}(p)$ for all $p \in S$. Thus, the space required to store this information amounts to only $\mathcal{O}(|\partial B| + |S|)$, which greatly improves over a naive $\mathcal{O}(|\partial B| \cdot |S|)$-sized tabulation.

Reverse information. By defining $L^{\mathrm{rev}}((x, y)) = -L((x, y)) = -x - y$, we obtain a labelling with symmetric properties. In particular, define $\mathrm{A}^{\mathrm{rev}}(q) := \min\{\mathrm{ind}(p) \mid p \in B^-, p \rightsquigarrow q\}$, $\mathrm{Z}^{\mathrm{rev}}(q) := \max\{\mathrm{ind}(p) \mid p \in B^-, p \rightsquigarrow q\}$ and the corresponding *reverse reachability interval* $\mathcal{I}^{\mathrm{rev}}(q) := [\mathrm{A}^{\mathrm{rev}}(q), \mathrm{Z}^{\mathrm{rev}}(q)]$. It is straightforward to prove the following symmetric variant of Corollary 5.13.

Corollary 5.14. *Let $p \in B^-$ and define $\ell^{\mathrm{rev}}(p) := \min\{L^{\mathrm{rev}}(q) \mid q \in B, p \rightsquigarrow q\}$. Then for any $q \in B$, we have*

$$p \rightsquigarrow q \qquad \text{if and only if} \qquad \mathrm{ind}(p) \in \mathcal{I}^{\mathrm{rev}}(q) \text{ and } \ell^{\mathrm{rev}}(p) \leq L^{\mathrm{rev}}(q).$$

Summary of reachability characterization. As a convenient reference, we collect here the main notation and results introduced in this section. For any $p \in B$, the *reachability interval* $\mathcal{I}(p)$ is defined as $[\mathrm{A}(p), \mathrm{Z}(p)]$ with

$$\mathrm{A}(p) = \min\{\mathrm{ind}(q) \mid q \in B^+, p \rightsquigarrow q\},$$
$$\mathrm{Z}(p) = \max\{\mathrm{ind}(q) \mid q \in B^+, p \rightsquigarrow q\}.$$

[6]In our language, this characterization is as follows: For any $p \in B^-, q \in B^+$, we have $p \rightsquigarrow q$ if and only if $\mathrm{ind}(q) \in \mathcal{I}(p)$ and there is some $p' \in B^-$ with $p' \rightsquigarrow q$. It is easy to see that this characterization no longer holds if we replace B^- by an arbitrary subset $S \supseteq B^-$; our approach instead relies on the reachability labelling to obtain a succinct and algorithmically tractable characterization.

(Note that $\mathcal{I}(p)$ might be empty if $\mathrm{A}(p) = \infty, \mathrm{Z}(p) = -\infty$.) For any $q \in B^+$, its *reachability level* $\ell(q)$ is defined as

$$\ell(q) = \min\{L(p) \mid p \in B, p \rightsquigarrow q\},$$

where $L((x, y)) = x + y$. For any $p \in B, q \in B^+$, we have the reachability characterization that

$$p \rightsquigarrow q \qquad \text{if and only if} \qquad \mathrm{ind}(q) \in \mathcal{I}(p) \text{ and } \ell(q) \le L(p).$$

For any $q \in B$, we have the *reverse reachability interval* $\mathcal{I}^{\mathrm{rev}}(q) = [\mathrm{A}^{\mathrm{rev}}(q), \mathrm{Z}^{\mathrm{rev}}(q)]$ with

$$\mathrm{A}^{\mathrm{rev}}(q) = \min\{\mathrm{ind}(p) \mid p \in B^-, p \rightsquigarrow q\},$$
$$\mathrm{Z}^{\mathrm{rev}}(q) = \max\{\mathrm{ind}(p) \mid p \in B^-, p \rightsquigarrow q\}.$$

(Again, $\mathcal{I}^{\mathrm{rev}}(q)$ might be empty if $\mathrm{A}^{\mathrm{rev}}(q) = \infty, \mathrm{Z}^{\mathrm{rev}}(q) = -\infty$.) For any $p \in B^-$, its *reverse reachability level* $\ell^{\mathrm{rev}}(p)$ is defined as

$$\ell^{\mathrm{rev}}(p) = \min\{L^{\mathrm{rev}}(q) \mid q \in B, p \rightsquigarrow q\},$$

where $L^{\mathrm{rev}}((x, y)) = -x - y$. For any $p \in B^-, q \in B$, we have the reachability characterization that

$$p \rightsquigarrow q \qquad \text{if and only if} \qquad \mathrm{ind}(p) \in \mathcal{I}^{\mathrm{rev}}(q) \text{ and } \ell^{\mathrm{rev}}(p) \le L^{\mathrm{rev}}(q).$$

5.3.3 Information Stored at Canonical Block B

Using the characterization given in Corollaries 5.13 and 5.14, we can now describe which information we need to store for any canonical block $B \in \mathcal{B}$.

Definition 5.15. *Let $B \in \mathcal{B}$. The* information stored at B *(which we denote as $\mathcal{D}_{M,\mathcal{T}}(B)$) consists of the following information: First, we store* forward reachability information *consisting of,*

- *for every $p \in B^- \cup \mathcal{T}_B$, the interval $\mathcal{I}(p)$, and*
- *for every $q \in B^+$, the reachability level $\ell(q)$.*

Symmetrically, we store reverse reachability information *consisting of,*

- *for every $q \in B^+ \cup \mathcal{T}_B$, the interval $\mathcal{I}^{\mathrm{rev}}(q)$, and*
- *for every $p \in B^-$, the reverse reachability level $\ell^{\mathrm{rev}}(p)$.*

Finally, if B has children $B_1, B_2 \in \mathcal{B}$, where B_1 is the lower or left sibling of B_2, we additionally store

- *an orthogonal range minimization data structure $\mathcal{OR}_B$ storing, for each* free $q \in B^{\mathrm{mid}} = B_1^+ \cap B_2^-$*, the value $\ell_2^{\mathrm{rev}}(q)$ under the key* $(\mathrm{ind}(q), \ell_1(q))$. *Here $\ell_1(q)$ denotes the forward reachability level in B_1, and $\ell_2^{\mathrm{rev}}(q)$ denotes the reverse reachability level in B_2.*

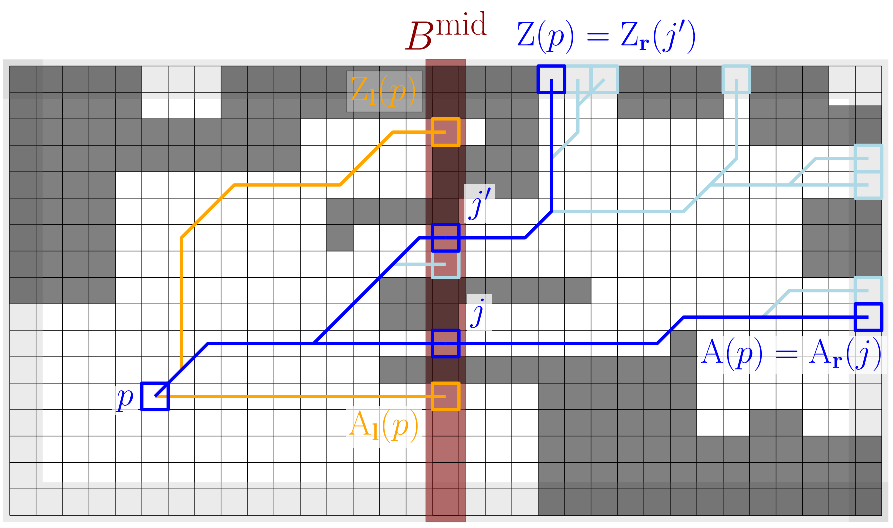

B^{mid}
$\mathrm{Z}(p) = \mathrm{Z}_{\mathbf{r}}(j')$
$\mathrm{Z}_{\mathbf{l}}(p)$
j'
j
$\mathrm{A}(p) = \mathrm{A}_{\mathbf{r}}(j)$
p
$\mathrm{A}_{\mathbf{l}}(p)$

Figure 5.6: Computation of $\mathcal{I}(p)$. To determine the smallest (largest) reachable index on $B^+ \cap B_{\mathbf{r}}$, we optimize, over all $j \in B^{\mathrm{mid}}$ with $p \rightsquigarrow j$, the smallest (largest) reachable index $\mathrm{A}_{\mathbf{r}}(j)$ ($\mathrm{Z}_{\mathbf{r}}(j)$) on $B^+_{\mathbf{r}}$. In this diagram, bright (dark) cells show free (non-free) positions. B^{mid} is the boundary shared by B_l (left of B^{mid}) and B_r (right of B^{mid}); see Figure 5.4 as a reminder of how we split boxes.

5.3.4 Computing Information at Parent From Information at Children

We show how to construct the information stored at the blocks quickly in a recursive fashion.

Lemma 5.16. *Let $B \in \mathcal{B}$ with children B_1, B_2. Given the information stored at B_1 and B_2, we can compute the information stored at B in time $\mathcal{O}((|\partial B| + |\mathcal{T}_B|) \log |\partial B|)$.*

Proof. Without loss of generality, we assume that B_1, B_2 are obtained from B by a vertical split (the other case is analogous) – let $B_{\mathbf{l}}, B_{\mathbf{r}}$ denote the left and right child, respectively. As a convention, we equip the information stored at $B_{\mathbf{l}}, B_{\mathbf{r}}$ with the subscript $\mathbf{l}, \mathbf{r}$, respectively, and write the information stored at B without subscript. Furthermore, we let $B^{\mathrm{mid}}_{\mathrm{free}}$ denote the set of *free* positions of the splitting boundary $B^{\mathrm{mid}} = B^+_{\mathbf{l}} \cap B^-_{\mathbf{r}}$.

Computation of $\mathcal{I}(p)$. Let $p \in B^- \cup \mathcal{T}_B$ be arbitrary. We first explain how to compute $\mathrm{A}(p)$ (see Figure 5.6 for an illustration). If $p \in B_{\mathbf{r}}$, then $\mathrm{A}(p) = \mathrm{A}_{\mathbf{r}}(p)$, since by monotonicity any $q \in B^+$ with $p \rightsquigarrow q$ satisfies $q \in B^+_{\mathbf{r}}$. Thus, it remains to consider $p \notin B_{\mathbf{r}}$.

We claim that for $p \notin B_{\mathbf{r}}$, we have $A(p) = \min\{A_1(p), A_2(p)\}$, where

$$A_1(p) := \min_{\substack{q \in B^+ \cap B_{\mathbf{l}},\\ p \rightsquigarrow q}} \mathrm{ind}(q)$$

$$A_2(p) := \min_{\substack{j \in B^{\mathrm{mid}}_{\mathrm{free}},\\ p \rightsquigarrow j}} \min_{\substack{q \in B^+ \cap B_{\mathbf{r}},\\ j \rightsquigarrow q}} \mathrm{ind}(q)$$

Indeed, this follows since each path starting in $p \in B_{\mathbf{l}}$ and ending in B^+ must end in $B_{\mathbf{l}}$, or cross B^{mid} at some free $j \in B^{\mathrm{mid}}$ and end in $B_{\mathbf{r}}$.

```
1: Build $\mathcal{OR}_\mathrm{A}$ storing $\mathrm{A}_\mathbf{r}(j)$ under the key $(\mathrm{ind}(j), \ell_\mathbf{l}(j))$ for all $j \in B^\mathrm{mid}_\mathrm{free}$ (for minimization queries)
2: Build $\mathcal{OR}_\mathrm{Z}$ storing $\mathrm{Z}_\mathbf{r}(j)$ under the key $(\mathrm{ind}(j), \ell_\mathbf{l}(j))$ for all $j \in B^\mathrm{mid}_\mathrm{free}$ (for maximization queries)
3: Build $\mathcal{OR}_\mathrm{top}$ storing $\mathrm{ind}(q)$ under the key $(\mathrm{ind}(q), \ell_\mathbf{l}(q))$ for all $q \in B^+ \cap B_\mathbf{l}$ (for both queries)
4: for $p \in (B^- \cup \mathcal{T}_B)$ do
5:     if $p \in B_\mathbf{r}$ then
6:         $\mathcal{I}(p) \leftarrow \mathcal{I}_\mathbf{r}(p)$
7:     else
8:         $\mathrm{A}_1(p) \leftarrow \mathcal{OR}_\mathrm{top}.\min([\mathrm{A}_\mathbf{l}(p), \mathrm{Z}_\mathbf{l}(p)] \times (-\infty, L(p)])$
9:         $\mathrm{A}_2(p) \leftarrow \mathcal{OR}_\mathrm{A}.\min([\mathrm{A}_\mathbf{l}(p), \mathrm{Z}_\mathbf{l}(p)] \times (-\infty, L(p)])$
10:        $\mathrm{A}(p) \leftarrow \min\{\mathrm{A}_1(p), \mathrm{A}_2(p)\}$

11:        $\mathrm{Z}_1(p) \leftarrow \mathcal{OR}_\mathrm{top}.\max([\mathrm{A}_\mathbf{l}(p), \mathrm{Z}_\mathbf{l}(p)] \times (-\infty, L(p)])$
12:        $\mathrm{Z}_2(p) \leftarrow \mathcal{OR}_\mathrm{Z}.\max([\mathrm{A}_\mathbf{l}(p), \mathrm{Z}_\mathbf{l}(p)] \times (-\infty, L(p)])$
13:        $\mathrm{Z}(p) \leftarrow \max\{\mathrm{Z}_1(p), \mathrm{Z}_2(p)\}$
```

Algorithm 9: Computing $\mathcal{I}(p) = [\mathrm{A}(p), \mathrm{Z}(p)]$ for all $p \in B^- \cup \mathcal{T}_B$.

To compute $A_1(p)$ note that Corollary 5.13 yields $A_1(p) = \min\{\mathrm{ind}(q) \mid q \in B^+ \cap B_\mathbf{l}, \mathrm{ind}(q) \in [\mathrm{A}_\mathbf{l}(p), \mathrm{Z}_\mathbf{l}(p)], \ell_\mathbf{l}(q) \le L(p)\}$, which can be expressed as an orthogonal range minimization query.

Likewise, to compute $A_2(p)$, note that $B^+ \cap B_\mathbf{r} = B^+_\mathbf{r}$. Thus,

$$\mathrm{A}_2(p) = \min_{\substack{j \in B^\mathrm{mid}_\mathrm{free},\\ p \rightsquigarrow j}} \min_{\substack{q \in B^+_\mathbf{r},\\ j \rightsquigarrow q}} \mathrm{ind}(q) = \min_{\substack{j \in B^\mathrm{mid}_\mathrm{free},\\ p \rightsquigarrow j}} \mathrm{A}_\mathbf{r}(j) = \min_{\substack{j \in B^\mathrm{mid}_\mathrm{free},\\ \mathrm{ind}(j) \in \mathcal{I}_\mathbf{l}(p), \ell_\mathbf{l}(j) \le L(p)}} \mathrm{A}_\mathbf{r}(j),$$

where the second and last equalities follow from the definition of $\mathrm{A}_\mathbf{r}$ and Corollary 5.13, respectively. It follows that we can compute $\mathrm{A}_2(p)$ using a simple orthogonal range minimization query.

Switching the roles of minimization and maximization, we obtain the analogous statements for computing $\mathrm{Z}(p)$. We summarize the resulting algorithm for computing the reachability intervals $\mathcal{I}(p)$ for all $p \in B^- \cup \mathcal{T}_B$ formally in Algorithm 9. Its correctness follows from the arguments above.

Let us analyze the running time of Algorithm 9: Observe that $|B^\mathrm{mid}_\mathrm{free}| \le |\partial B|$. Thus, we can construct the orthogonal range data structures $\mathcal{OR}_\mathrm{A}$, $\mathcal{OR}_\mathrm{Z}$, and $\mathcal{OR}_\mathrm{top}$ in time $\mathcal{O}(|\partial B| \log |\partial B|)$ (see Section 5.1.2). For each $p \in B^- \cup \mathcal{T}_B$, we perform at most a constant number of two-dimensional orthogonal range minimization/maximization queries, which takes time $\mathcal{O}(\log |\partial B|)$, followed by constant-time computation. The total running time amounts to $\mathcal{O}((|\partial B| + |\mathcal{T}_B|) \log |\partial B|)$.

Computation of $\boldsymbol{\ell(q)}$. Let $q \in B^+$ be arbitrary. If $q \in B_\mathbf{l}$, then $\ell(q) = \ell_\mathbf{l}(p)$, since by monotonicity every $p \in B$ with $p \rightsquigarrow q$ is contained in $B_\mathbf{l}$. Thus, we may assume that $q \notin B_\mathbf{l}$.

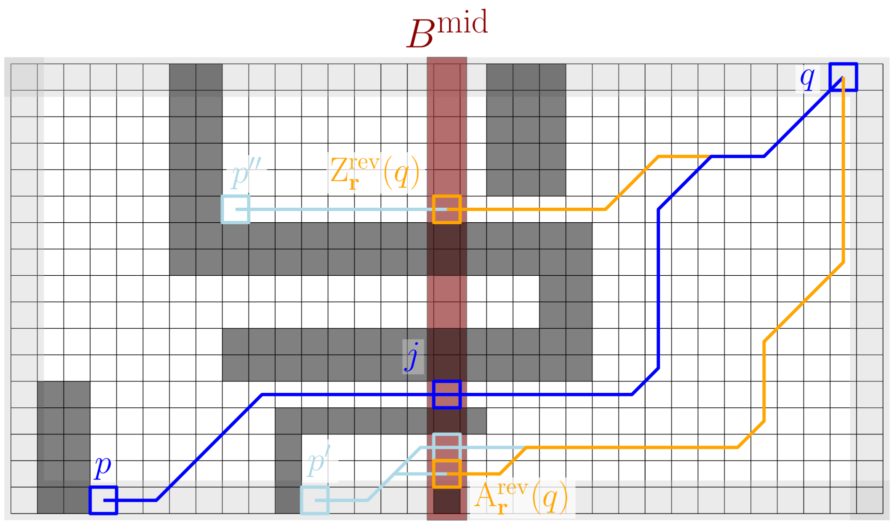

B^{mid}
q
p''
$Z_{\mathbf{r}}^{\mathrm{rev}}(q)$
j
p
p'
$A_{\mathbf{r}}^{\mathrm{rev}}(q)$

Figure 5.7: Computation of $\ell(q)$. To determine the smallest label of a position in $B_{\mathbf{l}}$ reaching q, we optimize, over all $j \in B^{\mathrm{mid}}$ with $j \rightsquigarrow q$, the smallest label $\ell_{\mathbf{l}}(j)$ of a position $p \in B_{\mathbf{l}}$ reaching j.

We claim that for $q \notin B_{\mathbf{l}}$, we have $\ell(q) = \min\{\ell_1(q), \ell_2(q)\}$, where

$$\ell_1(q) \coloneqq \min_{\substack{p \in B_{\mathbf{r}},\\ p \rightsquigarrow q}} L(p)$$

$$\ell_2(q) \coloneqq \min_{\substack{j \in B^{\mathrm{mid}}_{\mathrm{free}},\\ j \rightsquigarrow q}} \min_{\substack{p \in B_{\mathbf{l}},\\ p \rightsquigarrow j}} L(p)$$

Indeed, this follows since each path starting in B and ending in $q \in B_{\mathbf{r}}$ must start in $B_{\mathbf{r}}$, or start in $B_{\mathbf{l}}$ and cross B^{mid} at some free $j \in B^{\mathrm{mid}}$.

Observe that the definition of $\ell_1(q)$ coincides with the definition of $\ell_{\mathbf{r}}(q)$. Thus it only remains to compute $\ell_2(q)$. We write

$$\ell_2(q) = \min_{\substack{j \in B^{\mathrm{mid}}_{\mathrm{free}},\\ j \rightsquigarrow q}} \min_{\substack{p \in B_{\mathbf{l}},\\ p \rightsquigarrow j}} L(p) = \min_{\substack{j \in B^{\mathrm{mid}}_{\mathrm{free}},\\ j \rightsquigarrow q}} \ell_{\mathbf{l}}(j) = \min_{\substack{j \in B^{\mathrm{mid}}_{\mathrm{free}},\\ \mathrm{ind}(j) \in \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(q), \ell^{\mathrm{rev}}_{\mathbf{r}}(j) \le L^{\mathrm{rev}}(q)}} \ell_{\mathbf{l}}(j),$$

where the second and last equalities follow from the definition of $\ell_{\mathbf{l}}(j)$ and Corollary 5.14, respectively. It follows that we can compute $\ell_2(p)$ using a simple orthogonal range minimization query. For an illustration of $\ell_2(q)$, we refer to Figure 5.7.

We summarize the resulting algorithm for computing the reachability levels $\ell(q)$ for all $q \in B^+$ formally in Algorithm 10. Its correctness follows from the arguments above.

To analyze the running time of Algorithm 10, observe that $|B^{\mathrm{mid}}_{\mathrm{free}}| \le |\partial B|$. Thus, we can construct $\mathcal{OR}_\ell$ in time $\mathcal{O}(|\partial B| \log |\partial B|)$ (see Section 5.1.2). For each $q \in B^+$, we then perform at most one minimization query to $\mathcal{OR}_\ell$ in time $\mathcal{O}(\log |\partial B|)$, followed by a constant-time computation. Thus, the total running time amounts to $\mathcal{O}(|\partial B| \log |\partial B|)$.

Computation of reverse information. Switching the direction of reach traversals (which switches roles of inputs and outputs, $B_{\mathbf{l}}$ and $B_{\mathbf{r}}$, etc.) as well as L and L^{rev}, we

```
1: Build $\mathcal{OR}_\ell$ storing $\ell_{\mathbf{l}}(j)$ under the key $(\mathrm{ind}(j), \ell_{\mathbf{r}}^{\mathrm{rev}}(j))$ for all $j \in B_{\mathrm{free}}^{\mathrm{mid}}$ (for minimiza-
   tion queries)
2: for $q \in B^+$ do
3:     if $q \in B_{\mathbf{l}}$ then
4:         $\ell(q) \leftarrow \ell_{\mathbf{l}}(q)$
5:     else
6:         $\ell_2(q) \leftarrow \mathcal{OR}_\ell.\min([\mathrm{A}_{\mathbf{r}}^{\mathrm{rev}}(q), \mathrm{Z}_{\mathbf{r}}^{\mathrm{rev}}(q)] \times (-\infty, L^{\mathrm{rev}}(q)])$
7:         $\ell(q) \leftarrow \min\{\ell_{\mathbf{r}}(q), \ell_2(q)\}$
```

Algorithm 10: Computing $\ell(q)$ for all $q \in B^+$.

can use the same algorithms to compute the reverse reachability information in the same running time of $\mathcal{O}((|\partial B| + |\mathcal{T}_B|)\log|\partial B|)$.

Computation of $\mathcal{OR}_B$. Finally, we need to construct the two-dimensional orthogonal range minimization data structure $\mathcal{OR}_B$: Recall that $\mathcal{OR}_B$ stores, for each $q \in B_{\mathrm{free}}^{\mathrm{mid}}$, the value $\ell_{\mathbf{r}}^{\mathrm{rev}}(q)$ under the key $(\mathrm{ind}(q), \ell_{\mathbf{l}}(q))$ for minimization queries. Since $|B_{\mathrm{free}}^{\mathrm{mid}}| \leq |\partial B|$, this can be done in time $\mathcal{O}(|\partial B|\log|\partial B|)$ (cf. Section 5.1.2).

Summary. In summary, we can compute the information stored at B (according to Definition 5.15) from the information stored at B_1 and B_2 in time $\mathcal{O}((|\partial B| + |\mathcal{T}_B|)\log|\partial B|)$, as desired. □

5.3.5 Initialization and Updates

We show how to construct our reachability data structure (using Lemma 5.16 that shows how to compute the information stored at some canonical block B given the information stored at both children). Specifically, the following lemma proves i) of Lemma 5.3.

Lemma 5.17. *We can construct $\mathcal{D}_{M,\mathcal{T}}$ in time $\mathcal{O}(n^2 + |\mathcal{T}|\log^2 n)$.*

Proof. We use the obvious recursive algorithm to build $\mathcal{D}_{M,\mathcal{T}}$ in a bottom-up fashion using Lemma 5.16. Recall that $n = 2^\kappa + 1$ for some $\kappa \in \mathbb{N}$. Note that for the blocks $B \in \mathcal{B}_{2\kappa}$ in the lowest level, we can compute the information stored in B in constant time, which takes time $\mathcal{O}(|\mathcal{B}_{2\kappa}|) = \mathcal{O}(n^2)$ in total.

It remains to bound the running time to compute $\mathcal{D}_{M,\mathcal{T}}(B)$ for $B \in \mathcal{B}_\ell$ for $0 \leq \ell < 2\kappa$. Observe that this running time is bounded by $\mathcal{O}(\sum_{\ell=0}^{2\kappa-1}\sum_{B\in\mathcal{B}_\ell} c_B)$ by Lemma 5.16, where $c_B := |\partial B|\log|\partial B| + |\mathcal{T}_B|\log|\partial B|$.

Let $0 \leq \ell < 2\kappa$. By construction, we have $|\mathcal{B}_\ell| = 2^\ell$. Furthermore, for any $B \in \mathcal{B}_\ell$, observe that its side lengths are bounded by $2^{\kappa - \lfloor \ell/2 \rfloor} + 1$, and thus $|\partial B| \leq 4 \cdot 2^{\kappa - \lfloor \ell/2 \rfloor} \leq 2^{\kappa - \ell/2 + 3}$. Hence, we may compute

$$\begin{aligned}
\sum_{\ell=0}^{2\kappa-1} \sum_{B \in \mathcal{B}_\ell} |\partial B| \log |\partial B| &\leq \sum_{\ell=0}^{2\kappa-1} |\mathcal{B}_\ell| 2^{\kappa - \ell/2 + 3}(\kappa - \ell/2 + 3) \\
&= \sum_{\ell=0}^{2\kappa-1} 2^{\kappa + \ell/2 + 3}(\kappa - \ell/2 + 3) \qquad (5.1) \\
&\leq 2 \left(\sum_{i=0}^{\kappa} 2^{\kappa + i + 3}(\kappa - i + 3) \right) \\
&= 2(2^{2(\kappa+3)} - 2^{\kappa+3}(\kappa + 5)) = \mathcal{O}(2^{2\kappa}) = \mathcal{O}(n^2).
\end{aligned}$$

Furthermore, we have

$$\sum_{\ell=0}^{2\kappa-1} \sum_{B \in \mathcal{B}_\ell} |\mathcal{T}_B| \log |\partial B| \leq \sum_{\ell=0}^{2\kappa-1} 4|\mathcal{T}|(\kappa - \ell/2 + 3) = \mathcal{O}(|\mathcal{T}|\kappa^2) = \mathcal{O}(|\mathcal{T}| \log^2 n),$$

where we used that $\sum_{B \in \mathcal{B}_\ell} |\mathcal{T}_B| \leq 4|\mathcal{T}|$ (as any position in $[n] \times [n]$ is shared by at most 4 blocks at the same level). In total, we obtain a running time bound of $\mathcal{O}(n^2 + |\mathcal{T}| \log^2 n)$, as desired. $\square$

With very similar arguments, we can prove iii) of Lemma 5.3.

Lemma 5.18. *Let M, M' be any $n \times n$ 0-1-matrices differing in at most k positions and $\mathcal{T}, \mathcal{T}' \subseteq [n] \times [n]$ be any sets of terminals of size k. Given the data structure $\mathcal{D}_{M,\mathcal{T}}$, the set $\mathcal{T}'$, as well as the set Δ of positions in which M and M' differ, we can update $\mathcal{D}_{M,\mathcal{T}}$ to $\mathcal{D}_{M',\mathcal{T}'}$ in time $\mathcal{O}(n\sqrt{k} \log n + k \log^2 n)$.*

Proof. Set $X := \Delta \cup \mathcal{T} \cup \mathcal{T}'$ and note that $|X| = \mathcal{O}(k)$. Observe that for any $B \in \mathcal{B}$ with $B \cap X = \emptyset$, we have $\mathcal{D}_{M,\mathcal{T}}(B) = \mathcal{D}_{M',\mathcal{T}'}(B)$, since the information stored at this block does not depend on any changed entry in M and does not contain any of the old or new terminals. Thus, we only need to update $\mathcal{D}_{M,\mathcal{T}}(B)$ to $\mathcal{D}_{M',\mathcal{T}'}(B)$ for all $B \in \mathcal{B}$ with $B \cap X \neq \emptyset$. We do this by computing the information for these blocks in a bottom-up fashion analogously to Lemma 5.17. Specifically, for any lowest-level block $B \in \mathcal{B}_{2\kappa}$ with $B \cap X \neq \emptyset$, we can compute the information stored in B in constant time. Since there are at most $4|X|$ such blocks, this step takes time $\mathcal{O}(|X|) = \mathcal{O}(k)$ in total.

It remains to bound the running time to compute $\mathcal{D}_{M,\mathcal{T}}(B)$ for $B \in \mathcal{B}_\ell$ with $B \cap X \neq \emptyset$, where $0 \leq \ell < 2\kappa$. For any such B, we let again $c_B := |\partial B| \log |\partial B| + |\mathcal{T}_B| \log |\partial B|$. Observe that the running time for the remaining task is thus bounded by $\mathcal{O}(\sum_{\ell=0}^{2\kappa-1} \sum_{B \in \mathcal{B}_\ell, B \cap X \neq \emptyset} c_B)$ by Lemma 5.16.

We do a case distinction into $0 \le \ell < \bar{\ell}$ and $\bar{\ell} \le \ell < 2\kappa$ where $\bar{\ell} := \lfloor \log k \rfloor$. For the first case, we bound

$$\begin{aligned}
\sum_{\ell=0}^{\bar{\ell}-1} \sum_{\substack{B\in\mathcal{B}_\ell,\\ B\cap X\neq\emptyset}} |\partial B| \log |\partial B| &\le \sum_{\ell=0}^{\bar{\ell}-1} \sum_{B\in\mathcal{B}_\ell} |\partial B| \log |\partial B| \\
&\le \sum_{i=0}^{\bar{\ell}-1} 2^{\kappa+i/2+3}(\kappa - i/2 + 3) \\
&\le \left(\sum_{i=0}^{\bar{\ell}-1} 2^{i/2}\right) 2^{\kappa+3}\kappa \\
&= (1+\sqrt{2})(2^{\bar{\ell}/2}-1)2^{\kappa+3}\kappa = \mathcal{O}(\sqrt{k}n\log n),
\end{aligned}$$

where the second inequality is derived as in (5.1). Recall that for any $0 \le \ell < 2\kappa$, there are at most $4|X|$ blocks $B \in \mathcal{B}_\ell$ with $B \cap X \neq \emptyset$ and for any $B \in \mathcal{B}_\ell$, we have $|\partial B| \le 2^{\kappa-\ell/2+3}$. We compute

$$\begin{aligned}
\sum_{\ell=\bar{\ell}}^{2\kappa-1} \sum_{\substack{B\in\mathcal{B}_\ell,\\ B\cap X\neq\emptyset}} |\partial B| \log |\partial B| &\le \sum_{\ell=\bar{\ell}}^{2\kappa-1} 4|X|2^{\kappa-\ell/2+3}(\kappa - \ell/2 + 3) \\
&\le 4|X|2^{\kappa-\bar{\ell}/2+3}(\kappa+3) \cdot \sum_{\ell=0}^{2\kappa-\bar{\ell}-1} 2^{-\ell/2} \\
&= \mathcal{O}\left(|X|\frac{n}{\sqrt{k}}\log n\right) = \mathcal{O}(\sqrt{k}n\log n).
\end{aligned}$$

Furthermore, as in the proof of Lemma 5.17, we again compute

$$\sum_{\ell=0}^{2\kappa-1} \sum_{\substack{B\in\mathcal{B}_\ell,\\ B\cap X\neq\emptyset}} |\mathcal{T}_B| \log |\partial B| \le \sum_{\ell=0}^{2\kappa-1} 4|\mathcal{T}|(\kappa - \ell/2 + 3) = \mathcal{O}(|\mathcal{T}|\kappa^2) = \mathcal{O}(|\mathcal{T}|\log^2 n).$$

Thus, in total we obtain a running time of $\mathcal{O}(k + n\sqrt{k}\log n + |\mathcal{T}|\log^2 n) = \mathcal{O}(n\sqrt{k}\log n + k\log^2 n)$. ☐

5.3.6 Reachability Queries

It remains to show how to use the information stored at all canonical blocks to answer reachability queries quickly. Specifically, the following lemma proves ii) of Lemma 5.3.

Lemma 5.19. *Given $\mathcal{D}_{M,\mathcal{T}} = (\mathcal{D}_{M,\mathcal{T}}(B))_{B\in\mathcal{B}}$, we can answer reachability queries for $F \subseteq \mathcal{T}$ in time $\mathcal{O}(|\mathcal{T}|\log^3 n)$.*

Recall that we aim to determine whether there is a monotone path in M using only positions (i,j) with $M_{i,j} = 1$ or $(i,j) \in F$, i.e., we view F as a set of *free terminals* (typically, $(i,j) \in F$ is a *non-free* position). In this section we assume, without loss of generality, that $(1,1),(n,n) \in \mathcal{T}_B$ (whenever we construct/update to the data structure $\mathcal{D}_{M,\mathcal{T}}$, we may construct/update to $\mathcal{D}_{M,\mathcal{T}\cup\{(1,1),(n,n)\}}$ in the same asymptotic running time).

For any block $B \in \mathcal{B}$, $S \subseteq F \subseteq \mathcal{T}_B$, we define the function $\text{Reach}(B,S,F)$ that returns the set

$$R := \{t \in F \mid \exists f_1,\dots,f_\ell \in F : f_1 \in S, f_\ell = t, f_1 \rightsquigarrow f_2 \rightsquigarrow \cdots \rightsquigarrow f_\ell\},$$

i.e., we interpret S as a set of admissible *starting positions* for a reach traversal and ask for the set of positions reachable from S using only free positions or free terminals. We call any such position *F-reachable from S*. (Recall that in the definition of $p \rightsquigarrow q$, only the intermediate points on a reach traversal from p and q are required to be free, while the endpoints p and q are allowed to be non-free.)

We show that $\text{Reach}(B,S,F)$ can be computed in time $\mathcal{O}(|\mathcal{T}_B|\log^3 n)$. Given this, we can answer any reachability query in the same asymptotic running time: the reachability query asks whether there is a sequence $f_1,\dots,f_\ell \in F \cup \{(1,1),(n,n)\}$ such that (i) $f_1 = (1,1)$ and $f_\ell = (n,n)$, (ii) both $(1,1)$ and (n,n) are free positions or contained in F and (iii) $f_1 \rightsquigarrow f_2 \rightsquigarrow \cdots \rightsquigarrow f_\ell$. Since (ii) can be checked in constant time, it remains to determine whether

$$(n,n) \in \text{Reach}([n]\times[n],\ \{(1,1)\},\ F \cup \{(1,1),(n,n)\}).$$

Computation of Reach(B, S, F)

To compute $\text{Reach}(B,S,F)$, we work on the recursive block structure of $\mathcal{D}_{M,\mathcal{T}}$. Specifically, consider any canonical block $B \in \mathcal{B}$ (containing some free terminal) with children B_1, B_2. The (somewhat simplified) approach is the following: We first (recursively) determine all free terminals that are F-reachable from S in B_1 and call this set R_1. Then, we determine all free terminals in B_2 that are (directly) reachable from R_1 and call this set T_2. Finally, we (recursively) determine all free terminals in B_2 that are F-reachable from $T_2 \cup (S \cap B_2)$ and call this set R_2. The desired set of free terminals that are F-reachable from S is then $R_1 \cup R_2$. The main challenge in this process is the computation of the set T_2; this task is solved by the following lemma.

Lemma 5.20. *Let $B \in \mathcal{B}$ be a block with children B_1, B_2. Given $S \subseteq B_1 \setminus B^{\text{mid}}$ and $F \subseteq B_2 \setminus B^{\text{mid}}$ with $S, F \subseteq \mathcal{T}_B$, we can compute the set*

$$T = \{t \in F \mid \exists s \in S : s \rightsquigarrow t\}$$

in time $\mathcal{O}(|\mathcal{T}_B|\log^2 n)$. We call this procedure $\text{SingleStepReach}(B,S,F)$.

We postpone the proof of this lemma to Section 5.3.6 and first show how this yields an algorithm for Reach, and thus, for reachability queries.

Proof of Lemma 5.19. We claim that Algorithm 11 computes R in time $\mathcal{O}(|\mathcal{T}|\log^3 n)$.

To ease the analysis, we introduce the shorthand that $s \rightsquigarrow_F t$ if and only if there are $f_1,\dots,f_\ell \in F$ with $f_1 = s, f_\ell = t$ and $f_1 \rightsquigarrow f_2 \rightsquigarrow \cdots \rightsquigarrow f_\ell$, i.e., t is F-reachable from s.

```
1: function Reach($B, S, F$)
2:     if $F = \emptyset$ then
3:         return $\emptyset$
4:     else if $B$ is a $2 \times 2$ block then
5:         Compute $R$ by checking all possibilities
6:         return $R$
7:     else                                  ▷ $B$ splits into child blocks $B_1, B_2$
8:         $S_1 \leftarrow S \cap B_1$, $S_2 \leftarrow S \cap B_2$
9:         $R_1 \leftarrow$ Reach($B_1, S_1, F \cap B_1$)
10:        $T_2 \leftarrow$ SingleStepReach($B, R_1 \setminus B^{\mathrm{mid}}, F \setminus B_1$)
11:        $R_2 \leftarrow$ Reach($B_2, S_2 \cup T_2 \cup (R_1 \cap B^{\mathrm{mid}}), F \cap B_2$)
12:        return $R_1 \cup R_2$
```

Algorithm 11: Computing Reach(B, S, F) for $B \in \mathcal{B}$, $S \subseteq F \subseteq \mathcal{T}_B$.

We show that Algorithm 11 computes $R = \{t \in F \mid \exists s \in S, s \rightsquigarrow_F t\}$ inductively: The base case for 2×2 blocks B holds trivially. Otherwise, by inductive assumption, we have

$$R_1 = \{t \in F \cap B_1 \mid \exists s \in S \cap B_1, s \rightsquigarrow_{F \cap B_1} t\}.$$

Note that by definition of SingleStepReach, we furthermore have

$$T_2 = \{t \in F \setminus B_1 \mid \exists s \in R_1 \setminus B^{\mathrm{mid}}, s \rightsquigarrow t\}.$$

Finally, by inductive assumption,

$$\begin{aligned} R_2 = &\{t \in F \cap B_2 \mid \exists s \in S \cap B_2, s \rightsquigarrow_{F \cap B_2} t\} \cup \\ &\{t \in F \cap B_2 \mid \exists s \in T_2, s \rightsquigarrow_{F \cap B_2} t\} \cup \\ &\{t \in F \cap B_2 \mid \exists s \in R_1 \cap B^{\mathrm{mid}}, s \rightsquigarrow_{F \cap B_2} t\}. \end{aligned}$$

First, we show that any $t \in R_1 \cup R_2$ is contained in R: If $t \in R_1$, then there is some $s \in S \cap B_1 \subseteq S$ with $s \rightsquigarrow_{F \cap B_1} t$ (trivially implying $s \rightsquigarrow_F t$), and thus $t \in R$. Likewise, if $t \in T_2$, then there is some $t' \in R_1 \setminus B^{\mathrm{mid}}$ with $t' \rightsquigarrow t$. Since $t' \in R_1$, there must exist some $s \in S$ with $s \rightsquigarrow_F t'$. Thus $s \rightsquigarrow_F t'$ and $t' \rightsquigarrow t$ yields $s \rightsquigarrow_F t$ and $t \in R$. Finally, if $t \in R_2$, there exists some t' with $t' \rightsquigarrow_{F \cap B_2} t$ and either $t' \in S \cap B_2$, $t' \in T_2$, or $t' \in R_1 \cap B^{\mathrm{mid}}$. In all these cases, there is some $s \in S$ with $s \rightsquigarrow_F t'$. Hence $s \rightsquigarrow_F t'$ and $t' \rightsquigarrow_{F \cap B_2} t$ imply $s \rightsquigarrow_F t$, placing t in R.

We proceed to show the converse direction that any $t \in R$ is contained in $R_1 \cup R_2$: Let $s \in S$ with $s \rightsquigarrow_F t$. If $t \in F \cap B_1$, then $s \rightsquigarrow_F t$ is equivalent to $s \rightsquigarrow_{F \cap B_1} t$ and $s \in S \cap B_1$ (by monotonicity). Thus, $t \in R_1$. It only remains to consider the case that $t \in F \setminus B_1$. If $s \in S \cap B_2$, then again my monotonicity $s \rightsquigarrow_{F \cap B_2} t$ must hold, which implies $t \in R_2$. Otherwise, we have $s \in S \setminus B_2$. Since additionally $t \in F \setminus B_1$, there must exist either (1) some $r \in F \cap B^{\mathrm{mid}}$ with $s \rightsquigarrow_{F \cap B_1} r$ and $r \rightsquigarrow_{F \cap B_2} t$ or (2) some $t' \in F \setminus B_2$, $t'' \in F \setminus B_1$ with $s \rightsquigarrow_{F \cap B_1} t' \rightsquigarrow t'' \rightsquigarrow_{F \cap B_2} t$ (by monotonicity). For (1), note that $r \in R_1$ (as shown above), and thus $t \in R_2$. For (2), note that $t' \in R_1 \setminus B_2 = R_1 \setminus B^{\mathrm{mid}}$ (as $s \in S \cap B_1, t' \in F \setminus B_2$ and $s \rightsquigarrow_{F \cap B_1} t'$), $t'' \in T_2$ (as $t' \in R_1 \setminus B^{\mathrm{mid}}$, $t'' \in F \setminus B_1$ and $t' \rightsquigarrow t''$) and finally $t \in R_2$ (as $t'' \in T_2$ and $t'' \rightsquigarrow_{F \cap B_2} t$), as desired.

We analyze the running time of a call Reach(B_0, S_0, F_0). Let $T(B) = \mathcal{O}(|\mathcal{T}_B| \log^2 n)$ denote the running time of SingleStepReach(B, S, F) for arbitrary S, F. Observe that the running time of Reach(B_0, S_0, F_0) is bounded by

$$\sum_{\substack{B \in \mathcal{B}, \\ \mathcal{T}_B \neq \emptyset}} \mathcal{O}(T(B)), \tag{5.2}$$

as for $\mathcal{T}_B = \emptyset$, we have $F \subseteq \mathcal{T}_B = \emptyset$, which is a base case of Reach$(\cdot)$. To bound the above term, fix any ℓ, and note that for any $t \in \mathcal{T}$, there are at most 4 level-ℓ blocks $B \in \mathcal{B}_\ell$ with $t \in \mathcal{T}_B$ (if t is on the boundary of some block $B \in \mathcal{B}_\ell$, it is shared between different blocks; however, any position is shared by at most 4 blocks). Thus $\sum_{B \in \mathcal{B}_\ell} |\mathcal{T}_B| \leq 4|\mathcal{T}|$. Thus, (5.2) is bounded by

$$\sum_{\ell=0}^{2\log(n-1)} \sum_{\substack{B \in \mathcal{B}_\ell, \\ \mathcal{T}_B \neq \emptyset}} \mathcal{O}(|\mathcal{T}_B| \log^2 n) = \sum_{\ell=0}^{2\log(n-1)} \mathcal{O}(|\mathcal{T}| \log^2 n) = \mathcal{O}(|\mathcal{T}| \log^3 n).$$

$\square$

Computation of SingleStepReach(B, S, F)

It remains to prove Lemma 5.20 to conclude the proof of Lemma 5.19.

Proof of Lemma 5.20. Consider $B \in \mathcal{B}$. We only consider the case that B is split vertically (the other case is symmetric); let $B_\mathbf{l}, B_\mathbf{r}$ denote its left and right sibling, respectively. Let $S \subseteq B_\mathbf{l} \setminus B^{\text{mid}}$, $F \subseteq B_\mathbf{r} \setminus B^{\text{mid}}$ with $S, F \subseteq \mathcal{T}_B$ be arbitrary. We use notation (subscripts $\mathbf{l}, \mathbf{r}$, etc.) as in the proof of Lemma 5.16.

Observe that for any $s \in S, f \in F$, we have that $s \rightsquigarrow f$ if and only if there exists some $j \in B^{\text{mid}}_{\text{free}}$ with $s \rightsquigarrow j$ and $j \rightsquigarrow f$. To introduce some convenient conventions, let $J^{\text{mid}} = \{j_1, \dots, j_N\}$, where $j_1, \dots, j_N$ is the sorted sequence of ind(q) with $q \in B^{\text{mid}}_{\text{free}}$. We call $J \subseteq J^{\text{mid}}$ an interval of J^{mid} if $J = \{j_a, j_{a+1}, \dots, j_b\}$ for some $1 \leq a \leq b \leq N$ and write it as $J = [j_a, j_b]_{J^{\text{mid}}}$ (i.e., $[j_a, j_b]_{J^{\text{mid}}}$ simply disregards any indices in $[j_a, j_b]$ representing positions not in $B^{\text{mid}}_{\text{free}}$).

Consider any interval J of J^{mid} with the property that for all $s \in S$ we either have $J \cap \mathcal{I}_\mathbf{l}(s) = J$ or $J \cap \mathcal{I}_\mathbf{l}(s) = \emptyset$ and for all $f \in F$ we either have $J \cap \mathcal{I}^{\text{rev}}_\mathbf{r}(f) = J$ or $J \cap \mathcal{I}^{\text{rev}}_\mathbf{r}(f) = \emptyset$. We call such a J an (S, F)-*reach-equivalent* interval. Note that by splitting J^{mid} right before and right after all points $\mathrm{A}(s), \mathrm{Z}(s)$ with $s \in S$ and $\mathrm{A}^{\text{rev}}(f), \mathrm{Z}^{\text{rev}}(f)$ with $f \in F$, we obtain a partition of J^{mid} into (S, F)-reach-equivalent intervals $J_1, \dots, J_\ell$ with $\ell = \mathcal{O}(|S \cup F|) = \mathcal{O}(|\mathcal{T}_B|)$.[7]

Claim 5.21. *Let J be an (S, F)-reach-equivalent interval J. Let R^J be the set of $t \in F$ reachable from S via J, i.e., $R^J := \{t \in F \mid \exists s \in S, j \in J : s \rightsquigarrow j \rightsquigarrow t\}$. Define*

$$\ell^J := \min_{\substack{j \in J, \\ \exists s \in S: s \rightsquigarrow j}} \ell^{\text{rev}}_\mathbf{r}(j).$$

[7] To be more precise, we start with the partition $\mathcal{J}$ consisting of the singleton J^{mid}. We then iterate over any point j among $\mathrm{A}(s), \mathrm{Z}(s), s \in S$ and $\mathrm{A}^{\text{rev}}(f), \mathrm{Z}^{\text{rev}}(f), f \in F$, and replace the interval $J = [j_a, j_b]_{J^{\text{mid}}} \in \mathcal{J}$ containing j by the three intervals $[j_a, j)_{J^{\text{mid}}}, \{j\}, (j, j_b]_{J^{\text{mid}}}$, where the first and the last interval may be empty.

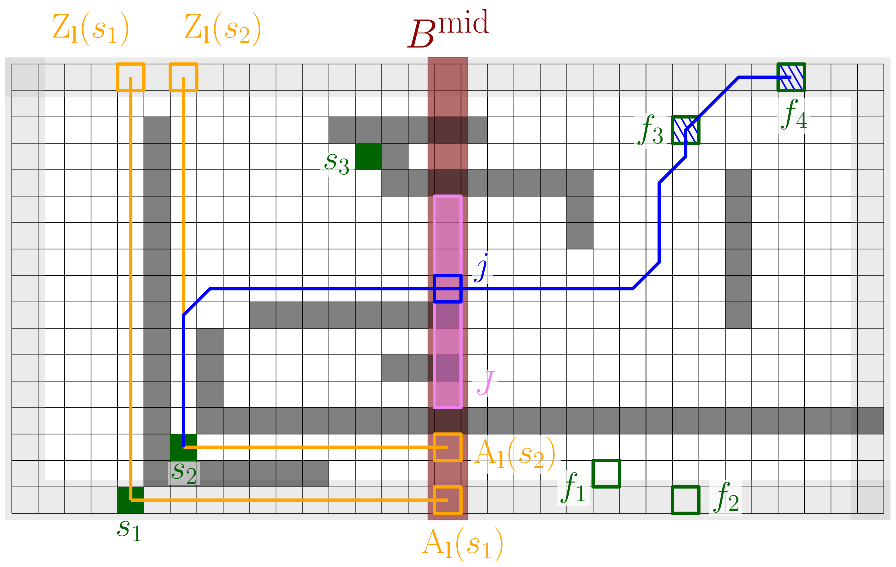


Figure 5.8: Computation of R^J for an (S, F)-reach equivalent interval J. Intuitively, we first determine, among indices in J reachable from some $s \in S$, the index $j \in J$ with the best reachability towards F. We then determine all $f \in F$ reachable from j (indicated by hatched boxes).

We have

$$R^J = \{t \in F \mid J \subseteq \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t), \ell^J \leq L^{\mathrm{rev}}(t)\}. \tag{5.3}$$

Proof. See Figure 5.8 for an illustration. Indeed, for any $t \in F$ with $J \subseteq \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t)$ and $\ell^J \leq L^{\mathrm{rev}}(t)$, consider any $j \in J$ with $\ell^{\mathrm{rev}}_{\mathbf{r}}(j) = \ell^J$ and $s \rightsquigarrow j$ for some $s \in S$. Then we have $j \in J \subseteq \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t)$ and $\ell^{\mathrm{rev}}_{\mathbf{r}}(j) = \ell^J \leq L^{\mathrm{rev}}(t)$. Thus by Corollary 5.14, $j \rightsquigarrow t$, which together with $s \rightsquigarrow j$ implies $s \rightsquigarrow j \rightsquigarrow t$, as desired. For the converse, let $t \in F$ with $s \rightsquigarrow j \rightsquigarrow t$ for some $s \in S, j \in J$. Then by definition of ℓ^J, we obtain $\ell^J \leq \ell^{\mathrm{rev}}_{\mathbf{r}}(j)$. Furthermore, by Corollary 5.14, $j \rightsquigarrow t$ implies that $j \in \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t)$ with $L^{\mathrm{rev}}(t) \geq \ell^{\mathrm{rev}}_{\mathbf{r}}(j) \geq \ell^J$. Note that $j \in \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t)$ implies $J \subseteq \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t)$ (as J is (S, F)-reach-equivalent), thus we obtain that $J \subseteq \mathcal{I}^{\mathrm{rev}}_{\mathbf{r}}(t)$ and $\ell^J \leq \ell^{\mathrm{rev}}_{\mathbf{r}}(j)$, as desired. □

Thus, after computing ℓ^J, an orthogonal range reporting query can be used to report all $t \in F$ reachable from S via J. To compute ℓ^J, we observe that for any $j \in J$, we have

$$\begin{aligned} \exists s \in S : s \rightsquigarrow j \quad &\overset{\text{Cor. 5.13}}{\iff} \quad \exists s \in S : j \in \mathcal{I}_{\mathbf{l}}(s), \ell_{\mathbf{l}}(j) \leq L(s) \\ &\iff \quad \ell_{\mathbf{l}}(j) \leq \max_{\substack{s \in S,\\ j \in \mathcal{I}_{\mathbf{l}}(s)}} L(s) =: L^j. \end{aligned}$$

Noting (by (S, F)-reach-equivalence of J) that $j \in \mathcal{I}_{\mathbf{l}}(s)$ if and only if $J \subseteq \mathcal{I}_{\mathbf{l}}(s)$, we have that L^j is independent of $j \in J$, and, in particular, equal to

$$L^J := \max_{\substack{s \in S,\\ J \subseteq \mathcal{I}_{\mathbf{l}}(s)}} L(s), \tag{5.4}$$

```
1: function SingleStepReach($B, S, F$)
2:     Compute a partition of $J^{\mathrm{mid}}$ into $(S,F)$-reach-equivalent intervals $J_1, \dots, J_\ell$
3:     Build $\mathcal{OR}_S$ storing $L(s)$ under the key $(\mathrm{A}_{\mathbf{l}}(s), \mathrm{Z}_{\mathbf{l}}(s))$ for all $s \in S$ (for maximization queries)
4:     Build $\mathcal{OR}_F$ storing $\mathrm{ind}(f)$ under the key $(\mathrm{A}^{\mathrm{rev}}_{\mathbf{r}}(f), \mathrm{Z}^{\mathrm{rev}}_{\mathbf{r}}(f), L^{\mathrm{rev}}(f))$ for all $f \in F$
                                                        (for decremental range reporting queries)
5:     Recall: (precomputed) $\mathcal{OR}_B$ stores $\ell_2^{\mathrm{rev}}(q)$ under the key $(\mathrm{ind}(q), \ell_1(q))$ for all $q \in B^{\mathrm{mid}}_{\mathrm{free}}$
                                                        (for minimization queries)
6:     for $i = 1, \dots, \ell$ do                                   ▷ consider $J = [a_i, b_i]_{J^{\mathrm{mid}}}$
7:         $L^J \leftarrow \mathcal{OR}_S.\max((-\infty, a_i] \times [b_i, \infty))$
8:         $\ell^J \leftarrow \mathcal{OR}_B.\min([a_i, b_i] \times (-\infty, L^J])$
9:         $R_i \leftarrow \mathcal{OR}_F.\mathrm{report}((-\infty, a_i] \times [b_i, \infty) \times [\ell^J, \infty))$
10:        $\mathcal{OR}_F.\mathrm{delete}(R_i)$
11:    return $\bigcup_{i=1}^{\ell} R_i$
```

Algorithm 12: Computing SingleStepReach(B, S, F) for $B \in \mathcal{B}$, $S \subseteq B_{\mathbf{l}} \setminus B^{\mathrm{mid}}, F \subseteq B_{\mathbf{r}} \setminus B^{\mathrm{mid}}$.

which can be computed by a single orthogonal range minimization query. Equipped with this value, we may determine ℓ^J as

$$\ell^J = \min_{\substack{j \in J, \\ \ell_1(j) \le L^J}} \ell^{\mathrm{rev}}_{\mathbf{r}}(j). \tag{5.5}$$

Note that given ℓ^J, we may determine R^J by a single orthogonal range reporting query; by (5.3).

We obtain the algorithm specified in Algorithm 12, whose correctness we summarize as follows: in the i-th loop, we consider the i-th (S,F)-reach-equivalent interval $J_i =: [a_i, b_i]_{J^{\mathrm{mid}}}$ in the above partition of J^{mid}. Observe that we determine L^{J_i} according to its definition in (5.4), and ℓ^{J_i} according to (5.5). Finally, we include in the set R_i all elements of R^{J_i} that have not yet been reported in a previous iteration, exploiting (5.3).

Let us analyze the running time. Recall that $\ell = \mathcal{O}(|S \cup F|) = \mathcal{O}(|\mathcal{T}_B|)$. Thus, we can compute $J_1, \dots, J_\ell$ in time $\mathcal{O}(\ell \log \ell) = \mathcal{O}(|\mathcal{T}_B| \log n)$. Furthermore, as discussed in Section 5.1.2, we can build $\mathcal{OR}_S$ in time $\mathcal{O}(|S| \log |S|) = \mathcal{O}(|\mathcal{T}_B| \log n)$ to support maximization queries in time $\mathcal{O}(\log |S|) = \mathcal{O}(\log n)$. Also, we can build $\mathcal{OR}_F$ in time $\mathcal{O}(|F| \log^2 |F|) = \mathcal{O}(|\mathcal{T}_B| \log^2 |\mathcal{T}_B|)$ to support deletions in time $\mathcal{O}(\log^2 |F|) = \mathcal{O}(\log^2 |\mathcal{T}_B|)$ and queries in time $\mathcal{O}(\log^2 |F| + k) = \mathcal{O}(\log^2 |\mathcal{T}_B| + k)$ where k denotes the number of reported elements. Observe that $\mathcal{OR}_B$ is already precomputed, as it belongs to the information stored at block B (see Definition 5.15).

We perform $\ell = \mathcal{O}(|\mathcal{T}_B|)$ iterations of the following form: First, we make a query to $\mathcal{OR}_S$ running in time $\mathcal{O}(\log n)$, followed by a query to $\mathcal{OR}_B$ running in time $\mathcal{O}(\log |\partial B|) = \mathcal{O}(\log n)$. Then we obtain a set R_i by a reporting query to $\mathcal{OR}_F$ running in time $\mathcal{O}(\log^2 |\mathcal{T}_B| + |R_i|)$. Afterwards, we delete all reported elements in time $\mathcal{O}(|R_i| \log^2 |\mathcal{T}_B|)$. Thus, the total running time is bounded by $\mathcal{O}(|\mathcal{T}_B| \log n + \sum_{i=1}^{\ell} |R_i| \log^2 |\mathcal{T}_B|)$. Observe that we report each element in $\mathcal{T}_B$ at most once, which results in $\sum_{i=1}^{\ell} |R_i| \le |\mathcal{T}_B|$.

Hence, the total running time is bounded by $\mathcal{O}(|\mathcal{T}_B|(\log n + \log^2 |\mathcal{T}_B|)) = \mathcal{O}(|\mathcal{T}_B| \log^2 n)$, as desired. $\square$

PART II

Lower Bounds

CHAPTER 6

Hausdorff Distance Under Translation

For a technical overview of this chapter see Section 3.3. In this chapter we consider finite point sets which lie in $\mathbb{R}^2$. For any $p \in \mathbb{R}^2$, we use p_x and p_y to refer to its first and second component, respectively.

6.1 OV Based $(nm)^{1-o(1)}$ Lower Bound for L_p

We now present a conditional lower bound of $(nm)^{1-o(1)}$ for the Hausdorff distance under translation — first for L_1 and L_∞, and then we discuss how to generalize this bound to L_p. We present the first lower bound only for the L_1 case, as the same construction carries over to the L_∞ case via a rotation of the input sets by $\frac{\pi}{4}$. Our lower bound is based on the hypothesized hardness of the Orthogonal Vectors problem.

Definition 6.1 (Orthogonal Vectors Problem (OV))**.** *Given two sets $U, V \subset \{0,1\}^D$ with $|U| = n, |V| = m$, decide whether there exist $u \in U$ and $v \in V$ with $u \cdot v = 0$.*

A popular hypothesis from fine-grained complexity theory is as follows.

Definition 6.2 (Orthogonal Vectors Hypothesis (OVH))**.** *The Orthogonal Vectors problem cannot be solved in time $\mathcal{O}((nm)^{1-\varepsilon} poly(D))$ for any $\varepsilon > 0$.*

This hypothesis is typically stated and used for the balanced case $n = m$. However, it is known that the hypothesis for the balanced case is equivalent to the hypothesis for any unbalanced case $m = n^\alpha$ for any fixed constant $\alpha > 0$, see, e.g, [41, Lemma 5.1 in Arxiv version].

We now describe a reduction from Orthogonal Vectors to Hausdorff distance under translation. To this end, we are given two sets of D-dimensional binary vectors $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_m\}$ with $|U| = n$ and $|V| = m$, and we construct an instance of the undirected Hausdorff distance under translation defined by point sets A and B and a decision distance $\delta = 1$. First, we describe the high-level structure of our reduction. The point set A consists only of Vector Gadgets, which encode the vectors of U using $2nd$ points. The point set B consists of three types of gadgets:

- *Vector Gadgets:* They encode the vectors from V, very similar to the Vector Gadgets of A.
- *Translation Gadget:* It restricts the possible translations of the point set B.
- *Undirected Gadget:* It makes our reduction work for the undirected Hausdorff distance under translation by ensuring that the maximum over the directed Hausdorff distances is always attained by $d_{\vec{H}}(B + \tau, A)$.

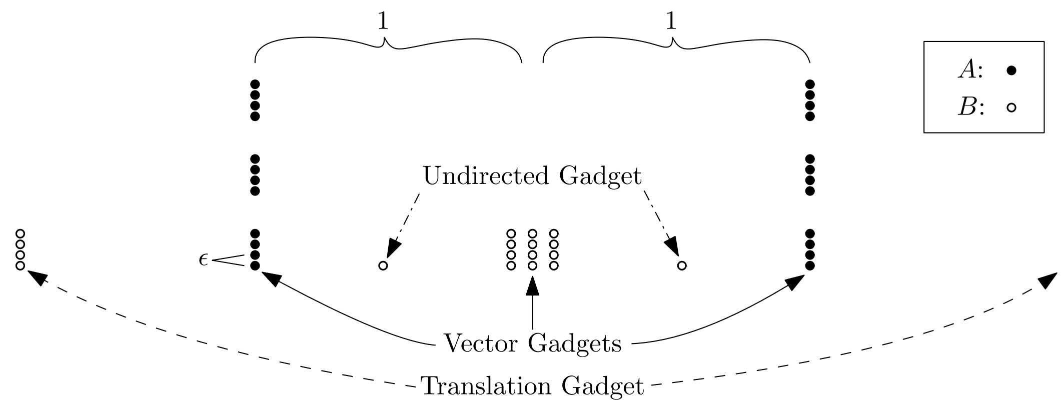


Figure 6.1: Sketch of the reduction from OV to the undirected Hausdorff distance under translation. The microtranslations in the order of ε^2 are not shown in this sketch.

Intuitively, the two dimensions of the translation choose the vectors $u \in U$ and $v \in V$ by aligning a Vector Gadget from A with a Vector Gadget from B in a certain way. An alignment of distance at most 1 is only possible if u and v are orthogonal. See Figure 6.1 for an overview of the reduction.

6.1.1 Gadgets

We now describe the gadgets in detail. Let $\varepsilon > 0$ be a sufficiently small constant, e.g., think of $\varepsilon = \frac{1}{20nmD}$. Recall that the distance for which we want to solve the decision problem is $\delta = 1$. Furthermore, we denote the ith component of a vector v by $v[i]$ and we use 0^D and 1^D to denote the D-dimensional all-zeros and all-ones vector, respectively.

Vector Gadget. We define a general Vector Gadget, which we then use at several places by translating it. Given a vector $v \in \{0,1\}^D$, the Vector Gadget consists of the points $p_1, \dots, p_D \in \mathbb{R}^2$:

$$p_i = \begin{cases} (\varepsilon^2, i\varepsilon), & \text{if } v[i] = 0 \\ (0, i\varepsilon), & \text{if } v[i] = 1 \end{cases}$$

We denote the Vector Gadget created from vector v by $G(v)$. Additionally, we define a mirrored version of the gadget G as

$$\overline{G}(v) \coloneqq G(\bar{v}),$$

where $\bar{v}$ is the inversion of v, i.e., each bit is flipped.

Lemma 6.3. *Given two vectors $v_1, v_2 \in \{0,1\}^D$ and corresponding Vector Gadgets $G_1 = G(v_1)$ and $G_2 = \overline{G}(v_2) + (1,0)$, we have $d_H(G_1, G_2) \le 1$ if and only if $v_1 \cdot v_2 = 0$.*

Proof. Let the points of G_1 (resp. G_2) be denoted as $p_1, \dots, p_D$ (resp. $q_1, \dots, q_D$). First, note that $\|p_i - q_j\|_1 = 1 + |i - j|\,\varepsilon + (v_1[i] + v_2[j] - 1)\varepsilon^2 > 1$ for $i \neq j$. Thus, for the Hausdorff distance to be at most 1, we have to match p_i to q_i for all $i \in [D]$. This is possible if and only if $v_1[i] = 0$ or $v_2[i] = 0$, as p_i and q_i are only in distance larger than 1 for $v_1[i] = 1$ and $v_2[i] = 1$. $\square$

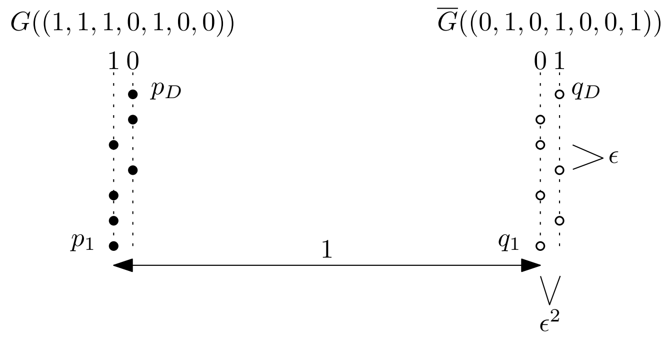


Figure 6.2: A depiction of the two types of Vector Gadgets and how they are placed to check for orthogonality.

See Figure 6.2 for an example. Note that if we swap both gadgets and invert both vectors (i.e., flip all their bits), the Hausdorff distance does not change and thus an analogous version of Lemma 6.3 holds in this case, as we are just performing a double inversion.

Lemma 6.4. *Given two vectors $v_1, v_2 \in \{0,1\}^D$ and corresponding Vector Gadgets $G_1 = \overline{G}(v_1)$ and $G_2 = G(v_2) + (1,0)$, we have $d_H(G_1, G_2) \leq 1$ if and only if $\bar{v}_1 \cdot \bar{v}_2 = 0$, where $\bar{v}_1, \bar{v}_2$ are the inversions of v_1, v_2.*

For any $x, y, \Delta \in \mathbb{R}$, we call Vector Gadgets $G_1 = G(v_1) + (x, y)$ and $G_2 = \overline{G}(v_2) + (x + \Delta, y)$ *vertically aligned*, or more precisely, *vertically aligned in distance* Δ.

Translation Gadget. To ensure that B cannot be translated arbitrarily, we introduce a gadget to restrict the translations to the regime we require. The Translation Gadget T consists of two translated Vector Gadgets of the zero vector:

$$T := (\overline{G}(1^D) - (2 - m\varepsilon, 0)) \cup (\overline{G}(0^D) + (2 + 2\varepsilon, 0)).$$

We show that restricting the coordinates of the points of the other set involved in the Hausdorff distance under translation instance, already restricts the feasible translations significantly.

Lemma 6.5. *Let $P \subset [-1 - \frac{1}{2}\varepsilon, 1 + \frac{1}{2}\varepsilon] \times \mathbb{R}$ be a point set. If $d^T_{\vec{H}}(T, P) \leq 1$, then $\tau^*_x \in [-(n + \frac{1}{2})\varepsilon - \varepsilon^2, -\frac{3}{2}\varepsilon]$, where τ^* is any translation satisfying $d_{\vec{H}}(T, P + \tau^*) \leq 1$.*

Proof. We show the contrapositive. Therefore, assume the converse, i.e., that τ^*_x is not contained in $[-(n + \frac{1}{2})\varepsilon - \varepsilon^2, -\frac{3}{2}\varepsilon]$. If $\tau^*_x < -(m + \frac{1}{2})\varepsilon - \varepsilon^2$, then $-1 - \frac{1}{2}\varepsilon - (-2 + m\varepsilon + \varepsilon^2 + \tau^*_x) > 1$ and thus the left part of T cannot contain any point of P in distance at most 1. If $\tau^*_x > -\frac{3}{2}\varepsilon$, then $2 + 2\varepsilon + \tau^*_x - (1 + \frac{1}{2}\varepsilon) > 1$ and thus the right part of T cannot contain any point of P in distance at most 1. Thus, $d^T_{\vec{H}}(T, P) > 1$. □

Undirected Gadget. To ensure that each point in A can be matched to a point in B within distance at most 1, we add auxiliary points to B. The Undirected Gadget is defined by the point set

$$F := \{(-\frac{1}{2}, 0), (\frac{1}{2}, 0)\}.$$

Lemma 6.6. *Given a set of points $P \subset [-1-\frac{1}{2}\varepsilon, 1+\frac{1}{2}\varepsilon] \times [-\frac{1}{8}, \frac{1}{8}]$, it holds that $d_{\vec{H}}(P, F+\tau) \leq 1$ for any $\tau \in [-(m+\frac{1}{2})\varepsilon - \varepsilon^2, (m+\frac{1}{2})\varepsilon + \varepsilon^2] \times [-\frac{1}{8}, \frac{1}{8}]$.*

Proof. By symmetry, we can restrict to proving that the distance of the point set

$$P' = P \cap [0, (m+\frac{1}{2})\varepsilon + \varepsilon^2] \times [-\frac{1}{8}, \frac{1}{8}]$$

to $(\frac{1}{2}, 0) + \tau$ is at most 1. For any $p' \in P'$, we have $\left|p'_x - (\frac{1}{2} + \tau_x)\right| \leq \frac{1}{2} + \mathcal{O}(m\varepsilon)$ and $\left|p'_y - \tau_y\right| \leq \frac{1}{4}$. Thus, $\left\|p' - ((\frac{1}{2}, 0) + \tau)\right\|_1 = \frac{3}{4} + \mathcal{O}(m\varepsilon)$, which is less than 1 for small enough ε. □

6.1.2 Reduction and correctness

We now describe the reduction and prove its correctness. We construct the point sets of our Hausdorff distance under translation instance as follows. The first set, i.e., set A, consists only of Vector Gadgets:

$$A := \left(\bigcup_{i \in [n]} G(u_i) + (-1 - \frac{1}{2}\varepsilon, i \cdot 2D\varepsilon) \right) \cup \left(\bigcup_{i \in [n]} G(1^D) + (1 + \frac{1}{2}\varepsilon, i \cdot 2D\varepsilon) \right)$$

The second set, i.e., set B, consists of Vector Gadgets, the Translation Gadget, and the Undirected Gadget:

$$B := \left(\bigcup_{j \in [m]} \overline{G}(v_j) + (j\varepsilon, 0) \right) \cup T \cup F$$

See Figure 6.1 for a sketch of the above construction. To reference the vector gadgets as they are used in the reduction, we use the notation

$$G_r(u_i) := G(u_i) + (-1 - \frac{1}{2}\varepsilon, i \cdot 2D\varepsilon) \quad \text{and} \quad \overline{G_r}(v_j) := \overline{G}(v_j) + (j\varepsilon, 0).$$

We can now prove correctness of our reduction. In the reduction, we return some canonical positive instance, if the 0^D vector is contained in any of the two OV sets. This allows us to drop all 1^D vectors from the input, as they cannot be orthogonal to any other vector. Thus, we can assume that all vectors in our input contain at least one 0-entry and at least one 1-entry.

Theorem 6.7. *Computing the directed or undirected Hausdorff distance under translation in L_1 or L_∞ for two point sets of size n and m in the plane cannot be solved in time $\mathcal{O}((nm)^{1-\gamma})$ for any $\gamma > 0$, unless the Orthogonal Vectors Hypothesis fails.*

Proof. Recall that we only have to consider the L_1 case. We first prove that there is a pair of orthogonal vectors $u \in U$ and $v \in V$ if and only if $d_H^T(A, B) \leq 1$. To prove the theorem for the directed and undirected Hausdorff distance under translation at the same time, it suffices to show "⇒" for the undirected version and "⇐" for the directed version.

⇒: Assume that there exist $u_i \in U$, $v_j \in V$ with $u_i \cdot v_j = 0$. Then consider the translation $\tau = (-(j+\frac{1}{2})\varepsilon, i \cdot 2D\varepsilon)$ which vertically aligns the Vector Gadgets $G_r(u_i)$ and

$\overline{G_r}(v_j) + \tau$ in distance 1. As u_i and v_j are orthogonal, it follows from Lemma 6.3 that $d_{\vec{H}}(\overline{G_r}(v_j) + \tau, A) \leq 1$. It remains to show that all remaining points of $B + \tau$ have a point in distance at most 1. The Vector Gadgets $\overline{G_r}(v_{j'}) + \tau$ with $j' < j$ are strictly to the left of $\overline{G_r}(v_j) + \tau$ and are thus also in Hausdorff distance at most 1 from $G_r(u_i)$. If $j = m$, then we are done with the Vector Gadgets. Otherwise, consider the Vector Gadget $\overline{G_r}(v_{j+1}) + \tau$. We claim that each point of it is in distance at most 1 from $G(1^D) + (1 + \frac{1}{2}\varepsilon, i \cdot 2D\varepsilon)$. As the two gadgets are vertically aligned, we just have to check their horizontal distance, which is

$$1 + \frac{1}{2}\varepsilon - ((j+1)\varepsilon - (j + \frac{1}{2})\varepsilon) = 1.$$

Thus, by Lemma 6.3, we have $d_{\vec{H}}(\overline{G_r}(v_{j+1}) + \tau, A) \leq 1$. Now, by the same argument as above, all gadgets $\overline{G_r}(v_{j'}) + \tau$ with $j' > j+1$ are in directed Hausdorff distance at most 1 from A.

As the points of the Undirected Gadget $F + \tau$ are closer by a distance of almost $\frac{1}{2}$ to A than the Vector Gadgets in $B + \tau$, also $d_{\vec{H}}(F + \tau, A) \leq 1$ holds. Finally, we have to show that the Translation Gadget $T + \tau$ is in distance at most 1 from A. As the left part of T and $G_r(u_i)$ are aligned vertically, we only have to check the horizontal distance. The horizontal distance is

$$-1 - \frac{1}{2}\varepsilon - (-2 + m\varepsilon - (j + \frac{1}{2})\varepsilon) = 1 - (m - j)\varepsilon \leq 1$$

for any $j \in [m]$. Similarly, the distance of the right part of the Translation Gadget from the vertically aligned $G(1^D)$ in A is

$$2 + 2\varepsilon - (j + \frac{1}{2})\varepsilon - (1 + \frac{1}{2}\varepsilon) = 1 - (j-1)\varepsilon \leq 1$$

for any $j \in [m]$. Thus, by Lemma 6.3 and Lemma 6.4, it holds that $d_{\vec{H}}(T + \tau, A) \leq 1$. As $\tau \in [-(n+\frac{1}{2})\varepsilon - \varepsilon^2, -\frac{3}{2}\varepsilon] \times [-\frac{1}{8}, \frac{1}{8}]$, we know by Lemma 6.6 that $d_{\vec{H}}(A, B + \tau) \leq 1$ and thus also $d_H^T(A, B) \leq 1$.

$\Leftarrow$: Now, assume that $d_H^T(A, B) \leq 1$ and let τ be any translation for which $d_{\vec{H}}(B + \tau, A) \leq 1$. Note that we used the directed Hausdorff distance in the previous statement on purpose, as we prove hardness for both versions. Lemma 6.5 implies that $\tau_x \in [-(n + \frac{1}{2})\varepsilon - \varepsilon^2, -\frac{3}{2}\varepsilon]$.

Let $\overline{G_r}(v_j) + \tau, \overline{G_r}(v_{j+1}) + \tau$ be the Vector Gadgets such that $\overline{G_r}(v_j) + \tau$ has directed Hausdorff distance at most 1 to the left Vector Gadgets of A and $\overline{G_r}(v_{j+1}) + \tau$ has directed Hausdorff distance at most 1 to the right Vector Gadgets of A. This is well-defined as the left Vector Gadgets of A and the right Vector Gadgets of A are in distance at least $2 + \varepsilon - \varepsilon^2$ from each other, and thus no Vector Gadget of $B + \tau$ can be in distance at most 1 from both sides. Furthermore, as $\tau_x \leq -\frac{3}{2}\varepsilon$, there has to be a Vector Gadget $\overline{G_r}(v_j) + \tau$ that has directed Hausdorff distance at most 1 to the left Vector Gadgets of A, as

$$j\varepsilon - \frac{3}{2}\varepsilon - (-1 - \frac{1}{2}\varepsilon) = 1 + (j-1)\varepsilon \leq 1$$

for $j = 1$. If $j = m$, then $\overline{G_r}(v_{j+1}) + \tau$ is undefined.

As $d_{\vec{H}}(B + \tau, A) \leq 1$, we know that $\overline{G_r}(v_j) + \tau$ has directed Hausdorff distance at most 1 to a gadget $G_r(u)$ for some $u \in U$. We claim that this distance cannot be closer than 1 as $\overline{G_r}(v_{j+1}) + \tau$ must have a directed Hausdorff distance at most 1 from the right side of A or, in case $j = m$, due to the restrictions imposed by the Translation Gadget. Let us consider the case $j \neq m$ first. Any translation τ' which places $\overline{G_r}(v_{j+1}) + \tau'$ in directed Hausdorff distance at most 1 from the right side of A needs to fulfill

$$1 + \frac{1}{2}\varepsilon - ((j+1)\varepsilon + \tau_1') \leq 1$$

and thus $\tau_1' \geq -(j + \frac{1}{2})\varepsilon$, using the fact that each vector in V contains at least one 0-entry. This, on the other hand, implies that $\overline{G_r}(v_j) + \tau'$ is in Hausdorff distance at least

$$j\varepsilon - (j + \frac{1}{2})\varepsilon - (-1 - \frac{1}{2}\varepsilon) = 1$$

from $G_r(u)$. Now consider the case $j = m$. As by Lemma 6.5 we have $\tau_x \geq -(m + \frac{1}{2})\varepsilon - \varepsilon^2$, it follows that $\overline{G_r}(v_m) + \tau$ is in Hausdorff distance at least

$$m\varepsilon - (m + \frac{1}{2})\varepsilon - (-1 - \frac{1}{2}\varepsilon) = 1$$

from $G_r(u)$, using the fact that each vector in V contains at least one 0-entry (this is the reason why the ε^2 disappears).

By the arguments above, the two gadgets $\overline{G_r}(v_j) + \tau$ and $G_r(u)$ have to be horizontally aligned as required by Lemma 6.3. They also have to be vertically aligned as a vertical deviation would incur a Hausdorff distance larger than 1 for the pair of points in the two gadgets that are in horizontal distance 1. Then, applying Lemma 6.3, it follows that u and v_j are orthogonal.

It remains to argue why the above reduction implies the lower bound stated in the theorem. Assume we have an algorithm that computes the Hausdorff distance under translation for L_1 or L_∞ in time $(nm)^{1-\gamma}$ for some $\gamma > 0$. Then, given an Orthogonal Vectors instance U, V with $|U| = n$ and $|V| = m$, we can use the described reduction to obtain an equivalent Hausdorff under translation instance with point sets A, B of size $|A| = \mathcal{O}(nD)$ and $|B| = \mathcal{O}(mD)$ and solve it in time $\mathcal{O}((nm)^{1-\gamma}\text{poly}(D))$, contradicting the Orthogonal Vectors Hypothesis.

□

6.1.3 Generalization to L_p

We can extend the above construction such that it works for all L_p norms with $p \neq \infty$ by changing the spacing between 0 and 1 points of the Vector Gadgets and also set ε accordingly. More precisely, we can set $\varepsilon = \frac{1}{40pnmD}$ (instead of $\frac{1}{20nmD}$) and use ε^{2p} as spacing (instead of ε^2), i.e., the Vector Gadget for a vector $v \in \{0,1\}^D$ then consists of the points $p_1, \ldots, p_D \in \mathbb{R}^2$:

$$p_i = \begin{cases} (\varepsilon^{2p}, i\varepsilon), & \text{if } v[i] = 0 \\ (0, i\varepsilon), & \text{if } v[i] = 1 \end{cases}$$

We prove that these modifications suffice in the remainder of this section.

To this end, first note that in the proof of Theorem 6.7, the proof for "$\Rightarrow$" for L_p already follows from the L_1 case as the L_1 norm is an upper bound on all L_p norms. Thus, we only have to modify the proof of "$\Leftarrow$". To show "$\Leftarrow$", note that the only place where we use the L_1 norm in the proof is in the invocation of Lemma 6.3. Otherwise, we only argue via distances with respect to a single dimension, which carries over to L_p as $\|(x,0)\|_p = |x|$. Thus, we now prove Lemma 6.3 for the general L_p case.

Proof of Lemma 6.3 for L_p. To adapt the proof of Lemma 6.3 to the L_p case, we only have to argue that we cannot match any p_i, q_j for $i \neq j$, as the remaining arguments merely argue about distances in a single dimension. We have that

$$\|p_i - q_j\|_p = \left((|i-j|\,\varepsilon)^p + (1-(v_1[i]+v_2[j]-1)\varepsilon^{2p})^p\right)^{1/p} \geq \left(\varepsilon^p + (1-\varepsilon^{2p})^p\right)^{1/p},$$

which is greater than 1 if $\varepsilon^p + (1-\varepsilon^{2p})^p > 1$, which we obtain by using Bernoulli's inequality:

$$\varepsilon^p + (1-\varepsilon^{2p})^p \geq \varepsilon^p + 1 - p\varepsilon^{2p} \geq 1 + \left(\frac{1}{40pnmD}\right)^p - p\left(\frac{1}{40pnmD}\right)^{2p} > 1.$$

The remainder of the proof is analogous to the remainder of the proof of Lemma 6.3. □

By all of the above arguments, the following theorem follows.

Theorem 6.8 (Theorem 6.7 for L_p). *Computing the directed or undirected Hausdorff distance under translation in L_p for two point sets of size n and m in the plane cannot be solved in time $\mathcal{O}((nm)^{1-\gamma})$ for any $\gamma > 0$, unless the Orthogonal Vectors Hypothesis fails.*

6.2 3Sum Based $n^{2-o(1)}$ Lower Bound for $m \in \mathcal{O}(1)$

We now present a hardness result for the unbalanced case of the directed and undirected Hausdorff distance under translation. We base our hardness on another popular hypothesis of fined-grained complexity theory: the 3Sum Hypothesis. Before stating the hypothesis, let us first introduce the 3Sum problem.

Definition 6.9 (3Sum). *Given three sets of positive integers X, Y, Z all of size n, do there exist $x \in X, y \in Y, z \in Z$ such that $x + y = z$?*

The corresponding hardness assumption is the 3Sum Hypothesis.

Definition 6.10 (3Sum Hypothesis). *There is no $\mathcal{O}(n^{2-\varepsilon})$ algorithm for 3Sum for any $\varepsilon > 0$.*

There are several equivalent variants of the 3Sum problem. Most important for us is the convolution 3Sum problem, abbreviated as Conv3Sum [63, 138].

Definition 6.11 (Conv3SUM). *Given a sequence of positive integers $X = (x_0, \ldots, x_{n-1})$ of size n, do there exist i, j such that $x_i + x_j = x_{i+j}$?*

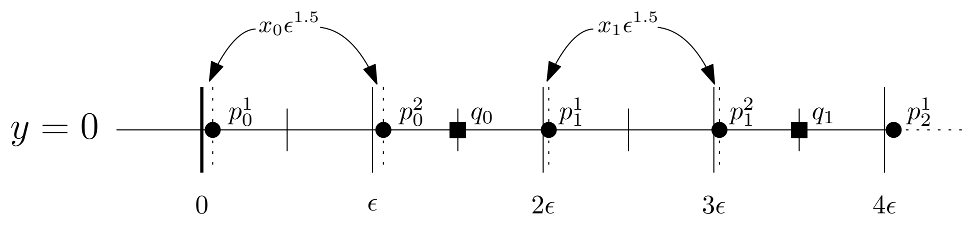


Figure 6.3: The A set of the low-level gadget of the 3Sum reduction, which is used to build the high-level gadgets. We just show the leftmost part of the gadget, but the remainder is similar.

This problem has a trivial $\mathcal{O}(n^2)$ algorithm and, assuming the 3Sum Hypothesis, this is also optimal up to lower order factors. As 3Sum and Conv3Sum are equivalent, a lower bound conditional on Conv3Sum implies a lower bound conditional on 3Sum.

Therefore, given a Conv3Sum instance defined by the sequence of integers X with $|X| = n$, we create an equivalent instance of the directed Hausdorff distance under translation for L_2 by constructing two sets of points A and B with $|A| = \mathcal{O}(n)$ and $|B| = \mathcal{O}(1)$ and providing a decision distance δ. Intuitively, we define a low-level gadget from which we build three high-level gadgets by rotation and scaling. Recall that in the Conv3Sum problem we have to find values i, j which fulfill the equation $x_i + x_j = x_{i+j}$. We encode the choice of these two values into the two dimensions of the translation. The three high-level gadgets then verify whether the Conv3Sum equation is fulfilled. In the remainder of this section, we present the details of our reduction and prove that it implies the claimed lower bound.

6.2.1 Construction

Given a Conv3Sum instance with $X \subset [M]$ where $n = |X|$, we now describe the construction of the Hausdorff distance under translation instance with point sets A, B and threshold distance δ. We use a small enough ε, e.g., $\varepsilon = (4Mn^2)^{-4}$, as value for microtranslations. Furthermore, we set $\delta = 1 + 4n^2\varepsilon^2$. The additional $4n^2\varepsilon^2$ term compensates for the small variations in distance that occur on microtranslations due to the curvature of the L_2-ball.

Low-Level Gadget

We use a single low-level gadget, which is then scaled and rotated to obtain high-level gadgets. This gadget consists of two point sets A_l and B_l. The point set A_l contains what we call *number points* p_i^1, p_i^2 and *filling points* q_i for $0 \le i < n$. The set B_l just contains two points: r_1 and r_2. The number points p_i^1, p_i^2 encode the number x_i, while the filling points make sure that no other translations than the desired ones are possible. See Figure 6.3 for an overview. All of the points in this gadget are of the form $(x, 0)$. The number points are

$$p_i^1 = \left(2i\varepsilon + x_i\varepsilon^{1.5}, 0\right), \quad p_i^2 = p_i^1 + (\varepsilon, 0)$$

for $0 \leq i < n$. The filling points are

$$q_i = \left(\left(2i + \frac{3}{2}\right)\varepsilon, 0\right)$$

for $0 \leq i < n$.

The points in B_l should introduce a gap to only allow alignment of the number gadgets such that the microtranslations (i.e., those in the order of $\varepsilon^{1.5}$) correspond to the number of the gap in the number gadget. To this end, B_l contains the points

$$r_1 = (-1, 0), \quad r_2 = (1 + \varepsilon, 0).$$

Before we prove properties of the low-level gadget, we first prove that the error due to the curvature of the L_2-ball is small.

Lemma 6.12. *Let $(p_x, p_y), (q_x, q_y) \in \mathbb{R}^2$ be two points with $|p_x - q_x| \in [\frac{1}{2}, 2]$ and $p_y = q_y$. For any $\tau \in [0, (2n-1)\varepsilon]^2$, we have*

$$|p_x - (q_x + \tau_x)| \leq \|p - (q + \tau)\|_2 \leq |p_x - (q_x + \tau_x)| + 4n^2\varepsilon^2.$$

Proof. As each component is a lower bound for the L_2 norm, the first inequality follows. Thus, let us prove the second inequality. We first transform

$$\|p - (q + \tau)\|_2 = \sqrt{(p_x - (q_x + \tau_x))^2 + \tau_y^2} = |p_x - (q_x - \tau_x)| \sqrt{1 + \tau_y^2/(p_x - (q_x + \tau_x))^2}.$$

As $\sqrt{1 + x} \leq 1 + \frac{x}{2}$ for any $x \geq 0$, we have

$$\|p - (q + \tau)\|_2 \leq |p_x - (q_x - \tau_x)| + \tau_y^2/(2\,|p_x - (q_x - \tau_x)|).$$

As $\tau_y \leq 2(n-1)\varepsilon$ and $|p_x - (q_x - \tau_x)| \geq \frac{1}{2}$, we obtain the desired upper bound. □

An analogous statement holds when swapping the x and y coordinates. Note that the $4n^2\varepsilon^2$ term also occurs in the value of δ that we chose, as this is how we compensate for these errors in our construction. While we have to consider this error in the following arguments, it should already be conceivable that it will be insignificant due to its magnitude.

We now state two lemmas which show how the Hausdorff distance under translation decision problem is related to the structure of the low-level gadget.

Lemma 6.13. *Given a low-level gadget A_l, B_l as constructed above and the translation being restricted to $\tau \in [0, (2n-1)\varepsilon]^2$, it holds that if $d_{\vec{H}}(A_l, B_l + \tau) \leq \delta$, then*

$$\exists i \in \mathbb{N} : \tau_x = 2i\varepsilon + x_i\varepsilon^{1.5} \pm 4n^2\varepsilon^2.$$

Proof. Let $\tau \in [0, (2n-1)\varepsilon]^2$ and assume $d_{\vec{H}}(A_l, B_l + \tau) \leq \delta$. Then all points in A_l are in distance at most δ from one of the two points in B_l. Furthermore, both points in $B_l + \tau$ also have at least one close point in A_l, as

$$\left\|r_1 + \tau - p_0^1\right\|_2 \leq 1 - \tau_x + 4n^2\varepsilon^2 \leq \delta$$

$$\text{and} \quad \|r_2 + \tau - q_{n-1}\|_2 \leq 1 + \tau_x - (2n - \frac{3}{2})\varepsilon + 4n^2\varepsilon^2 < \delta,$$

using that $n \geq 1$ and Lemma 6.12.

The gaps between neighboring points in A_l either have width close to $\frac{1}{2}\varepsilon$, if the gap is between a number point and a filling point (p_i^1 and q_{i-1}, or p_i^2 and q_i), or they have a width of ε, if the gap is between two number points (p_i^1 and p_i^2). Furthermore, the two points in B_l have distance $2 + \varepsilon$, so there is an $\varepsilon - 8n^2\varepsilon^2$ gap between their δ-balls. Thus, there is an i such that p_i^1 has distance at most δ to r_1, and p_i^2 has distance at most δ to r_2. This alignment of the gadgets can only be realized by a translation τ for which

$$\tau_x = 2i\varepsilon + x_i\varepsilon^{1.5} \pm 4n^2\varepsilon^2,$$

which completes the proof. □

Lemma 6.14. *Given a low-level gadget A_l, B_l as constructed above and the translation being restricted to $\tau \in [0, (2n-1)\varepsilon]^2$, it holds that if*

$$\exists i \in \mathbb{N} : \tau_x = 2i\varepsilon + x_i\varepsilon^{1.5},$$

then $d_H(A_l, B_l + \tau) \leq \delta$.

Proof. Let $i \in \mathbb{N}$ and let $\tau_x = 2i\varepsilon + x_i\varepsilon^{1.5}$. Consider any translations $\tau \in \{\tau_x\} \times [0, 2(n-1)\varepsilon]$. Due to the restricted translation and Lemma 6.12, we can disregard the error terms that arise from the vertical translation τ_y as they are compensated for by δ. Then all the points in A_l before and including p_i^1 are in distance at most δ from $r_1 \in B_l + \tau$ and all the points afterwards are in distance at most δ from $r_2 \in B_l + \tau$. Clearly, both points in $B_l + \tau$ then also have points from A_l in distance δ, and thus $d_H(A_l, B_l + \tau) \leq \delta$. □

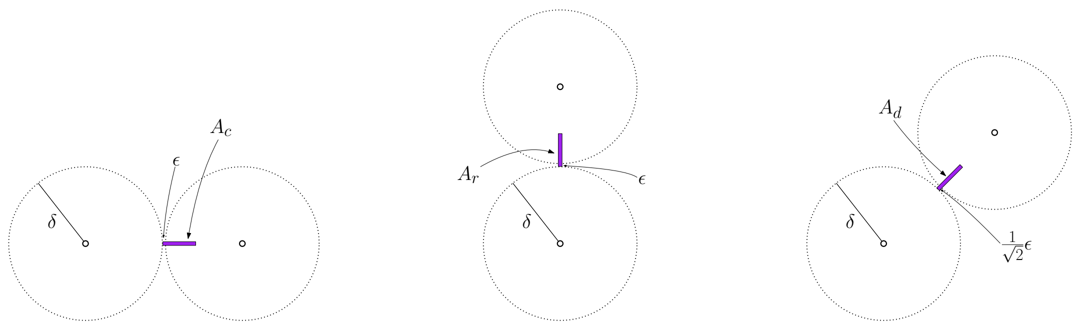
High-Level Gadgets

This construction is inspired by the hard instance that was given in [144]. We want to obtain a grid of translations of spacing ε with some microtranslations in the $\mathcal{O}(\varepsilon^{1.5})$ range. We already defined the low-level gadget above, and we now define the high-level gadgets.

Column Gadget. The column gadget induces columns in translational space, i.e., it enforces that valid translations have to lie on one of these columns. The column gadget is actually the low-level gadget we already described above. You can see a sketch of this gadget in Figure 6.4(a). To semantically distinguish it from the low-level gadget, we refer to the point sets of the column gadget as A_c and B_c.

Row Gadget. The row gadget induces rows in translational space, i.e., it enforces that valid translations have to lie on one of these rows. We obtain the row gadget by rotating all points in the low-level gadget around the origin by $\pi/2$ counterclockwise. You can see a sketch of this gadget in Figure 6.4(b). We call the point sets of the row gadget A_r and B_r.

Diagonal Gadget. The diagonal gadget induces diagonals in translational space, i.e., it enforces that valid translations have to lie on one of these diagonals. As opposed to the column and row gadget, the diagonal gadget also has to be scaled. We scale the sets A_l and B_l separately. We scale A_l such that the gap between the number point pairs p_i^1, p_i^2



(a) Column Gadget **(b)** Row Gadget **(c)** Diagonal Gadget

Figure 6.4: Three of the high-level gadgets. The points of A are all in the low-level gadgets, while the points in B are explicitly shown including their δ-ball.

becomes $\frac{1}{\sqrt{2}}\varepsilon$. And we scale B_l such that the gap between the points becomes $2 + \frac{1}{\sqrt{2}}\varepsilon$. After scaling, we rotate the points counterclockwise around the origin by $\pi/4$. You can see a sketch of this gadget in Figure 6.4(c). We call the point sets of the diagonal gadget A_d and B_d.

Translation Gadget. To restrict the translations for the directed Hausdorff distance under translation, we introduce another gadget. The first set of points A_t contains

$$z_l := (-1 + (2n-1)\varepsilon, 0), \quad z_r := (1, 0), \quad z_b := (0, -1 + (2n-1)\varepsilon), \quad z_t := (0, 1).$$

The second point set B_t only contains the origin $z_c := (0, 0)$. We want to make sure that this gadget behaves well in a certain range.

Lemma 6.15. *Given $\tau \in [0, (2n-1)\varepsilon]^2$, it holds that $d_H(A_t, B_t + \tau) \leq \delta$.*

Proof. As B_t has a point on all sides, clearly $d_{\vec{H}}(B_t + \tau, A_t) \leq \delta$. Furthermore,

$$\|z_l - (z_c + \tau)\|_2 \leq 1 + 4n^2\varepsilon^2 \leq \delta \quad \text{and} \quad \|z_r - (z_c + \tau)\|_2 \leq \delta,$$

using Lemma 6.12. Analogous statements hold for z_b and z_t. Thus, also $d_{\vec{H}}(A_t, B_t + \tau) \leq \delta$. □

Complete Construction

To obtain the final sets of the reduction, we now place all four described high-level gadgets (i.e., column gadget, row gadget, diagonal gadget, and translation gadget) far enough apart. More explicitly, the point sets A, B of the Hausdorff distance under translation instance are defined as

$$A := A_c \cup (A_r + (10, 0)) \cup (A_d + (20, 0)) \cup (A_t + (30, 0))$$

and

$$B := B_c \cup (B_r + (10, 0)) \cup (B_d + (20, 0)) \cup (B_t + (30, 0)).$$

The far placement ensures that the two point sets of the respective gadgets have to be matched to each other when the Hausdorff distance under translation is at most delta δ.

6.2.2 Proof of correctness

First, we want to ensure that everything relevant happens in a very small range of translations.

Lemma 6.16. *Let $\tau \in \mathbb{R}^2$. If $d_{\vec{H}}(A, B+\tau) \le \delta$, then $\tau \in [0,(2n-1)\varepsilon]^2$.*

Proof. Note that for a Hausdorff distance at most δ, the sets A_c and B_c have to matched to each other and analogously for A_r, B_r, and A_d, B_d, and A_t, B_t. To show the contrapositive, assume $\tau \notin [0,(2n-1)\varepsilon]^2$. For simplicity, we refer to the points in the high-level gadgets with the notation of the low-level gadget. Due to the translation gadget, we have

$$\|z_l - (z_c + \tau)\|_2 > \delta \quad \text{for} \quad \tau_x > (2n-1)\varepsilon + 4n^2\varepsilon^2,$$

and

$$\|z_r - (z_c + \tau)\|_2 > \delta \quad \text{for} \quad \tau_x < -4n^2\varepsilon^2.$$

We now show that under these restricted translations and as $d_{\vec{H}}(A, B+\tau) \le \delta$, both points r_1, r_2 in B_c have at least one point of A_c in distance δ. In the column gadget for $\tau_x \in [-4n^2\varepsilon^2, 0)$, we have

$$\left\|(r_1 + \tau) - p_0^1\right\|_2 \ge \left|-1 - (p_0^1)_x + \tau_x\right| > \delta$$

and

$$\left\|(r_2 + \tau) - p_0^1\right\|_2 \ge 1 + \varepsilon - \mathcal{O}(\varepsilon^{1.5}) > \delta$$

for small enough ε and as $x_0 > 0$ and thus there is a component of order $\varepsilon^{1.5}$. On the other hand, for $\tau_x \in ((2n-1)\varepsilon, (2n-1)\varepsilon + 4n^2\varepsilon^2]$, we have

$$\left\|r_2 + \tau - p_{n-1}^2\right\|_2 \ge 1 + \varepsilon + \tau_x - (2n-1)\varepsilon > \delta$$

and

$$\left\|r_1 + \tau - p_{n-1}^2\right\|_2 \ge 1 + \mathcal{O}(\varepsilon^{1.5}) - 4n^2\varepsilon^2 > \delta$$

for small enough ε. An analogous argument holds for the row gadget and τ_y, as the row gadget is just a rotated version of the column gadget and the translation gadget is symmetric with respect to these gadgets. □

We can now prove the main result of this section.

Theorem 6.17. *Computing the directed or undirected Hausdorff distance under translation in L_2 for two sets of size n and $\mathcal{O}(1)$ cannot be solved in time $\mathcal{O}(n^{2-\gamma})$ for any $\gamma > 0$, unless the* 3Sum *Hypothesis fails.*

Proof. We construct a Hausdorff under translation instance from a Conv3Sum instance as described previously in this section, and then show that they are equivalent. We first consider how to apply Lemma 6.13 and Lemma 6.14 to the diagonal gadget. More precisely, we consider which translations align the gaps of A_d and B_d as is used in these two lemmas. Due to the scaling of the gadget, these translations are of the form $\sqrt{2}\tau_x = 2k\varepsilon + x_k\varepsilon^{1.5}$. By the rotation, we then obtain translations of the form

$$\|\tau\|_1 = \frac{\sqrt{2}(\tau_x + \tau_y)}{\sqrt{2}} = 2k\varepsilon + x_k\varepsilon^{1.5}.$$

$\Leftarrow$: Assume X is a positive Conv3Sum instance. Then there exist x_i, x_j such that $x_i + x_j = x_{i+j}$. Consider $\tau = (2i\varepsilon + x_i\varepsilon^{1.5}, 2j\varepsilon + x_j\varepsilon^{1.5})$ as translation. Due to Lemma 6.14, we have that $d_H(A_c, B_c + \tau) \leq \delta$ and analogously $d_H(A_r, B_r + \tau) \leq \delta$. By the initial observation, we can also apply Lemma 6.14 to the diagonal gadget, and thus $d_H(A_d, B_d + \tau) \leq \delta$. Finally, by Lemma 6.15, we also have that $d_H(A_t, B_t + \tau) \leq \delta$ for the given τ.

$\Rightarrow$: Assume $d_{\vec{H}}^T(A, B) \leq \delta$. From Lemma 6.16, it follows that $\tau \in [0, (2n-1)\varepsilon]^2$. Then, due to Lemma 6.13 and the initial observation about the diagonal gadget, we have that there exist i, j, k that fulfill

$$\begin{aligned} \tau_x &= 2i\varepsilon + x_i\varepsilon^{1.5} \pm 4n^2\varepsilon^2, \\ \tau_y &= 2j\varepsilon + x_j\varepsilon^{1.5} \pm 4n^2\varepsilon^2, \\ \tau_x + \tau_y &= 2k\varepsilon + x_k\varepsilon^{1.5} \pm 4n^2\varepsilon^2. \end{aligned}$$

It follows that

$$2i\varepsilon + x_i\varepsilon^{1.5} + 2j\varepsilon + x_j\varepsilon^{1.5} \pm 8n^2\varepsilon^2 = 2k\varepsilon + x_k\varepsilon^{1.5} \pm 4n^2\varepsilon^2,$$

and thus $i + j = k$ and $x_i + x_j = x_k$.

It remains to argue why the above reduction implies the lower bound stated in the theorem. Assume we have an algorithm that computes the Hausdorff distance under translation in L_2 in time $\mathcal{O}(n^{2-\gamma})$ for some $\gamma > 0$. Then, given a Conv3Sum instance X with $|X| = n$, we can use the described reduction to obtain an equivalent Hausdorff under translation instance with point sets A, B of size $|A| = \mathcal{O}(n)$ and $|B| = \mathcal{O}(1)$ and solve it in time $\mathcal{O}(n^{2-\gamma})$, contradicting the 3Sum Hypothesis. □

6.3 Conclusion

In this chapter, we provide matching lower bounds for the running time of two important cases of the fundamental distance measure Hausdorff distance under translation. These lower bounds are based on popular standard hypotheses from fine-grained complexity theory. Interestingly, we use two different hypotheses to show hardness. For the Hausdorff distance under translation for L_p, we show a lower bound of $(nm)^{1-o(1)}$ using the Orthogonal Vectors Hypothesis, while for the imbalanced case of $m = \mathcal{O}(1)$ in L_2, we show an $n^{2-o(1)}$ lower bound using the 3Sum Hypothesis. We leave it as an open problem whether Hausdorff distance under translation for the balanced case admits a strongly subcubic algorithm or if conditional hardness can be shown.

CHAPTER 7

Continuous Fréchet Approximate Near Neighbor Search

For a technical overview of this chapter see Section 3.1. In this section we show several conditional lower bounds for $(2-\varepsilon)$ and $(3-\varepsilon)$-approximate nearest neighbor data structures. We use the well-known Orthogonal Vectors problem as base for our hardness results. To be precise, we reduce from unbalanced OV instances to show stronger hardness results. For convenience, we introduce some additional notation. For a vector $v \in \{0,1\}^D$, we use $v[i]$ to refer to its ith entry, where the entries are 0-index, i.e., $v = (v[0], \ldots, v[D-1])$. Recall that we use the "$\circ$" operator to concatenate curves and that the curve P where each point is translated by τ is denoted as $P+\tau$. Instead of reducing directly from OV, we introduce a novel problem called OneSidedSparseOV and show that it is hard under OV. Subsequently, we reduce from this problem to the ANN problems introduced above.

7.1 OneSidedSparseOV

This problem can be thought of as a variant of OV with an additional restriction on one of the input sets. More precisely, for one set we allow at most k non-zero entries in each vector.

Definition 7.1 (OneSidedSparseOV). *Given a value $k \in \mathbb{N}$ and two sets of vectors $U, V \subseteq \{0,1\}^D$ where each $u \in U$ contains at most k non-zero entries, do there exist two vectors $u \in U, v \in V$ such that $\langle u, v\rangle = 0$?*

We also refer to OneSidedSparseOV with parameter k as OneSidedSparseOV(k). We now show that this problem is hard under OV, interestingly, this is already the case for $k \in \omega(1)$.

Lemma 7.2. *Assume OVH holds true. For every $\alpha \in (0,1)$, $\varepsilon > 0$ there is a $c > 0$ such that for any $k \in \omega(1) \cap o(\log|U|)$ there is no algorithm solving* OneSidedSparseOV(k) *instances $U, V \subset \{0,1\}^D$ with $|V| = |U|^\alpha$ and $D = k \cdot |U|^{c/k}$ in time $\mathcal{O}(|U|^{1+\alpha-\varepsilon})$.*

Proof. For any $\alpha \le 1, \varepsilon > 0$, let $c > 0$ be the constant from Lemma 2.9. Thus, unless OVH fails, we cannot solve OV instances $U, V \subset \{0,1\}^D$ with $|V| = |U|^\alpha$ and $D = c\log|U|$ in time $\mathcal{O}(|U|^{1+\alpha-\varepsilon})$. For any $k \in \omega(1) \cap o(\log|U|)$, we now reduce to OneSidedSparseOV(k) as follows. We convert U to a *set of sparse vectors* U' and V to a set V' such that U', V' is an equivalent OneSidedSparseOV(k) instance. To achieve this, we increase the dimensionality of the vectors in the OneSidedSparseOV instance.

Given a vector $u \in U$, partition the dimensions of u into k blocks of size D/k.[1] More precisely, let

$$a_i = \left(u\left[i \cdot \frac{D}{k}\right], u\left[i \cdot \frac{D}{k} + 1\right], \dots, u\left[i \cdot \frac{D}{k} + \frac{D}{k} - 1\right] \right)$$

for $i \in \{0, \dots, k-1\}$. Let $\hat{u}_i \in \left[2^{D/k}\right]$ be defined as the binary vector a_i interpreted as a binary number. We now construct the corresponding $u' \in U'$ as follows. We choose the dimension of the vectors in U', V' as $D' = k \cdot 2^{D/k}$ — note that this equals $k \cdot |U|^{c/k}$ as stated in the lemma. For each $i \in \{0, \dots, k-1\}$, we set $u'[i \cdot 2^k + \hat{u}_i] = 1$. All other entries of u' are set to 0. Thus, each vector $u' \in U'$ contains exactly k 1-entries. The vectors $v' \in V'$ we construct as follows. Given a vector $v \in V$, we also partition its dimensions the same way as we did for $u \in U$ and obtain vectors $b_0, \dots, b_{k-1}$. For each $i \in \{0, \dots, k-1\}$ and all $\beta \in \{0,1\}^{D/k}$, where we again interpret β as a vector or as a binary number $\hat{\beta}$, we set $v'[i \cdot 2^k + \hat{\beta}] = 1$ if $\langle b_i, \beta \rangle > 0$, otherwise we set it to zero. This completes the description of the reduction. Note that while we changed the dimension of the vectors, the size of the sets remained the same, that is $|U'| = |U|$ and $|V'| = |V|$.

Note that for any vectors $u \in U$ and $v \in V$ with $\langle u, v \rangle > 0$ there exist parts a_i, b_i and a coordinate ℓ such that $a_i[\ell] = b_i[\ell] = 1$, and thus $\langle a_i, b_i \rangle > 0$. Hence, by construction of v', there exists a dimension in u' and v' where both have a 1. On the other hand, if u' and v' contain a 1 in the same dimension, then by construction of v' there have to be two parts a_i, b_i such that $\langle a_i, b_i \rangle > 0$ and thus $\langle u, v \rangle > 0$.

The total running time of this reduction consists of constructing the vectors in U' — which takes time proportional to the number of entries — and the inner product computation between vectors of dimensionality D/k for each of the $k \cdot 2^{D/k}$ dimensions of each vector in V':

$$\begin{aligned} \mathcal{O}\left(|U'| \cdot k \cdot 2^{D/k} + |V'| \cdot k \cdot 2^{D/k} \cdot \frac{D}{k} \right) &= \mathcal{O}\left(|U| \cdot 2^{c \log |U| / k} \cdot c \log |U| \right) \\ &= \mathcal{O}\left(|U|^{1+c/k} \cdot c \log |U| \right), \end{aligned}$$

which simplifies to $|U|^{1+o(1)}$. Thus, if we can solve OneSidedSparseOV(k) in time $\mathcal{O}(|U'|^{1+\alpha-\varepsilon})$ and add the running time of the reduction, then we can solve unbalanced OV in time

$$\mathcal{O}(|U'|^{1+\alpha-\varepsilon}) + |U|^{1+o(1)} = \mathcal{O}(|U|^{1+\alpha-\varepsilon}),$$

which would refute OVH. □

Using this insight, we now proceed to proving hardness results for different approximation ratios for ANN under the continuous Fréchet distance.

7.2 Hardness of $(2-\varepsilon)$-Approximation in 1D

In this section we present our first hardness result. We note that the gadgets that we use to encode our vectors are inspired by [84].

[1] If D is not divisible by k, increase the dimension until this is the case and fill these dimensions with zeros.

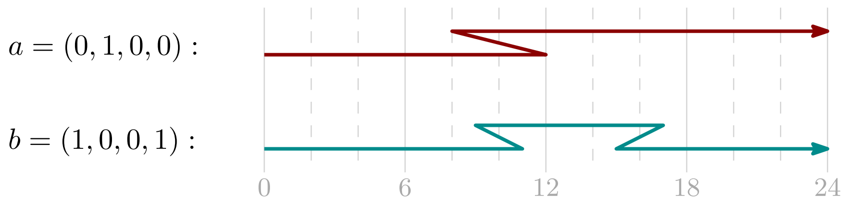


Figure 7.1: Visualization of the $2-\varepsilon$ lower bound in 1D.

Theorem 7.3. *Assume OVH holds true. For any $\varepsilon, \varepsilon' > 0$ there is a $c > 0$, such that there is no $(2-\varepsilon)$-ANN for the continuous Fréchet distance supporting query curves of any complexity $k \in \omega(1) \cap o(\log n)$ and storing n one-dimensional curves of complexity $m = k \cdot n^{c/k}$ with preprocessing time poly(n) and query time $\mathcal{O}(n^{1-\varepsilon'})$.*

Proof. We show the hardness by a reduction from OneSidedSparseOV(k). To that end, let $U, V \subset \{0,1\}^D$ be a OneSidedSparseOV(k) instance with $|V| = |U|^\alpha$ for a constant $\alpha \leq 1$ that we specify later, $k \in \omega(1) \cap o(\log |U|)$, and $D = k \cdot |U|^{c/k}$ with a constant $c > 0$ that we later choose sufficiently large. Recall that, by Lemma 7.2, there exists a $c > 0$ such that OneSidedSparseOV(k) is OV-hard in this regime. The goal is to use the k-sparsity of the vectors in U to obtain short query curves of length $\mathcal{O}(k)$.

Let us first give the reduction. To that end, we define the following subcurves:

$$0_U := \langle 0, 6 \rangle, \quad 1_U := \langle 0, 6, 2, 6 \rangle, \quad 0_V := \langle 0, 5, 3, 6 \rangle, \quad 1_V := \langle 0, 6 \rangle$$

Now, given a OneSidedSparseOV(k) instance U, V, we create the input set $\mathcal{P}$ and the query set $\mathcal{Q}$ of a $(2-\varepsilon)$-ANN instance with distance threshold $\delta = 1$ as follows. For each vector $u \in U$, we add the curve Q_u to $\mathcal{Q}$ which is defined as

$$Q_u := \bigcirc_{i=0}^{D-1} G_u^i \quad \text{with} \quad G_u^i := u[i]_U + 6i,$$

where $u[i]_U$ is either 0_U or 1_U, depending on the value of $u[i]$, and the "$+6i$" is a translation of each point of the curve by $6i$. For each vector $v \in V$, we add the curve P_v to $\mathcal{P}$ which is defined as

$$P_v := \bigcirc_{i=0}^{D-1} G_v^i \quad \text{with} \quad G_v^i := v[i]_V + 6i.$$

where $v[i]_V$ is either 0_V or 1_V, depending on the value of $v[i]$. It is crucial that we make the resulting curves non-degenerate by removing all degenerate vertices. In particular, all connecting vertices between gadget curves will be removed and any sequence of consecutive gadgets 0_U will be turned into a single line segment. Thus, the curves in $\mathcal{Q}$ will have complexity $\mathcal{O}(k)$. See Figure 7.1 for an example of the construction.

We now show correctness of the reduction. Let $P_v \in \mathcal{P}$ and $Q_u \in \mathcal{Q}$ be any curves in these sets. Note that if $d_F(P_v, Q_u) < 2$, then if the traversal is a distance 2 into the gadget G_u^i, then we also have to be in the gadget G_v^i, as there is no other gadget in distance less than 2. The same statement holds for G_u^i and G_v^i exchanged. Thus, we traverse the gadgets synchronously. Now consider the case $\langle u, v \rangle = 0$. As $d_F(0_U, 0_V) = d_F(0_U, 1_V) = d_F(1_U, 0_V) = 1$, also $d_F(P_v, Q_u) = 1$, as there is no $i \in \{0, \dots, D-1\}$ for which the gadget G_u^i is of type 1_U and G_v^i is of type 1_V. Conversely, consider the case

$\langle u, v\rangle = 1$. Then there exist an $i \in \{0, \ldots, D-1\}$ such that G_u^i is of type 1_U and G_v^i is of type 1_V. As we traverse the gadgets synchronously and as $\mathrm{d_F}(G_v^i, G_u^i) = 2$, we have $\mathrm{d_F}(P_v, Q_u) = 2$. Thus, if we have a $(2-\varepsilon)$-ANN, then we can use it to check if there exist orthogonal vectors $u \in U$ and $v \in V$ by the above reduction.

It remains to show that this reduction implies the claimed lower bound. The time to compute the reduction is linear in the output size and thus negligible. Recall that $\mathcal{P}$ is the input set, i.e., it is the set that we preprocess, and we run a query for each curve in $\mathcal{Q}$. Note that by the construction of the above reduction we have $|\mathcal{P}| = |U|^\alpha$, and $|\mathcal{Q}| = |U|$. Towards a contradiction, assume that we can solve $(2-\varepsilon)$-ANN with preprocessing time $\mathcal{O}(|\mathcal{P}|^{\alpha'})$ for some $\alpha' > 0$ and query time $\mathcal{O}(|\mathcal{P}|^{1-\varepsilon'})$ for some $\varepsilon' > 0$. Choosing $\alpha = 1/\alpha'$, we obtain preprocessing time $\mathcal{O}(|\mathcal{P}|^{\alpha'}) = \mathcal{O}(|U|^{\alpha\alpha'}) = \mathcal{O}(|U|)$ and total query time

$$\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{P}|^{1-\varepsilon'}) = \mathcal{O}(|U| \cdot |U|^{\alpha(1-\varepsilon')}) = \mathcal{O}(|U|^{1+\alpha-\varepsilon'\alpha}).$$

Thus, we could solve $\textsc{OneSidedSparseOV}(k)$ in time $\mathcal{O}(|U|^{1+\alpha-\varepsilon'\alpha})$. However, by Lemma 7.2, there exists a $c > 0$ such that this contradicts OVH. □

7.3 Hardness of $(3-\varepsilon)$-Approximation in 1D

We now show the first of two hardness results that rule out certain preprocessing and query times for $(3-\varepsilon)$-approximations. Note that ruling out higher approximation ratios is not possible using gadgets that encode the single coordinates, as the distance between the gadgets that encode 1-entries can be at most 3 times the threshold distance due to the triangle inequality between the other gadgets, for details see [56]. For one-dimensional curves we obtain the following lower bound. We note that the gadgets that we use to encode our vectors are inspired by [56].

Theorem 7.4. *Assume OVH holds true. For any $\varepsilon, \varepsilon' > 0$ there is a $c > 0$, such that there is no $(3-\varepsilon)$-ANN for the continuous Fréchet distance storing n one-dimensional curves of complexity m and supporting query curves of complexity k with $m = k = c\log n$ such that we have preprocessing time poly(n) and query time $\mathcal{O}(n^{1-\varepsilon'})$.*

Proof. We show the hardness by a reduction from OV. To that end, let $U, V \subset \{0,1\}^D$ be an OV instance with $|V| = |U|^\alpha$ for a constant $\alpha \leq 1$ that we specify later and $D = c\log|U|$ for a constant $c > 0$ that we later choose sufficiently large. Recall that, by Lemma 2.9, there exists a $c > 0$ such that this problem is OV-hard. We now create the input set $\mathcal{P}$ and query set $\mathcal{Q}$ of a $(3-\varepsilon)$-ANN instance with distance threshold $\delta = 1$ as follows. For convenience, we define the curves

$$1_U \coloneqq \langle 0, 6, 0\rangle, \quad 0_V \coloneqq \langle 0, 7, 0\rangle, \quad 0_U \coloneqq \langle 0, 8, 0\rangle, \quad 1_V \coloneqq \langle 0, 9, 0\rangle.$$

First, for each vector $u \in U$ we create a new curve $Q_u \in \mathcal{Q}$ defined as

$$Q_u \coloneqq \bigcirc_{i=0}^{D-1} G_u^i \quad \text{with} \quad G_u^i \coloneqq u[i]_U,$$

where $u[i]_U$ is either 1_U or 0_U, depending on the value of $u[i]$. Second, for each vector $v \in V$ we create a new curve $P_v \in \mathcal{P}$ defined as

$$P_v \coloneqq \bigcirc_{i=0}^{D-1} G_v^i \quad \text{with} \quad G_v^i \coloneqq v[i]_V.$$

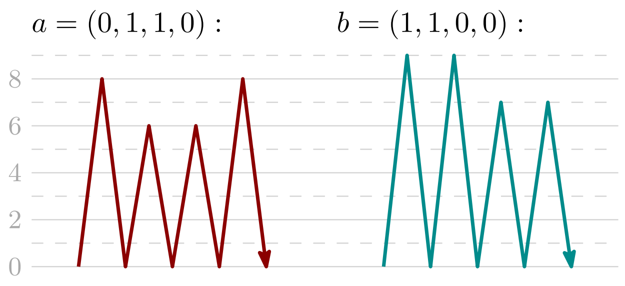


Figure 7.2: Visualization of the $3-\varepsilon$ lower bound in 1D.

where $v[i]_V$ is either 0_V or 1_V, depending on the value of $v[i]$. See Figure 7.2 for examples of these curves.

We now prove the correctness of the reduction. Consider any $Q_u \in \mathcal{Q}$ and $P_v \in \mathcal{P}$. We first show that if $d_F(Q_u, P_v) < 3$, then any traversal realizing this distance has to visit vertices of both curves synchronously. More precisely, a traversal can be in the gadgets G_u^i and G_v^j with $i \neq j$ only if the positions on both curves are strictly less than 6 in image space. Towards a contradiction, consider the first point in the traversal where this occurs and without loss of generality let the traversal be at position 6 in Q_u. As the traversal on Q_u visited 0 before, the traversal on P_v has to be below 3 and thus the positions on Q_u and P_v are within distance more than 3, which is a contradiction. Thus, when traversing gadgets G_u^i and G_v^j above 6, then $i = j$.

We now proceed with showing that for all $u \in U$ and $v \in V$ it holds that $d_F(Q_u, P_v) \leq 1$ if and only if $\langle u, v\rangle = 0$, and $d_F(Q_u, P_v) \geq 3$ otherwise. Assume that $\langle u, v\rangle = 0$, then, by traversing all G_u^i, G_v^i for $i \in \{0, \dots, D-1\}$ synchronously, they can always stay within distance at most 1, as $d_F(0_U, 0_V) = d_F(0_U, 1_V) = d_F(1_U, 0_V) = 1$. However, if $\langle u, v\rangle > 0$, then there exists an index $i \in \{0, \dots, D-1\}$ such that $u[i] = v[i] = 1$. If $d_F(Q_u, P_v) < 3$, then we have to traverse these G_u^i and G_v^i synchronously but as $d_F(1_U, 1_V) = 3$, there is a point in the traversal where the curves have distance at least 3 and thus $d_F(Q_u, P_v) \geq 3$. It follows that, if we have a $(3-\varepsilon)$-ANN, then it would find if there exists orthogonal vectors in U and V by querying each $Q \in \mathcal{Q}$.

Let us now show that this implies the desired lower bounds. The time to compute the reduction is linear in the output size and thus negligible. Note that by construction we have $m = k = \mathcal{O}(c \log |U|) \leq c' \log |U|$ for some constant $c' > 0$. By adding dummy vertices, say many points close to the starting point, we can ensure $m = k = c' \log |U|$ (we could also achieve any intended value $m \geq k$, but this is not necessary for the theorem statement). Moreover, $|\mathcal{P}| = |U|^\alpha$ and $|\mathcal{Q}| = |U|$. Towards a contradiction, assume that we can solve $(3-\varepsilon)$-ANN with preprocessing time $\mathcal{O}(|\mathcal{P}|^{\alpha'})$ for some $\alpha' > 0$ and query time $\mathcal{O}(|\mathcal{P}|^{1-\varepsilon'})$ for some $\varepsilon' > 0$. Choosing $\alpha = 1/\alpha'$, we obtain preprocessing time $\mathcal{O}(|\mathcal{P}|^{\alpha'}) = \mathcal{O}(|U|^{\alpha\alpha'}) = \mathcal{O}(|U|)$ and total query time

$$\mathcal{O}(|\mathcal{Q}| \cdot |\mathcal{P}|^{1-\varepsilon'}) = \mathcal{O}(|U| \cdot |U|^{\alpha(1-\varepsilon')}) = \mathcal{O}(|U|^{1+\alpha-\varepsilon'\alpha}).$$

Thus, we could solve unbalanced OV in time $\mathcal{O}(|U|^{1+\alpha-\varepsilon'\alpha})$. However, by Lemma 2.9, there exists a $c > 0$ such that this contradicts OVH. $\square$

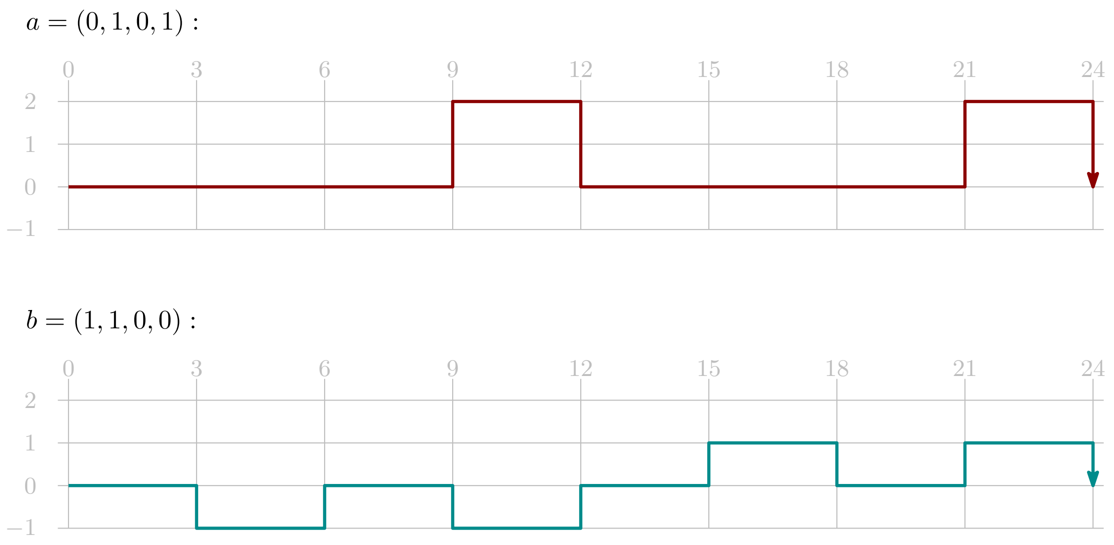


Figure 7.3: Visualization of the $3 - \varepsilon$ lower bound in 2D.

7.4 Hardness of $(3 - \varepsilon)$-Approximation in 2D

While until here we only considered algorithmic and hardness results for one-dimensional curves, we now show a hardness result for *two-dimensional* curves. This is the only technical section in this chapter where we consider two-dimensional curves. Note that in Chapter 2 we defined most of our notation for curves in $\mathbb{R}^d$ and thus the notation of the previous hardness results carries over. For two-dimensional curves we obtain the following lower bound.

Theorem 7.5. *Assume OVH holds true. For any $\varepsilon, \varepsilon' > 0$ there is a $c > 0$, such that there is no $(3 - \varepsilon)$-ANN for the continuous Fréchet distance supporting query curves of any complexity $k \in \omega(1) \cap o(\log n)$ and storing n two-dimensional curves of complexity $m = k \cdot n^{c/k}$ with preprocessing time poly(n) and query time $\mathcal{O}(n^{1-\varepsilon'})$.*

Proof. This proof is very similar to the proof of Theorem 7.3. The significant difference is the gadgets that we construct. To this end, consider a OneSidedSparseOV(k) instance U, V, where we again use the k-sparsity of the vectors in U to obtain short query curves of length $\mathcal{O}(k)$. We define the generic subcurve

$$G(y) := \langle (0, 0), (3, 0), (3, y), (6, y), (6, 0) \rangle$$

to then define the usual gadgets

$$0_U := G(0), \quad 1_U := G(2), \quad 0_V := G(1), \quad 1_V := G(-1).$$

Now, given a OneSidedSparseOV(k) instance U, V, we create the input set $\mathcal{P}$ and query set $\mathcal{Q}$ of a $(3 - \varepsilon)$-ANN with distance threshold $\delta = 1$ as follows. For each vector $u \in U$, we add the curve Q_u to $\mathcal{Q}$ which is defined as

$$Q_u := \bigcirc_{i=0}^{D-1} G_u^i \quad \text{with} \quad G_u^i := u[i]_U + (6i, 0),$$

where $u[i]_U$ is either 0_U or 1_U, depending on the value of $u[i]$, and the "$+(6i,0)$" is a translation of each point of the curve by $(6i,0)$. For each vector $v \in V$, we add the curve P_v to $\mathcal{P}$ which is defined as

$$P_v \coloneqq \bigcirc_{i=0}^{D-1} G_v^i \quad \text{with} \quad G_v^i \coloneqq v[i]_V + (6i,0).$$

where $v[i]_V$ is either 0_V or 1_V, depending on the value of $v[i]$. It is crucial that we make the resulting curves non-degenerate by removing all degenerate vertices. In particular, any sequence of consecutive gadgets 0_U will be turned into a single line segment. Thus, the curves in $\mathcal{Q}$ will have complexity $\mathcal{O}(k)$. See Figure 7.3 for an example of the construction.

We now prove correctness of the reduction. Consider the case of two orthogonal vectors $u \in U$ and $v \in V$ such that there is an $i \in \{0,\dots,D-1\}$ with $u[i] = v[i] = 1$. Note that for $\mathrm{d_F}(Q_u, P_v) < 3$, there has to be a point in the traversal where we are in some point $(x_u, 2)$ in Q_u and in some point $(x_v, -1)$ in P_v as otherwise the distance of the x-coordinate would be at least 3. However, the y-distance of these points is 3 and thus $\mathrm{d_F}(Q_u, P_v) \geq 3$. On the other hand, if $u \in U$ and $v \in V$ are orthogonal, then we can traverse the two curves with the same speed in x-direction — i.e., staying at the same x-coordinate at every point in time — and obtain a Fréchet distance at most 1 as $\mathrm{d_F}(0_U, 0_V) = \mathrm{d_F}(0_U, 1_V) = \mathrm{d_F}(1_U, 0_V) = 1$, where the described traversal realizes these distances.

The remainder of the proof, i.e., the derivation of the claimed lower bound, is the same as in the proof of Theorem 7.3 and we thus omit it for brevity. $\square$

CHAPTER 8

Fréchet Distance Under Translation

For a technical overview of this chapter see Section 3.2. In this section we prove a lower bound of $n^{4-o(1)}$ for the discrete Fréchet distance under translation for two curves of length n in the Euclidean plane conditional on the Strong Exponential Time Hypothesis, or more precisely the 4-OV Hypothesis. To this end, we reduce 4-OV to the discrete Fréchet distance under translation.

8.1 Overview of Reduction

Let us first have a closer look at 4-OV. Given four sets of N vectors $V_1, \ldots, V_4 \subseteq \{0,1\}^D$, the 4-OV problem can be expressed as

$$\exists v_1 \in V_1, \ldots, v_4 \in V_4 \; \forall j \in [D] \; \exists i \in \{1, \ldots, 4\} : v_i[j] = 0. \tag{8.1}$$

Recall from the introduction that we encode choosing the vectors $v_1, \ldots, v_4$ by the canonical translation $\tau = (\tau_1, \tau_2) = (h_1 \cdot \varepsilon, h_2 \cdot \varepsilon)$ with $h_1, h_2 \in \{0, \ldots, N^2 - 1\}$ for some constant $\varepsilon > 0$ which is sufficiently small. To be concrete, let

$$\varepsilon \coloneqq 0.001/N^4$$

for the remaining section. Choosing $v_1 \in V_1$ and $v_2 \in V_2$, we define

$$h_1 \coloneqq h(v_1, v_2) \coloneqq \mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N,$$

where $\mathrm{ind}(v_i)$ is the index of vector v_i in the set V_i; similarly for $v_3 \in V_3, v_4 \in V_4$ we define $h_2 \coloneqq h(v_3, v_4)$. To perform the reduction, we want to construct two curves π and σ whose discrete Fréchet distance decision for some δ is equivalent to the following expression, which is equivalent to (8.1):

$$\exists \tau \in \mathbb{R}^2 \; \forall j \in [D] \; \exists i \in \{1,2\}, v \in V_{2i-1}, v' \in V_{2i} : (v[j] = 0 \vee v'[j] = 0) \wedge (h(v,v') \cdot \varepsilon = \tau_i). \tag{8.2}$$

The expressions (8.1) and (8.2) are equivalent as the three quantifiers encode the same choices and we evaluate if there exists a zero in one of the chosen vectors. In (8.2) we additionally need to make sure that the translation chosen by the outermost quantifier indeed is consistent with the vectors that are chosen by the innermost quantifier, which is done by requiring $h(v,v') \cdot \varepsilon = \tau_i$.

We can further transform this expression to make it easier to create gadgets for the reduction:

$$\exists \tau \in [0, (N^2-1) \cdot \varepsilon] \times [0, (N^2-1) \cdot \varepsilon] : \bigwedge_{j \in [D]} \; \bigvee_{\substack{i \in \{1,2\} \\ v \in V_{2i-1}, v' \in V_{2i} : \\ v[j]=0 \text{ or } v'[j]=0}} [h(v,v') \cdot \varepsilon = \tau_i].$$

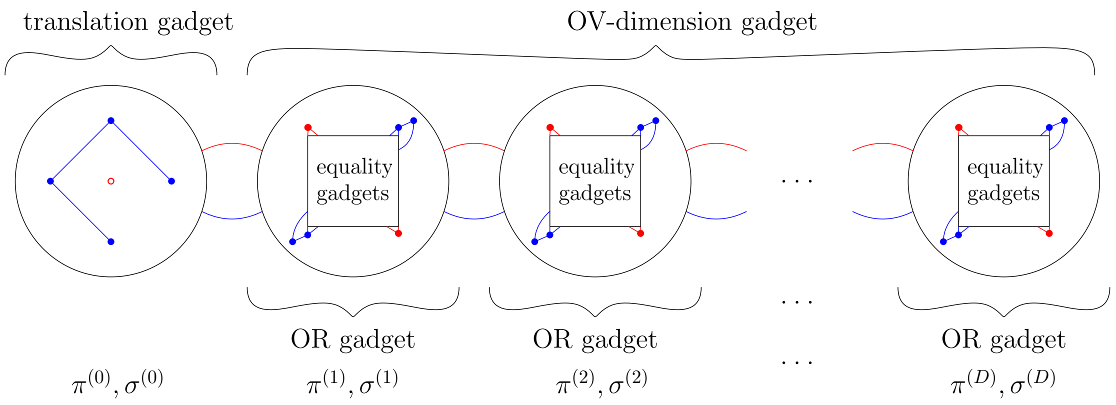


Figure 8.1: Overview of how the different gadgets are used in the curves that result from the reduction. We use one translation gadget, one OV-dimension gadget, D OR gadgets, and $\mathcal{O}(ND)$ equality gadgets.

According to this formula, we will construct gadgets. However, we cannot exactly ensure the equality $h(v, v') \cdot \varepsilon = \tau_i$. Therefore, we resort to an approximate equality which still fulfills the intended usage of mapping translations to vector choices. The approximate values just snap to the closest canonical values. The gadgets we construct are the following:

- *Translation gadget:* It ensures that $\tau \in [-\frac{1}{4} \cdot \varepsilon, (N^2 - \frac{3}{4}) \cdot \varepsilon] \times [-\frac{1}{4} \cdot \varepsilon, (N^2 - \frac{3}{4}) \cdot \varepsilon]$, i.e., we are always close to the points in the ε-grid of translations that choose our vectors $v_1, \ldots, v_4$.
- *OV-dimension gadget:* AND over all $j \in [D]$.
- *OR gadget:* The big OR in the formula.
- *Equality gadget:* This gadget is only traversable if the two vectors it was created for correspond to τ, i.e., it ensures that $h(v, v') \cdot \varepsilon \approx \tau_i$.

We use the above mentioned gadgets as follows. The constructed curves π and σ start with the translation gadget consisting of the curves $\pi^{(0)}, \sigma^{(0)}$. They are followed by D different parts that form the OV-dimension gadget. Each of the D parts is an OR gadget and we call the respective curves $\pi^{(j)}$ and $\sigma^{(j)}$ for $j \in [D]$. Each of the OR gadgets $(\pi^{(j)}, \sigma^{(j)})$ contains several equality gadgets. We will use different variations of the equality gadget (one for each set of vectors $V_1, \ldots, V_4$) but they are all of very similar structure. We need four different types of equality gadgets because for a certain $v_i \in V_i$ a part of the gadget is only inserted if $v_i[d] = 0$. Thus, if we traverse an equality gadget later, we know that it corresponds to one zero entry and also to the current translation. See Figure 8.1 for an overview of the whole construction.

8.2 Gadgets

Without loss of generality, assume that for all dimensions $j \in [D]$ at least one vector in $V_1 \cup \cdots \cup V_4$ contains a 0 in dimension j. Now we give the detailed construction of the gadgets and the proofs of correctness. The instance of the discrete Fréchet distance under translation that we construct in the reduction uses a threshold distance of $\delta = 2 + \frac{1}{4}\varepsilon$, i.e., we want to know for the constructed curves π and σ if their discrete Fréchet distance under translation is not more than δ.

Translation Gadget. First we have to restrict the possible translations. To this end, we build a gadget to ensure $\tau \in \mathcal{T}$ for

$$\mathcal{T} := [-\frac{1}{4} \cdot \varepsilon, (N^2 - \frac{3}{4}) \cdot \varepsilon] \times [-\frac{1}{4} \cdot \varepsilon, (N^2 - \frac{3}{4}) \cdot \varepsilon].$$

This is realized by a gadget where curve $\pi^{(0)}$ consists of only one vertex and curve $\sigma^{(0)}$ consists of four vertices:

$$\pi^{(0)} := \langle (0,0) \rangle,$$
$$\sigma^{(0)} := \langle (2 - (N^2 - 1)\varepsilon, 0), (0, 2 - (N^2 - 1)\varepsilon), (-2, 0), (0, -2) \rangle.$$

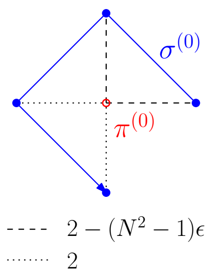


Figure 8.2: Translation gadget

This gadget is sketched in Figure 8.2.

Lemma 8.1. *Given two curves $\pi = \pi^{(0)} \circ \pi'$ and $\sigma = \sigma^{(0)} \circ \sigma'$ (with $\pi^{(0)}, \sigma^{(0)}$ as defined above), such that each $p \in \pi^{(0)}$ has distance greater than 10 to each $p' \in \pi'$, the following holds:*

(i) if $\tau \in [0, (N^2 - 1)\varepsilon] \times [0, (N^2 - 1)\varepsilon]$, then $\mathrm{d_{dF}}(\pi^{(0)}, \sigma^{(0)} + \tau) \le \delta$

(ii) if $\mathrm{d_{dF}}(\pi, \sigma + \tau) \le \delta$, then $\tau \in [-\frac{1}{4} \cdot \varepsilon, (N^2 - \frac{3}{4}) \cdot \varepsilon] \times [-\frac{1}{4} \cdot \varepsilon, (N^2 - \frac{3}{4}) \cdot \varepsilon]$

Proof. We start with showing (i), so assume $\tau \in [0, (N^2-1)\varepsilon] \times [0, (N^2-1)\varepsilon]$. Note that the maximal distance $\max_{q \in \sigma^{(0)}} \max_\tau \left\| \pi^{(0)} - (q + \tau) \right\|$ is an upper bound on $\mathrm{d_{dF}}(\pi^{(0)}, \sigma^{(0)} + \tau)$. By a simple calculation we obtain the desired result:

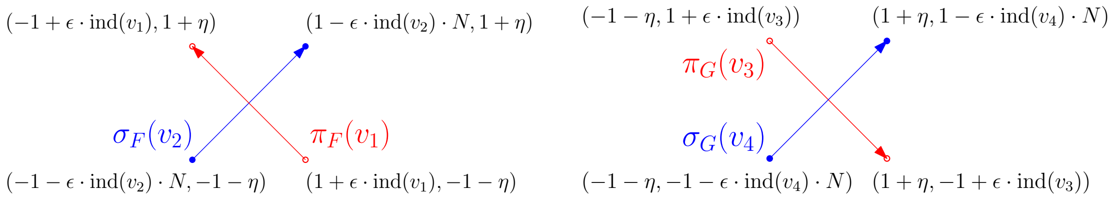
$$\max_{q \in \sigma^{(0)}} \max_{\tau} \left\| \pi^{(0)} - (q + \tau) \right\| < \sqrt{2^2 + \varepsilon^2 N^4} < \sqrt{2^2 + \varepsilon + \frac{1}{16}\varepsilon^2} = 2 + \frac{1}{4}\varepsilon,$$

where we used $\varepsilon \le N^{-4}$.

Now we prove (ii). Note that the start points of π and $\sigma + \tau$ have to be in distance $\le \delta$, thus $\tau \in [-5, 5]^2$ (using a very rough estimate). As all points of π' are further than 10 from any point in $\pi^{(0)}$ and thus all points on the postfix π' are further than δ from $\sigma^{(0)} + \tau$, we have to stay in $\pi^{(0)}$ while traversing $\sigma^{(0)}$. Thus, the following inequalities hold for $\tau_i > (N^2 - \frac{3}{4})\varepsilon$ or $\tau_i < -\frac{1}{4}\varepsilon$ and $i \in \{1, 2\}$ (where $\|v\|_\infty$ denotes the infinity norm of v):

$$\mathrm{d_{dF}}(\pi, \sigma + \tau) \ge \mathrm{d_{dF}}(\pi^{(0)}, \sigma^{(0)} + \tau) \ge \max_{i \in [4]} \left\{ \left\| \pi_1^{(0)} - (\sigma_i^{(0)} + \tau) \right\|_\infty \right\} > \delta,$$

which is the contrapositive of (ii). $\square$


$(-1+\epsilon\cdot\mathrm{ind}(v_1), 1+\eta)$
$(1-\epsilon\cdot\mathrm{ind}(v_2)\cdot N, 1+\eta)$
$\sigma_F(v_2)$
$\pi_F(v_1)$
$(-1-\epsilon\cdot\mathrm{ind}(v_2)\cdot N, -1-\eta)$
$(1+\epsilon\cdot\mathrm{ind}(v_1), -1-\eta)$
$(-1-\eta, 1+\epsilon\cdot\mathrm{ind}(v_3))$
$(1+\eta, 1-\epsilon\cdot\mathrm{ind}(v_4)\cdot N)$
$\pi_G(v_3)$
$\sigma_G(v_4)$
$(-1-\eta, -1-\epsilon\cdot\mathrm{ind}(v_4)\cdot N)$
$(1+\eta, -1+\epsilon\cdot\mathrm{ind}(v_3))$

(a) Equality Gadget for V_1 and V_2 **(b)** Equality Gadget for V_3 and V_4

Figure 8.3: The equality gadgets for F and G. The equality gadgets F' and G' are simply shifted.

OV-dimension Gadget. For every 4-OV dimension $j \in [D]$, we construct separate gadgets $\pi^{(1)}, \dots, \pi^{(D)}$ for π and $\sigma^{(1)}, \dots, \sigma^{(D)}$ for σ. We want to connect these gadgets in a way that the two curves are in distance not more than δ if and only if all gadgets have distance not more than δ for a given translation τ. This is done by simply placing the gadgets in distance greater than $\delta + N^2 \cdot \varepsilon$ from each other and concatenating them.

Lemma 8.2. *Given a translation $\tau \in \mathcal{T}$ and curves $\pi = \pi^{(1)}, \dots, \pi^{(D)}$ and $\sigma = \sigma^{(1)}, \dots, \sigma^{(D)}$ where for all $j \in [D]$ all points of $\pi^{(j)}$ are further than $\delta + 2N^2 \cdot \varepsilon$ from each point of $\sigma^{(j')}$ with $j \neq j'$, then $\delta(\pi, \sigma + \tau) \leq \delta$ if and only if $\mathrm{d_{dF}}(\pi^{(j)}, \sigma^{(j)} + \tau) \leq \delta$ for all $j \in [D]$.*

Proof. First, note that whatever τ we choose in the given range, $\sigma^{(j)} + \tau$ is still in distance greater than δ from every $\pi^{(j')}$ with $j' \neq j$.

Now, assume that for all $j \in [D]$ the curves $\pi^{(j)}, \sigma^{(j)} + \tau$ have distance at most δ. Then we can traverse the gadgets in order and do simultaneous jumps between them. Thus, also the distance of the whole curves π and $\sigma + \tau$ is at most δ. For the other direction, assume that for at least one $j \in [D]$ the distance between $\pi^{(j)}$ and $\sigma^{(j)} + \tau$ is greater than δ. On the one hand, if we do not traverse simultaneously (i.e., at one point the traversal is in $\pi^{(j)}$ and $\sigma^{(j')}$ for $j \neq j'$), then due to large distances of $\pi^{(j)}, \sigma^{(j')} + \tau$ for $j \neq j'$, we have distance greater than δ for this traversal. On the other hand, if we traverse $\pi^{(j)}$ and $\sigma^{(j)}$ together for all j, we also have distance greater than δ due to the gadget with distance greater than δ. □

For the remaining gadgets we define for convenience:

$$\eta := 3 \cdot N^2 \varepsilon.$$

Equality Gadget. An equality gadget $F(v_1, v_2)$ for the vectors $v_1 \in V_1, v_2 \in V_2$ is a pair of two line segments, $\pi_F(v_1)$ and $\sigma_F(v_2)$, see Figure 8.3(a):

$$\pi_F(v_1) := \langle (1 + \varepsilon \cdot \mathrm{ind}(v_1), -1 - \eta), (-1 + \varepsilon \cdot \mathrm{ind}(v_1), 1 + \eta) \rangle,$$
$$\sigma_F(v_2) := \langle (-1 - \varepsilon \cdot \mathrm{ind}(v_2) \cdot N, -1 - \eta), (1 - \varepsilon \cdot \mathrm{ind}(v_2) \cdot N, 1 + \eta) \rangle.$$

Note that this gives us N^2 different gadgets consisting of $2N$ different line segments. We later use the line segments $\pi_F(v_1)$ in π and the line segments $\sigma_F(v_2)$ in σ where they can be combined to form an equality gadget.

Lemma 8.3. *Given curves $\pi_F(v_1), \sigma_F(v_2)$ for some $v_1 \in V_1$ and $v_2 \in V_2$, and given a translation $\tau \in \mathcal{T}$, the following properties hold:*

(i) if $\tau_1 = \varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N)$, then $\mathrm{d_{dF}}(\pi_F(v_1), \sigma_F(v_2) + \tau) \le \delta$

(ii) if $\mathrm{d_{dF}}(\pi_F(v_1), \sigma_F(v_2) + \tau) \le \delta$, then $|\varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) - \tau_1| \le \frac{1}{3}\varepsilon$

Proof. To prove (i), it suffices to give a valid traversal. We traverse $\pi_F(v_1) = (p_1, p_2)$ and $\sigma_F(v_2) = (q_1, q_2)$ simultaneously. Thus, we just want an upper bound on the distance between the (translated) first points $p_1, q_1 + \tau$ and the distance between the (translated) second points $p_2, q_2 + \tau$ to get an upper bound on $\mathrm{d_{dF}}(\pi_F(v_1), \sigma_F(v_2) + \tau)$. These distances are

$$\begin{aligned}\|p_1 - (q_1 + \tau)\|^2 &= (2 + \varepsilon \cdot \mathrm{ind}(v_1) + \varepsilon \cdot \mathrm{ind}(v_2) \cdot N - \tau_1)^2 + \tau_2^2 \\ &= 4 + \tau_2^2 \le 4 + \varepsilon + \frac{1}{16}\varepsilon^2 = \delta^2\end{aligned}$$

and

$$\|p_2 - (q_2 + \tau)\|^2 = (-2 + \varepsilon \cdot \mathrm{ind}(v_1) + \varepsilon \cdot \mathrm{ind}(v_2) \cdot N - \tau_1)^2 + \tau_2^2 = 4 + \tau_2^2 \le \delta^2,$$

where we used $|\tau_2| \le N^2\varepsilon$ and thus $\tau_2^2 \le N^4\varepsilon^2 \le \varepsilon$ since $\varepsilon \le N^{-4}$. Both distances are at most δ and thus the discrete Fréchet distance is at most δ as well.

For proving (ii), first note that the first (respectively second) point of $\pi_F(v_1)$ is far from the second (respectively first) point of $\sigma_F(v_2)$, due to $\eta \ge N^2\varepsilon$. Thus, we have to traverse the gadget simultaneously. Let $\Delta := \varepsilon \cdot \mathrm{ind}(v_1) + \varepsilon \cdot \mathrm{ind}(v_2) \cdot N - \tau_1$, it remains to show that $\Delta \le \frac{1}{3}\varepsilon$. For p_1, q_1 we then get

$$\begin{array}{rrcl} \|p_1 - (q_1 + \tau)\|^2 = & (2 + \varepsilon \cdot \mathrm{ind}(v_1) + \varepsilon \cdot \mathrm{ind}(v_2) \cdot N - \tau_1)^2 + \tau_2^2 & \le & (2 + \frac{1}{4}\varepsilon)^2 \\ \Leftrightarrow & (2 + \Delta)^2 + \tau_2^2 & \le & 4 + \varepsilon + \frac{1}{16}\varepsilon^2 \\ \Leftrightarrow & 4 + 4\Delta + \Delta^2 + \tau_2^2 & \le & 4 + \varepsilon + \frac{1}{16}\varepsilon^2 \\ \Rightarrow & 4\Delta & \le & \varepsilon + \frac{1}{16}\varepsilon^2 \\ \Rightarrow & \Delta & \le & \frac{1}{4}\varepsilon + \frac{1}{64}\varepsilon^2 \le \frac{1}{3}\varepsilon. \end{array}$$

The last inequality follows from plugging in $\varepsilon = 0.001/N^4$ and using the fact that $N \ge 1$. With a similar calculation for p_2, q_2 we obtain that $\Delta \ge -\frac{1}{3}\varepsilon$, and thus $|\Delta| \le \frac{1}{3}\varepsilon$. □

Now we introduce three gadgets which have the same properties as the equality gadget but are slightly different. The aim is to have four types of gadgets which are pairwise further than a discrete Fréchet distance of δ apart such that we can use them together in one big OR expression.

Shifted Equality Gadget. As described in the introduction of this section, we want to use the curves $\pi_F(v_1), \sigma_F(v_2)$ in case $v_1[j] = 0$ and we need an additional gadget for $v_2[j] = 0$. However, those two gadgets should not be too close such that the curves cannot be matched but also not too far such that the OR gadget (which we introduce later) still works. Thus, we introduce another gadget $F'(v_1, v_2)$ which consists of a pair of curves

$\pi_{F'}(v_1), \sigma_{F'}(v_2)$ that are just shifted versions of $\pi_F(v_1), \sigma_F(v_2)$; shifted by $N^2\varepsilon$ in the first dimension. More formally,

$$\pi_{F'}(v_1) \coloneqq \pi_F(v_1) + (N^2\varepsilon, 0),$$
$$\sigma_{F'}(v_2) \coloneqq \sigma_F(v_2) + (N^2\varepsilon, 0).$$

Before proving the desired properties, we introduce the remaining two variants of the equality gadget.

Equality Gadget for V_3 and V_4. The above introduced equality gadgets only work for vectors in V_1 and V_2 but we also need a gadget for vectors in V_3 and V_4. Therefore, we introduce the gadget $G(v_3, v_4)$, which is a mirrored equality gadget consisting of the curves $\pi_G(v_3)$ and $\sigma_G(v_4)$, see Figure 8.3(b):

$$\pi_G(v_3) \coloneqq \langle(-1-\eta, 1+\varepsilon\cdot\mathrm{ind}(v_3)), (1+\eta, -1+\varepsilon\cdot\mathrm{ind}(v_3))\rangle,$$
$$\sigma_G(v_4) \coloneqq \langle(-1-\eta, -1-\varepsilon\cdot\mathrm{ind}(v_4)\cdot N), (1+\eta, 1-\varepsilon\cdot\mathrm{ind}(v_4)\cdot N)\rangle.$$

Shifted Equality Gadget for V_3 and V_4. We define $G'(v_3, v_4)$ similarly to $F'(v_1, v_2)$, i.e., we shift the curves of G by $N^2\varepsilon$, but in contrast to F' we shift it in the second dimension. More formally:

$$\pi_{G'}(v_3) \coloneqq \pi_G(v_3) + (0, N^2\varepsilon),$$
$$\sigma_{G'}(v_4) \coloneqq \sigma_G(v_4) + (0, N^2\varepsilon).$$

Due to the similar structure of the curve pairs of $F(v_1, v_2)$ and $F'(v_1, v_2)$, $G(v_3, v_4)$, $G'(v_3, v_4)$, analogous statements to Lemma 8.3 also hold for the curve pairs from $F'(v_1, v_2)$, $G(v_3, v_4)$, and $G'(v_3, v_4)$. Specifically, we have:

Lemma 8.4. *Given curves $\pi_{F'}(v_1), \sigma_{F'}(v_2)$ for some $v_1 \in V_1$ and $v_2 \in V_2$, and given a translation $\tau \in \mathcal{T}$, the following properties hold:*

(i) if $\tau_1 = \varepsilon\cdot(\mathrm{ind}(v_1) + \mathrm{ind}(v_2)\cdot N)$, then $\mathrm{d_{dF}}(\pi_{F'}(v_1), \sigma_{F'}(v_2) + \tau) \le \delta$

(ii) if $\mathrm{d_{dF}}(\pi_{F'}(v_1), \sigma_{F'}(v_2) + \tau) \le \delta$, then $|\varepsilon\cdot(\mathrm{ind}(v_1) + \mathrm{ind}(v_2)\cdot N) - \tau_1| \le \frac{1}{3}\varepsilon$

Lemma 8.5. *Given curves $\pi_G(v_3), \sigma_G(v_4)$ for some $v_3 \in V_3$ and $v_4 \in V_4$, and given a translation $\tau \in \mathcal{T}$, the following properties hold:*

(i) if $\tau_2 = \varepsilon\cdot(\mathrm{ind}(v_3) + \mathrm{ind}(v_4)\cdot N)$, then $\mathrm{d_{dF}}(\pi_G(v_3), \sigma_G(v_4) + \tau) \le \delta$

(ii) if $\mathrm{d_{dF}}(\pi_G(v_3), \sigma_G(v_4) + \tau) \le \delta$, then $|\varepsilon\cdot(\mathrm{ind}(v_3) + \mathrm{ind}(v_4)\cdot N) - \tau_2| \le \frac{1}{3}\varepsilon$

Lemma 8.6. *Given curves $\pi_{G'}(v_3), \sigma_{G'}(v_4)$ for some $v_3 \in V_3$ and $v_4 \in V_4$, and given a translation $\tau \in \mathcal{T}$, the following properties hold:*

(i) if $\tau_2 = \varepsilon\cdot(\mathrm{ind}(v_3) + \mathrm{ind}(v_4)\cdot N)$, then $\mathrm{d_{dF}}(\pi_{G'}(v_3), \sigma_{G'}(v_4) + \tau) \le \delta$

(ii) if $\mathrm{d_{dF}}(\pi_{G'}(v_3), \sigma_{G'}(v_4) + \tau) \le \delta$, then $|\varepsilon\cdot(\mathrm{ind}(v_3) + \mathrm{ind}(v_4)\cdot N) - \tau_2| \le \frac{1}{3}\varepsilon$

We now show that all subcurves of different equality gadgets are pairwise further apart than δ. Here we say that the curves $\pi_F(v_1)$ and $\sigma_F(v_2)$ have *type* F. Similarly, the other curves constructed above have type F', G, or G'.

Lemma 8.7. *For any vectors $v_1 \in V_1, \dots, v_4 \in V_4$ and any translation $\tau \in \mathcal{T}$, for any curves $\pi \in \{\pi_F(v_1), \pi_{F'}(v_1), \pi_G(v_3), \pi_{G'}(v_3)\}$ and $\sigma \in \{\sigma_F(v_2), \sigma_{F'}(v_2), \sigma_G(v_4), \sigma_{G'}(v_4)\}$ of different type, we have $\mathrm{d_{dF}}(\pi, \sigma + \tau) > \delta$.*

Proof. We first consider $\pi_{F'}(v_1)$ and $\sigma_F(v_2)$. Consider the first point of $\sigma_F(v_2)$ which we call q. This point is further than $2 + N^2\varepsilon$ from both points of $\pi_{F'}(v_1)$. When translating σ with $\tau \in \mathcal{T}$, the distance is still greater than $2 + \frac{3}{4}\varepsilon > \delta$. Thus, $\sigma_F(v_2)$ and $\pi_{F'}(v_1)$ are in discrete Fréchet distance greater than δ for any valid τ.

Similarly, consider $\pi_F(v_1)$ and $\sigma_{F'}(v_2)$, and let p be the second point of $\pi_F(v_1)$. The point p has distance greater than $2+\varepsilon$ from $\sigma_{F'}(v_2)$. With translation $\tau \in \mathcal{T}$ this distance is still greater than $2+\frac{3}{4}\varepsilon > \delta$ and thus $\pi_F(v_1)$ and $\sigma_{F'}(v_2)$ are in discrete Fréchet distance greater than δ for any valid τ. The proof for types G and G' is symmetric.

Now we prove the lemma for types F and G. First note that every point of $\pi_F(v_1)$ is in distance $1 + \eta$ of the first coordinate axis and every point of $\sigma_G(v_4)$ is in distance $1 + \eta$ of the second coordinate axis. Additionally, no point of $\pi_F(v_1)$ is closer than $1 - 2N^2\varepsilon$ to the second coordinate axis while no point of $\sigma_G(v_4)$ is closer than $1 - 2N^2\varepsilon$ to the first coordinate axis. This means that every point of $\pi_F(v_1)$ is in distance at least $2 + \eta - 2N^2\varepsilon = 2 + N^2\varepsilon$ of any point of $\sigma_G(v_4)$. Even with translation this distance is at least $2 + \frac{3}{4}\varepsilon > \delta$. Thus, also the discrete Fréchet distance is greater than δ. The proofs for the remaining cases are symmetric. ☐

We moreover observe that our equality gadgets lie in very restricted regions. Specifically, call a curve *diagonal* if all of its vertices are in $R_1 \cup R_2$ with

$$R_1 := [-1 - 2\eta, -1 + 2\eta]^2, \; R_2 := [1 - 2\eta, 1 + 2\eta]^2,$$

and we call it *anti-diagonal* if all of its vertices are contained in $R_3 \cup R_4$ with

$$R_3 := [-1 - 2\eta, -1 + 2\eta] \times [1 - 2\eta, 1 + 2\eta], \; R_4 := [1 - 2\eta, 1 + 2\eta] \times [-1 - 2\eta, -1 + 2\eta].$$

See Figure 8.4. Also, note that the order in which the curves visit the regions is not specified in the definition of (anti-)diagonal.

Observation 8.8. *The curves $\pi_F(v_1), \pi_{F'}(v_1), \pi_G(v_3), \pi_{G'}(v_3)$ are anti-diagonal, and the curves $\sigma_F(v_2), \sigma_{F'}(v_2), \sigma_G(v_4), \sigma_{G'}(v_4)$ are diagonal.*

Proof. We observe that each coordinate of a vertex of any of these curves differs from 1 or -1 by at most $\varepsilon N^2 + \max\{\eta, \varepsilon N^2\}$, by bounding $0 \le \mathrm{ind}(v_i) \le N$. Recalling $\eta = 3 \cdot N^2\varepsilon$, we have $\varepsilon N^2 \le \eta$. Therefore, any coordinate differs from 1 or -1 by at most 2η, that is, each coordinate lies in $R_1 \cup R_2 \cup R_3 \cup R_4$. The general shape of being (anti-)diagonal can be inferred from Figure 8.3. ☐

We are now ready to describe the last gadget. For proving its correctness, we will essentially only use the diagonal and anti-diagonal property of the curves.

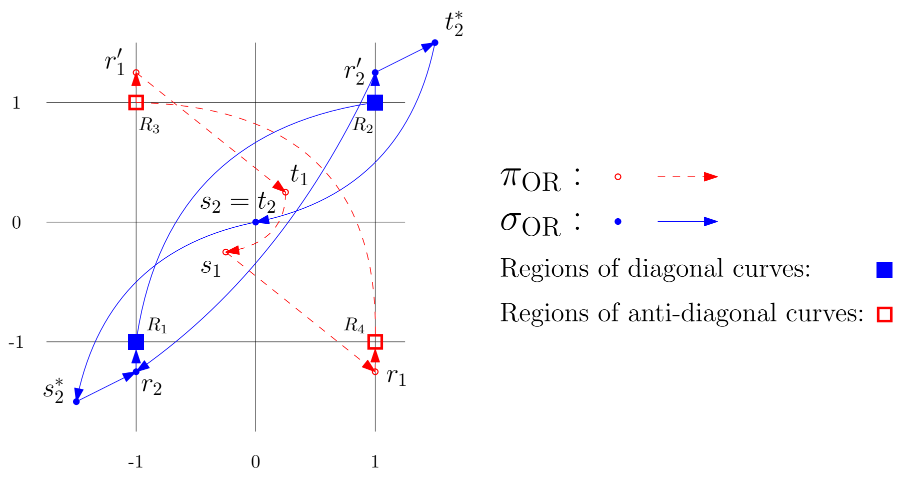


Figure 8.4: The OR gadget for general diagonal and anti-diagonal curves.

OR Gadget. We construct an OR gadget over diagonal and anti-diagonal curves which we will later apply to equality gadgets. Before introducing the gadget itself, we define various auxiliary points whose meaning will become clear later. Here we keep notation close to [38], although the details of our construction are quite different.

$$s_1 := \left(-\tfrac{1}{4}, -\tfrac{1}{4}\right), t_1 := \left(\tfrac{1}{4}, \tfrac{1}{4}\right), r_1 := \left(\tfrac{99}{100}, -\tfrac{5}{4}\right), r'_1 := \left(-\tfrac{99}{100}, \tfrac{5}{4}\right),$$
$$s_2 := (0,0), s_2^* := \left(-\tfrac{3}{2}, -\tfrac{3}{2}\right), t_2^* := \left(\tfrac{3}{2}, \tfrac{3}{2}\right), t_2 := (0,0), r_2 := \left(-\tfrac{99}{100}, -\tfrac{5}{4}\right), r'_2 := \left(\tfrac{99}{100}, \tfrac{5}{4}\right).$$

Now, given diagonal curves $\hat{\sigma}^1, \dots, \hat{\sigma}^\ell$ and anti-diagonal curves $\hat{\pi}^1, \dots, \hat{\pi}^k$, we define the two curves of the OR gadget as

$$\pi_{\mathrm{OR}} := \bigcirc_{i \in [k]} s_1 \circ r_1 \circ \hat{\pi}^i \circ r'_1 \circ t_1,$$
$$\sigma_{\mathrm{OR}} := s_2 \circ s_2^* \circ \Big(\bigcirc_{j \in [\ell]} r_2 \circ \hat{\sigma}^j \circ r'_2 \Big) \circ t_2^* \circ t_2.$$

See Figure 8.4 for a visualization. Now let us prove correctness of the gadget.

Lemma 8.9. *Given an OR gadget over diagonal curves $\hat{\sigma}^1, \dots, \hat{\sigma}^\ell$ and anti-diagonal curves $\hat{\pi}^1, \dots, \hat{\pi}^k$, for any translation $\tau \in \mathcal{T}$ we have $\mathrm{d_{dF}}(\pi_{\mathrm{OR}}, \sigma_{\mathrm{OR}} + \tau) \le \delta$ if and only if $\mathrm{d_{dF}}(\hat{\pi}^i, \hat{\sigma}^j + \tau) \le \delta$ for some i, j.*

Proof. We first observe that for none of the auxiliary points $p \in \pi_{\mathrm{OR}}$ and $q \in \sigma_{\mathrm{OR}}$ we have that $\|p - q\| \in [1.99, 2.01]$. This can be verified by calculating all distances, but we omit this due to readability of the proof. Also observe that $\mathcal{T} \subset [-0.001, 0.001]$ and $\delta \in [2, 2.001]$. It follows from the above observations that the translation τ does not change whether auxiliary points are closer than δ or not. Thus, we can ignore the translation for distances between auxiliary points in this proof. For reference we state which auxiliary points are closer than δ for all $\tau \in \mathcal{T}$. For each auxiliary point in π_{OR} we list its close auxiliary points in σ_{OR}:

$$\begin{aligned} s_1 :&\quad s_2, s_2^*, t_2, r_2, r_2', \\ t_1 :&\quad s_2, t_2, r_2, r_2', \\ r_1 :&\quad s_2, t_2, r_2, \\ r_1' :&\quad s_2, t_2, r_2'. \end{aligned}$$

All other pairs are in distance greater than δ. Note that for the remainder of the proof, we do not have to consider the specific value for τ anymore.

We first show that if $\mathrm{d_{dF}}(\hat{\pi}^i, \hat{\sigma}^j + \tau) \leq \delta$ for some i, j, then $\mathrm{d_{dF}}(\pi_{\mathrm{OR}}, \sigma_{\mathrm{OR}} + \tau) \leq \delta$ by giving a valid traversal. We start in $s_1, s_2 + \tau$. Then we traverse π_{OR} until the copy of s_1 which comes before the subcurve $\hat{\pi}^i$. While staying in s_1, we traverse $\sigma_{\mathrm{OR}} + \tau$ until we reach the copy of $r_2 + \tau$ right before the subcurve $\hat{\sigma}^j + \tau$. We then do one step on π_{OR} to r_1. Now we step to the first nodes of $\hat{\pi}^i$ and $\hat{\sigma}^j + \tau$ simultaneously, and then traverse these two subcurves in distance δ, which is possible due to $\mathrm{d_{dF}}(\hat{\pi}^i, \hat{\sigma}^j + \tau) \leq \delta$. We then step to the copies of r_1' and $r_2' + \tau$ simultaneously. We then step to t_1 on π_{OR}, while staying at $r_2' + \tau$ in $\sigma_{\mathrm{OR}} + \tau$. Subsequently, while staying in t_1, we traverse $\sigma_{\mathrm{OR}} + \tau$ until we reach its last point, namely $t_2 + \tau$. Now we can traverse the remainder of π_{OR}. One can check that this traversal stays within distance δ.

We now show that if $\mathrm{d_{dF}}(\pi_{\mathrm{OR}}, \sigma_{\mathrm{OR}} + \tau) \leq \delta$, then there exist i, j such that $\mathrm{d_{dF}}(\hat{\pi}^i, \hat{\sigma}^j + \tau) \leq \delta$. Pick any valid traversal for which $\mathrm{d_{dF}}(\pi_{\mathrm{OR}}, \sigma_{\mathrm{OR}} + \tau) \leq \delta$. We reconstruct in the following how it passed through π_{OR} and $\sigma_{\mathrm{OR}} + \tau$. Consider the point when $s_2^* + \tau$ is reached. At that point, we have to be in some copy of s_1 as this is the only type of node of π_{OR} which is in distance at most δ from $s_2^* + \tau$. Let $\hat{\pi}^i$ be the subcurve right after this copy of s_1. When we step to the copy of r_1 right after this s_1, there are only three types of nodes from $\sigma_{\mathrm{OR}} + \tau$ in distance δ: $s_2 + \tau, t_2 + \tau, r_2 + \tau$. Note that we already passed $s_2 + \tau$, and we cannot have reached $t_2 + \tau$ yet, as $t_2^* + \tau$ is neither in reach of s_1 nor r_1. Thus, we are in $r_2 + \tau$. Let the curve right after $r_2 + \tau$ be $\hat{\sigma}^j + \tau$. The only option now is to do a simultaneous step to the first nodes of $\hat{\pi}^i$ and $\hat{\sigma}^j + \tau$. Now, consider the point when either r_1' or $r_2' + \tau$ is first reached. All points of $\hat{\pi}^i$ are far from $r_2' + \tau$ and all points of $\hat{\sigma}^j + \tau$ are far from r_1' and thus we have to be in r_1' and $r_2' + \tau$ at the same time. This implies that we traversed $\hat{\pi}^i$ and $\hat{\sigma}^j + \tau$ from the start to the end nodes in distance δ and therefore $\mathrm{d_{dF}}(\hat{\pi}^i, \hat{\sigma}^j + \tau) \leq \delta$. □

8.3 Final Construction and Correctness

We now show how to construct the final curves that result from the reduction using the above gadgets and then prove correctness of the reduction.

Assembling $\boldsymbol{\pi^{(j)}}$ and $\boldsymbol{\sigma^{(j)}}$. We can apply the OR gadget to the equality gadgets in the following way. For each of the D dimensions we construct an OR gadget. The OR gadget for dimension $j \in [D]$ contains as anti-diagonal curves all $\pi_F(v_1)$ with $v_1[j] = 0$, all $\pi_{F'}(v_1)$, all $\pi_G(v_3)$ with $v_3[j] = 0$, and all $\pi_{G'}(v_3)$; and as diagonal curves it contains all $\sigma_F(v_2)$, all $\sigma_{F'}(v_2)$ with $v_2[j] = 0$, all $\sigma_G(v_4)$, and all $\sigma_{G'}(v_4)$ with $v_4[j] = 0$. Note that these curves fulfill the requirements stated in Observation 8.8 for usage in the OR gadget as (anti-)diagonal curves. We denote the resulting curves by $\pi^{(j)}$ and $\sigma^{(j)}$, and we write $H(j) = (\pi^{(j)}, \sigma^{(j)})$. This yields the following lemma.

Lemma 8.10. *Given a 4-OV instance $V_1, \dots, V_4$, and consider the corresponding OR gadget $H(j) = (\pi^{(j)}, \sigma^{(j)})$ for some $j \in [D]$. It holds that:*

(i) For any vectors $v_1 \in V_1, \dots, v_4 \in V_4$ with $v_1[j] \cdot v_2[j] \cdot v_3[j] \cdot v_4[j] = 0$ we have $\mathrm{d_{dF}}(\pi^{(j)}, \sigma^{(j)} + \tau) \le \delta$ for $\tau = ((\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) \cdot \varepsilon, (\mathrm{ind}(v_3) + \mathrm{ind}(v_4) \cdot N) \cdot \varepsilon)$.

(ii) If $\mathrm{d_{dF}}(\pi^{(j)}, \sigma^{(j)} + \tau) \le \delta$ for some $\tau \in \mathcal{T}$, then

- $\exists v_1 \in V_1, v_2 \in V_2 : v_1[j] \cdot v_2[j] = 0$ *and* $|\varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) - \tau_1| \le \frac{1}{3}\varepsilon$
  or
- $\exists v_3 \in V_3, v_4 \in V_4 : v_3[j] \cdot v_4[j] = 0$ *and* $|\varepsilon \cdot (\mathrm{ind}(v_3) + \mathrm{ind}(v_4) \cdot N) - \tau_2| \le \frac{1}{3}\varepsilon$

Proof. For (i), from $v_1[j] \cdot v_2[j] \cdot v_3[j] \cdot v_4[j] = 0$ it follows that at least one gadget of $F(v_1, v_2)$, $F'(v_1, v_2)$, $G(v_3, v_4)$, $G'(v_3, v_4)$ is contained in $H(j)$. By Lemmas 8.3 to 8.6, we know that the discrete Fréchet distance of this gadget is small. By Lemma 8.9 it then follows that $\mathrm{d_{dF}}(\pi^{(j)}, \sigma^{(j)} + \tau) \le \delta$.

For (ii), from $\mathrm{d_{dF}}(\pi^{(j)}, \sigma^{(j)} + \tau) \le \delta$ it follows by Lemmas 8.9 and 8.7 that there exists a gadget Γ for which the discrete Fréchet distance is at most δ. From Lemmas 8.3 to 8.6 it then follows that

$$|\varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) - \tau_1| \le \frac{1}{3}\varepsilon \quad \text{or} \quad |\varepsilon \cdot (\mathrm{ind}(v_3) + \mathrm{ind}(v_4) \cdot N) - \tau_2| \le \frac{1}{3}\varepsilon.$$

for some vectors $v_1 \in V_1, \dots, v_4 \in V_4$. As Γ is contained in the OR gadget, we additionally have that $v_1[j] \cdot v_2[j] = 0$ or $v_3[j] \cdot v_4[j] = 0$, respectively. □

Final curves. The final curves π and σ are now defined as follows. We start with the translation gadget $\pi^{(0)}$ ($\sigma^{(0)}$). Then the curves $\pi^{(j)}$ ($\sigma^{(j)}$) follow for $j \in [D]$. Note that we have to translate these curves to fulfill the requirements of Lemmas 8.1 and 8.2, thus, we translate $\pi^{(j)}$ ($\sigma^{(j)}$) by $(100 \cdot j, 0)$. More explicitly, the final curves are

$$\pi := \pi^{(0)} \bigcirc_{j \in [D]} \pi^{(j)} + (100 \cdot j, 0),$$
$$\sigma := \sigma^{(0)} \bigcirc_{j \in [D]} \sigma^{(j)} + (100 \cdot j, 0).$$

We are now ready to prove Theorem 3.3. Recall its statement:

Theorem 3.3. *The discrete Fréchet distance under translation of curves of length n in the plane requires time $n^{4-o(1)}$, unless the Strong Exponential Time Hypothesis fails.*

Also recall that it suffices to prove a lower bound under the 4-OV hypothesis. For clarity of structure, we split the proof into Lemma 8.11 and Lemma 8.12 which together imply Theorem 3.3.

Lemma 8.11. *Given a YES-instance of 4-OV, the curves π and σ constructed in our reduction have discrete Fréchet distance under translation at most δ, i.e., $\min_\tau \mathrm{d_{dF}}(\pi, \sigma + \tau) \le \delta$.*

Proof. Let $v_1 \in V_1, \ldots, v_4 \in V_4$ be orthogonal vectors and let $\tau = ((\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) \cdot \varepsilon, (\mathrm{ind}(v_3) + \mathrm{ind}(v_4) \cdot N) \cdot \varepsilon)$ be the translation corresponding to those vectors. From Lemma 8.1 we know that $\mathrm{d_{dF}}(\pi^{(0)}, \sigma^{(0)} + \tau) \leq \delta$, and thus there is a valid traversal to the endpoints of the translation gadget. Then we simultaneously step to the start of $\pi^{(1)}$ and $\sigma^{(1)}$. From Lemma 8.10 we know that there also exist traversals of $\pi^{(1)}, \ldots, \pi^{(D)}$ and $\sigma^{(1)} + \tau, \ldots, \sigma^{(D)} + \tau$ of distance at most δ. It follows from Lemma 8.2 that we can also traverse those gadgets sequentially in distance δ and thus $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$. □

Lemma 8.12. *If the curves π and σ constructed in our reduction have discrete Fréchet distance under translation at most δ, then the given 4-OV instance is a YES-instance.*

Proof. Let τ be a translation such that $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$. We know from Lemma 8.1 that $\tau \in \mathcal{T}$. Furthermore, from Lemma 8.2 we know that for all $j \in [D]$ it holds that $\mathrm{d_{dF}}(\pi^{(j)}, \sigma^{(j)} + \tau) \leq \delta$. It follows from Lemma 8.10 that for every $j \in [D]$ there exist $v_1 \in V_1, v_2 \in V_2$ such that $v_1[j] \cdot v_2[j] = 0$ and $|\varepsilon \cdot (\mathrm{ind}(v_1) + \mathrm{ind}(v_2) \cdot N) - \tau_1| \leq \frac{1}{3}\varepsilon$ or there exist $v_3 \in V_3, v_4 \in V_4$ such that $v_3[j] \cdot v_4[j] = 0$ and $|\varepsilon \cdot (\mathrm{ind}(v_3) + \mathrm{ind}(v_4) \cdot N) - \tau_2| \leq \frac{1}{3}\varepsilon$. Therefore, every dimension $j \in [D]$ gives us constraints on either v_1, v_2 or v_3, v_4. Due to Lemma 8.10 these constraints have to be consistent. If in total this gives us constraints for $v_1, \ldots, v_4$, then we are done. Otherwise, if this only gives us constraints for v_1, v_2, then we already found v_1, v_2 which are orthogonal and thus we can pick arbitrary $v_3 \in V_3, v_4 \in V_4$ to obtain an orthogonal set of vectors. The case of only v_3, v_4 being constrained is symmetric. □

Proof of Theorem 3.3. The Strong Exponential Time Hypothesis implies the k-OV hypothesis. The reduction above from a 4-OV instance of size N over $\{0,1\}^D$ to an instance of the discrete Fréchet distance under translation in $\mathbb{R}^2$ results in two curves of length $\mathcal{O}(D \cdot N)$. Lemmas 8.11 and 8.12 show correctness of this reduction. Hence, any $\mathcal{O}(n^{4-\varepsilon})$-time algorithm for the discrete Fréchet distance under translation would imply an algorithm for 4-OV in time $\mathcal{O}((D \cdot N)^{4-\varepsilon}) = \mathcal{O}(\mathrm{poly}(D) \cdot N^{4-\varepsilon})$, refuting the k-OV hypothesis. □

PART III

Algorithm Engineering

CHAPTER 9

Fréchet Distance

For a technical overview of this chapter see Section 3.4. This chapter is organized as follows. First, in Section 9.1, we present all the core definitions. Subsequently, we explain our complete decider in Section 9.2. The following section then explains the decider and its filtering steps. Then, in Section 9.4, we present a query data structure which enables us to compare to the GIS Cup submissions. Section 9.5 contains some details regarding the implementation to highlight crucial points that are relevant for similar implementations. We conduct extensive experiments in Section 9.6, detailing the improvements over the current state of the art by our implementation. Finally, in Section 9.7, we describe how we make our implementation certifying and evaluate the certifying code experimentally.

9.1 Preliminaries

Our implementation as well as the description are restricted to two dimensions, however, the approach can also be generalized to polygonal curves in d dimensions. In the remainder, we denote the *number of vertices* of π (resp. σ) with n (resp. m) if not stated otherwise. We denote the length (not the complexity!) of a curve π by $\|\pi\|$, i.e., the sum of the Euclidean lengths of its line segments. Additionally, we use $\|v\|$ for the Euclidean norm of a vector $v \in \mathbb{R}^2$. Given two curves π, σ and a query distance δ, we call them *close* if $d_F(\pi, \sigma) \le \delta$ and *far* otherwise. There are two problem settings that we consider in this chapter:

Decider Setting: Given curves π, σ and a distance δ, decide whether $d_F(\pi, \sigma) \le \delta$. (With such a decider, we can compute the exact distance by using parametric search in theory and binary search in practice.)

Query Setting: Given a curve dataset $\mathcal{D}$, build a data structure that on query (π, δ) returns all $\sigma \in \mathcal{D}$ with $d_F(\pi, \sigma) \le \delta$.

We mainly focus on the decider in this chapter. To allow for a comparison with previous implementations (which are all in the query setting), we also run experiments with our decider plugged into a data structure for the query setting.

9.1.1 Preprocessing

When reading the input curves we immediately compute additional data which is stored with each curve:

Prefix Distances: To be able to quickly compute the curve length between any two vertices of π, we precompute the prefix lengths, i.e., the curve lengths $\|\pi[1, i]\|$ for every $i \in \{2, \dots, n\}$. We can then compute the curve length for two indices $i < i'$ on π by $\|\pi[i, i']\| = \|\pi[1, i']\| - \|\pi[1, i]\|$.

Bounding Box: We compute the bounding box of all curves, which is just a coordinate-wise maximum and minimum computation.

Both of these preprocessing steps are extremely cheap as they only require a single pass over all curves, which we anyway do when parsing them. In the remainder of this chapter we assume that this additional data was already computed, in particular, we do not measure it in our experiments as it is dominated by reading the curves.

9.2 Complete Decider

The key improvement of this chapter lies in the complete decider via free-space exploration. Here, we use a divide-and-conquer interpretation of the algorithm of Alt and Godau [19] which is similar to [30] where a free-space diagram is built recursively. This interpretation allows us to prune away large parts of the search space by designing powerful *pruning rules* identifying parts of the search space that are irrelevant for determining the correct output. Before describing the details, we formally define the free-space diagram.

9.2.1 Free-Space Diagram

The free-space diagram was first defined in [19]. Given two polygonal curves π and σ and a distance δ, it is defined as the set of all pairs of indices of points from π and σ that are close to each other, i.e.,

$$F := \{(p, q) \in [1, n] \times [1, m] \mid \|\pi(p) - \sigma(q)\| \leq \delta\}.$$

For an example see Figure 9.1. A *path* from a to b in the free-space diagram F is defined as a continuous mapping $P : [0, 1] \to F$ with $P(0) = a$ and $P(1) = b$. A path P in the free-space diagram is *monotone* if $P(x)$ is component-wise at most $P(y)$ for any $0 \leq x \leq y \leq 1$. The *reachable space* is then defined as

$$R := \{(p, q) \in F \mid \text{there exists a monotone path from } (1, 1) \text{ to } (p, q) \text{ in } F\}.$$

Figure 9.2 shows the reachable space for the free-space diagram of Figure 9.1. It is well known that $\mathrm{d_F}(\pi, \sigma) \leq \delta$ if and only if $(n, m) \in R$.

This leads us to a simple dynamic programming algorithm to decide whether the Fréchet distance of two curves is at most some threshold distance. We iteratively compute R starting from $(1, 1)$ and ending at (n, m), using the previously computed values. As R is potentially a set of infinite size, we have to discretize it. A natural choice is to restrict to cells. The *cell* of R with coordinates $(i, j) \in \{1, \dots, n-1\} \times \{1, \dots, m-1\}$ is defined as $C_{i,j} := [i, i+1] \times [j, j+1]$. This is a natural choice as given $R \cap C_{i-1,j}$ and $R \cap C_{i,j-1}$, we can compute $R \cap C_{i,j}$ in constant time; this follows from the simple fact that $F \cap C_{i,j}$ is convex [19]. We call this computation of the outputs of a cell the *cell propagation.* This algorithm runs in time $\mathcal{O}(nm)$ and was introduced by Alt and Godau [19].

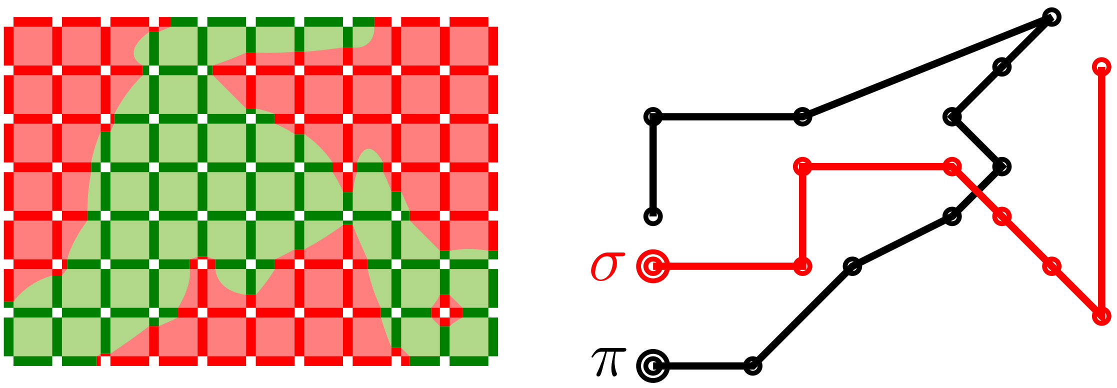


Figure 9.1: Example of a free-space diagram for curves π (black) and σ (red). Curve π is on the horizontal axis of the free-space diagram, while σ is on the vertical axis; we use this convention in the remainder. The doubly-circled vertices mark the start. The free-space, i.e., the pairs of indices of points which are close, is colored green. The non-free areas are colored red. The threshold distance δ is roughly the distance between the first vertex of σ and the third vertex of π.

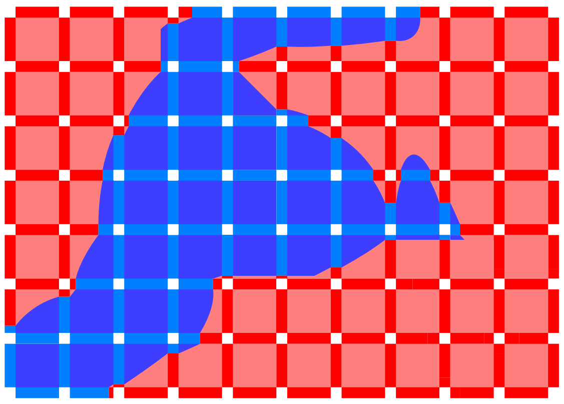


Figure 9.2: Reachable space of the free-space diagram in Figure 9.1. The reachable part is blue and the non-reachable part is red. Note that the reachable part is a subset of the free-space. We use this color scheme in the remainder of this chapter.

9.2.2 Basic Algorithm

For integers $1 \leq i \leq i' \leq n, 1 \leq j \leq j' \leq m$ we call the set $B = [i, i'] \times [j, j']$ a *box*. We denote the left/right/bottom/top *boundaries* of B by $B_l = \{i\} \times [j, j'], B_r = \{i'\} \times [j, j'], B_b = [i, i'] \times \{j\}, B_t = [i, i'] \times \{j'\}$. The *left input* of B is $B_l^R = B_l \cap R$, and its *bottom input* is $B_b^R = B_b \cap R$. Similarly, the *right/top output* of B is $B_r^R = B_r \cap R$, $B_t^R = B_t \cap R$. A box is a cell if $i + 1 = i'$ and $j + 1 = j'$. We always denote the lower left corner of a box by (i, j) and the top right by (i', j'), if not mentioned otherwise.

A recursive variant of the standard free-space decision procedure is as follows: Start with $B = [1, n] \times [1, m]$. At any recursive call, if B is a cell, then determine its outputs from its inputs in constant time, as described by [19]. Otherwise, split B vertically or horizontally into B_1, B_2 and first compute the outputs of B_1 from the inputs of B and then compute the outputs of B_2 from the inputs of B and the outputs of B_1. In the end, we just have to check $(n, m) \in R$ to decide whether the curves are close or far. This is a constant-time operation after calculating all outputs.

Now comes the main idea of our approach: we try to avoid recursive splitting by directly computing the outputs for non-cell boxes using certain rules. We call them *pruning rules* as they enable pruning large parts of the recursion tree induced by the divide-and-conquer approach. Our pruning rules are heuristic, meaning that they are not always applicable, however, we show in the experiments that on practical curves they apply very often and therefore massively reduce the number of recursive calls. The detailed pruning rules are described in Section 9.2.3. Using these rules, we change the above recursive algorithm as follows. In any recursive call on box B, we first try to apply the pruning rules. If this is successful, then we obtained the outputs of B and we are done with this recursive call. Otherwise, we perform the usual recursive splitting. Corresponding pseudocode is shown in Algorithm 13.

In the remainder of this section, we describe our pruning rules and their effects.

9.2.3 Pruning Rules

In this section we introduce the rules that we use to compute outputs of boxes which are above cell-level in certain special cases. Note that we aim at catching special cases which occur often in practice, as we cannot hope for improvements on adversarial instances due to the conditional lower bound of [38]. Therefore, we make no claims whether they are applicable, only that they are sound and fast. In what follows, we call a boundary *empty* if its intersection with R is $\emptyset$.

Rule I: Empty Inputs

The simplest case where we can compute the outputs of a box B is if both inputs are empty, i.e. $B_b^R = B_l^R = \emptyset$. In this case no propagation of reachability is possible and thus the outputs are empty as well, i.e. $B_t^R = B_r^R = \emptyset$. See Figure 9.3 for an example.

Rule II: Shrink Box

Instead of directly computing the outputs, this rule allows us to shrink the box we are currently working on, which reduces the problem size. Assume that for a box B we

```
1: procedure DecideFréchetDistance($\pi, \sigma, \delta$)
2:     ComputeOutputs($\pi, \sigma, [1, n] \times [1, m]$)
3:     return $[(n, m) \in R]$

4: procedure ComputeOutputs($\pi, \sigma, B = [i, i'] \times [j, j'], \delta$)
5:     if $B$ is a cell then
6:         compute outputs by cell propagation
7:     else
8:         use pruning rules I to IV to compute outputs of $B$
9:         if not all outputs have been computed then
10:            if $j' - j > i' - i$ then                         ▷ split horizontally
11:                $B_1 = [i, i'] \times [j, \lfloor (j + j')/2 \rfloor]$
12:                $B_2 = [i, i'] \times [\lfloor (j + j')/2 \rfloor, j']$
13:            else                                              ▷ split vertically
14:                $B_1 = [i, \lfloor (i + i')/2 \rfloor] \times [j, j']$
15:                $B_2 = [\lfloor (i + i')/2 \rfloor, i'] \times [j, j']$
16:            ComputeOutputs($\pi, \sigma, B_1$)
17:            ComputeOutputs($\pi, \sigma, B_2$)
```

Algorithm 13: Recursive Decider of the Fréchet Distance

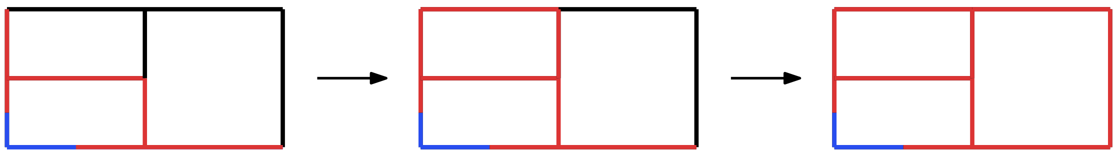


Figure 9.3: Output computation of a box when inputs are empty. First we can compute the outputs of the top left box and then the outputs of the right box. In this example, we then know that the curves have a Fréchet distance greater than δ as (n, m) is not reachable.

have that $B_b^R = \emptyset$ and the lowest point of B_l^R is $(i, j_{\min})$ with $j_{\min} > j$. In this case, no pair in $[i, i'] \times [j, j_{\min}]$ is reachable. Thus, we can shrink the box to the coordinates $[i, i'] \times [\lfloor j_{\min} \rfloor, j']$ without losing any reachability information. An equivalent rule can be applied if we swap the role of B_b and B_l. See Figure 9.4 for an example of applying this rule.

Rule III: Simple Boundaries

Simple boundaries are boundaries of a box that contain at most one free component. To define this formally, a set $\mathcal{I} \subseteq [1, n] \times [1, m]$ is called an *interval* if $\mathcal{I} = \emptyset$ or $\mathcal{I} = \{p\} \times [q, q']$ or $\mathcal{I} = [q, q'] \times \{p\}$ for real p and an interval $[q, q']$. In particular, the four boundaries of a box $B = [i, i'] \times [j, j']$ are intervals. We say that an interval $\mathcal{I}$ is *simple* if $\mathcal{I} \cap F$ is again an interval. Geometrically, we have a free interval of a point $\pi(p)$ and a curve $\sigma[q, q']$ (which is the form of a boundary in the free-space diagram) if the circle of radius

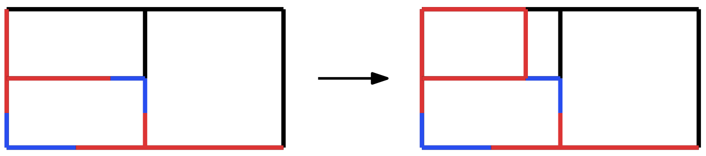


Figure 9.4: This is an example of shrinking a box in case one of the inputs is empty and the other one starts with an empty part. In this example the top left box has an empty input on the left and the start of the bottom input is empty as well. Thus, we can shrink the box to the right part.

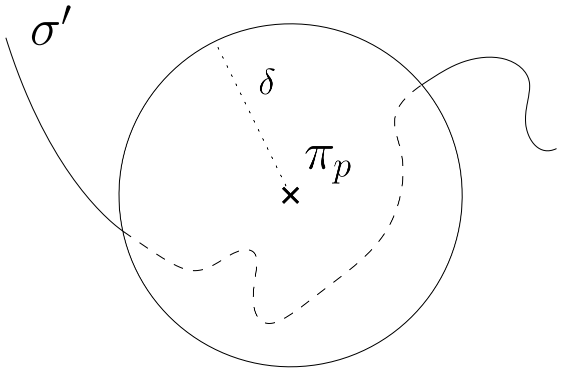


Figure 9.5: Example of a point $\pi(p)$ and a curve σ' which lead to a simple boundary.

δ around $\pi(p)$ intersects $\sigma[q, q']$ at most twice. See Figure 9.5 for an example. We call such a boundary simple because it is of low complexity, which we can exploit for pruning.

There are three pruning rules that we do based on simple boundaries (see Figure 9.6 for visualizations). They are stated here for the top boundary B_t, but symmetric rules apply to B_r. Later, in Section 9.2.4, we then explain how to actually compute simple boundaries, i.e., also how to compute $B_t \cap F$. The pruning rules are:

(a) If B_t is simple because $B_t \cap F$ is empty then we also know that the output of this boundary is empty. Thus, we conclude that $B_t^R = \emptyset$ and we are done with B_t.

(b) Suppose that B_t is simple and, more specifically, of the form that it first has a free and then a non-free part; in other words, we have $(i, j') \in B_t \cap F$. Due to our recursive approach, we already computed the left inputs of the box and thus know whether the top left corner of the box is reachable, i.e. whether $(i, j') \in R$. If this is the case, then we also know the reachable part of our simple boundary: Since $(i, j') \in R$ and $B_t \cap F$ is an interval containing (i, j'), we conclude that $B_t^R = B_t \cap F$ and we are done with B_t.

(c) Suppose that B_t is simple, but the leftmost point $(i_{\min}, j')$ of $B_t \cap F$ has $i_{\min} > i$. In this case, we try to certify that $(i_{\min}, j') \in R$, because then it follows that $B_t^R = B_t \cap F$ and we are done with B_t. To check for reachability of $(i_{\min}, j')$, we try to propagate the reachability through the inside of the box, which in this case means to propagate it from the bottom boundary. We test whether $(i_{\min}, j)$ is in the input, i.e., if $(i_{\min}, j) \in B_b^R$, and whether $\{i_{\min}\} \times [j, j'] \subseteq F$ (by slightly modifying

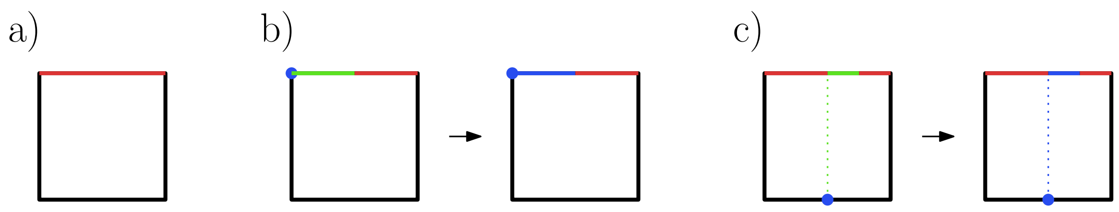


Figure 9.6: Visualization of the rules for computing outputs using simple boundaries. All three cases are visualized with the top boundary being simple. In a) the boundary is non-free and therefore no point on it can be reachable. In b) the boundary's beginning is free and reachable, enabling us to propagate the reachability to the entire free interval. In c) we can propagate the reachability of a point on the bottom boundary, using a free interval inside the box, to the beginning of the free interval of the top boundary and thus decide the entire boundary. The rules for the right boundary being simple are equivalent.

the algorithm for simple boundary computations). If this is the case, then we can reach every point in $B_t \cap F$ from $(i_{\min}, j)$ via $\{i_{\min}\} \times [j, j']$. Note that this is an operation in the complete decider where we explicitly use the *inside* of a box and not exclusively operate on its boundaries.

We also use symmetric rules by swapping "top" with "right" and "bottom" with "left".

Rule IV: Boxes at Free-Space Diagram Boundaries

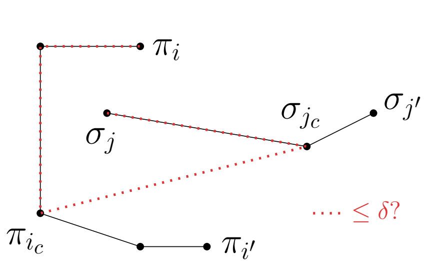
The boundaries of a free-space diagram are a special form of boundary which allows us to introduce an additional rule. Consider a box B which touches the top boundary of the free-space diagram, i.e., $B = [i, i'] \times [j, m]$. Suppose the previous rules allowed us to determine the output for B_r^R. Since any valid traversal from $(1, 1)$ to (n, m) passing through B intersects B_r, the output B_t^R is not needed anymore, and we are done with B. A symmetric rule applies to boxes which touch the right boundary of the free-space diagram.

9.2.4 Implementation Details of Simple Boundaries

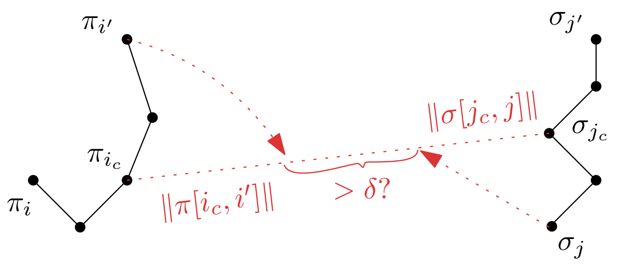
It remains to describe how we test whether a boundary is simple, and how we determine the free interval of a simple boundary. One important ingredient for the fast detection of simple boundaries are two simple heuristic checks that check whether two polygonal curves are close or far, respectively. The former check was already used in [30]. We first explain these heuristic checks, and then explain how to use them for the detection of simple boundaries.

Heuristic check whether two curves are close. Given two subcurves $\pi' := \pi[i, i']$ and $\sigma' := \sigma[j, j']$, this filter heuristically tests whether $d_F(\pi', \sigma') \leq \delta$. Let $i_c := \lfloor \frac{i+i'}{2} \rfloor$ and $j_c := \lfloor \frac{j+j'}{2} \rfloor$ be the indices of the midpoints of π' and σ' (with respect to hops). Then $d_F(\pi', \sigma') \leq \delta$ holds if

$$\max\{\|\pi[i, i_c]\|, \|\pi[i_c, i']\|\} + \|\pi_{i_c} - \sigma_{j_c}\| + \max\{\|\sigma[j, j_c]\|, \|\sigma[j_c, j']\|\} \leq \delta.$$



(a) HeurClose



(b) HeurFar

Figure 9.7: Visualizations of heuristic checks HeurClose and HeurFar.

The triangle inequality ensures that this is an upper bound on all distances between two points on the curves. For a visualization, see Figure 9.7(a). Observe that all curve lengths that need to be computed in the above equation can be determined quickly due to our preprocessing, see Section 9.1.1. We call this procedure HeurClose(π', σ', δ).

Heuristic check whether two curves are far. Symmetrically, we can test whether all pairs of points on π' and σ' are far by testing

$$\|\pi_{i_c} - \sigma_{j_c}\| - \max\{\|\pi[i, i_c]\|, \|\pi[i_c, i']\|\} - \max\{\|\sigma[j, j_c]\|, \|\sigma[j_c, j']\|\} > \delta.$$

We call this procedure HeurFar(π', σ', δ).

Computation of simple boundaries. Recall that an interval is defined as $I = \{p\} \times [q, q']$ (intervals of the form $[q, q'] \times \{p\}$ are handled symmetrically). The naive way to decide whether interval I is simple would be to go over all the segments of $\sigma[q, q']$ and compute the intersection with the circle of radius δ around $\pi(p)$. However, this is too expensive because (i) computing the intersection of a disc and a segment involves taking a square root, which is an expensive operation with a large constant running time, and (ii) iterating over all segments of $\sigma[q, q']$ incurs a linear factor in n for large boxes, while we aim at a logarithmic dependence on n for simple boundary detection.

We avoid these issues by resolving long subcurves $\sigma[j, j + s]$ using our heuristic checks (HeurClose, HeurFar). Here, s is an adaptive step size that grows whenever the heuristic checks were applicable, and shrinks otherwise. See Algorithm 14 for pseudocode of our simple boundary detection. It is straightforward to extend this algorithm to not only detect whether a boundary is simple, but also compute the free interval of a simple boundary; we call the resulting procedure SimpleBoundary.

9.2.5 Effects of Combined Pruning Rules

All the pruning rules presented above can in practice lead to a reduction of the number of boxes that are necessary to decide the Fréchet distance of two curves. We exemplify this on two real-world curves; see Figure 9.8 on page 148 for the curves and their corresponding free-space diagram. We explain in the following where the single rules come into play. For *Box 1* we apply Rule IIIb twice – for the top and right output. The top boundary of *Box 2* is empty and thus we computed the outputs according to Rule IIIa. Note that the

```
procedure isSimpleBoundary($\pi(p), \sigma[q, q']$)
    if HeurFar($\pi(p), \sigma[q, q'], \delta$) or HeurClose($\pi(p), \sigma[q, q'], \delta$) then
        return “simple”

    $C \leftarrow \begin{cases} \{\sigma(q)\} & \text{, if } \|\pi(p) - \sigma(q)\| \leq \delta \\ \emptyset & \text{, otherwise} \end{cases}$        ▷ set of change points
    $s \leftarrow 1, j \leftarrow q$
    while $j < q'$ do
        if HeurClose($\pi(p), \sigma[j, j+s], \delta$) then
            $j \leftarrow j + s$
            $s \leftarrow \min\{2s, q' - j\}$        ▷ double $s$ but do not overstep $q'$
        else if HeurFar($\pi(p), \sigma[j, j+s], \delta$) then
            $j \leftarrow j + s$
            $s \leftarrow \min\{2s, q' - j\}$        ▷ double $s$ but do not overstep $q'$
        else if $s > 1$ then
            $s \leftarrow s/2$
        else
            $P \leftarrow \{j' \in (j, j+1] \mid \|\pi(p) - \sigma(j')\| = \delta\}$
            $C \leftarrow C \cup P$
            $j \leftarrow j + 1$
            if $|C| > 2$ then
                return “not simple”

    return “simple”
```

Algorithm 14: Checks if the boundary in the free-space diagram corresponding to $\{p\} \times [q, q']$ is simple.

right boundary of this box is on the right boundary of the free-space diagram and thus we do not have to compute it according to Rule IV. For *Box 3* we again use Rule IIIb for the top, but we use Rule IIIc for the right boundary – the blue dotted line indicates that the reachability information is propagated through the box. For *Box 4* we first use Rule II to move the bottom boundary significantly up, until the end of the left empty part; we can do this because the bottom boundary is empty and the left boundary is simple, starting with an empty part. After two splits of the remaining box, we see that the two outputs of the leftmost box are empty as the top and right boundaries are non-free, using Rule IIIa. For the remaining two boxes we use Rule I as their inputs are empty.

This example illustrates how propagating through a box (in *Box 3*) and subsequently moving a boundary (in *Box 4*) leads to pruning large parts. Additionally, we can see how using simple boundaries leads to early decisions and thus avoids many recursive steps. In total, we can see how all the explained pruning rules together lead to a free-space diagram with only twelve boxes, i.e., twelve recursive calls, for curves with more than 50 vertices and more than 1500 reachable cells. Figure 9.9 shows what effects the pruning rules have by introducing them one by one in an example.

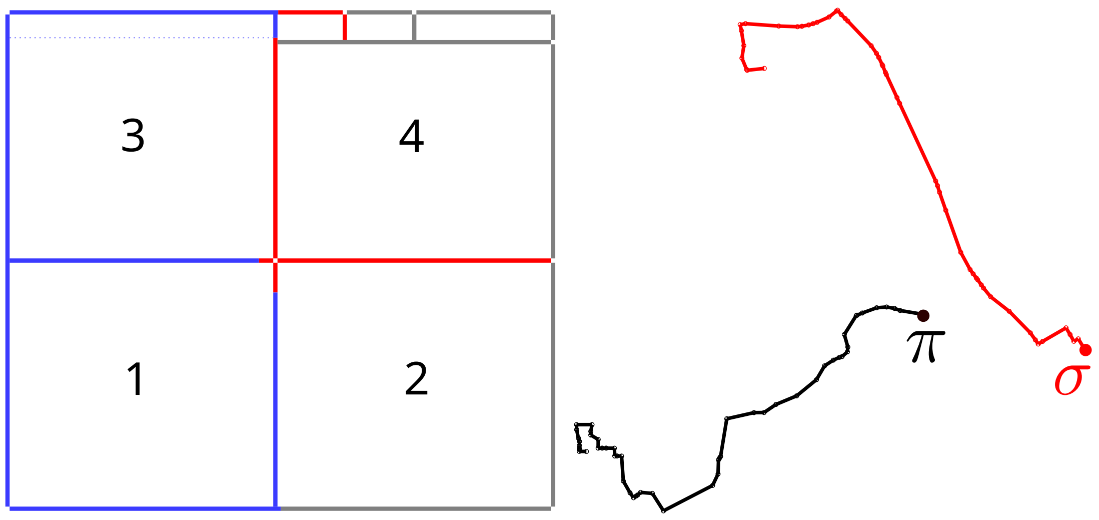


Figure 9.8: A free-space diagram as produced by our final implementation (left) with the corresponding curves (right). The beginnings of the curves are marked, and π is on the horizontal axis of the free-space diagram, while σ is on the vertical axis. The curves are taken from the SIGSPATIAL dataset. We number the boxes in the third level of the recursion from 1 to 4.

9.3 Decider with Filters

Now that we introduced the complete decider, we are ready to present the decider. We first give a high-level overview.

9.3.1 Decider

The decider can be divided into two parts:

(1) Filters (see this section)

(2) Complete decider via free-space exploration (see Section 9.2)

As outlined in Section 3.4, we first try to determine the correct output by using fast but incomplete filtering mechanisms and only resort to the slower complete decider presented in the last section if none of the heuristic deciders (*filters*) gave a result. The high-level pseudocode of the decider is shown in Algorithm 15.

The speed-ups introduced by our complete decider were already explained in Section 9.2. A second source for our speed-ups lies in the usage of a good set of filters. Interestingly, since our optimized complete decider via free-space exploration already solves many simple instances very efficiently, our filters have to be extremely fast to be useful – otherwise, the additional effort for an incomplete filter does not pay off. In particular, we cannot afford expensive preprocessing and ideally, we would like to achieve sublinear running times for our filters. To this end, we only use filters that can traverse large parts of the curves quickly. We achieve sublinear-type behavior by making previously used filters work with an adaptive step size (exploiting fast heuristic checks), and designing a new filter.

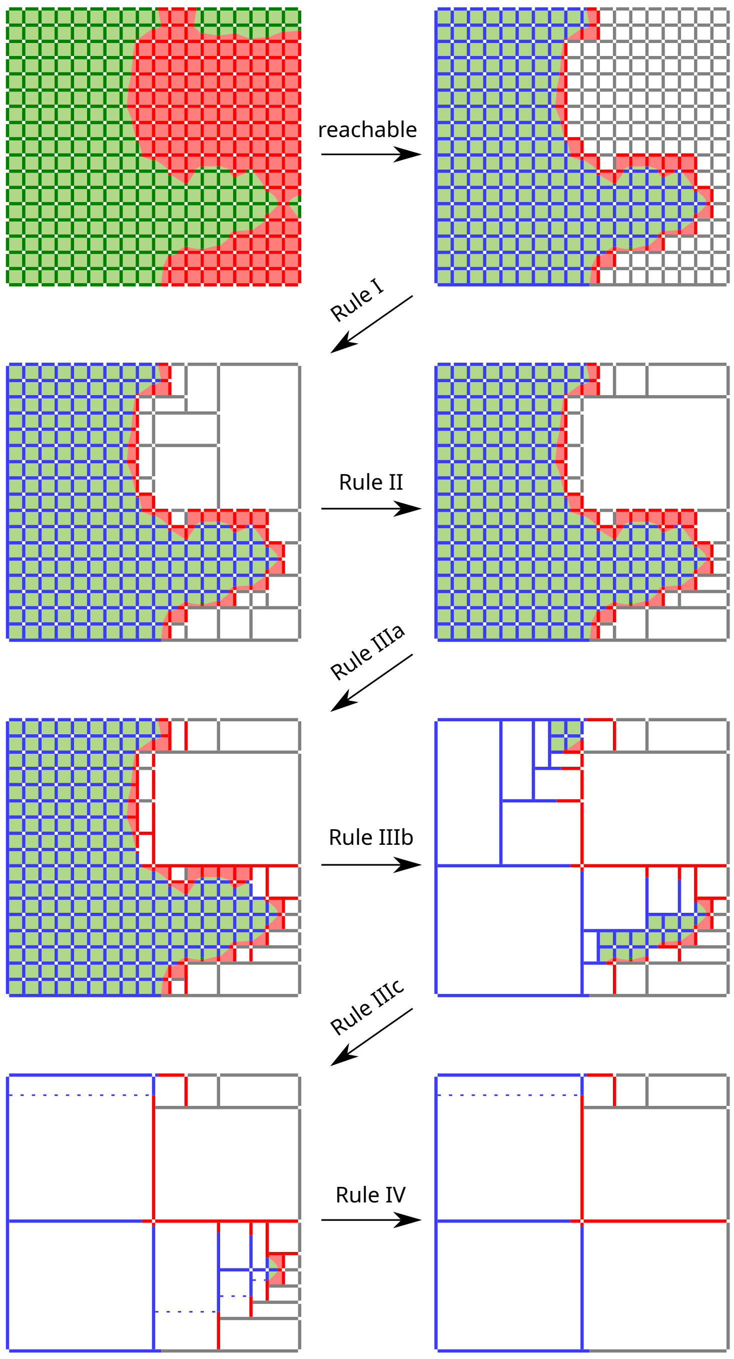


Figure 9.9: A decider example introducing the pruning rules one by one. They are introduced from top to bottom and left to right. The arrows denote the rules which are introduced in between the two corresponding free-space diagrams. The curves of this example are shown in Figure 9.8.

```
1: procedure DECIDER($\pi, \sigma, \delta$)
2:     if start points $\pi_1, \sigma_1$ or end points $\pi_n, \sigma_m$ are far then return “far”
3:     for all $f \in$ Filters do
4:         verdict = $f(\pi, \sigma, \delta)$
5:         if verdict $\in$ {“close”, “far”} then
6:             return verdict
7:     return COMPLETEDECIDER($\pi, \sigma, \delta$)
```

Algorithm 15: High-level code of the Fréchet decider.

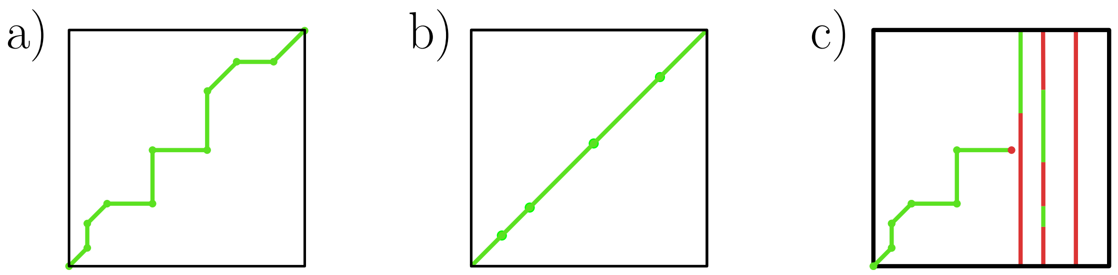


Figure 9.10: Sketches of the (a) greedy filter, (b) adaptive equal-time filter, and (c) negative filter. These sketches should be read as follows: the first dimension is the index on the first curve, while the second dimension is the index on the second curve. Recall that the green color indicates that the corresponding points are in distance at most δ; otherwise they are colored red. This visualization is similar to the free-space diagram.

In the remainder of this section, we describe all the filters that we use to heuristically decide whether two curves are close or far. There are two types of filters: *positive* filters check whether a curve is close to the query curve and return either “close” or “unknown”; *negative* filters check if a curve is far from the query curve and return either “far” or “unknown”.

9.3.2 Bounding Box Check

This is a positive filter already described in [87], which heuristically checks whether all pairs of points on π, σ are in distance at most δ. Recall that we compute the bounding box of each curve when we read it. We can thus check in constant time whether the furthest points on the bounding boxes of π, σ are in distance at most δ. If this is the case, then also all points of π, σ have to be close to each other and thus the free-space diagram is completely free and a valid traversal trivially exists.

9.3.3 Greedy

This is a positive filter. To assert that two curves π and σ are close, it suffices to find a traversal (f, g) satisfying $\max_{t\in[0,1]} \|\pi(f(t)) - \sigma(g(t))\| \le \delta$. We try to construct such a traversal staying within distance δ by making greedy steps that minimize the current distance. This may yield a valid traversal: if after at most $n + m$ steps we reach both endpoints and during the traversal the distance was always at most δ, we return “near”.

```
1: procedure GreedyFilter($\pi, \sigma, \delta$)
2:     $i, j, s \leftarrow 1$
3:     while $i < n$ or $j < m$ do
4:         $S \leftarrow \begin{cases} \{(i+1,j),(i,j+1),(i+1,j+1)\}, & \text{if } s = 1 \\ \{(i+s,j),(i,j+s)\}, & \text{if } s > 1 \end{cases}$        ▷ possible steps
5:         $P \leftarrow \{(i',j') \in S \mid i' \leq n \;\&\; j' \leq m \;\&\; \text{HeurClose}(\pi[i,i'], \sigma[j,j'], \delta)\}$
6:         if $P = \emptyset$ then
7:             if $s = 1$ then
8:                 return “unknown”
9:             else
10:                $s \leftarrow s/2$
11:        else
12:            $(i,j) \leftarrow \arg\min_{(i',j') \in P} \left\| \pi_{i'} - \sigma_{j'} \right\|$
13:            $s \leftarrow 2s$
14:    return “close”
```

Algorithm 16: Greedy filter with adaptive step size.

We can also get stuck: if a step on each of the curves would lead to a distance greater than δ, we return “unknown”. A similar filter was already used in [30] and is a standard idea (see, e.g., [48, 105]), however, here we present a variant with adaptive step size. This means that instead of just advancing to the next node in the traversal, we try to make larger steps, leveraging the heuristic checks presented in Section 9.2.4. We adapt the step size depending on the success of the last step. For pseudocode of the greedy filter see Algorithm 16, and for a visualization see Figure 9.10a.

9.3.4 Adaptive Equal-Time

We also consider a variation of Greedy Filter, which we call Adaptive Equal-Time Filter. The only difference to Algorithm 16 is that the allowed steps are now:

$$S := \begin{cases} \{(i+1,j),(i,j+1),(i+1,j+1)\}, & \text{if } s = 1, \\ \left\{ \left(i+s, j + \left\lfloor \frac{m-j}{n-i} \cdot s \right\rfloor \right) \right\}, & \text{if } s > 1. \end{cases}$$

In contrast to Greedy Filter, this searches for a traversal that stays as close as possible to the diagonal.

9.3.5 Negative

A negative filter was already used in [30] and [87]. However, changing this filter to use an adaptive step size does not seem to be practical when used with our approach. Preliminary tests showed that this filter would dominate our running time. Therefore, we developed a new negative filter which is more suited to be used with an adaptive step size and thus can be used with our approach.

Let (π_i, σ_j) be the points at which Greedy Filter got stuck. We check whether some point $\pi_{i+2^k}, k \in \mathbb{N}$, is far from all points of σ using HeurFar. If so, we conclude that

```
1: procedure NegativeFilter($\pi, \sigma, \delta$)
2:     $(i, j) \leftarrow$ last indices of close points in greedy filter
3:     $s \leftarrow 1$
4:     while $i + s \leq n$ do
5:         if SimpleBoundary($\pi_{i+s}, \sigma, \delta$) is non-free then
6:             return “far”
7:         $s \leftarrow 2s$
8:
9:     Repeat lines 3 to 7 with the roles of $\pi$ and $\sigma$ swapped
10:
11:    return “unknown”
```

Algorithm 17: Negative filter, where in the two if statements we do a search with adaptive step size on σ and π, respectively.

```
1: procedure FindCloseCurves($\pi, \delta$)
2:     $C \leftarrow$ kdtree.query($\pi, \delta$)
3:     $R \leftarrow \emptyset$
4:     for all $\sigma \in C$ do
5:         if FrechetDistanceDecider($\pi, \sigma, \delta$) = “close” then
6:             $R \leftarrow R \cup \{\sigma\}$
7:     return $R$
```

Algorithm 18: The function for answering a range query.

$d_F(\pi, \sigma) > \delta$. We do the same with the roles of π and σ exchanged. See Algorithm 17 for the pseudocode of this filter; for a visualization see Figure 9.10c.

9.4 Query Data Structure

In this section we give the details of extending the fast decider to compute the Fréchet distance in the query setting. Recall that in this setting we are given a *curve dataset* $\mathcal{D}$ that we want to preprocess for the following queries: Given a polygonal curve π (the *query curve*) and a threshold distance δ, report all $\sigma \in \mathcal{D}$ that are δ-close to π. To be able to compare our new approach to existing work (especially the submissions of the GIS Cup) we present a query data structure here, which is influenced by the one presented in [30].

The most important component that we need additionally to the decider to obtain an efficient query data structure is a mechanism to quickly determine a set of candidate curves on which we can then run the decider presented above. The candidate selection is done using a kd-tree on 8-dimensional points, similar to the octree used in [30], see 9.4.1 for more details. The high-level structure of the algorithm for answering queries is shown in Algorithm 18.

9.4.1 Kd-Tree

Fetching an initial set of candidate curves via a space-partitioning data structure was already used in [30, 54, 87]. We use a kd-tree which contains 8-dimensional points, each corresponding to one of the curves in the data set. Four dimensions of the 8-dimensional points are used for the start point and end point of the curve (two dimensions each). Note that two curves can only be close with respect to the Fréchet distance if their start points are close (and equivalently for the end points). Especially, if any of these four dimensions of two curves differ by more than δ, then these curves have a Fréchet distance larger than δ. The remaining four dimensions are used for the maximum/minimum coordinates in x/y direction. This is because, if the extremal coordinate in one direction of one curve compared to another is larger than δ, then the point that induces this large extremal coordinate cannot be matched to any point on the other curve. We can then query this kd-tree with the threshold distance δ and obtain a set of candidate curves. Note that this query does not have any false negatives, but might contain false positives, which we then filter out in the later stages of our algorithm.

9.5 Implementation Details

Square root. Computing which parts are close and which are far between a point and a segment involves intersecting a circle and a line segment, which in turn requires computing a square root. As square roots are computationally quite expensive, we avoid them by:

- filtering out simple comparisons by heuristic checks not involving square roots
- testing $x < a^2$ instead of $\sqrt{x} < a$ (and analogous for other comparisons)

While these changes seem trivial, they have a significant effect on the running time due to the large amount of distance computations in the implementation.

Recursion. Note that the complete decider (Algorithm 13) is currently formulated as a recursive algorithm. Indeed, our implementation is also recursive, which is feasible due to the logarithmic depth of the recursion. An iterative variant that we implemented turned out to be equally fast but more complicated, thus we settled for the recursive variant.

9.6 Experiments

In the experiments, we aim to substantiate the following two claims. First, we want to verify that our main contribution, the decider, actually is a significant improvement over the state of the art. To this end, we compare our implementation with the – to our knowledge – currently fastest Fréchet distance decider, namely [30]. Second, we want to verify that our improvements in the decider setting also carry over to the query setting, also significantly improving the state of the art. To show this, we compare to the top three submissions of the GIS Cup.

data set	type	#curves	mean hops	stddev hops
Sigspatial	synthetic GPS-like	20199	247.8	154.0
Characters	handwritten	2858	120.9	21.0
GeoLife	GPS (multi-modal)	16966	1080.4	1844.1

Table 9.1: Information about the data sets used in the benchmarks.

We use three different data sets: the GIS Cup set (Sigspatial) [9], the handwritten characters (Characters) [65], and the GeoLife data set (GeoLife) [102]. For all experiments, we used a laptop with an Intel i5-6440HQ processor with 4 cores and 16GB of RAM.

Hypotheses. In what follows, we verify the following hypotheses:

(1) Our implementation is significantly faster than the fastest previously known implementation in the query and in the decider setting.

(2) Our implementation is fast on a wide range of data sets.

(3) Each of the described improvements of the decider speeds up the computation significantly.

(4) The running time of the complete decider is proportional to the number of recursive calls.

The first two we verify by running time comparisons on different data sets. The third we verify by leaving out single pruning rules and then comparing the running time with the final implementation. Finally, we verify the fourth hypothesis by correlating the running time for different decider computations against the number of recursive calls encountered during the computation.

9.6.1 Data Sets Information

Some properties of the data sets are shown in Table 9.1. Sigspatial has the most curves, while GeoLife has by far the longest. Characters is interesting as it does not stem from GPS data. By this selection of data sets, we hope to cover a sufficiently diverse set of curves.

Hardware. We used standard desktop hardware for our experiments. More specifically, we used a laptop with an Intel i5-6440HQ processor with 4 cores (2.6 to 3.1 GHz) with cache sizes 256KiB, 1MiB, and 6MiB (L1, L2, L3).

Code. The implementation is written in modern C++ and only has the standard library and OpenMP as dependencies. The target platforms are Linux and OS X, with little work expected to adapt it to other platforms. The code was optimized for speed as well as readability (as we hope to give a reference implementation).

9.6.2 Decider Setting

In this section we test the running time performance of our new decider algorithm (Algorithm 15). We first describe our new benchmark using the three data sets, and then discuss our experimental findings, in particular how the performance and improvement over the state of the art varies with the distance and also the "neighbor rank" in the data set.

Benchmark. For the decider, we want to specifically test how the decision distance δ and how the choice of the second curve σ influences the running time of the decider. To experimentally evaluate this, we create a benchmark for each data set $\mathcal{D}$ in the following way. We select a random curve $\pi \in \mathcal{D}$ and sort the curves in the data set $\mathcal{D}$ by their distance to π in increasing order, obtaining the sequence $\sigma_1, \ldots, \sigma_n$. We define the *neighbor rank* of σ_i with respect to π to be its index i in the ordering. To create the benchmark, for all $k \in \{1, \ldots, \lfloor \log n \rfloor\}$ we

- select a curve $\sigma \in \{\sigma_{2^k}, \ldots, \sigma_{2^{k+1}-1}\}$ uniformly at random[1],
- compute the exact distance $\delta^* := \mathrm{d_F}(\pi, \sigma)$,
- for each $l \in \{-10, \ldots, 0\}$, add benchmark tests $(\pi, \sigma, (1 - 2^l) \cdot \delta^*)$ and $(\pi, \sigma, (1 + 2^l) \cdot \delta^*)$.

By repeating this process for 1000 uniformly random curves $\pi \in \mathcal{D}$, we create 1000 test cases for every pair of k and l.

Running times. First we show how our implementation performs in this benchmark. In Figure 9.11 we depict timings for running our implementation on the benchmark for all data sets. We can see that decision distances larger than the exact Fréchet distance are harder than smaller decision distances. This effect is most likely caused by the fact that decider instances with positive result need to find a path through the free-space diagram, while negative instances might be resolved earlier as it already becomes clear close to the lower left corner of the free-space diagram that there cannot exist such a path. Also, the performance of the decider is worse for computations on $(\pi, \sigma, c \cdot \delta^*)$ when σ has a small neighbor rank (with respect to π) and c is close to 1. This seems natural, as curves which are closer are more likely in the data set to actually be of similar shape, and similar shapes often lead to bottlenecks in the free-space diagram (i.e., small regions where a witness path can barely pass through), which have to be resolved in much more detail and therefore lead to a higher number of recursive calls. It follows that the benchmark instances for low k and l close to 0 are the hardest; this is the case for all data sets. In CHARACTERS we can also see that for $k = 7$ there is suddenly a rise in the running time for certain distance factors. We assume that this comes from the fact that the previous values of k all correspond to the same written character and this changes for $k = 7$.

We also run the original code of the winner of the GIS Cup, namely [30], on our benchmark and compare it with the running time of our implementation. See Figure 9.12

[1]Note that for $k = \lfloor \log n \rfloor$ some curves might be undefined as possibly $2^{k+1} - 1 > n$. In this case we select a curve uniformly at random from $\{\sigma_{2^k}, \ldots, \sigma_n\}$.

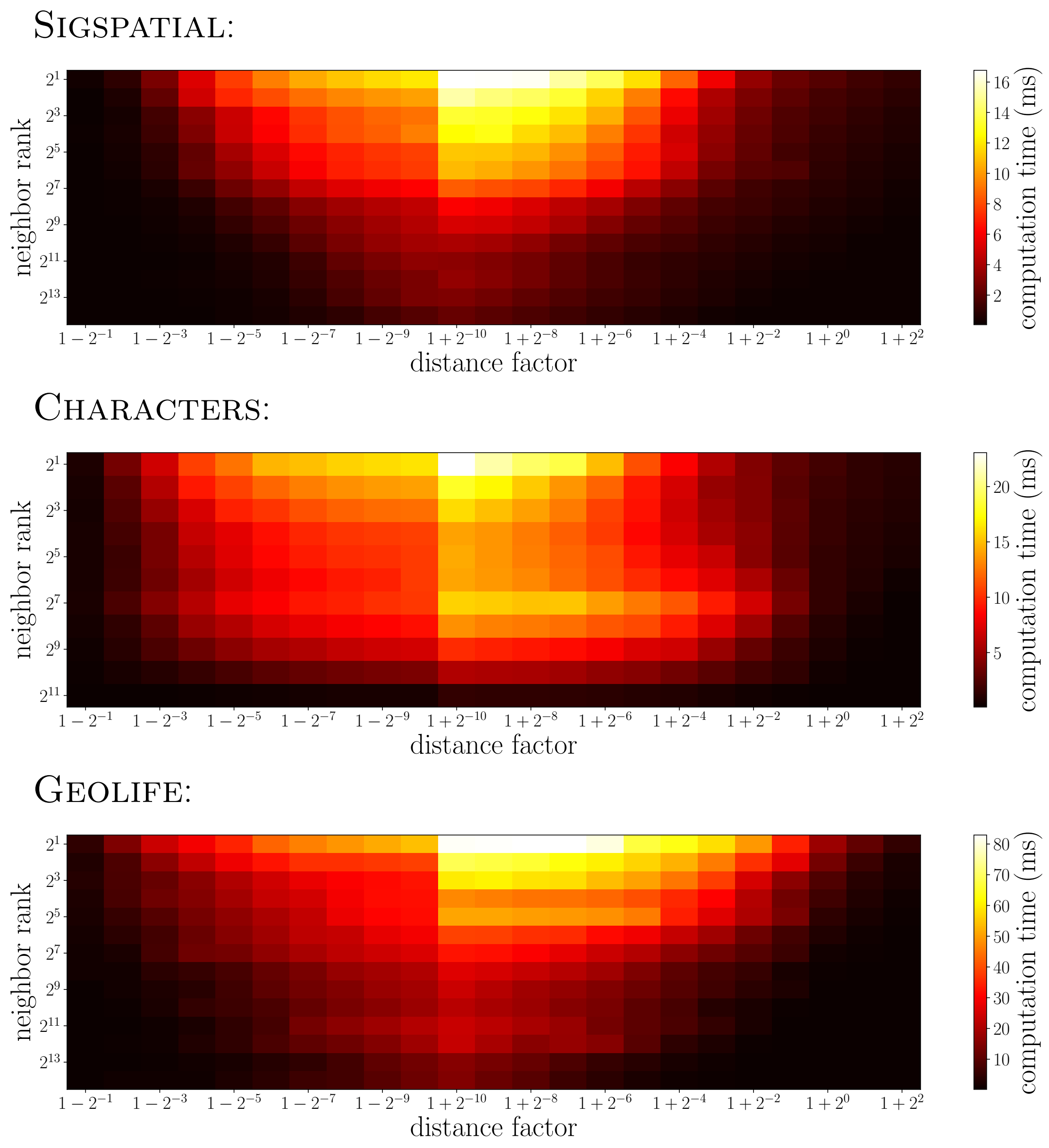


Figure 9.11: Running times of the decider benchmark when we run our implementation on it.

for the speed-up factors of our implementation over the GIS Cup winner implementation. The speed-ups obtained depend on the data set. While for every data set a significant amount of benchmarks for different k and l are more than one order of magnitude faster, for GeoLife even speed-ups by 2 orders of magnitude are reached. Speed-ups tend to be higher for larger distance factors. The results on GeoLife suggest that for longer curves, our implementation becomes significantly faster relative to the current state of the art. Note that there also are situations where our decider shows similar performance to the one of [30]; however, those are cases where both deciders can easily recognize that the curves are far (due to, e.g., their start or end points being far). We additionally show the percentage of instances that are already decided by the filters in Figure 9.13.

9.6.3 Influence of the Individual Pruning Rules

We also verified that the improvements that we introduced indeed are all necessary. In Section 9.2.3 we introduced six pruning rules. Rule I, i.e., “Empty Inputs”, is essential. If we were to omit it, we would hardly improve over the naive free-space exploration algorithm. The remaining five rules can potentially be omitted. Thus, for each of these pruning rules, we let our implementation run on the decider benchmark with this single rule disabled; and once with all rules enabled. See Table 9.2 for the results. Clearly, all pruning rules yield significant improvements when considering the timings of the GeoLife benchmark. All rules, except Rule IV, also show significant speed-ups for the other two data sets. Additionally, note that omitting Rule IIIb drastically increases the running time. This effect results from Rule IIIb being the main rule to prune large reachable parts, which we otherwise have to explore completely. One can clearly observe this effect in Figure 9.9.

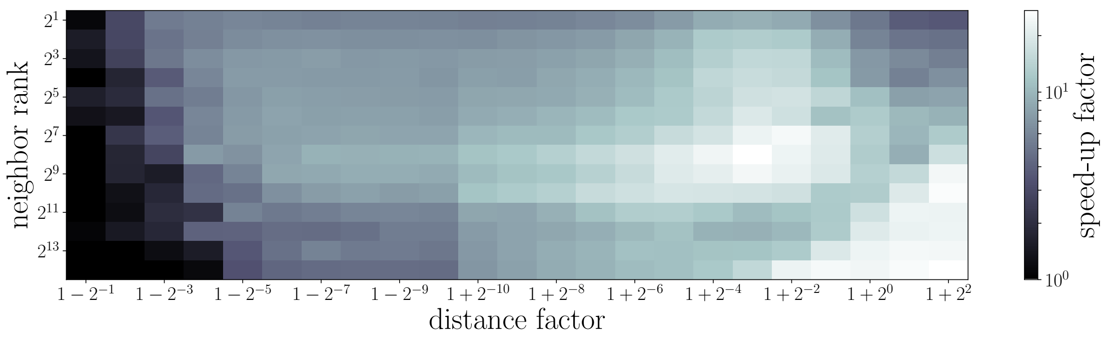
Filters. In Figure 9.13 we show what percentage of the queries are decided by the filters. We can see that the closer we get to the actually distance δ^* of two curves, the less likely it gets that the filters can make a decision. Furthermore, for the distances that are greater than δ^* the filters perform worse than for distances less than δ^*. We additionally observe that on Characters the filters perform significantly worse than on the other two data sets. Also the running times are inversely correlated with the percentage of decisions of the filters as returning earlier in the decider naturally reduces the overall runtime.

9.6.4 Query Setting

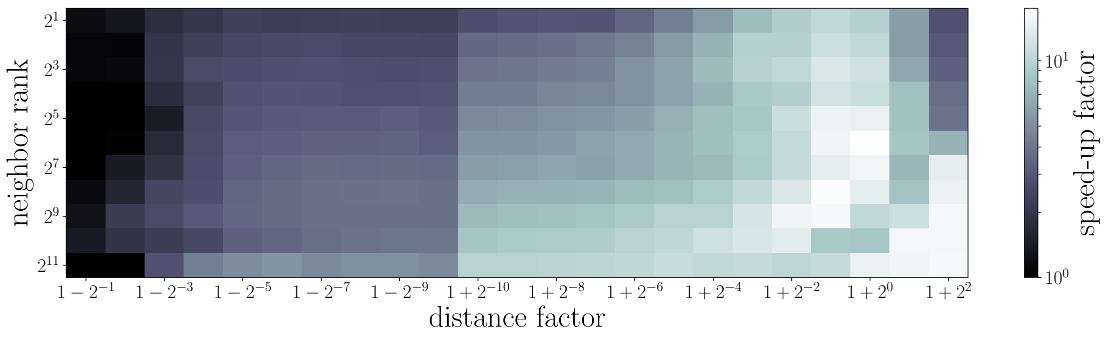
We now turn to the experiments conducted for our query data structure, which we explained in Section 9.4.

Benchmark. We build a query benchmark similar to the one used in [30]. For each $k \in \{0, 1, 10, 100, 1000\}$, we select a random curve $\pi \in \mathcal{D}$ and then pick a threshold distance δ such that a query of the form (π, δ) returns exactly $k + 1$ curves (note that the curve π itself is also always returned). We repeat this 1000 times for each value of k and also create such a benchmark for each of the three data sets.

Sigspatial:



Characters:



Geolife:

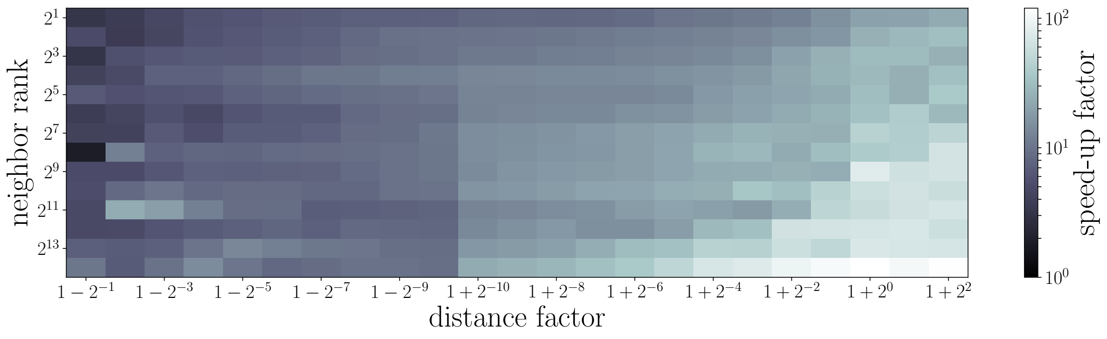


Figure 9.12: The speed-up factors obtained over the GIS Cup winner on the decider benchmark.

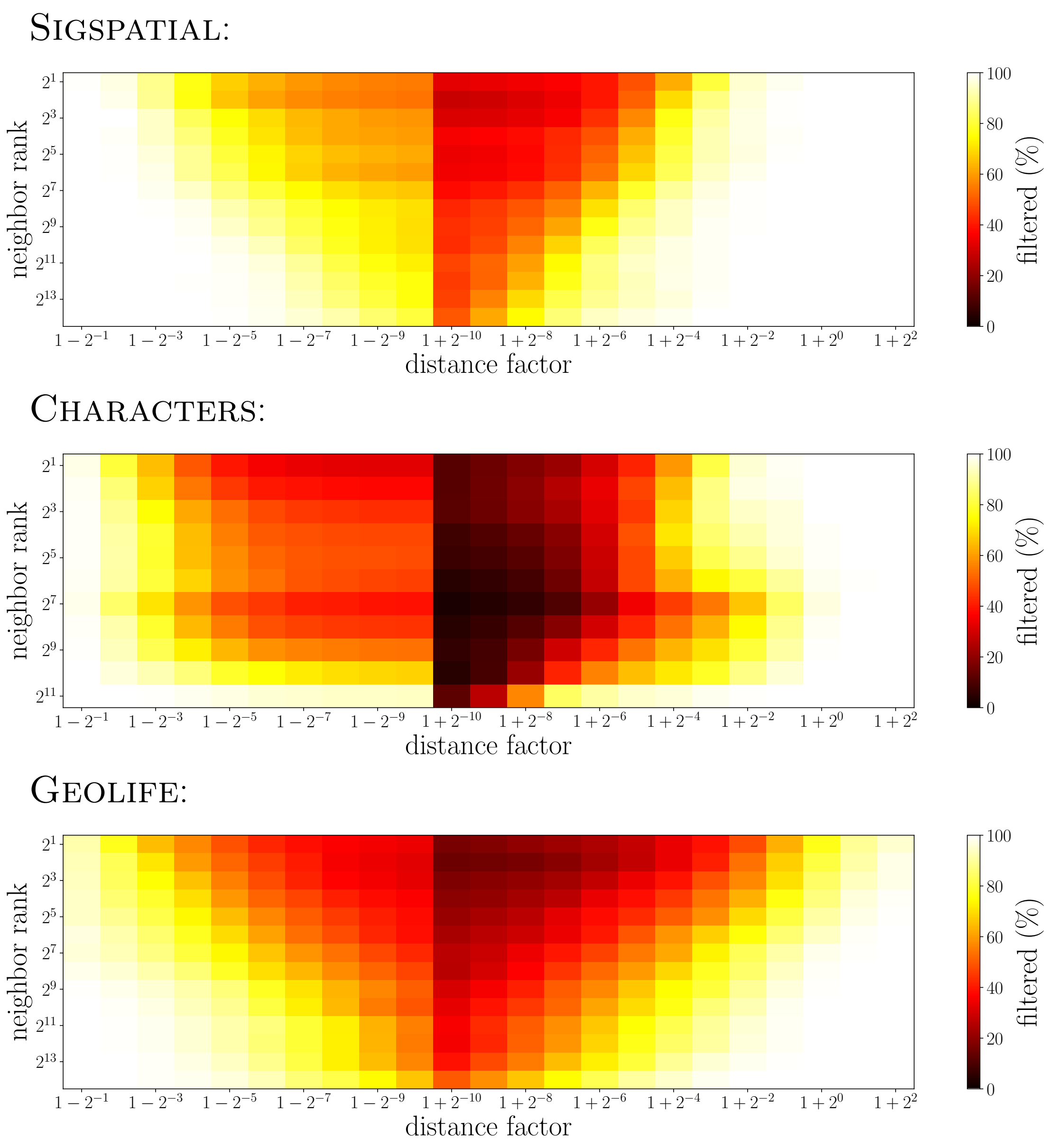


Figure 9.13: The percentage of queries that are decided by the filters on the decider benchmark.

	Sigspatial	Characters	GeoLife
omit none	99.085	153.195	552.661
omit Rule II	112.769	204.347	1382.306
omit Rule IIIa	193.437	296.679	1779.810
omit Rule IIIb	5317.665	1627.817	385031.421
omit Rule IIIc	202.469	273.146	2049.632
omit Rule IV	110.968	161.142	696.382

Table 9.2: Times (in ms) for running the decider benchmarks with leaving out pruning steps. We only ran the first 100 queries for each k and l due to large running times when omitting the third rule.

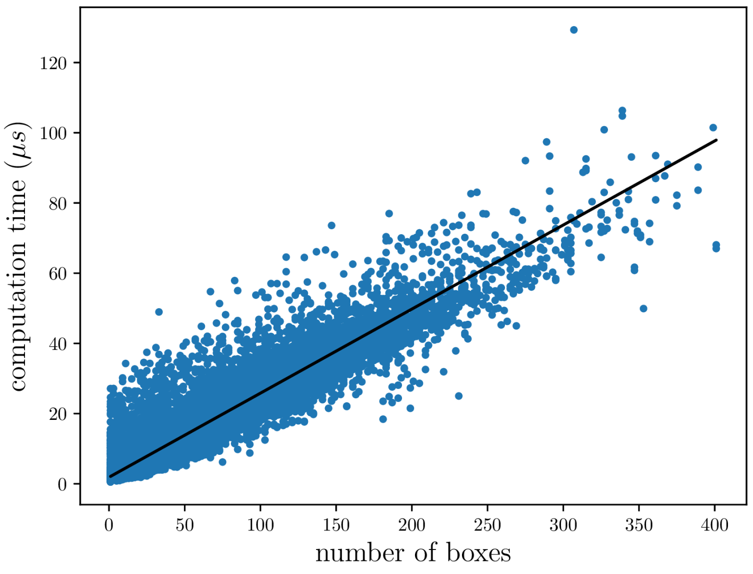


Figure 9.14: Shows how much time a call to the complete decider takes plotted over the number of boxes that the free-space diagram creates in total (i.e., even if a box is later split, it is still counted). The data are all exact computations (i.e., those where neither kd-tree nor filter decided) issued for the Sigspatial query benchmark. The black line is the linear regression ($r^2 = 0.91$).

Running times. We compare our implementation with the top three implementations of the GIS Cup on this benchmark. The results are shown in Table 9.3. Again the running time improvement of our implementation depends on the data set. For Characters the maximal improvement factor over the second best implementation is 14.6, for Sigspatial 17.3, and for GeoLife 29.1. For Sigspatial and Characters it is attained at $k = 1000$, while for GeoLife it is reached at $k = 100$ but $k = 1000$ shows a very similar but slightly smaller factor.

To give deeper insights about the single parts of our decider, a detailed analysis of the running times of the single parts of the algorithm is shown in Table 9.4. Again we witness different behavior depending on the data set. It is remarkable that for Sigspatial the running time for $k = 1000$ is dominated by the greedy filter. This suggests that improving the filters might still lead to a significant speed-up in this case. However, for most of the remaining cases the running time is clearly dominated by the complete decider, suggesting that our efforts of improving the state of the art focused on the right part of the algorithm.

9.6.5 Other Experiments

The main goal of the complete decider was to reduce the number of recursive calls that we need to consider during the computation of the free-space diagram. Due to our optimized algorithm to compute simple boundaries with adaptive step size, we expect roughly a constant (or possibly polylogarithmic) running time effort per box, essentially independent of the size of the box. To test this hypothesis, we ask whether the number of recursive calls is indeed correlated with the running time. To test this, we measured the time for each complete decider call in the query benchmark and plotted it over the number of boxes that were considered in this call. The result of this experiment is shown in Figure 9.14. We can see a practically (near-)linear correlation between the number of boxes and the running time.

9.7 Certificates

Whenever we replace a naive implementation in favor of a fast, optimized, but typically more complex implementation, it is almost unavoidable to introduce bugs to the code. As a useful countermeasure the concept of *certifying algorithms* has been introduced; we refer to [131] for a survey. In a nutshell, we aim for an implementation that outputs, apart from the desired result, also a proof of correctness of the result. Its essential property is that the certificate should be *simple* to check (i.e., much simpler than solving the original problem). In this way, the certificate gives any user of the implementation a simple means to check the output for any conceivable instance.

Following this philosophy, we have made our implementation of the Fréchet decider *certifying*: for any input curves π, σ and query distance δ, we are able to return, apart from the output whether the Fréchet distance of π and σ is at most δ, also a certificate c. On our realistic benchmarks, constructing this certificate slows down the Fréchet decider by roughly 50%. The certificate c can be checked by a simple verification procedure consisting of roughly 200 lines of code.

In Sections 9.7.1 and 9.7.2, we define our notion of YES and NO certificates, prove that they indeed certify YES and NO instances and discuss how our implementation finds

	Sigspatial					Characters					GeoLife				
k	0	1	10	100	1000	0	1	10	100	1000	0	1	10	100	1000
[30]	0.094	0.123	0.322	1.812	8.408	0.187	0.217	0.421	2.222	17.169	0.298	0.741	4.327	33.034	109.44
[54]	0.421	0.618	1.711	7.86	35.704	0.176	0.28	0.611	3.039	17.681	3.627	6.067	26.343	120.509	415.548
[87]	0.197	0.188	0.643	5.564	76.144	0.142	0.147	0.222	1.849	22.499	2.614	4.112	16.428	166.206	1352.19
ours	0.017	0.007	0.026	0.130	0.490	0.004	0.020	0.058	0.301	1.176	0.027	0.089	0.341	1.108	3.642

Table 9.3: Comparing the running times (in s) of the queries of the top three implementations of the GIS Cup 2017 with our new implementation on the query benchmark on all data sets (1000 queries per entry).

	Sigspatial					Characters					GeoLife				
k	0	1	10	100	1000	0	1	10	100	1000	0	1	10	100	1000
spatial hashing	0.002	0.003	0.005	0.017	0.074	0.002	0.002	0.004	0.011	0.032	0.006	0.009	0.016	0.032	0.091
greedy filter	0.004	0.006	0.024	0.143	0.903	0.004	0.010	0.032	0.153	0.721	0.009	0.017	0.060	0.273	1.410
equal-time filter	0.000	0.001	0.006	0.030	0.088	0.001	0.004	0.018	0.088	0.424	0.005	0.017	0.063	0.273	1.211
negative filter	0.001	0.002	0.010	0.044	0.107	0.003	0.012	0.038	0.152	0.309	0.008	0.020	0.069	0.200	0.606
complete decider	0.002	0.011	0.044	0.214	0.330	0.005	0.030	0.109	0.671	2.639	0.062	0.210	0.998	3.025	8.760

Table 9.4: Timings (in s) of the single parts of our query algorithm on the query benchmark on all three data sets. To avoid confusion, note that the sum of the times in this table do not match the entries in Table 9.3 as those are parallelized timings and additionally the timing itself introduces some overhead.

them. In Section 9.7.3, we describe the simple checking procedure for our certificates. Finally, we conclude with an experimental evaluation in Section 9.7.4.

9.7.1 Certificate for YES Instances

To verify that $\mathrm{d_F}(\pi, \sigma) \leq \delta$, by definition it suffices to give a feasible traversal, i.e., monotone and continuous functions $f : [0,1] \to [1,n]$ and $g : [0,1] \to [1,m]$ such that for all $t \in [0,1]$, we have $(\pi(f(t)), \sigma(g(t))) \in F$, where $F = \{(p,q) \in [1,n] \times [1,m] \mid \|\pi(p) - \sigma(q)\| \leq \delta\}$ denotes the free-space (see Section 9.2.1). We slightly simplify this condition by discretizing $(f(t), g(t))_{t \in [0,1]}$, as follows.

Definition 9.1. *We call $T = (t_1, \dots, t_\ell)$ with $t_i \in [1,n] \times [1,m]$ a YES certificate if it satisfies the following conditions: (See also Figure 9.15 for an example.)*

(1) (start) $t_1 = (1,1) \in F$,

(2) (end) $t_\ell = (n,m) \in F$,

(3) (step) For any $t_k = (p,q)$ and $t_{k+1} = (p', q')$, we have either

(a) $p' = p$ and $q' > q$: we require that $(p, \bar{q}) \in F$ for all $\bar{q} \in \{q, \lceil q \rceil, \dots, \lfloor q' \rfloor, q'\}$,

(b) $q' = q$ and $p' > p$: we require that $(\bar{p}, q) \in F$ for all $\bar{p} \in \{p, \lceil p \rceil, \dots, \lfloor p' \rfloor, p'\}$,

(c) $i \leq p < p' \leq i+1$, $j \leq q < q' \leq j+1$ for some $i \in \{1, \dots, n\}, j \in \{1, \dots, m\}$: we require that $(p,q), (p',q') \in F$.

It is straightforward to show that a YES certificate T proves correctness for YES instances as follows.

Proposition 9.2. *Any YES certificate $T = (t_1, \dots, t_\ell)$ with $t_i \in [1,n] \times [1,m]$ proves that $\mathrm{d_F}(\pi, \sigma) \leq \delta$.*

Proof. View T as a polygonal curve in $[1,n] \times [1,m]$ and let $\tau : [0,1] \to [1,n] \times [1,m]$ be a reparameterization of T. Let f, g be the projection of τ to the first and second coordinate, respectively. Note that by the assumption on T, f and g are monotone and satisfy $(f(0), g(0)) = (1,1)$ and $(f(1), g(1)) = (n,m)$. We claim that $(f(t), g(t)) \in F$ for all $t \in [0,1]$, which thus yields $\mathrm{d_F}(\pi, \sigma) \leq \delta$ by definition.

To see the claim, we recall that for any *cell* $[i, i+1] \times [j, j+1]$, the free-space restricted to this cell, i.e., $F \cap [i, i+1] \times [j, j+1]$, is convex (as it is the intersection of an ellipse with $[i, i+1] \times [j, j+1]$, see [19]). Observe that for any segment from $t_k = (p,q)$ to $t_{k+1} = (p', q')$, we (implicitly) decompose it into subsegments contained in single cells (e.g., for $p' = p$ and $q' > q$, the segment from (p,q) to (p,q') is decomposed into the segments connecting the sequence $(p,q), (\lceil p \rceil, q), \dots, (\lfloor p' \rfloor, q), (p'q')$. As each such subsegment is contained in a single cell, by convexity we see that the whole subsegment is contained in F if the corresponding endpoints of the subsegment are in F. This concludes the proof. $\square$

It is not hard to prove that for YES instances, such a certificate always exists (in fact, there always is a certificate of length $O(n+m)$). Furthermore, for each YES instance in our benchmark set, our implementation indeed finds and returns a YES instance, in a way we describe next.

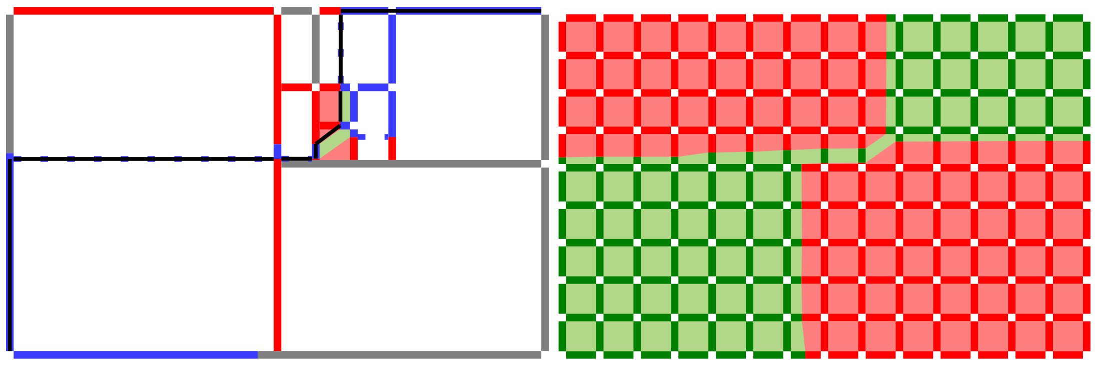


Figure 9.15: Example of a YES instance and its certificate. The right picture shows the free-space of the instance. The left picture illustrates the parts of the free-space explored by our algorithm and indicates the computed YES certificate by black lines.

Certifying positive filters. It is straightforward to construct YES certificates for instances that are resolved by our positive filters (Bounding Box Check, Greedy and Adaptive Equal-Time): All of these filters implicitly construct a feasible traversal. In particular, for any instance for which the Bounding Box Check applies (which shows that any pair of points of π and σ are within distance δ), already the sequence $((1,1),(n,1),(n,m))$ yields a YES certificate.

For Greedy, note that the sequence of positions (i,j) visited in Algorithm 16 yields a YES certificate: Indeed, any step from (i,j) is either a vertical step to $(i,j+s)$ (corresponding to case (3a)), a horizontal step to $(i+s,j)$ (corresponding to case (3b)), or a diagonal step *within a cell* to $(i+1,j+1)$ (corresponding to Case (3c) of Definition 9.1). Furthermore, such a step is only performed if it stays within the free-space.

Finally, for Adaptive Equal-Time, we also record the sequence of positions (i,j) visited in Algorithm 16 (recall that here, we change the set of possible steps for $s > 1$ to $S = \{(i+s, j+s')\}$ with $s' = \lfloor \frac{m-j}{n-i} \cdot s \rfloor$) – with the only difference that we need to replace any step from (i,j) to $(i+s,j+s')$ by the sequence $(i,j),(i+s,j),(i+s,j+s')$. Note that this sequence satisfies Condition (step) of Definition 9.1, as Adaptive Equal-Time only performs this step if it can verify that *all pairwise distances* between $\pi[i,i+s]$ and $\sigma[j,j+s']$ are bounded by δ.

Certifying YES instances in the complete decider. Recall that the complete decider via free-space exploration decides an instance by recursively determining, given the inputs B_l^R, B_b^R of a box B, the corresponding outputs B_r^R, B_t^R. In particular, YES instances are those with $(n,m) \in B_t^R$ (or equivalently $(n,m) \in B_r^R$) for the box $B = [1,n] \times [1,m]$. To certify such instances, we memorize for each point in B_r^R and B_t^R a predecessor of a feasible traversal from $(1,1)$ to this point. Note that here, it suffices to memorize such a predecessor only for the *first*, i.e., lowest or leftmost, *point of each interval* in B_r^R and B_t^R (as any point in this interval can be reached by traversing to the first point of the interval and then along this reachable interval to the destination point). This gives rise to a straightforward recursive approach to determine a feasible traversal.

In the complete decider, whenever we determine some output B_t^R, it is because of one of the following reasons: (1) one of of our pruning rules is successful, (2) the box B is on the cell-level, or (3) we determine B_t^R as the union of the outputs $(B_1)_t^R$, $(B_2)_t^R$ of the boxes B_1, B_2 obtained by splitting B vertically. Note that we only need to consider the case in which B_t^R is determined as non-empty (otherwise nothing needs to be memorized). Let us consider each case separately.

If reason (1) determines a non-empty B_t^R, then this happens either by Rule IIIb or by Rule IIIc. Note that in both cases, B_t^R consists of a single interval. If Rule IIIb applies, then the last, i.e., topmost, point on B_l is reachable and proves that the *free prefix* of B_t^R is reachable. Thus, we store *the last interval of* B_l^R as the responsible interval for the (single) interval in B_t^R. Similarly, if Rule IIIc applies, then consider the first, i.e., leftmost, point $(i, j_{\max})$ on B_t. Since the rule applies, the opposite point $(i, j_{\min})$ on B_l must be reachable and the path $\{i\} \times [j_{\min}, j_{\max}]$ must be free. Thus, we can store *the interval of* B_b^R *containing* $(i, j_{\min})$ as the responsible interval for the (single) interval in B_t^R.

If reason (2) determines a non-empty B_t^R, then we are on a cell-level. In this case, either B_l^R or B_b^R is a non-empty interval, and we can store such an interval as the responsible interval for the (single) interval in B_t^R. Finally, if reason (3) determines a non-empty B_t^R, then we simply keep track of the responsible interval for each interval in $(B_1)_t^R$ and $(B_2)_t^R$ (to be precise, if the last interval of $(B_1)_t^R$ and the first interval of $(B_2)_t^R$ overlap by the boundary point, we merge the two corresponding intervals and only keep track of the responsible interval of the last interval of $(B_1)_t^R$ and can safely forget about the responsible interval of the first interval of $(B_2)_t^R$.

Note that we proceed analogously for outputs B_r^R. Furthermore, the required memorization overhead is very limited.

It is straightforward to use the memorized information to compute a YES certificate recursively: Specifically, to compute a YES certificate reaching some point x on an output interval I, we perform the following steps. Let J be the responsible interval of I. We recursively determine a YES certificate reaching the first point J. Then we append a point to the certificate to traverse to the point of J from which we can reach the first point of I (this point is easily determined by distinguishing whether we are on the cell-level, and whether J is opposite to I or intersects I in a corner point). We append the first point of I to the certificate, and finally append the point x to the certificate.[2] By construction, the corresponding traversal never leaves the free-space. Using this procedure, we can compute a YES certificate by computing a YES certificate reaching (n, m) on the last interval of B_t^R for the initial box $B = [1, n] \times [1, m]$.

9.7.2 Certificate for NO Instances

We say that a point (p, q) lies on the bottom boundary if $q = 1$, on the right boundary if $p = n$, on the top boundary if $q = m$, and on the left boundary if $p = 1$. Likewise, we say that a point (p', q') lies to the lower right of a point (p, q), if $p \le p'$ and $q \ge q'$.

Definition 9.3. *We call $T = (t_1, \dots, t_\ell)$ with $t_i \in [1, n] \times [1, m]$ a NO certificate if it satisfies the following conditions: (See also Figure 9.16 for an example.)*

(1) (start) t_1 lies on the right or bottom boundary and $t_1 \notin F$,

(2) (end) t_ℓ lies on the left or upper boundary and $t_\ell \notin F$,

[2]To be precise, we only append a point if it is different from the last point of the current certificate.

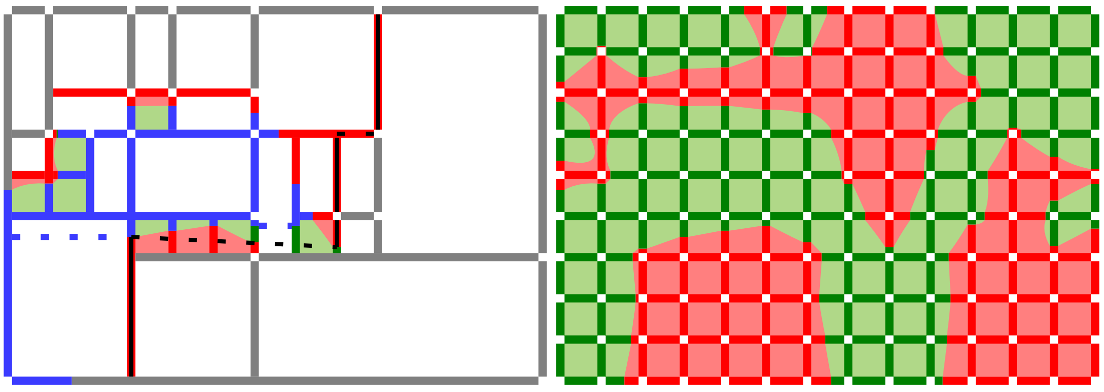


Figure 9.16: Example of a NO instance and its certificate. The right picture shows the free-space of the instance. The left picture illustrates the parts of the free-space explored by our algorithm and indicates the computed NO certificate by black lines.

(3) (step) For any $t_k = (p, q)$ and $t_{k+1} = (p', q')$, we have either

- *(a) $p' = p$ and $q' > q$: for any neighboring elements $\bar{q}_1, \bar{q}_2$ in $q, \lceil q \rceil, \dots, \lfloor q' \rfloor, q'$, we require that $(\{p\} \times [\bar{q}_1, \bar{q}_2]) \cap F = \emptyset$,*
- *(b) $q' = q$ and $p' < p$: for any neighboring elements $\bar{p}_1, \bar{p}_2$ in $p', \lceil p' \rceil, \dots, \lfloor p \rfloor, p$, we require that $([\bar{p}_1, \bar{p}_2] \times \{q\}) \cap F = \emptyset$,*
- *(c) t_{k+1} lies to the lower right of t_k, i.e., $p \leq p'$ and $q \geq q'$.*

We prove that a NO certificate T proofs correctness for NO instances as follows.

Proposition 9.4. *Any NO certificate $T = (t_1, \dots, t_\ell)$ with $t_i \in [1, n] \times [1, m]$ proves that* $d_F(\pi, \sigma) > \delta$.

Proof. We inductively prove that no feasible traversal from $(1, 1)$ to (n, m) can visit any point to the lower right of t_i, for all $1 \leq i \leq \ell$. As an immediate consequence, $d_F(\pi, \sigma) > \delta$, since t_ℓ lies on the left or upper boundary and thus *any* feasible traversal must visit a point to the lower right of t_ℓ – hence, such a traversal cannot exists.

As base case, note that t_1 lies on the right or bottom boundary and is not contained in the free-space. Thus, by monotonicity, no feasible traversal can visit any point to the lower right of t_1. Thus, assume that the claim is true for $t_i = (p, q)$ and consider the next point $t_{i+1} = (p', q')$ in the sequence. If t_{i+1} lies to the lower right of t_i, the claim is trivially fulfilled for t_{i+1} by monotonicity. If, however, $p' = p$ and $q' > q$, then Condition (3a) of Definition 9.3 is equivalent to $(p \times [q, q']) \cap F = \emptyset$. Note that any feasible traversal visiting a point to the lower right of t_{i+1} must either visit a point to the lower right of t_i – which is not possible by inductive assumption – or must cross the path $\{p\} \times [q, q']$ – which is not possible as $(\{p\} \times [q, q']) \cap F = \emptyset$. We argue symmetrically for the case that $q' = q$ and $p' < p$. This concludes the proof. □

Note that our definition of NO certificate essentially coincides with the definition of a cut of positive width in [56]. For NO instances, such a NO certificate always exists (in contrast to YES certificates, the shortest such certificate is of length $\Theta(n^2)$ in the worst case). For all NO instances in our benchmark sets, our implementation manages to find and return such a NO certificate, in a way we describe next.

```
procedure ComputeNOCertificate($\pi, \sigma, \delta$)
    $N \leftarrow$ non-free segments determined by CompleteDecider($\pi, \sigma, \delta$)
    $Q \leftarrow \{I \in N \mid$ lowerRight($I$) lies on bottom or right boundary$\}$
    Build orthogonal range search data structure $D$,
                                storing all $I \in N \setminus Q$ under the key lowerRight($I$).
    while $Q \neq \emptyset$ do
        Pop any element $I$ from $Q$
        if upperLeft($I$) lies on top or left boundary then
            Reconstruct sequence of intervals leading to $I$
            return corresponding NO certificate
        else
            $Q' \leftarrow D$.ReportAndDelete(upperLeft($I$))
                    ▷ reports $J$ if lowerRight($J$) is to the lower right of upperLeft($I$)
            $Q \leftarrow Q \cup Q'$
```

Algorithm 19: High-level code for computing a NO certificate.

Certifying the negative filter. It is straightforward to compute a NO certificate for instances resolved by our negative filter. Note that this filter, if successful, determines an index $p \in \{1, \dots, n\}$ such that π_p is far from all points on σ, or symmetrically an index $q \in \{1, \dots, m\}$ such that σ_q is far from all points on π. Thus, in these cases, we can simply return the NO certificate $((p, 1), (p, m))$ or $((n, q), (1, q))$, respectively.

Certifying NO instances in the complete decider. Whenever the complete decider via free-space exploration returns a negative answer, the explored parts of the free-space diagram must be sufficient to derive a negative answer. This gives rise to the following approach: Consider all non-free segments computed by the complete decider. We start from a non-free segment touching the bottom or right boundary and traverse non-free segments (possibly also making use of monotonicity steps according to Case (3c) of Definition 9.3) and stop as soon as we have found a non-free segment touching the left or top boundary.

Formally, consider Algorithm 19. Here, we use the notation that lowerRight(I) denotes the lower right endpoint of I, i.e., the right endpoint if I is a horizontal segment and the lower endpoint if I is a vertical segment. Analogously, upperLeft(I) denotes the upper left endpoint of I.

The initial set of non-free segments in Algorithm 19 consists of the non-free segments of *all simple boundaries* determined by the complete decider via free-space exploration. We maintain a queue Q of non-free segments, which initially contains all non-free segments touching the right or bottom boundary. Furthermore, we maintain a data structure D of yet *unreached* non-free intervals. Specifically, we require D to store intervals I under the corresponding key lowerRight(I) $\in [1, n] \times [1, m]$ in a way to support the query ReportAndDelete(p): Such a query returns all $I \in D$ such that lowerRight(I) lies to the lower right of p and deletes all returned intervals from D.

Equipped with such a data structure, we can traverse all elements in the queue as follows: We delete any interval I from Q and check whether it reaches the upper or left boundary. If this is the case, we have (implicitly) found a NO certificate, which we then reconstruct (by memorizing why each element of the queue was put into the queue). Otherwise, we add to Q all intervals from D that can be reached by a monotone step (according to Case (3c) of Definition 9.3) from upperLeft(I); these intervals are additionally deleted from D.

To implement D, we observe that it essentially asks for a 2-dimensional orthogonal range search data structure where the ranges are unbounded in two directions (and bounded in the other two). Already for the case of 2-dimensional ranges with only a single unbounded direction (sometimes called 1.5-dimensional), a very efficient solution is provided by a classic data structure due to McCreight, the priority search tree data structure [132]. We can adapt it in a straightforward manner to implement D such that it (1) takes time $O(d \log d)$ and space $O(d)$ to construct D on an initial set of size d and (2) supports ReportAndDelete(p) queries in time $O(k + \log d)$, where k denotes the number of reported elements. Thus, Algorithm 19 can be implemented to run in time $O(|N| \log |N|)$.

9.7.3 Certificate Checker

It remains to describe how to check the correctness of a given certificate $T = (t_1, \dots, t_\ell)$. For this, we simply verify that all properties of Definition 9.1 or Definition 9.3 are satisfied.

Checking YES certificates. Observe that the only conditions in the definition of YES instances are either simple comparisons of neighboring elements t_k, t_{k+1} in the sequence or *freeness tests*, specifically, whether a give position $p \in [1, n] \times [1, m]$ is free, i.e, whether π_{p_1} and σ_{p_2} have distance at most δ. The latter test only requires interpolation along a curve segment (to obtain π_{p_1} and σ_{p_2}) and a Euclidean distance computation. Thus, YES certificates are extremely simple to check.

Checking NO certificates. Checking NO certificates involves a slightly more complicated geometric primitive than the freeness tests of YES certificates. Apart from simple comparisons of neighboring elements t_k, t_{k+1}, the conditions in the definition involve the following *non-freeness tests*: Given a (sub)segment $\pi[p, p']$ with $i \leq p \leq p' \leq i + 1$ for some $i \in [n]$, as well as a point $\sigma(q)$ with $q \in [1, m]$, determine whether all points on $\pi[p, p']$ have distance strictly larger than δ from $\sigma(q)$. Besides the (simple) interpolation along a line segment to obtain $\sigma(q)$, we need to determine intersection points of the line containing $\pi[p, p']$ and the circle of radius δ around $\sigma(q)$ (if these exist). From these intersection points, we verify that $\pi[p, p']$ and the circle do not intersect, concluding the check.

Summary. In summary, certificate checkers are straightforward and simple to implement.

9.7.4 Certification Experiments

We evaluate the overhead introduced by computing certificates using our benchmark sets for the query setting. In particular, as our implementation can be compiled both as a certifying and a non-certifying version, we compare the running times of both

	SIGSPATIAL				
k	0	1	10	100	1000
computation without certification	**6.9**	**21.3**	**84.5**	**429.4**	**1409.1**
certifying computation	**10.0**	**29.6**	**117.8**	**553.8**	**1840.2**
–computation of certificates	1.0	3.7	12.0	40.9	65.7
–YES certificates (complete decider)	0.0	0.4	1.6	8.2	12.1
–NO certificates (complete decider)	1.0	3.3	10.0	31.4	50.0
checking certificates	**6.4**	**13.9**	**63.7**	**426.5**	**3803.2**
–checking filter certificates	6.0	11.3	51.6	361.2	3666.7
–checking complete decider certificates	0.4	2.6	12.1	65.3	136.5

	CHARACTERS				
k	0	1	10	100	1000
computation without certification	**12.1**	**55.5**	**205.8**	**1052.8**	**4080.3**
certifying computation	**20.3**	**91.6**	**311.2**	**1589.8**	**5895.8**
–computation of certificates	4.0	20.0	59.6	220.7	470.2
–YES certificates (complete decider)	0.0	0.5	2.3	24.9	181.3
–NO certificates (complete decider)	3.9	19.2	56.3	186.2	259.4
checking certificates	**6.3**	**21.6**	**76.8**	**457.7**	**2626.1**
–checking filter certificates	5.3	14.8	49.7	278.0	1759.8
–checking complete decider certificates	1.0	6.8	27.2	179.6	866.3

	GEOLIFE				
k	0	1	10	100	1000
computation without certification	**82.2**	**251.1**	**1156.6**	**3663.1**	**11452.4**
certifying computation	**142.1**	**414.6**	**1834.4**	**5304.2**	**16248.7**
–computation of certificates	40.1	100.7	388.0	767.7	1827.8
–YES certificates (complete decider)	0.0	3.2	20.0	87.6	247.5
–NO certificates (complete decider)	39.7	96.7	364.5	664.8	1517.1
checking certificates	**70.9**	**185.2**	**733.7**	**3595.4**	**20188.4**
–checking filter certificates	45.2	85.9	283.8	1754.0	12987.5
–checking complete decider certificates	25.7	99.4	450.0	1841.4	7200.9

Table 9.5: Certificate computation and check times on query setting benchmark (in ms). The first and second bold lines show the running time of our implementation compiled without and with certification, respectively. For the certifying variant, we also give the times to compute YES and NO certificates of the complete decider (note that filter certificates are computed on the fly by the filters and hence cannot be separately measured; also, this certificate computation time does not include the overhead to record additional information during the complete decision procedure). Finally, we give running times for checking correctness of certificates.

versions. The results are depicted in Table 9.5. Notably, the slowdown factor introduced by computing certificates ranges between 1.29 and 1.46 (Sigspatial), 1.44 and 1.67 (Characters) and 1.42 and 1.73 (GeoLife). As expected, the certificate computation time is dominated by the task of generating NO certificates (which is more complex than computing YES certificates), even for large values of k for which most unfiltered instances are YES instances.

At first sight, it might be surprising that checking the certificates takes longer than computing them. However, this is due to the fact that our filters often display sublinear running time behavior (by using the heuristic checks and adaptive step sizes). However, to keep our certificate checker elementary, we have not introduced any such improvements to the checker, which thus has to traverse essentially all points on the curves. This effect is particularly prominent for large values of k.

9.8 Conclusion

In this chapter we presented an implementation for computing the Fréchet distance which beats the state-of-the-art by one to two orders of magnitude in running time in the query as well as the decider setting. Furthermore, it can be used to compute certificates of correctness with little overhead. To facilitate future research, we created two benchmarks on several data sets – one for each setting – such that comparisons can easily be conducted. Given the variety of applications of the Fréchet distance, we believe that this result will also be of broader interest and implies significant speed-ups for other computational problems in practice.

This enables a wide range of future work. An obvious direction to continue research is to take it back to theory and show that our pruning approach provably has subquadratic runtime on a natural class of realistic curves. On the other hand, one could try to find further pruning rules or replace the divide-and-conquer approach by some more sophisticated search. To make full use of the work presented here, it would make sense to incorporate this algorithm in software libraries. Currently, we are not aware of any library with a non-naive implementation of a Fréchet distance decider or query. Finally, another possible research direction would be to work on efficient implementations for similar problems like the Fréchet distance under translation, rotation or variants of map matching with respect to the Fréchet distance. In summary, this chapter should lay ground to a variety of improvements for practical aspects of curve similarity.

CHAPTER 10

Fréchet Distance Under Translation

For a technical overview of this chapter see Section 3.5. This chapter is structured as follows. We discuss our approach in Section 10.2 and present the details of our decision algorithm in Section 10.3. We develop our approach also for the related, but different task to compute the distance value up to a given precision in Section 10.4, and evaluate our solutions for both settings in comparison to baseline approaches in Section 10.5.

10.1 Preliminaries

Throughout this chapter, we only consider trajectories in the Euclidean plane and we mostly consider the *discrete* Fréchet distance. To avoid confusion between the discrete Fréchet distance and the discrete Fréchet distance under translation, we also refer to the Fréchet distance as the *fixed-translation* Fréchet distance. In this chapter we view the discrete Fréchet distance under translation as a two-dimensional optimization problem with objective function $f(\tau) \coloneqq \mathrm{d_{dF}}(\pi, \sigma + \tau)$. Specifically, we consider the task to decide $\min_{\tau \in \mathbb{R}^2} f(\tau) \le \delta$*?* (*exact decider*) or to return a value in the range $[(1-\varepsilon)\min_{\tau\in\mathbb{R}^2} f(\tau), (1+\varepsilon)\min_{\tau\in\mathbb{R}^2} f(\tau)]$ (*approximate value computation*, multiplicative version). In fact, for implementation reasons (see Section 10.4 for the details), our implementation returns a value in $[\min_{\tau\in\mathbb{R}^2} f(\tau) - \varepsilon, \min_{\tau\in\mathbb{R}^2} f(\tau) + \varepsilon]$ (*approximate value computation*, additive version) using a straightforward adaptation of our approach.

Apart from a black-box Fréchet oracle answering decision queries $\mathrm{d_{dF}}(\pi, \sigma + \tau) \le \delta$*?*, we only use the fact that the Fréchet distance under translation is Lipschitz (Observation 2.3) and the following simple property: We obtain a 2-approximation of the Fréchet distance under translation as follows.

Observation 10.1. *Let* $\tau_{\text{start}} \coloneqq \pi_1 - \sigma_1$ *be the translation of* σ *that aligns the first points of* π *and* σ*. Then* $\mathrm{d_{dF}}(\pi, \sigma + \tau_{\text{start}}) \le 2 \cdot \mathrm{d^T_{dF}}(\pi, \sigma)$*. Analogously, for* $\tau_{\text{end}} \coloneqq \pi_n - \sigma_m$*, we have* $\mathrm{d_{dF}}(\pi, \sigma + \tau_{\text{end}}) \le 2 \cdot \mathrm{d^T_{dF}}(\pi, \sigma)$*.*

Proof. Let $\delta^* \coloneqq \mathrm{d^T_{dF}}(\pi, \sigma)$ and let τ^* be such that $\mathrm{d_{dF}}(\pi, \sigma + \tau^*) = \delta^*$, which implies in particular that $\|\pi_1 - (\sigma_1 + \tau^*)\| \le \delta^*$. Thus, $\|\tau_{\text{start}} - \tau^*\| = \|\pi_1 - (\sigma_1 + \tau^*)\| \le \delta^*$. Thus by Observation 2.3, we obtain $\mathrm{d_{dF}}(\pi, \sigma + \tau_{\text{start}}) \le \mathrm{d_{dF}}(\pi, \sigma + \tau^*) + \delta^* = 2\delta^*$. □

Note that the above observation gives a formal guarantee of a simple heuristic: translate the curves such that the start points match, and compute the corresponding fixed-translation Fréchet distance. Unfortunately, this worst-case guarantee is tight[1] – a correspondingly large discrepancy is also observed on our data sets.

[1]To see this, take any segment in the plane and let π traverse it in one direction, and σ in the other. Then the heuristic would return as estimate two times the segment length (the distance of the translated end points), while the optimal translation aligns the segments and achieves the segment length as Fréchet distance.

10.2 Our Approach: Lipschitz Meets Fréchet

To obtain a fast exact decider, we approach the problem from two different angles: First, we review previous problem-specific approaches to the discrete Fréchet distance under translation, all relying on the construction of an arrangement of circles as an essential tool from computational geometry. Second, we cast the problem into the framework of global Lipschitz optimization with its rich literature on fast, numerical solutions. In isolation, both approaches are inadequate to obtain a fast, exact decider (as the arrangement can be prohibitively large even for realistic data sets, and black-box Lipschitz optimization methods cannot return an exact optimum). We then describe how to combine both approaches to obtain a fast implementation of an exact decider for the discrete Fréchet distance under translation in the plane. We evaluate our approach, including comparisons to (typically computationally infeasible) baseline approaches, on a data set that we craft from sets of handwritten character and (synthetic) GPS trajectories used in the ACM SIGSPATIAL GIS Cup 2017 [9, 65]. We believe that our approach will inspire similar combinations of fast, inexact methods from continuous optimization with expensive, but exact approaches from computational geometry also in other contexts.

10.2.1 View I: Arrangement-Based Algorithms

Previous algorithms for the discrete Fréchet distance under translation in the plane work as follows. Given two polygonal curves π, σ and a decision distance δ, consider the set of circles

$$\mathcal{C} := \{C_\delta(\pi_i - \sigma_j) \mid \pi_i \in \pi,\ \sigma_j \in \sigma\},$$

where $C_r(p)$ denotes the circle of radius $r \in \mathbb{R}$ around $p \in \mathbb{R}^2$. Define the arrangement $\mathcal{A}_\delta$ as the partition of $\mathbb{R}^2$ induced by $\mathcal{C}$. The decision of $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ is then uniform among all $\tau \in \mathbb{R}$ in the same face of $\mathcal{A}_\delta$ (for a detailed explanation, we refer to [33, Section 3] or Chapter 5). Thus, it suffices to check, for each face f of $\mathcal{A}_\delta$, an arbitrarily chosen translation $\tau_f \in f$. Specifically, the discrete Fréchet distance under translation is bounded by δ if and only if there is some face f of $\mathcal{A}_\delta$ such that $\mathrm{d_{dF}}(\pi, \sigma + \tau_f) \leq \delta$. Since the arrangement $\mathcal{A}_\delta$ has size $\mathcal{O}(n^4)$ and can be constructed in time $\mathcal{O}(n^4)$ [120], using the standard $\mathcal{O}(n^2)$-time algorithm for the fixed-translation Fréchet distance [19, 89] to decide $\mathrm{d_{dF}}(\pi, \sigma + \tau_f) \leq \delta$ for each face f, we immediately arrive at an $\mathcal{O}(n^6)$-time algorithm.

Subsequent improvements — see [33] and Chapter 5 — speed up the decision of $\mathrm{d_{dF}}(\pi, \sigma + \tau_f) \leq \delta$ for all faces f by choosing an appropriate ordering of the translations τ_f and designing data structures that avoid recomputing some information for "similar" translations, leading to an $\mathcal{O}(n^{4.667})$-time algorithm. Still, these works rely on computing the arrangement $\mathcal{A}_\delta$ of worst-case size $\Theta(n^4)$, and a conditional lower bound indeed rules out $\mathcal{O}(n^{4-\varepsilon})$-time algorithms, see Chapter 8.

Drawback: The arrangement size bottleneck. Despite the worst-case arrangement size of $\Theta(n^4)$ and the conditional lower bound in Chapter 8, which indeed constructs such large arrangements, one might hope that realistic instances often have much smaller arrangements. If so, a combination with a practical implementation of the fixed-translation

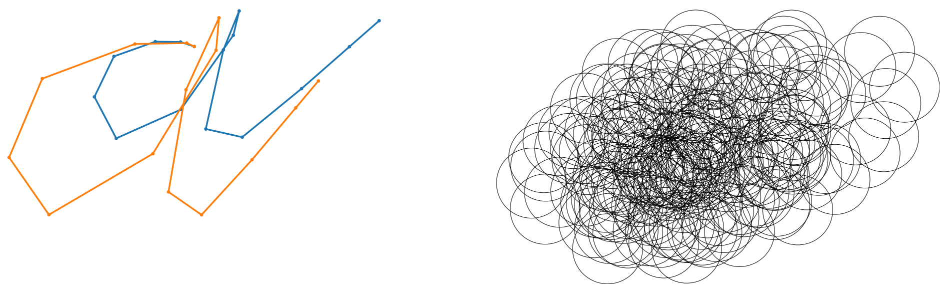


Figure 10.1: Example curves π, σ (left) together with their arrangement $\mathcal{A}_\delta$ (right), $\delta = \mathrm{d}_{\mathrm{dF}}^{\mathrm{T}}(\pi, \sigma)$.

Fréchet distance could already give an algorithm with reasonable running time. Unfortunately, this is not the case: our experiments in this chapter exhibit typical arrangement sizes between 10^6 to 10^8 for curves of length $n \approx 200$, see Figure 10.5 in Section 10.5. Also see Figure 10.1 which illustrates a large arrangement already on curves with 15 vertices, subsampled from our benchmark sets of realistic curves.

This renders a purely arrangement-based approach infeasible: As existing implementations for the Fréchet distance typically answer queries within few microseconds, we would expect an average decision time between a few seconds and several minutes already for a single decision query for the discrete Fréchet distance under translation. Thus, a reasonable approximation of the distance value via binary search would take between a minute and over an hour.

10.2.2 View II: A Global Lipschitz Optimization Problem

A second view on the discrete Fréchet distance under translation results from a simple observation: For any polygonal curves π, σ and any translation $\tau \in \mathbb{R}^2$, we have $|\,\mathrm{d}_{\mathrm{dF}}(\pi, \sigma + \tau) - \mathrm{d}_{\mathrm{dF}}(\pi, \sigma)| \le \|\tau\|_2$, see Section 10.1. As a consequence, the discrete Fréchet distance under translation is the minimum of a function $f(\tau) := \mathrm{d}_{\mathrm{dF}}(\pi, \sigma + \tau)$ that is 1-Lipschitz (i.e., $|f(x) - f(x+y)| \le \|y\|_2$ for all x, y). This suggests to study the problem also from the viewpoint of the generic algorithms developed for optimizing Lipschitz functions by the continuous optimization community.

Following the terminology of [108], in an *unconstrained bivariate global Lipschitz optimization problem*, we are given an objective function $f : \mathbb{R}^2 \to \mathbb{R}$ that is 1-Lipschitz, and the aim is to minimize $f(x)$ over $x \in B := [a_1, b_1] \times [a_2, b_2]$; we can access f only by evaluating it on (as few as possible) points $x \in B$. Note that in this abstract setting, we cannot optimize f exactly, so we are additionally given an error parameter $\varepsilon > 0$ and the precise task is to find a point $x \in B$ such that $f(x) \le \min_{z \in B} f(z) + \varepsilon$.

Global Lipschitz optimization techniques have been studied from an algorithmic perspective for at least half a century [139]. This suggest to explore the use of the fast algorithms developed in this context to obtain at least an *approximate* decider for the discrete Fréchet distance under translation. Indeed, our problem fits into the above framework, if we take the following considerations into account:

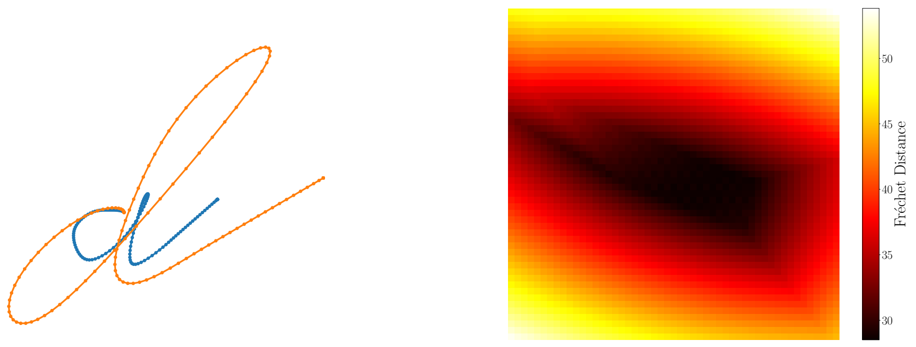


Figure 10.2: Example curves π, σ (left) together with a plot of the resulting non-convex objective function $f(\tau) = \mathrm{d_{dF}}(\pi, \sigma + \tau)$. For a closer look at the area close to the optimal translation (and highly non-convex small-scale artefacts), we refer to Figure 10.3.

(1) **Finite Box Domain:** While we seek to minimize $f(\tau) = \mathrm{d_{dF}}(\pi, \sigma+\tau)$ over $\tau \in \mathbb{R}^2$, the above formulation assumes a finite box domain B. To reconcile this difference, observe that any translation τ achieving a discrete Fréchet distance of at most δ must translate the first (last) point of σ such that the first (last) point of π is within distance at most δ. Thus, any feasible translation τ must be contained in the intersection of the two corresponding disks, and we can use any bounding box of this intersection as our box domain B.

(2) **(Approximate) Decision Problem:** While we seek to decide "$\min_\tau f(\tau) \leq \delta$", the above formulation solves the corresponding minimization problem. Note that approximate minimization can be used to *approximately* solve the decision problem, but *exactly* solving the decision problem is impossible in the above framework.

(3) **Oracle Access to $f(\tau)$:** Evaluation of $f(\tau)$ corresponds to computing the discrete Fréchet distance of π and $\sigma+\tau$, for which we can use previous fast implementations, see [30, 54, 87] or Chapter 9. (Actually, these algorithms were designed to answer decision queries of the form "$f(\tau) \leq \delta$?"; we discuss this aspect at the end of this section.)

In Figure 10.2, we illustrate our view of the discrete Fréchet distance under translation as Lipschitz optimization problem. As the figure suggests, on many realistic instances, the problem appears well-behaved (almost convex) at a global scale; using the Lipschitz property, one should be able to quickly narrow down the search space to small regions of the search space[2]. Particularly for this task, it is very natural to consider branch-and-bound approaches, as pioneered by Galperin [98, 99, 100, 101] and formalized by Horst and Tuy [110, 111, 112], since these have been applied very successfully for low-dimensional Global Lipschitz optimization (and non-convex optimization in general).

On a high level, in this approach we maintain a global upper bound $\tilde{\delta}$ and a list of search boxes $B_1, \ldots, B_b$ with lower bounds $\ell_1, \ldots, \ell_b$ (i.e., $\min_{\tau \in B_i} f(\tau) \geq \ell_i$) obtained via

[2]For an illustration that highly non-convex behavior may still occur at a local level, we refer to Figure 10.3.

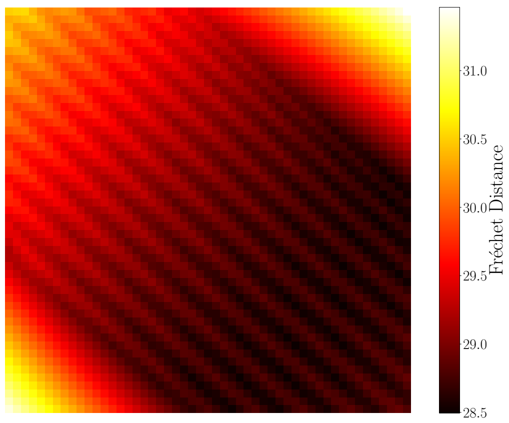


Figure 10.3: Highly non-convex artefacts of the objective function at a local scale, resulting particularly from the notion of traversals in the *discrete* Fréchet distance.

the Lipschitz condition. We iteratively pick some search box B_i and first try to improve the global upper bound $\tilde{\delta}$ or the local lower bound ℓ_i using a small number of queries $f(\tau)$ with $\tau \in B_i$ (and exploiting the Lipschitz property). If the local lower bound exceeds the global upper bound, i.e., $\ell_i > \tilde{\delta}$, we drop the search box B_i, otherwise, we split B_i into smaller search boxes. The procedure stops as soon as $\tilde{\delta} \leq (1+\varepsilon)\min_i \ell_i$, which proves that $\tilde{\delta}$ gives a $(1+\varepsilon)$-approximation to the global minimum.

Specifically, we arrive at the following branch-and-bound strategy proposed by Gourdin, Hansen and Jaumard [104]. We specify it by giving the rules with which it (i) attempts to update the global upper bound, (ii) selects the next search box from the set of current search boxes, (iii) splits a search box if it remains active after bounding, and (iv) determines the local lower bounds.[3]

(i) **Upper Bounding Rule:** We evaluate f at the center τ_i of the current search box B_i.

(ii) **Selection Rule:** We pick the search box with the smallest lower bound (ties are broken arbitrarily).

(iii) **Branching Rule:** We split the current search box along its longest edge into 2 equal-sized subproblems.

(iv) **Lower Bounding Rule:** We obtain the local lower bound ℓ_i as $f(\tau_i) - d$ where d is the half-diameter of the current box. (Since f is 1-Lipschitz, we indeed have $\min_{\tau \in B_i} f(\tau) \geq \ell_i$.)

One may observe that the chosen selection rule (also known as Best-Node First) is a no-regret strategy in the sense that no other selection rule, *even with prior knowledge of the global optimum*, considers fewer search boxes (see, e.g., [162, Section 7.4]).

[3]See [108] for a precise formalization of the generic branch-and-bound algorithm that leaves open the instantiation of these rules. In any case, we give a self-contained description of our algorithms in Section 10.3 and 10.4.

Drawback: Inexactness. Unfortunately, the above branch-and-bound approach for Lipschitz optimization fundamentally cannot return an exact global optimum, and thus yields only an approximate decider.

In a somewhat similar vein, in the above framework we assume that we can evaluate $f(\tau)$ quickly. Previous implementations for the fixed-translation Fréchet distance focus on the decision problem "$f(\tau) \leq \delta$?", not on determining the value $f(\tau)$. Both precise computations (via parametric search) or approximate computations (using a binary search up to a desired precision) are significantly more costly, raising the question how to make optimal use of the cheaper decision queries.

10.3 Contribution I: An Exact Decider by Combining Both Views

Our first main contribution is engineering an exact decider for the discrete Fréchet distance under translation by combining the two approaches. On a high level, we *globally* perform the branch-and-bound strategy described in the Lipschitz optimization view in Section 10.2.2, but use as a base case a *local* version of the arrangement-based algorithms of Section 10.2.1 once the arrangement size in a search box is sufficiently small. As each search box is thus resolved exactly, this yields an exact decider. More precisely, our final algorithm is a result of the following steps and adaptations:

(1) **Fréchet Decision Oracle.** We adapt the currently fastest implementation of a decider for the continuous fixed-translation Fréchet distance, see Chapter 9, to the discrete fixed-translation Fréchet distance. Furthermore, to handle many queries for the same curve pair *under different translations* quickly, we incorporate an implicit translation so that curves do not need to be explicitly translated for each query translation τ.

(2) **Objective Function Evaluation.** For our exact decider, the branch-and-bound strategy in Section 10.2.2 simplifies significantly: We do not maintain a global upper bound and local lower bounds ℓ_i, but for each box only test whether $f(\tau_i) \leq \delta$ (if so, we return YES) or whether $f(\tau_i) > \delta + d$ (this corresponds to updating the local lower bound beyond δ, i.e., we may drop the box completely). Therefore, we may use an arbitrary selection rule. Note that we only require decision queries to the fixed-translation Fréchet algorithm.

(3) **Base Case.** We implement a *local* arrangement-based algorithm: For a given search box B_i, we (essentially) construct the arrangement $\mathcal{A} \cap B_i$ using CGAL [156], and test, for each face f of $\mathcal{A} \cap B_i$, some translation $\tau' \in f$ for $f(\tau') \leq \delta$. This yields the algorithm that we may use as a base case.

(4) **Base Case Criterion.** For each search box, we compute an estimate of its arrangement complexity. If this estimate is smaller than a (tunable) parameter γ_{size}, or the depth of the branch-and-bound recursion for the current search box exceeds a parameter γ_{depth}, then we use the localized arrangement-based algorithm.

```
procedure Decider($\pi, \sigma, \delta$)
    decide trivial NO instances with empty initial search box quickly
    $Q \leftarrow$ Fifo(initial search box)
    while Q $\neq \emptyset$ do
        $B \leftarrow$ extract front of search box queue $Q$
        if FréchetDistance($\pi, \sigma + \tau_B$) $> \delta + d_B/2$ then        ▷ Lower Bounding
            skip $B$
        if FréchetDistance($\pi, \sigma + \tau_B$) $\leq \delta$ then        ▷ Upper Bounding
            return YES

        $u \leftarrow$ upper bound on arrangement size inside $B$
        if $u = 0$ then        ▷ Arrangement-based Base Case
            skip $B$
        else if $u \leq \gamma_{\text{size}}$ or layer of $B$ is $\gamma_{\text{depth}}$ then
            if local arrangement-based algorithm on $\pi, \sigma, \delta, B$ returns YES then
                return YES
            else
                skip $B$

        halve $B$ along longest edge and push resulting child boxes to $Q$ ▷ Branching
    return NO
```

Algorithm 20: Algorithm for deciding the discrete Fréchet distance under translation. We use τ_B to denote the center of the box B and d_B to denote the length of the diagonal of B.

(5) **Benchmark and Choice of Parameters.** We choose the size and depth parameters $\gamma_{\text{size}}, \gamma_{\text{depth}}$ guided by a benchmark set that we create from a set of handwritten characters and synthetic GPS trajectories.

The pseudocode of the resulting algorithm is shown in Algorithm 20. In the remainder of this section, we describe the details of our Fréchet-under-translation decider. We first describe the details of the *local* arrangement-based algorithm which serves as the base case for our decider.

10.3.1 Local Arrangement-Based Algorithm

Recall that given two polygonal curves π, σ and a decision distance δ, the set of circles of the arrangement $\mathcal{A}_\delta$ is

$$\mathcal{C} := \{C_\delta(\pi_i - \sigma_j) \mid \pi_i \in \pi,\ \sigma_j \in \sigma\},$$

where $C_r(p)$ denotes the circle of radius $r \in \mathbb{R}$ around $p \in \mathbb{R}^2$. The arrangement is then defined as the partition of $\mathbb{R}^2$ induced by $\mathcal{C}$. In particular, the decision of $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ is uniform for each $\tau \in \mathbb{R}$ in the same face of $\mathcal{A}_\delta$ (for a detailed explanation, we refer to [33, Section 3] or Chapter 5). Thus, as already described in Section 10.2.1, it suffices to evaluate a representative translation from each face of the arrangement by running a fixed-translation Fréchet decider query on it to reach a discrete Fréchet under translation query decision.

For integration into our branch-and-bound approach where each node in the branch-and-bound tree corresponds to a search box B, the base case task is to decide whether there is some $\tau \in B$ with $\mathrm{d}_{\mathrm{dF}}(\pi, \sigma + \tau) \leq \delta$. For this task, we consider *local* arrangements, i.e., arrangements restricted to B. A circle $C \in \mathcal{C}$ contributes to the local arrangement of B if the boundary of C intersects the box. In other words, C is relevant for the arrangement of B if C either is completely contained in B or C intersects the boundary of B. In particular, C does not contribute to the local arrangement if it contains B completely.

Estimation of local arrangement sizes. Given a search box B, a simple way to estimate the size of the local arrangement for B, i.e., the arrangement restricted to B, is to consider the number of circles in $\mathcal{C}$ that contribute to it. We can obtain this number naively, by iterating over all $|\mathcal{C}| \leq nm$ circles of the global arrangement and check if they contribute to the local arrangement (by checking for intersection and containment as described above). Let this number be denoted by c. The maximal number of nodes in the arrangement is then bounded by $u := 2(c + c^2)$, as this is the maximal number of intersections between two circles and a circle and the box. In particular, if $u = 0$, then the arrangement in the box belongs to a single face and all translations in B are equivalent for our decision question.

As a simple optimization, we may stop counting contributing circles once our estimate exceeds the threshold γ_{size}. A more sophisticated optimization builds a geometric data structure (specifically a kd-tree) to quickly retrieve all contributing circles without checking all circles in $\mathcal{C}$ naively. We discuss this approach in Section 10.4, as the expensive preprocessing for constructing this data structure only amortizes in the value computation setting.

Construction of local arrangement. For a search box B with an estimate smaller than γ_{size}, we construct an arrangement $\mathcal{A}_B$. To this end, we adapt our arrangement size estimation to also return the set $\mathcal{C}_B$ of circles intersecting B or being contained in B. Note that computing topologically correct geometric arrangements on such a circle set is a challenging task, as it requires the usage of arbitrary precision numbers to reliably test for intersections and orderings of those intersections. Thus, we use the state-of-the-art computational geometry library CGAL [156] to build our circle arrangements.[4] Unfortunately, CGAL only provides methods for building a global arrangement and not an arrangement restricted to a bounding box, thus we always build the whole arrangement of the circles in $\mathcal{C}_B$ instead of just the arrangement restricted to the box B. Alternatively, we could indeed compute circular arcs restricted to the bounding box and then build the arrangement of those arcs. However, due to the rather expensive construction of these arcs, this seems wasteful compared to a direct computation. Thus, a practical performance improvement of our approach could be achieved by directly computing an arrangement

[4]Specifically, we use the exact predicates and exact computation kernels as this is necessary for CGAL arrangements. The significantly faster kernel for inexact computation is not suitable for the CGAL arrangement package (although, surprisingly, for most instances it actually worked). Being able to use a faster kernel for arrangements should significantly improve our implementation's performance.

with a box restriction. Furthermore, we use the standard bulk-insertion interface for building the arrangement.

Resulting local arrangement-based algorithm. Finally, given the arrangement $\mathcal{A}_B$ of the circles $\mathcal{C}_B$, we may simply test a translation τ for each face f of $\mathcal{A}_B$ that intersects B. In fact, for efficiency, we do this by testing each vertex τ of $\mathcal{A}_B$ (even for vertices outside of B, as due to the expensive construction of $\mathcal{A}_B$, it pays off to make the rather cheap tests for positive witnesses also outside of B); observe that this ensures that each face f is indeed tested. We return YES if and only if some vertex τ of $\mathcal{A}_B$ achieves $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$.

10.3.2 Decision Algorithm

Now, we describe our decider (whose pseudocode is given in Algorithm 20) in more detail. Recall that an exact decider, given curves $\pi = (\pi_1, \ldots, \pi_n), \sigma = (\sigma_1, \ldots, \sigma_m)$ and a distance δ, decides whether the discrete Fréchet distance under translation of π and σ is at most δ, i.e., whether $\mathrm{d}^{\mathrm{T}}_{\mathrm{dF}}(\pi, \sigma) \leq \delta$.

Preprocessing. As a first step, we aim to determine an initial search box. Since any $\tau \in \mathbb{R}^2$ with $\mathrm{d_{dF}}(\pi, \sigma + \tau) \leq \delta$ implies that $\|\pi_1 - (\sigma_1 + \tau)\|, \|\pi_n - (\sigma_m + \tau)\| \leq \delta$, we must have that τ is in the intersection $I \coloneqq D_\delta(\pi_1 - \sigma_1) \cap D_\delta(\pi_n - \sigma_m)$, where $D_r(p)$ denotes the disk of radius r around p. If this intersection is empty, i.e., $\pi_1 - \sigma_1$ and $\pi_n - \sigma_m$ have a distance more than 2δ, we return NO immediately. Otherwise, we take a bounding box of the intersection.[5]

Branch-and-bound. We implement the recursive branch-and-bound strategy using a FIFO queue Q of search boxes (corresponding to a breadth-first search) that is initialized with the initial search box. As long as there are undecided boxes in the queue, we take the first such box B and try to resolve it using the upper bounding rule (point (i) in View II) and the lower bounding rule (point (iv) in View II), which are both derived by queries to the fixed-translation Fréchet distance decider using the center point τ_B of the box as translation. Specifically, if $\mathrm{d_{dF}}(\pi, \sigma + \tau_B) \leq \delta$ (line 6 in Algorithm 20), we have found a witness translation and can return YES. The lower bounding rule (line 8) tests if $\mathrm{d_{dF}}(\pi, \sigma + \tau_B) > \delta + d_B/2$, i.e., if the distance at the center point is larger than the test distance δ plus the maximal distance of any point in the box to the center τ_B, i.e., the half-diagonal length $d_B/2$. If so, by the Lipschitz property, we know that the any translation in B yields a discrete Fréchet distance larger than δ and thus we can drop B.

If neither rule applies, we check our termination criterion of the branch-and-bound strategy. To this end, in line 11, we calculate a good upper bound u on the size of the local arrangement for B as described in Section 10.3.1. If $u = 0$, the arrangement for B consists of a single face, i.e., each translation $\tau \in B$ is equivalent for our decision problem, and we can skip the box since we have already tested the translation $\tau_B \in B$. Otherwise, in line 14, if $u \neq 0$, we check if the number is bounded by a size parameter γ_{size} or the

[5]In fact, we use a slightly more refined search box by incorporating additionally the extreme points of both curves.

depth of the current search box (in the implicit recursion tree) is bounded by a depth parameter γ_{depth}. If so, we run the local arrangement-based algorithm to decide B.

If none of the above rules decide the search box B, we split it along its longer side into two equal-sized child boxes and push them to the queue. If all boxes have been dropped without finding a witness translation, we have verified that any translation $\tau \in B$ yields $\mathrm{d_{dF}}(\pi, \sigma + \tau) > \delta$ and may safely return NO.

Low-level optimizations. For further practical speed-ups, we employ several low-level optimizations, which we briefly mention here (for further details, we refer to the source code of our implementation).

For each box in the branch-and-bound tree we need a differently translated curve. However, often we barely access the nodes of the translated curve. For example, if already the start nodes of the curves are too far, we do not need to consider the remainder. Thus, it seems wasteful to translate each point of the curves before calling the fixed-translation Fréchet decider. To avoid this overhead, we lazily translate the necessary parts of a curve on access. In fact, while the currently fastest implementation of the fixed-translation Fréchet distance decider, see Chapter 9, uses a preprocessing of the curves that computes all prefix lengths and extrema of the curves, we only need to perform this preprocessing once, as all computed information is either invariant under translations (for the prefix lengths) or can just be shifted by the translation (for the extrema).

Furthermore, while the initial bounding box is derived from the discs around the translation between the start nodes and the translation between the end nodes, later child boxes in the branch-and-bound tree might violate this condition. We therefore re-check this condition on creating child boxes. Additionally, in line 14 of Algorithm 20 we check if the depth parameter γ_{depth} is reached. This can actually already be done before line 8, which we also do in the implementation, but for the sake of brevity, we present it differently in the pseudocode.

10.4 Contribution II: Approximate Computation of the Distance Value

In this section we present our second main contribution: an algorithm for computing the value of the discrete Fréchet distance under translation. Thus, we now focus on the functional task of computing the value $\mathrm{d^T_{dF}}(\pi, \sigma) = \min_{\tau \in \mathbb{R}^2} \mathrm{d_{dF}}(\pi, \sigma + \tau)$, in contrast to the previously discussed decision problem "$\mathrm{d^T_{dF}}(\pi, \sigma) \leq \delta$?". In theory, one could use the paradigm of parametric search [133], see [33] and Chapter 5 for details for the discrete case. However, it is rarely used in practice as it is non-trivial to code, and computationally costly. Instead, as in most conceivable settings an estimate with small multiplicative error $(1 \pm \varepsilon)$ with, e.g., $\varepsilon = 10^{-7}$, suffices, we consider the problem of computing an estimate in $(1 \pm \varepsilon)\, \mathrm{d^T_{dF}}(\pi, \sigma)$.

There are several possible approaches to obtain an approximation with multiplicative error $(1 \pm \varepsilon)$ for arbitrarily small $\varepsilon > 0$:

(1) **ε-approximate Set:** A natural approach underlying previous approximation algorithms [22] is to generate a set of $f(1/\varepsilon)$ candidate translations T such that the best translation τ among this set gives a $(1 + \varepsilon)$-approximation for the discrete

```
1: procedure LMF($\pi, \sigma$)
2:     Preprocessing: build data structures for fast arrangement estimation and construction
3:     compute initial distance interval $[\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$ containing $\mathrm{d}_{\mathrm{dF}}^{\mathrm{T}}(\pi, \sigma)$
4:     initialize global upper bound $\tilde{\delta} \leftarrow \delta_{\mathrm{UB}}$
5:     $Q \leftarrow$ PriorityQueue(initial search box $B_1$ with local lower bound $\ell_{B_1} \leftarrow \delta_{\mathrm{LB}}$)

6:     while $Q \neq \emptyset$ do
7:         $B \leftarrow$ box with smallest local lower bound $\ell_B$ in $Q$
8:         if $\tilde{\delta} \leq \ell_B(1+\varepsilon)$ then
9:             skip $B$
10:        if FréchetDistance($\pi, \sigma + \tau_B$) $\leq \tilde{\delta}$ then        ▷ Upper/Lower Bounding
11:            compute value $\mathrm{d}_{\mathrm{dF}}(\pi, \sigma + \tau_B)$ with high precision and update $\tilde{\delta}$ and $\ell_B$
12:        else
13:            if FréchetDistance($\pi, \sigma + \tau_B$) $> \tilde{\delta} + d_B/2$ then
14:                skip $B$
15:            compute value $\mathrm{d}_{\mathrm{dF}}(\pi, \sigma + \tau_B)$ with coarse precision and update $\ell_B$
16:        if $\tilde{\delta} \leq \ell_B(1+\varepsilon)$ then
17:            skip $B$
18:        $u \leftarrow$ upper bound on arrangement size inside $B$ for $\delta \in [\ell_B, \tilde{\delta}]$
19:        if $u = 0$ then                                   ▷ Arrangement-based Base Case
20:            skip $B$
21:        else if $u \leq \gamma_{\mathrm{size}}$ or layer of $B$ is $\gamma_{\mathrm{depth}}$ then
22:            update $\tilde{\delta}$ via binary search over arrangement algorithm on $B$ and $\delta \in [\ell_B, \tilde{\delta}]$

23:            skip $B$
24:
25:        push child boxes of $B$ to $Q$ with local lower bounds set to $\ell_B$      ▷ Branching
26:    return $\tilde{\delta}$
```

Algorithm 21: Algorithm of our Lipschitz-Meets-Fréchet (LMF) algorithm for approximate value computation. We use τ_B to denote the center of the box and d_B to denote the length of the diagonal.

Fréchet distance under translation. Specifically, it is simple to obtain a bounding box B of side length $\mathcal{O}(\delta)$ for the optimal translation τ^* (see the 2-approximation in Section 10.1 together with the preprocessing described in Section 10.3). We impose a grid of side length at most $(\varepsilon/\sqrt{2})\delta$ so that each each point in B is within distance $\varepsilon\delta$ of some grid point. Since the discrete Fréchet distance is Lipschitz, this yields a $(1+\varepsilon)$-approximate set. Unfortunately, this set is of size $\Theta(1/\varepsilon^2)$ which is prohibitively large for approximation guarantees such as $\varepsilon = 10^{-7}$.
Remark: In the context of global Lipschitz optimization, this approach is known as the *passive algorithm* whose performance generally is dominated by (the adaptive) branch-and-bound methods.

(2) **Binary Search via Decision Problem:** A further canonical approach is to reduce the $(1+\varepsilon)$-approximate computation task to the decision problem using a binary search. Formally, let δ^* denote the discrete Fréchet distance under translation. Starting from a simple 2-approximation δ_{UB} (see Section 10.1, or, more precisely, the initial estimates discussed later in this section), we use a binary search in the interval $[0.5 \cdot \delta_{\mathrm{UB}}, \delta_{\mathrm{UB}}]$, terminating as soon as we arrive at an interval of length $[a,b]$ with $b \leq (1+\varepsilon)a$. As this takes only $\mathcal{O}(\log(1/\varepsilon))$ iterations to obtain an $(1+\varepsilon)$-approximation, this approach is much more suitable to obtain a desired guarantee of $\varepsilon = 10^{-7}$.

(3) **Lipschitz-only Optimization:** The main drawback of the generic Lipschitz optimization algorithms discussed in Section 10.2.2 was that they cannot be used to derive an exact answer. This drawback no longer applies for approximate value computation. We can thus use a pure branch-and-bound algorithm for global Lipschitz optimization. In particular, we will use the same strategy as our fastest solution, however, we never use the arrangement-based algorithm, but only terminate at a search box once the local lower bound and global upper bound provide a $(1+\varepsilon)$-approximation.

(4) **Our solution, Lipschitz-meets-Fréchet:** We follow our approach of combining Lipschitz optimization with arrangement-based algorithms (described in Section 10.2) to compute a $(1+\varepsilon)$-approximation of the distance value. As opposed to the decision algorithm, we indeed maintain a global upper bound $\tilde{\delta}$ and local lower bounds ℓ_i for each search box B_i. To update these bounds, we approximately evaluate the objective function $f(\tau)$ using a tuned binary search[6] over the fixed-translation Fréchet decider algorithm. We stop branching in a search box B_i if either the global upper bound $\tilde{\delta}$ is at most $\ell_i(1+\varepsilon)$, or a base case criterion similar to the decision setting applies. As selection strategy, we employ the no-regret strategy of choosing the box with the smallest lower bound first. The base case performs a binary search using the local arrangement-based *decision* algorithm; thus, our upper bound on the arrangement size must hold for *all* δ in the search interval. The pseudocode of our solution is shown in Algorithm 21.

[6]We tune the binary search by distinguishing the precision with which we want to evaluate $f(\tau)$; intuitively, it pays off to evaluate $f(\tau)$ with high precision if this evaluation yields a better global upper bound, while for improvements of a local lower bound, a cheaper evaluation with coarser precision suffices.

We present the details of our approach in the remainder of this section. As our experiments reveal, our solution generally outperforms the above described alternatives, see Section 10.5.

Remark: To enable a fair comparison of the Lipschitz-meets-Fréchet (LMF) approach to the alternative approaches of Binary Search and Lipschitz-only optimization, we take care that the low-level optimizations for LMF described in the reminder of this section are also applied to these approaches, as far as applicable. In particular, we use the same method to obtain initial estimates for the desired value for LMF, Binary Search and Lipschitz-only optimization, and adapt the kd-tree-based data structure used to speed-up estimation and construction of arrangements for LMF also for Binary Search (note that these tasks do not apply to Lipschitz-only optimization).

We now present details of our solution for the (approximate) value computation setting, the LMF algorithm. We first consider the base case (which differs from the base case of the decider, given in Section 10.3.1), before we discuss further details.

10.4.1 Local Arrangement-Based Algorithm for Value Computation

Our base case problem is the following: Given curves π, σ, a *test distance interval* $I = [\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$ and a search box B, we let $\delta^* := \min_{\tau \in B} \mathrm{d_{dF}}(\pi, \sigma + \tau)$ and ask to determine whether $\delta^* \in I$, and if so, an estimate δ' with $|\delta' - \delta^*| \leq \varepsilon$.

The central idea is to solve this task via a binary search for $\delta^* \in I$ using our local arrangement-based algorithm of Section 10.3.1 to decide queries of the form "$\delta^* \leq \delta$?" for any given δ. For this algorithm to run quickly, we need that for *any* queried distance δ, the corresponding local arrangement for the test distance δ is small. To this end, we seek to obtain a strong upper bound for the local arrangement size over *worst-case* $\delta \in I$.

Estimation of local arrangement sizes. Given an interval $I = [\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$ of test distances, instead of the *circles* defined in Section 10.3.1, we consider the set of *annuli*

$$\mathcal{D} := \{D_{\delta_{\mathrm{UB}}}(\pi_i - \sigma_j) \setminus D_{\delta_{\mathrm{LB}}}(\pi_i - \sigma_j) \mid \pi_i \in \pi,\ \sigma_j \in \sigma\},$$

where $D_r(p)$ denotes the disk of radius $r \in \mathbb{R}$ around $p \in \mathbb{R}^2$. Clearly, if a circle $C_\delta(\pi_i - \sigma_j)$ contributes to the local arrangement of B for *some* test distance $\delta \in [\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$, then the corresponding annulus $D_{\delta_{\mathrm{UB}}}(\pi_i - \sigma_j) \setminus D_{\delta_{\mathrm{LB}}}(\pi - \sigma_j)$ intersects B or is contained in B. Thus by determining the number d of annuli $a \in \mathcal{D}$ that intersect B or are contained in B, we may bound the local arrangement size for B for any $\delta \in [\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$ by $u := 2(d + d^2)$ (analogously to Section 10.3.1).

To obtain the above upper bound efficiently, we implement a geometric search data structure based on the *kd-tree*. Specifically, we build a kd-tree on the set of center points of all annuli in $\mathcal{D}$. Given a search box B, we seek to determine all centers of annuli $a \in \mathcal{D}$ that intersect B or contain B. While this condition can be described using a constant (but large) set of simple primitives, evaluating this test frequently for many kd-tree nodes is costly. Thus, to determine whether a node in the kd-tree needs to be explored, we use a more permissive, but cheaper test which essentially approximates the search box B by its center point: we search for all candidate points that are contained in an annulus of width roughly $|I|$ plus half the diameter of B, centered at the center of B, and test for each such point whether the corresponding annulus in $\mathcal{D}$ indeed intersects B.

Again, we implement this search for contributing annuli such that we return the centers of all found annuli. This can subsequently be used by the local arrangement-based algorithm to quickly construct the arrangement for each query. Furthermore, we again stop the search as soon as the numbers of such annuli exceeds γ_{size}.

Binary search via local arrangement-based algorithm. To obtain the desired estimate for δ^* in the case that our size estimate is bounded by γ_{size}, we use a binary search via our local arrangement-based algorithm. As a low-level optimization to speed-up the construction of the local arrangement for a query distance δ, we pass the centers of contributing annuli to the local arrangement-based algorithm. Furthermore, as described in Section 10.3.1, we let the arrangement-based decision algorithm test *all* vertices in the arrangement of all circles $\mathcal{C}_B$ contributing to the search box B, not only vertices in B. As this can only decrease the returned estimate (by finding a corresponding witness), this does not affect correctness of the algorithm.

10.4.2 Overview and Details for LMF

The pseudocode of the LMF algorithm is shown in Algorithm 21. When referring to lines in the remainder of this section, we refer to lines in this algorithm. Before we address some aspects and optimizations in detail, we give a short overview of the algorithm. First, note that as our selection strategy is different from the decider setting, we now use a priority queue for the boxes, see line 5. In lines 8 to 17 the bounding happens and in lines 18 to 23 we check if the base case criterion applies, and if it does, determine the value for this box using the arrangement-based approach. Finally, in line 25 we branch if we did not already skip the box.

Initial estimates. In line 3 we calculate initial estimates for the upper and lower bound. To this end, we consider the translation τ_{start} (resp. τ_{end}) that aligns the first (resp. last) points of π, σ as it yields a 2-approximation $\delta_{\text{start}} := \mathrm{d_{dF}}(\pi, \sigma + \tau_{\text{start}})$ (resp. $\delta_{\text{end}} := \mathrm{d_{dF}}(\pi, \sigma + \tau_{\text{end}})$). Using the best of both approximations, our initial estimation interval for $\mathrm{d}^{\mathrm{T}}_{\mathrm{dF}}(\pi, \sigma)$ is $[\delta_{\text{LB}}, \delta_{\text{UB}}] := [\max\{\delta_{\text{start}}, \delta_{\text{end}}\}/2, \min\{\delta_{\text{start}}, \delta_{\text{end}}\}]$, see Section 10.1.

Priority queue. To implement our smallest-lower-bound-first selection rule, we use a priority queue to organize the search boxes, using the local lower bounds as keys. Recall that this yields a no-regret selection strategy for our branch-and-bound framework.

Objective function evaluation: Computing Fréchet distance via Fréchet decider. To update our global upper bound and local lower bounds, we need to determine discrete Fréchet distance *values* rather than decisions (which were sufficient for our decider), see lines 11 and 15. However, we do not always need a very precise calculation. While the upper bound is global and thus an improvement might lead to significant progress by dropping a number of search boxes, the lower bound only has an effect on the box itself and on its children. Thus, we use a coarse distance computation (i.e., an approximation up to a larger additive constant) for the lower bound in line 15, but a more precise calculation for the upper bound in line 11.

In two cases (lines 10 and 22) we are only interested in the exact discrete Fréchet distance value if it is smaller than the current global upper bound. Thus, as is hidden in the pseudocode, we first check if there is an improvement at all, and only if this is the case, we compute the actual value using a binary search.

Additive vs. multiplicative approximation. Due to rounding issues that occur at decisions depending on extremely small value differences when using fixed precision arithmetic, we use an additive approximation of $\varepsilon = 10^{-7}$ instead of a multiplicative approximation to ensure that these issues do not arise on usage of our implementation with arbitrary data sets. Note that all computed distances in our benchmarks have a value larger than 1, and thus also in terms of multiplicative approximation $(1 + \varepsilon')$, we have $\varepsilon' \leq 10^{-7}$.

10.5 Experiments

To engineer and evaluate our approach, we provide a benchmark on the basis of the curve datasets that were used to evaluate the currently fastest fixed-translation Fréchet decider implementation presented in Chapter 9. In particular, this curve set involves a set of handwritten characters (Characters, [65]) and the data set of the GIS Cup 2017 (Sigspatial, [9]). Table 10.1 gives statistics of these datasets.

Data set	Type	#Curves	Mean #vertices
Sigspatial [9]	synthetic GPS-like	20199	247.8
Characters [65]	20 handwritten characters	2858	120.9

Table 10.1: Information about the data sets used in the benchmarks.

The aim of our evaluations is to investigate the following main questions:

(1) Is our solution able to decide queries on realistic curve sets in an amount of time that is practically feasible, even when the size of the arrangement suggests infeasibility?

(2) Is our combination of Lipschitz optimization and arrangement-based algorithms for value computation superior to the alternative approaches described in Section 10.4?

Furthermore, we aim to provide an understanding of the performance of our novel algorithms.

Decider experiments. For decision queries of the form "$\mathrm{d}_{\mathrm{dF}}^{\mathrm{T}}(\pi, \sigma) \leq \delta$?", we generate a benchmark query set that distinguishes between how close the test distance is to the actual distance of the curves: Given a set of curves C, we sample 1000 curve pairs $\pi, \sigma \in C$ uniformly at random. Using our implementation, we determine an interval $[\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$ such that $\delta_{\mathrm{UB}} - \delta_{\mathrm{LB}} \leq 2 \cdot 10^{-7}$ and $\mathrm{d}_{\mathrm{dF}}^{\mathrm{T}}(\pi, \sigma) \in [\delta_{\mathrm{LB}}, \delta_{\mathrm{UB}}]$. For each $\ell \in \{-10, \ldots, -1\}$, we add "$\mathrm{d}_{\mathrm{dF}}^{\mathrm{T}}(\pi, \sigma) \leq (1 - 4^{\ell})\delta_{\mathrm{LB}}$?" to the query set C_{ℓ}^{NO}, which contains only NO instances.

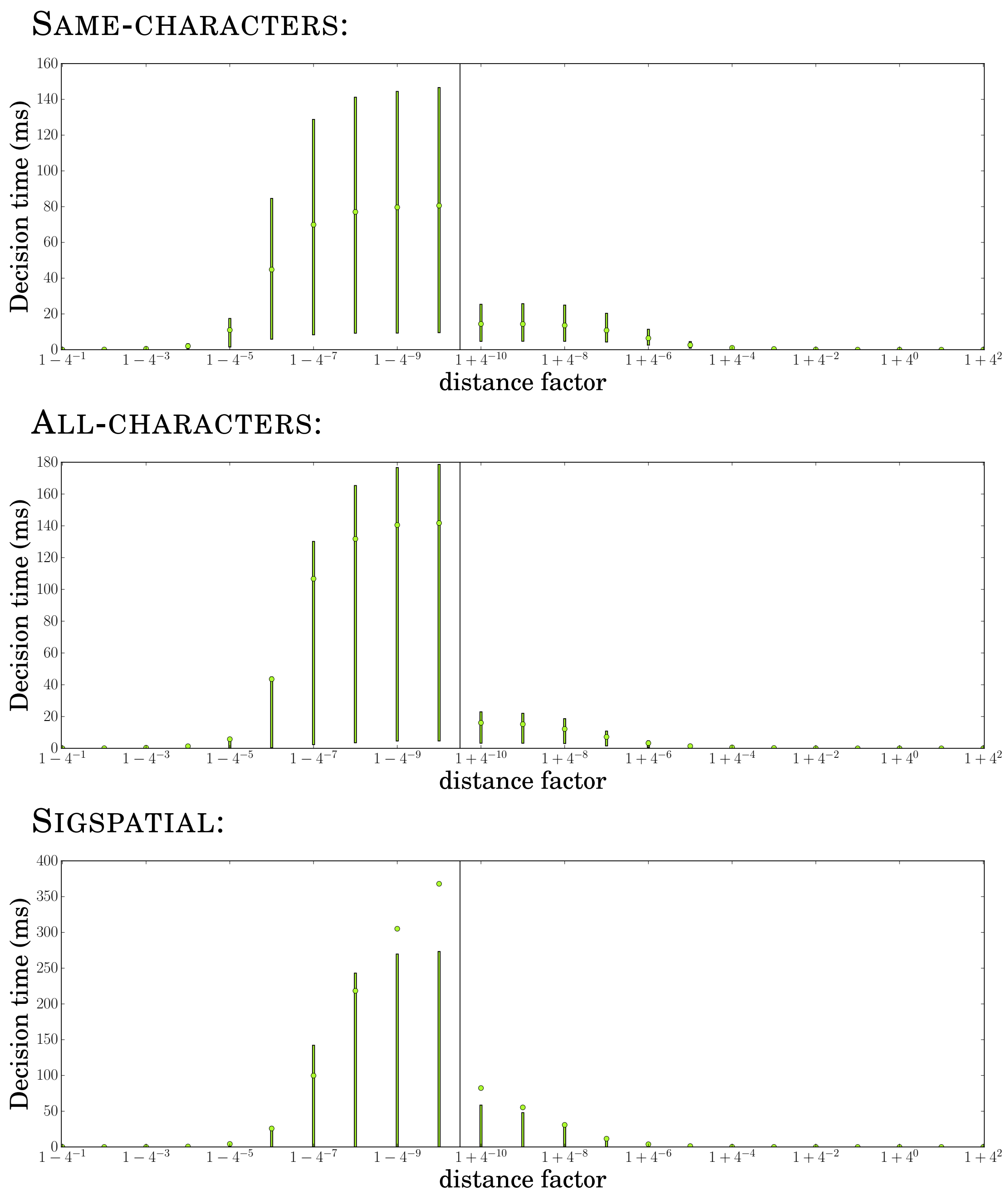


Figure 10.4: Running time for our decider. We plot the mean running times over 1000 NO (or YES) queries with a test distance of approximately $(1 - 4^{-\ell})$ (or $(1 + 4^{-\ell})$) times the true discrete Fréchet distance under translation, as well as the interval between the lower and upper quartile over the queries.

Similarly, for each $\ell \in \{-10, \ldots, 2\}$ we add "$d_{dF}^{T}(\pi, \sigma) \le (1 + 4^{\ell})\delta_{UB}$?" to the query set C_{ℓ}^{YES}, which contains only YES instances. We evaluate our decider on this benchmark created for the Characters and Sigspatial data sets. Furthermore, we give results for a further benchmark set generated from the Characters curve set by sampling, for each of the 20 characters c included in Characters, 50 curve pairs π, σ representing the *same character* c. This yields a benchmark that has the same size of 1000 query curve pairs, but compares only same-character curves. We show the mean running times on these three benchmark sets in Figure 10.4. As before, we also depict the number of black-box calls of our decider and, as a baseline, an estimate of the arrangement size (and thus the number of black-box calls of a naive approach) in Figure 10.5. Note that for small ranges of the test distance δ, it may happen that we decide a NO instance without a single black-box call by determining that the distance between $\pi_1 - \sigma_1$ and $\pi_n - \sigma_m$ is larger than 2δ; corresponding values below 1 call are not depicted in Figure 10.5.

To give an insight for the speed-up achieved over the baseline arrangement-based algorithm that makes a black-box call to the fixed-translation Fréchet decider for each face of the arrangement $\mathcal{A}_{\delta}$, in Figure 10.5 we depict both the number of black-box calls to the fixed-translation Fréchet decider made by our implementation, as well as an estimate[7] for the arrangement size, and thus the number of black-box calls of the baseline approach.

We observe that on the above sets, the average decision time ranges from below 1 ms to 142 ms, deciding our Characters benchmark (involving 23, 000 queries) in 628 seconds. Our estimation suggests that a naive implementation of the baseline arrangement-based algorithm would have been worse by more than *three orders of magnitude*, as for each set, the average number of black-box calls to the fixed-translation Fréchet decider is smaller by a factor of at least 1000 than our estimation of the arrangement size. See Table 10.2 for the detailed timing results of our decider on the benchmarks described above.

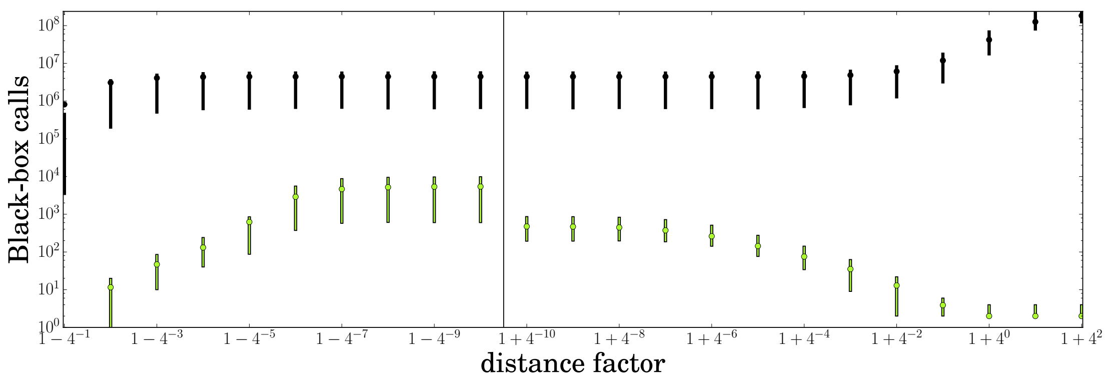
Approximate value computation experiments. We evaluate our implementation of the algorithm presented in Section 10.4 by computing an estimate $\tilde{\delta}$ such that $|\tilde{\delta} - d_{dF}^{T}(\pi, \sigma)| \le \varepsilon$ with a choice of $\varepsilon = 10^{-7}$.[8] In particular, we compare the performances of the different approaches discussed in Section 10.4:

- **Binary Search:** Binary search using our Fréchet-under-translation decider of Section 10.3.
- **Lipschitz-only:** Algorithm 21 without the arrangement, i.e., without lines 18 to 23.

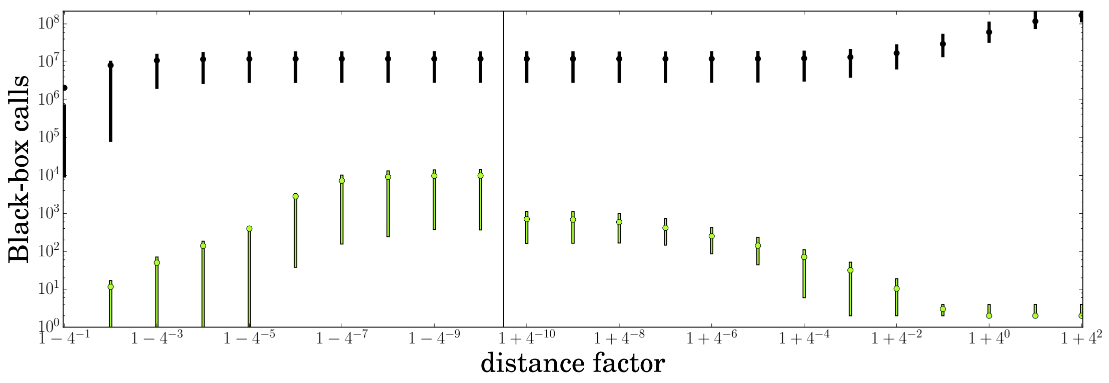
[7]We only give an estimate for the arrangement size, since the size of the arrangement is too large to be evaluated exactly for all our benchmark queries within a day. Specifically, we estimate the number of vertices of the arrangement which closely corresponds to the number of faces by Euler's formula. We give the following estimate: We first determine a search box B for the given decision instance $\pi = (\pi_1, \ldots, \pi_n), \sigma = (\sigma_1, \ldots, \sigma_m), \delta$ as described for our algorithm. We then sample $S = 100000$ tuples $i_1, i_2 \in \{1, \ldots, n\}, j_1, j_2 \in \{1, \ldots, m\}$ and count the number I of intersections of the circles of radius δ around $\pi_{i_1} - \sigma_{j_1}$ and $\pi_{i_2} - \sigma_{j_2}$ inside B. The number $(I/S) \cdot (nm)^2$ is the estimated number of circle-circle intersections in B. Adding the number of circle-box intersections, which we can compute exactly, yields our estimate.

[8]Here we use additive rather than multiplicative approximation for technical reasons. Since all computed distances are within $[1.6, 120.7]$, this also yields a multiplicative $(1 + \varepsilon)$-approximation with $\varepsilon \le 10^{-7}$.

Same-characters:



All-characters:



Sigspatial:

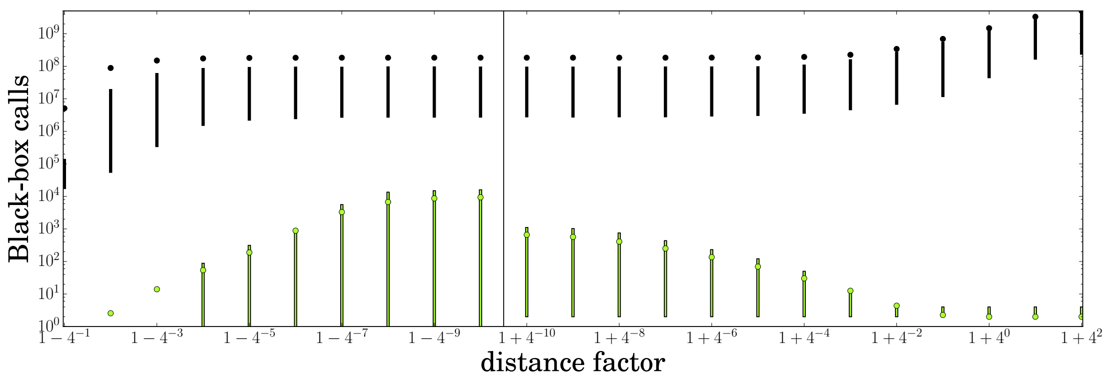


Figure 10.5: Number of black-box calls to the fixed-translation Fréchet decider made by our decider (below, in green), as well as an estimate of the arrangement complexity, i.e., number of calls of a naive algorithm (above, in black). We plot the mean number of calls and arrangement complexity over 1000 NO (or YES) queries with a test distance of approximately $(1-4^{-\ell})$ (or $(1+4^{-\ell})$) times the true discrete Fréchet distance under translation, as well as the interval between the lower and upper quartile over the queries.

SAME-CHARACTERS Time		Black-Box Calls
429,623 ms (18.7 ms per instance)		26,661,524 (1,159.2 per instance)
- Preprocessing	5 ms	
- Black-box calls (Lipschitz)	44,312 ms	
- Arrangement estimation	157,780 ms	
- Arrangement algorithm	226,469 ms	
* Construction	148,898 ms	
* Black-box calls	60,156 ms	

ALL-CHARACTERS Time		Black-Box Calls
628,043 ms (27.3 ms per instance)		42,781,931 (1,860.08 per instance)
- Preprocessing	5 ms	
- Black-box calls (Lipschitz)	50,462 ms	
- Arrangement estimation	191,177 ms	
- Arrangement algorithm	385,145 ms	
* Construction	237,043 ms	
* Black-box calls	120,149 ms	

SIGSPATIAL Time		Black-Box Calls
1,207,560 ms (52.5 ms per instance)		31,420,517 (1,366.11 per instance)
- Preprocessing	5 ms	
- Black-box calls (Lipschitz)	43,861 ms	
- Arrangement estimation	913,266 ms	
- Arrangement algorithm	249,268 ms	
* Construction	155,332 ms	
* Black-box calls	73,934 ms	

Table 10.2: Time measurements for the components of the decider over the complete decider benchmark sets. In parentheses, we give average values over the total of 23,000 decision instances.

Approach	Time	Black-Box Calls
LMF	148,032 ms (141.0 ms per instance)	13,323,232 (12,688.8 per instance)
Binary Search	536,853 ms (511.3 ms per instance)	45,909,628 (43,723.5 per instance)
Lipschitz-only	4,204,521 ms (4,004.3 ms per instance)	820,468,224 (781,398.3 per instance)

Table 10.3: Statistics for approximate value computation for $N_{\text{samples}} = 5$. In parentheses we show the mean values averaged over a total of 1050 instances.

- **Lipschitz-meets-Fréchet (LMF):** Our implementation as detailed in Section 10.4.

Since simple estimates show that the ε-approximate sets are clearly too costly for $\varepsilon = 10^{-7}$, we drop this approach from all further consideration. We took care to implement all approaches with a similar effort of low-level optimizations.

For our evaluation, we focus on the Characters data set which allows us to distinguish the rough shape of the curves: We subdivide the curve set into the subsets C_α for $\alpha \in \Sigma$ (where Σ is the set of 20 characters occurring in Characters). In particular for each character pair $\alpha, \beta \in \Sigma$, we create a sample of N_{samples} curve pairs (π, σ) chosen uniformly at random from $C_\alpha \times C_\beta$. For $N_{\text{samples}} = 5$, computing the value (up to $\varepsilon = 10^{-7}$) for all $N_{\text{samples}} \cdot (\binom{|\Sigma|}{2} + |\Sigma|) = 1050$ sampled curve pairs gives the statistics shown in Table 10.3.

Since already for this example the Lipschitz-only approach is dominated by almost a factor of 30 by LMF (and by a factor of almost 8 by binary search), we perform more detailed analyses with $N_{\text{samples}} = 100$ only for LMF and binary search. The overall performance is given in Table 10.4. Also here LMF is more than 3 times faster than binary search. To give more insights into the relationship of their running times, we give a scatter plot of the running times of LMF and binary search on the same instances over the complete benchmark in Figure 10.6, showing that binary search generally outperforms LMF only on instances which are comparably easy for LMF as well. The advantage of LMF becomes particularly clear on hard instances.

Apart from these general statistics for our value computation benchmarks, we depict individual mean computation times and mean number of black-box calls (over all N_{samples} samples) for each character pair $\alpha, \beta \in \Sigma$ in Figures 10.7 and 10.8.

Finally, we give the average distance values on our benchmark set both under a fixed translation (specifically, with start points of π and σ normalized to the origin) and under translation in Figure 10.9. Note that using naive approaches computing these tables would have been computationally extremely costly.

10.6 Conclusion

We engineer the first practical implementation for the discrete Fréchet distance under translation in the plane. While previous algorithmic solution for the problem solve it via expensive discrete methods, we introduce a new method from continuous optimization

LMF		
Time		**Black-Box Calls**
2,938,512 ms (140.0 ms per instance)		260,128,449 (12,387.1 per instance)
- Preprocessing	71,728 ms	
- Black-box calls (Lipschitz)	400,189 ms	
- Arrangement estimation	166,479 ms	
- Arrangement algorithm	2,250,493 ms	
* Construction	1,537,500 ms	
* Black-box calls	545,442 ms	
Binary Search		
Time		**Black-Box Calls**
10,555,630 ms (502.7 ms per instance)		875,424,988 (41,686.9 per instance)

Table 10.4: Statistics for approximate value computation for $N_{\text{samples}} = 100$. In parentheses, we give average values over the total of 21,000 curve pairs.

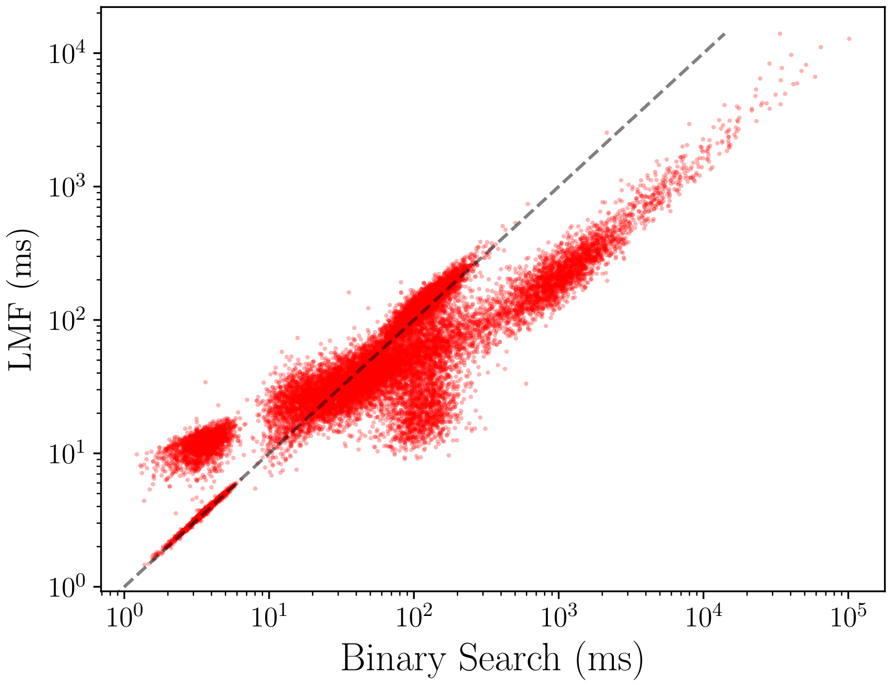
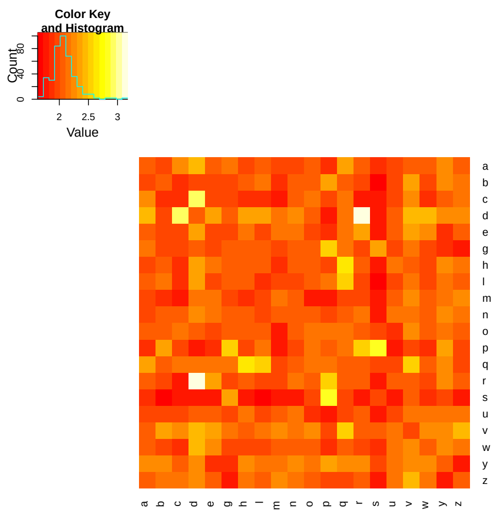


Figure 10.6: Running times of LMF and binary search on set of randomly sampled Characters curves.



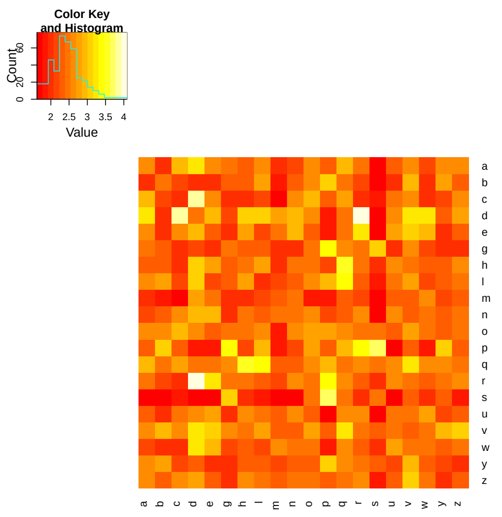
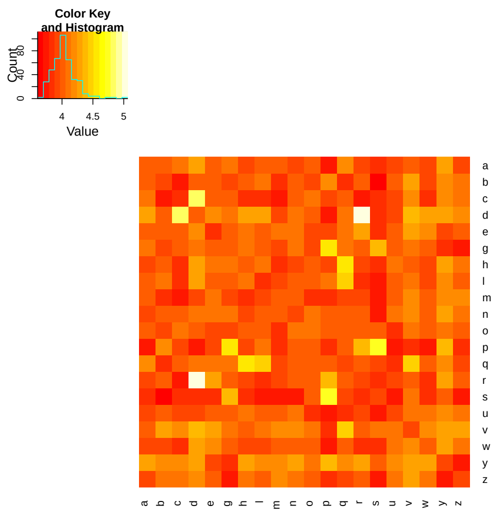


Figure 10.7: Log of mean value computation time in ms for LMF (left) and Binary Search (right).



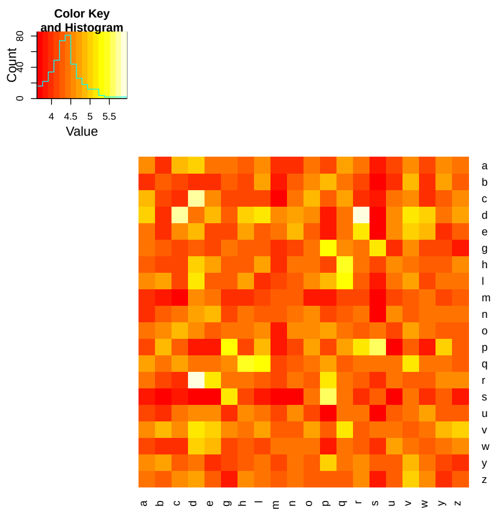
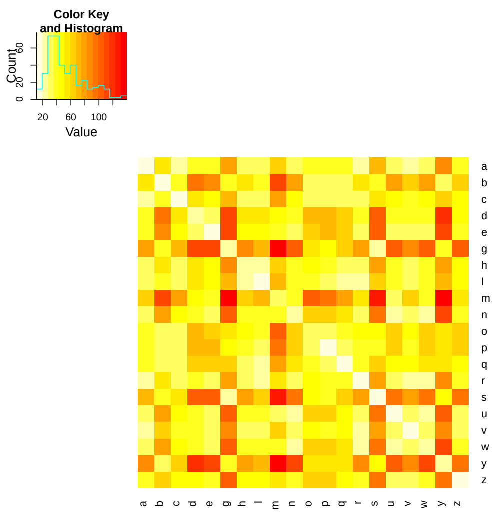
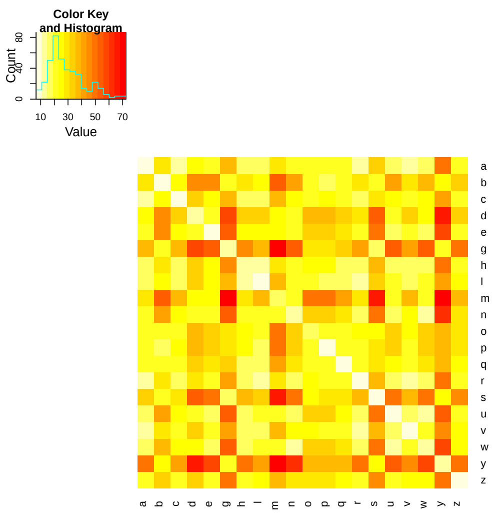


Figure 10.8: Log of mean number of black-box calls for LMF (left) and Binary Search (right).





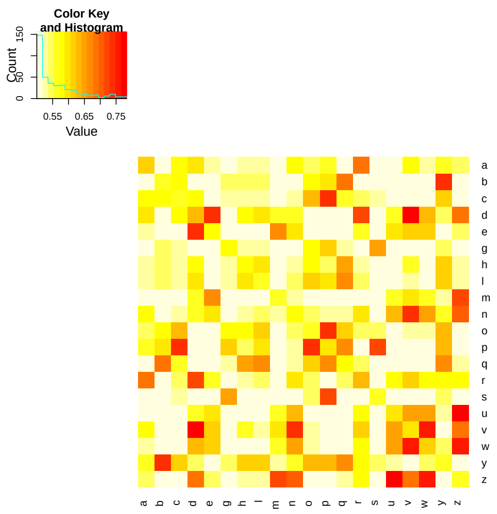


Figure 10.9: Average discrete Fréchet distance value (top left) and average discrete Fréchet distance under translation value (top right), as well as the quotient of these values (bottom left).

to achieve significant speed-ups on realistic inputs. This is analogous to the success of integer programming solvers which, while optimizing a discrete problem, choose to work over the reals to gain access to linear programming relaxations, cutting planes methods, and more. A novelty here is that we successfully apply such methods to obtain drastic speed-ups for a *polynomial-time problem*.

We leave as open problems to determine whether there are reasonable analogues of further ideas from integer programming, such as cutting plane methods or preconditioning, that could help to get further improved algorithms for our problem. More generally, we believe that this gives an exciting direction for algorithm engineering in general that should find wider applications. A particular direction in this vein is the use of our methods to compute rotation- or scaling-invariant versions of the Fréchet distance. Intuitively, by introducing additional dimensions in our search space, our methods can in principle also be used to optimize over such additional degrees of freedom. However, the Lipschitz condition changes significantly, and we leave it to future work to determine the applicability in these settings.

Bibliography

[1] A. Abboud, A. Backurs, K. Bringmann, and M. Künnemann. Fine-grained complexity of analyzing compressed data: quantifying improvements over decompress-and-solve. In C. Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 192–203. IEEE Computer Society, 2017. DOI: `10.1109/FOCS.2017.26`. URL: `https://doi.org/10.1109/FOCS.2017.26`.

[2] A. Abboud, A. Backurs, and V. V. Williams. Tight hardness results for LCS and other sequence similarity measures. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78. IEEE Computer Society, 2015. DOI: `10.1109/FOCS.2015.14`. URL: `https://doi.org/10.1109/FOCS.2015.14`.

[3] A. Abboud and K. Bringmann. Tighter connections between Formula-SAT and shaving logs. In I. Chatzigiannakis, C. Kaklamanis, D. Marx, and D. Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. DOI: `10.4230/LIPIcs.ICALP.2018.8`. URL: `https://doi.org/10.4230/LIPIcs.ICALP.2018.8`.

[4] A. Abboud and S. Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proc. IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS'16)*, pages 477–486, 2016.

[5] A. Abboud, A. Rubinstein, and R. R. Williams. Distributed PCP theorems for hardness of approximation in P. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 25–36, 2017. DOI: `10.1109/FOCS.2017.12`. URL: `https://doi.org/10.1109/FOCS.2017.12`.

[6] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014. DOI: `10.1109/FOCS.2014.53`. URL: `https://doi.org/10.1109/FOCS.2014.53`.

[7] A. Abboud, V. V. Williams, and O. Weimann. Consequences of faster alignment of sequences. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 39–51. Springer, 2014. DOI: `10.1007/978-3-662-43948-7\_4`. URL: `https://doi.org/10.1007/978-3-662-43948-7\_4`.

[8] I. Abraham, S. Chechik, and C. Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proc. 44th Annual ACM Symposium on Theory of Computing (STOC'12)*, pages 1199–1218, 2012.

[9] ACM SIGSPATIAL GIS Cup 2017 Data Set. `https://www.martinwerner.de/datasets/san-francisco-shortest-path.html`. Accessed: 2021-04-13.

[10] P. Afshani and A. Driemel. On the complexity of range searching among curves. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 898–917, 2018. DOI: `10.1137/1.9781611975031.58`. URL: `https://doi.org/10.1137/1.9781611975031.58`.

[11] P. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. In *Proc. 24th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'13)*, pages 156–167, 2013.

[12] ALCOM. `https://cordis.europa.eu/project/id/3075`. Accessed: 2021-09-28.

[13] ALCOM-FT. `https://cordis.europa.eu/project/id/IST-1999-14186`. Accessed: 2021-09-28.

[14] ALCOM II. `https://cordis.europa.eu/project/id/7141`. Accessed: 2021-09-28.

[15] ALCOM-IT. `https://cordis.europa.eu/project/id/20244`. Accessed: 2021-09-28.

[16] J. Alman and R. Williams. Probabilistic polynomials and hamming nearest neighbors. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 136–150, 2015. DOI: `10.1109/FOCS.2015.18`. URL: `https://doi.org/10.1109/FOCS.2015.18`.

[17] H. Alt, B. Behrends, and J. Blömer. Approximate matching of polygonal shapes. *Ann. Math. Artif. Intell.*, 13(3-4):251–265, 1995. DOI: `10.1007/BF01530830`. URL: `https://doi.org/10.1007/BF01530830`.

[18] H. Alt and M. Buchin. Can we compute the similarity between surfaces? *Discrete & Computational Geometry*, 43(1):78–99, 2010.

[19] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995. DOI: `10.1142/S0218195995000064`. URL: `https://doi.org/10.1142/S0218195995000064`.

[20] H. Alt, C. Knauer, and C. Wenk. Bounding the Fréchet distance by the Hausdorff distance. In *In Proceedings of the Seventeenth European Workshop on Computational Geometry*, pages 166–169, 2001.

[21] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004. DOI: `10.1007/s00453-003-1042-5`. URL: `https://doi.org/10.1007/s00453-003-1042-5`.

[22] H. Alt, C. Knauer, and C. Wenk. Matching polygonal curves with respect to the Fréchet distance. In *Proc. 18th Annual Symposium on Theoretical Aspects of Computer Science (STACS'01)*, pages 63–74, 2001.

[23] A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein. On hardness of jumbled indexing. In J. Esparza, P. Fraigniaud, T. Husfeldt, and E. Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 114–125. Springer, 2014. DOI: 10.1007/978-3-662-43948-7_10. URL: https://doi.org/10.1007/978-3-662-43948-7_10.

[24] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, Jan. 2008. ISSN: 0001-0782. DOI: 10.1145/1327452.1327494. URL: https://doi.org/10.1145/1327452.1327494.

[25] T. Asano, D. Kirkpatrick, K. Nakagawa, and O. Watanabe. $\widetilde{O}(\sqrt{n})$-space and polynomial-time algorithm for planar directed graph reachability. In *Proc. 39th International Symposium on Mathematical Foundations of Computer Science (MFCS'14)*, pages 45–56, 2014.

[26] R. Ashida and K. Nakagawa. $\widetilde{O}(n^{1/3})$-space algorithm for the grid graph reachability problem. In *Proc. 34th International Symposium on Computational Geometry (SoCG'18)*, 5:1–5:13, 2018.

[27] M. Astefanoaei, P. Cesaretti, P. Katsikouli, M. Goswami, and R. Sarkar. Multi-resolution sketches and locality sensitive hashing for fast trajectory processing. In *Proc. 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS)*, 2018.

[28] G. Ausiello. *Europe strikes back*. In *The Making of a New Science: A Personal Journey Through the Early Years of Theoretical Computer Science*. Springer International Publishing, Cham, 2018, pages 211–232. ISBN: 978-3-319-62680-2. DOI: 10.1007/978-3-319-62680-2_10. URL: https://doi.org/10.1007/978-3-319-62680-2_10.

[29] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(2):177–191, 1988.

[30] J. Baldus and K. 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'17, 99:1–99:4, Redondo Beach, CA, USA. ACM, 2017. ISBN: 978-1-4503-5490-5. DOI: 10.1145/3139958.3140062. URL: http://doi.acm.org/10.1145/3139958.3140062.

[31] G. Barequet and S. Har-Peled. Polygon containment and translational in-Hausdorff-distance between segment sets are 3SUM-hard. *International Journal of Computational Geometry & Applications*, 11(04):465–474, Aug. 2001. ISSN: 0218-1959. DOI: 10.1142/S0218195901000596. URL: https://www.worldscientific.com/doi/abs/10.1142/S0218195901000596 (visited on 11/19/2019).

[32] R. Ben Avraham, O. Filtser, H. Kaplan, M. J. Katz, and M. Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Transactions on Algorithms*, 11(4):29, 2015.

[33] R. Ben Avraham, H. Kaplan, and M. Sharir. A faster algorithm for the discrete Fréchet distance under translation. *ArXiv preprint `http://arxiv.org/abs/1501.03724`*, 2015.

[34] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In U. M. Fayyad and R. 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.

[35] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proc. 31st International Conf. Very Large Data Bases (VLDB'05)*, pages 853–864, 2005.

[36] M. Brankovic, K. Buchin, K. Klaren, A. Nusser, A. Popov, and S. Wong. (k, l)-medians clustering of trajectories using continuous dynamic time warping. In C. Lu, F. Wang, G. Trajcevski, Y. Huang, S. D. Newsam, and L. Xiong, editors, *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020*, pages 99–110. ACM, 2020. DOI: `10.1145/3397536.3422245`. URL: `https://doi.org/10.1145/3397536.3422245`.

[37] K. Bringmann. Fine-grained complexity theory (tutorial). In R. Niedermeier and C. Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, volume 126 of *LIPIcs*, 4:1–4:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI: `10.4230/LIPIcs.STACS.2019.4`. URL: `https://doi.org/10.4230/LIPIcs.STACS.2019.4`.

[38] K. 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, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. DOI: `10.1109/FOCS.2014.76`. URL: `https://doi.org/10.1109/FOCS.2014.76`.

[39] K. Bringmann, A. Driemel, A. Nusser, and I. 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. To appear.

[40] K. Bringmann and M. Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *Int. J. Comput. Geometry Appl.*, 27(1-2):85–120, 2017. DOI: `10.1142/S0218195917600056`. URL: `https://doi.org/10.1142/S0218195917600056`.

[41] K. Bringmann and M. Künnemann. Multivariate fine-grained complexity of longest common subsequence. In A. Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235. SIAM, 2018. DOI: `10.1137/1.9781611975031.79`. URL: `https://doi.org/10.1137/1.9781611975031.79`.

[42] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In V. Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97. IEEE Computer Society, 2015. DOI: `10.1109/FOCS.2015.15`. URL: `https://doi.org/10.1109/FOCS.2015.15`.

[43] K. Bringmann, M. Künnemann, and A. Nusser. Fréchet distance under translation: conditional hardness and an algorithm via offline dynamic grid reachability. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2902–2921. SIAM, 2019. DOI: `10.1137/1.9781611975482.180`. URL: `https://doi.org/10.1137/1.9781611975482.180`.

[44] K. Bringmann, M. Künnemann, and A. Nusser. Walking the dog fast in practice: algorithm engineering of the Fréchet distance. In G. Barequet and Y. Wang, editors, *35th International Symposium on Computational Geometry, SoCG 2019, June 18-21, 2019, Portland, Oregon, USA*, volume 129 of *LIPIcs*, 17:1–17:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. DOI: `10.4230/LIPIcs.SoCG.2019.17`. URL: `https://doi.org/10.4230/LIPIcs.SoCG.2019.17`.

[45] K. Bringmann, M. Künnemann, and A. Nusser. When Lipschitz walks your dog: algorithm engineering of the discrete Fréchet distance under translation. In F. Grandoni, G. Herman, and P. Sanders, editors, *28th Annual European Symposium on Algorithms, ESA 2020, September 7-9, 2020, Pisa, Italy (Virtual Conference)*, volume 173 of *LIPIcs*, 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. DOI: `10.4230/LIPIcs.ESA.2020.25`. URL: `https://doi.org/10.4230/LIPIcs.ESA.2020.25`.

[46] K. Bringmann, M. Künnemann, and A. Nusser. Code of "Walking the Dog Fast in Practice: Algorithm Engineering of the Fréchet Distance", Dec. 2018. DOI: `10.5281/zenodo.5644988`. URL: `https://doi.org/10.5281/zenodo.5644988`.

[47] K. Bringmann, M. Künnemann, and A. Nusser. Code of "When Lipschitz Walks Your Dog: Algorithm Engineering of the Discrete Fréchet Distance Under Translation", Apr. 2020. DOI: `10.5281/zenodo.5645105`. URL: `https://doi.org/10.5281/zenodo.5645105`.

[48] K. Bringmann and W. Mulzer. Approximability of the discrete Fréchet distance. *J. Comput. Geom.*, 7(2):46–76, 2016. DOI: `10.20382/jocg.v7i2a4`. URL: `https://doi.org/10.20382/jocg.v7i2a4`.

[49] K. Bringmann and A. Nusser. Translating Hausdorff is hard: fine-grained lower bounds for Hausdorff distance under translation. In K. Buchin and É. C. de Verdière, editors, *37th International Symposium on Computational Geometry, SoCG 2021, June 7-11, 2021, Buffalo, NY, USA (Virtual Conference)*, volume 189 of *LIPIcs*, 18:1–18:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. DOI: `10.4230/LIPIcs.SoCG.2021.18`. URL: `https://doi.org/10.4230/LIPIcs.SoCG.2021.18`.

[50] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *Internat. J. Comput. Geom. Appl.*, 21(3):253–282, 2011.

[51] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four soviets walk the dog - with an application to Alt's conjecture. In *Proc. 25th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA'14)*, pages 1399–1413, 2014.

[52] K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th Annu. ACM-SIAM Symp. Discrete Algorithms (SODA'09)*, pages 645–654. Society for Industrial and Applied Mathematics, 2009.

[53] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons. *Computational Geometry*, 41(1-2):2–20, 2008.

[54] K. Buchin, Y. Diez, T. van Diggelen, and W. 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'17, 101:1–101:4, Redondo Beach, CA, USA. ACM, 2017. ISBN: 978-1-4503-5490-5. DOI: `10.1145/3139958.3140064`. URL: `http://doi.acm.org/10.1145/3139958.3140064`.

[55] K. Buchin, A. Driemel, N. van de L'Isle, and A. Nusser. Klcluster: center-based clustering of trajectories. In F. B. Kashani, G. Trajcevski, R. H. Güting, L. Kulik, and S. D. Newsam, editors, *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2019, Chicago, IL, USA, November 5-8, 2019*, pages 496–499. ACM, 2019. DOI: `10.1145/3347146.3359111`. URL: `https://doi.org/10.1145/3347146.3359111`.

[56] K. Buchin, T. Ophelders, and B. Speckmann. SETH says: weak Fréchet distance is faster, but only if it is continuous and in one dimension. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2887–2901. SIAM, 2019. DOI: `10.1137/1.9781611975482.179`. URL: `https://doi.org/10.1137/1.9781611975482.179`.

[57] M. Buchin. *On the Computability of the Fréchet Distance Between Triangulated Surfaces.* PhD thesis, Freie Universität Berlin, 2007. DOI: `10.17169/refubium-6111`.

[58] J. Campbell, J. Tremblay, and C. Verbrugge. Clustering player paths. In *FDG*, 2015.

[59] M. Ceccarello, A. Driemel, and F. Silvestri. FRESH: Fréchet similarity with hashing. *CoRR*, abs/1809.02350, 2018. arXiv: `1809.02350`. URL: `http://arxiv.org/abs/1809.02350`.

[60] CGAL. `https://cordis.europa.eu/project/id/21957`. Accessed: 2021-09-28.

[61] E. Chambers, B. T. Fasy, Y. Wang, and C. Wenk. Map-matching using shortest paths. In *Proceedings of the 3rd International Workshop on Interactive and Spatial Computing*, pages 44–51. ACM, 2018.

[62] E. W. Chambers, É. Colin de Verdière, J. Erickson, S. Lazard, F. Lazarus, and S. Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010.

[63] T. M. Chan and Q. He. Reducing 3SUM to convolution-3SUM. In M. Farach-Colton and I. L. Gørtz, editors, *3rd Symposium on Simplicity in Algorithms, SOSA@SODA 2020, Salt Lake City, UT, USA, January 6-7, 2020*, pages 1–7. SIAM, 2020. DOI: `10.1137/1.9781611976014.1`. URL: `https://doi.org/10.1137/1.9781611976014.1`.

[64] T. M. Chan and K. Tsakalidis. Dynamic orthogonal range searching on the ram, revisited. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, 28:1–28:13, 2017.

[65] Character Trajectories Data Set. `https://archive.ics.uci.edu/ml/datasets/Character+Trajectories`. Accessed: 2021-04-13.

[66] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.

[67] D. Chen, A. Driemel, L. J. Guibas, A. Nguyen, and C. Wenk. Approximate map matching with respect to the Fréchet distance. In *2011 Proceedings of the Thirteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 75–83. SIAM, 2011.

[68] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In F. Özcan, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005*, pages 491–502. ACM, 2005. DOI: `10.1145/1066157.1066213`. URL: `https://doi.org/10.1145/1066157.1066213`.

[69] L. Chen, S. Goldwasser, K. Lyu, G. N. Rothblum, and A. Rubinstein. Fine-grained complexity meets IP = PSPACE. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1–20. SIAM, 2019. DOI: `10.1137/1.9781611975482.1`. URL: `https://doi.org/10.1137/1.9781611975482.1`.

[70] L. Chen and R. Williams. An equivalence class for orthogonal vectors. In T. M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 21–40. SIAM, 2019. DOI: `10.1137/1.9781611975482.2`. URL: `https://doi.org/10.1137/1.9781611975482.2`.

[71] L. P. Chew, D. Dor, A. Efrat, and K. Kedem. Geometric pattern matching in d-dimensional space. en. *Discrete & Computational Geometry*, 21(2):257–274, Feb. 1999. ISSN: 1432-0444. DOI: `10.1007/PL00009420`. URL: `https://doi.org/10.1007/PL00009420` (visited on 03/09/2021).

[72] L. P. Chew and K. Kedem. Improvements on geometric pattern matching problems. en. In O. Nurmi and E. Ukkonen, editors, *Algorithm Theory — SWAT '92*, Lecture Notes in Computer Science, pages 318–325. Springer Berlin Heidelberg, 1992. ISBN: 978-3-540-47275-9.

[73] I. R. Cleasby, E. D. Wakefield, B. J. Morrissey, T. W. Bodey, S. C. Votier, S. Bearhop, and K. C. Hamer. Using time-series similarity measures to compare animal movement trajectories in ecology. *Behavioral Ecology and Sociobiology*, 73(11):151, 2019. ISSN: 1432-0762. DOI: 10.1007/s00265-019-2761-1. URL: https://doi.org/10.1007/s00265-019-2761-1.

[74] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *Journal of the ACM*, 34(1):200–208, 1987.

[75] A. F. Cook and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):193–204, 2010.

[76] M. de Berg and A. F. Cook. Go with the flow: the direction-based Fréchet distance of polygonal curves. In A. Marchetti-Spaccamela and M. Segal, editors, *Theory and Practice of Algorithms in (Computer) Systems - First International ICST Conference, TAPAS 2011, Rome, Italy, April 18-20, 2011. Proceedings*, volume 6595 of *Lecture Notes in Computer Science*, pages 81–91. Springer, 2011. DOI: 10.1007/978-3-642-19754-3_10. URL: https://doi.org/10.1007/978-3-642-19754-3_10.

[77] M. de Berg, J. Gudmundsson, and A. 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, GIS 2017, Redondo Beach, CA, USA, November 7-10, 2017*, 48:1–48:4, 2017. DOI: 10.1145/3139958.3140023. URL: https://doi.org/10.1145/3139958.3140023.

[78] M. de Berg, A. F. Cook, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747 –755, 2013. ISSN: 0925-7721. DOI: https://doi.org/10.1016/j.comgeo.2012.11.006. URL: http://www.sciencedirect.com/science/article/pii/S0925772112001617.

[79] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL: http://archive.ics.uci.edu/ml.

[80] K. Diks and P. Sankowski. Dynamic plane transitive closure. In *Proc. 15th European Symposium on Algorithms (ESA'07)*, pages 594–604, 2007.

[81] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

[82] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012. ISSN: 1432-0444. DOI: 10.1007/s00454-012-9402-z. URL: https://doi.org/10.1007/s00454-012-9402-z.

[83] A. Driemel, A. Krivosija, and C. Sohler. Clustering time series under the Fréchet distance. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 766–785, 2016. DOI: 10.1137/1.9781611974331.ch55.

[84] A. Driemel and I. Psarros. $(2+\varepsilon)$-ANN for time series under the Fréchet distance. *CoRR*, abs/2008.09406, 2020. arXiv: 2008.09406. URL: https://arxiv.org/abs/2008.09406.

[85] A. Driemel and I. Psarros. ANN for time series under the Fréchet distance. In A. Lubiw and M. R. Salavatipour, editors, *Algorithms and Data Structures - 17th International Symposium, WADS 2021, Virtual Event, August 9-11, 2021, Proceedings*, volume 12808 of *Lecture Notes in Computer Science*, pages 315–328. Springer, 2021. DOI: `10.1007/978-3-030-83508-8\_23`. URL: `https://doi.org/10.1007/978-3-030-83508-8\_23`.

[86] A. Driemel and F. Silvestri. Locality-sensitive hashing of curves. In *33rd International Symposium on Computational Geometry, SoCG 2017, July 4-7, 2017, Brisbane, Australia*, 37:1–37:16, 2017. DOI: `10.4230/LIPIcs.SoCG.2017.37`. URL: `https://doi.org/10.4230/LIPIcs.SoCG.2017.37`.

[87] F. Dütsch and J. 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'17, 100:1–100:4, Redondo Beach, CA, USA. ACM, 2017. ISBN: 978-1-4503-5490-5. DOI: `10.1145/3139958.3140063`. URL: `http://doi.acm.org/10.1145/3139958.3140063`.

[88] A. Efrat, P. Indyk, and S. Venkatasubramanian. Pattern matching for sets of segments. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 295–304, 2001.

[89] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

[90] I. Z. Emiris and I. Psarros. Products of euclidean metrics, applied to proximity problems among curves: unified treatment of discrete Fréchet and dynamic time warping distances. *ACM Trans. Spatial Algorithms Syst.*, 6(4):27:1–27:20, 2020. URL: `https://dl.acm.org/doi/10.1145/3397518`.

[91] M. Farach-Colton and P. Indyk. Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 171–180. IEEE Computer Society, 1999. DOI: `10.1109/SFFCS.1999.814589`. URL: `https://doi.org/10.1109/SFFCS.1999.814589`.

[92] A. Fedorov, E. Billet, M. Prastawa, G. Gerig, A. Radmanesh, S. K. Warfield, R. Kikinis, and N. Chrisochoides. Evaluation of brain MRI alignment with the robust Hausdorff distance measures. In G. Bebis, R. D. Boyle, B. Parvin, D. Koracin, P. Remagnino, F. M. Porikli, J. Peters, J. T. Klosowski, L. L. Arns, Y. K. Chun, T. Rhyne, and L. Monroe, editors, *Advances in Visual Computing, 4th International Symposium, ISVC 2008, Las Vegas, NV, USA, December 1-3, 2008. Proceedings, Part I*, volume 5358 of *Lecture Notes in Computer Science*, pages 594–603. Springer, 2008. DOI: `10.1007/978-3-540-89639-5\_57`. URL: `https://doi.org/10.1007/978-3-540-89639-5\_57`.

[93] A. Filtser, O. Filtser, and M. J. Katz. Approximate nearest neighbor for curves - simple, efficient, and deterministic. In A. Czumaj, A. Dawar, and E. Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of

LIPIcs, 48:1–48:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. DOI: 10.4230/LIPIcs.ICALP.2020.48. URL: https://doi.org/10.4230/LIPIcs.ICALP.2020.48.

[94] M. M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 22(1):1–72, 1906.

[95] M. L. Fredman, J. Komlós, and E. Szemerédi. Storing a sparse table with $O(1)$ worst case access time. *J. ACM*, 31(3):538–544, 1984. DOI: 10.1145/828.1884. URL: https://doi.org/10.1145/828.1884.

[96] H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC'84)*, pages 135–143, 1984.

[97] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995. DOI: 10.1016/0925-7721(95)00022-2. URL: https://doi.org/10.1016/0925-7721(95)00022-2.

[98] E. A. Galperin. Precision, complexity, and computational schemes of the cubic algorithm. *Journal of Optimization Theory and Applications*, 57(2):223 –238, 1988. DOI: https://doi.org/10.1007/BF00938537.

[99] E. A. Galperin. The cubic algorithm. *Journal of Mathematical Analysis and Applications*, 112(2):635 –640, 1985. ISSN: 0022-247X. DOI: https://doi.org/10.1016/0022-247X(85)90268-9. URL: http://www.sciencedirect.com/science/article/pii/0022247X85902689.

[100] E. A. Galperin. The fast cubic algorithm. *Computers & Mathematics with Applications*, 25(10):147 –160, 1993. ISSN: 0898-1221. DOI: https://doi.org/10.1016/0898-1221(93)90289-8. URL: http://www.sciencedirect.com/science/article/pii/0898122193902898.

[101] E. A. Galperin. Two alternatives for the cubic algorithm. *Journal of Mathematical Analysis and Applications*, 126(1):229 –237, 1987. ISSN: 0022-247X. DOI: https://doi.org/10.1016/0022-247X(87)90088-6. URL: http://www.sciencedirect.com/science/article/pii/0022247X87900886.

[102] GeoLife GPS Trajectories. https://www.microsoft.com/en-us/download/details.aspx?id=52367. Accessed: 2021-04-13.

[103] O. Gold and M. Sharir. Dynamic time warping and geometric edit distance: breaking the quadratic barrier. *ACM Trans. Algorithms*, 14(4):50:1–50:17, 2018. DOI: 10.1145/3230734. URL: https://doi.org/10.1145/3230734.

[104] E. Gourdin, P. Hansen, and B. Jaumard. Global optimization of multivariate Lipschitz functions: survey and computational comparison, 1994.

[105] J. Gudmundsson, M. Mirzanezhad, A. Mohades, and C. Wenk. Fast Fréchet distance between curves with long edges. *Int. J. Comput. Geom. Appl.*, 29(2):161–187, 2019. DOI: 10.1142/S0218195919500043. URL: https://doi.org/10.1142/S0218195919500043.

[106] J. Gudmundsson, A. van Renssen, Z. Saeidi, and S. Wong. Translation invariant Fréchet distance queries. *CoRR*, abs/2102.05844, 2021. arXiv: 2102.05844. URL: https://arxiv.org/abs/2102.05844.

[107] J. Gudmundsson and S. Wong. Cubic upper and lower bounds for subtrajectory clustering under the continuous Fréchet distance. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2022. To appear.

[108] P. Hansen and B. Jaumard. Lipschitz optimization. In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, pages 407–493. Springer US, Boston, MA, 1995. DOI: `10.1007/978-1-4615-2025-2\_9`. URL: `https://doi.org/10.1007/978-1-4615-2025-2\_9`.

[109] S. Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, Boston, MA, USA, 2011. ISBN: 0821849115, 9780821849118.

[110] R. Horst. A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization. *Journal of Optimization Theory and Applications*, 51:271 –291, 1986. DOI: `https://doi.org/10.1007/BF00939825`.

[111] R. Horst and H. Tuy. *Global Optimization – Deterministic Approaches*. Springer Berlin Heidelberg, 3rd edition, 1996.

[112] R. Horst and H. Tuy. On the convergence of global methods in multiextremal optimization. *Journal of Optimization Theory and Applications*, 54:253 –271, 1987. DOI: `https://doi.org/10.1007/BF00939434`.

[113] D. P. Huttenlocher, K. Kedem, and M. Sharir. The upper envelope of Voronoi surfaces and its applications. *Discrete & Computational Geometry*, 9(3):267–291, 1993.

[114] D. P. Huttenlocher, G. A. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863, 1993. DOI: `10.1109/34.232073`. URL: `https://doi.org/10.1109/34.232073`.

[115] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. DOI: `10.1006/jcss.2000.1727`. URL: `https://doi.org/10.1006/jcss.2000.1727`.

[116] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001. DOI: `10.1006/jcss.2001.1774`. URL: `https://doi.org/10.1006/jcss.2001.1774`.

[117] P. Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proceedings of the 18th Annual Symposium on Computational Geometry, Barcelona, Spain, June 5-7, 2002*, pages 102–106, 2002. DOI: `10.1145/513400.513414`. URL: `https://doi.org/10.1145/513400.513414`.

[118] G. F. Italiano, A. Karczmarz, J. Łącki, and P. Sankowski. Decremental single-source reachability in planar digraphs. In *Proc. 49th Annual ACM Symposium on Theory of Computing*, pages 1108–1121, 2017.

[119] G. F. Italiano, Y. Nussbaum, P. Sankowski, and C. Wulff-Nilsen. Improved algorithms for min cut and max flow in undirected planar graphs. In *Proc. 43rd Annual ACM Symposium on Theory of Computing (STOC'11)*, pages 313–322, 2011.

[120] M. Jiang, Y. Xu, and B. Zhu. Protein structure–structure alignment with discrete Fréchet distance. *J. Bioinformatics and Computational Biology*, 6(01):51–64, 2008.

[121] E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005. DOI: `10.1007/s10115-004-0154-9`. URL: `https://doi.org/10.1007/s10115-004-0154-9`.

[122] S. Kim, S. Park, and W. W. Chu. An index-based approach for similarity search supporting time warping in large sequence databases. In D. Georgakopoulos and A. Buchmann, editors, *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 607–614. IEEE Computer Society, 2001. DOI: `10.1109/ICDE.2001.914875`. URL: `https://doi.org/10.1109/ICDE.2001.914875`.

[123] J. King. A survey of 3SUM-hard problems, 2004.

[124] P. N. Klein and S. Subramanian. A fully dynamic approximation scheme for shortest paths in planar graphs. *Algorithmica*, 22(3):235–249, 1998.

[125] C. Knauer and M. Scherfenberg. Approximate nearest neighbor search under translation invariant Hausdorff distance. *Int. J. Comput. Geom. Appl.*, 21(3):369–381, 2011. DOI: `10.1142/S0218195911003706`. URL: `https://doi.org/10.1142/S0218195911003706`.

[126] Y. Liu, K. Chiang, C. L. Corbett, R. Archibald, B. Mukherjee, and D. Ghosal. A novel audio steganalysis based on high-order statistics of a distortion measure with Hausdorff distance. In T. Wu, C. Lei, V. Rijmen, and D. Lee, editors, *Information Security, 11th International Conference, ISC 2008, Taipei, Taiwan, September 15-18, 2008. Proceedings*, volume 5222 of *Lecture Notes in Computer Science*, pages 487–501. Springer, 2008. DOI: `10.1007/978-3-540-85886-7\_33`. URL: `https://doi.org/10.1007/978-3-540-85886-7\_33`.

[127] A. Maheshwari, J. Sack, and C. Scheffer. Approximating the integral Fréchet distance. *Comput. Geom.*, 70-71:13–30, 2018. DOI: `10.1016/j.comgeo.2018.01.001`. URL: `https://doi.org/10.1016/j.comgeo.2018.01.001`.

[128] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry*, 44(2):110–120, 2011.

[129] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Improved Algorithms for Partial Curve Matching. en. *Algorithmica*, 69(3):641–657, July 2014. ISSN: 1432-0541. DOI: `10.1007/s00453-013-9758-3`. URL: `https://doi.org/10.1007/s00453-013-9758-3` (visited on 04/30/2020).

[130] H. Mao, Q. Yu, and T. Zhang. Matching SAR image to optical image using modified Hausdorff distance and genetic algorithms. In S. J. Maybank, M. Ding, F. Wahl, and Y. Zhu, editors, *MIPPR 2007: Pattern Recognition and Computer Vision*, volume 6788, pages 532 –537. International Society for Optics and Photonics, SPIE, 2007. URL: `https://doi.org/10.1117/12.750623`.

[131] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[132] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[133] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[134] P. B. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '94, 625–634, Montreal, Quebec, Canada. Association for Computing Machinery, 1994. ISBN: 0897916638. DOI: `10.1145/195058.195415`. URL: `https://doi.org/10.1145/195058.195415`.

[135] M. Mirzanezhad. On the approximate nearest neighbor queries among curves under the Fréchet distance. *CoRR*, abs/2004.08444, 2020. arXiv: `2004.08444`. URL: `https://arxiv.org/abs/2004.08444`.

[136] C. W. Mortensen. Fully dynamic orthogonal range reporting on RAM. *SIAM J. Comput.*, 35(6):1494–1525, 2006.

[137] M. E. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proc. 7th Intl. Conf. Comp. Vision (ICCV'99)*, pages 108–115, 1999.

[138] M. Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the Forty-Second ACM Symposium on Theory of Computing*, STOC '10, 603–610, Cambridge, Massachusetts, USA. Association for Computing Machinery, 2010. ISBN: 9781450300506. DOI: `10.1145/1806689.1806772`. URL: `https://doi.org/10.1145/1806689.1806772`.

[139] S. Piyavskii. An algorithm for finding the absolute extremum of a function. *USSR Computational Mathematics and Mathematical Physics*, 12(4):57 –67, 1972. ISSN: 0041-5553. DOI: `https://doi.org/10.1016/0041-5553(72)90115-2`. URL: `http://www.sciencedirect.com/science/article/pii/0041555372901152`.

[140] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In Q. Yang, D. Agarwal, and J. Pei, editors, *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 262–270. ACM, 2012. DOI: `10.1145/2339530.2339576`. URL: `https://doi.org/10.1145/2339530.2339576`.

[141] L. Roditty and U. Zwick. On dynamic shortest paths problems. In S. Albers and T. Radzik, editors, *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*, pages 580–591. Springer, 2004. DOI: `10.1007/978-3-540-30140-0\_52`. URL: `https://doi.org/10.1007/978-3-540-30140-0\_52`.

[142] G. Rote. Computing the minimum Hausdorff distance between two point sets on a line under translation. en. *Information Processing Letters*, 38(3):123–127, May 1991. ISSN: 0020-0190. DOI: `10.1016/0020-0190(91)90233-8`. URL: `http://www.sciencedirect.com/science/article/pii/0020019091902338` (visited on 10/25/2019).

[143] A. Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1260–1268, 2018. DOI: 10.1145/3188745.3188916. URL: https://doi.org/10.1145/3188745.3188916.

[144] W. J. Rucklidge. Lower bounds for the complexity of the graph of the Hausdorff distance as a function of transformation. en. *Discrete & Computational Geometry*, 16(2):135–153, Feb. 1996. ISSN: 1432-0444. DOI: 10.1007/BF02716804. URL: https://doi.org/10.1007/BF02716804 (visited on 07/16/2019).

[145] W. Rucklidge. Efficiently locating objects using the Hausdorff distance. *Int. J. Comput. Vis.*, 24(3):251–270, 1997. DOI: 10.1023/A:1007975324482. URL: https://doi.org/10.1023/A:1007975324482.

[146] P. Sanders. *Algorithm engineering – an attempt at a definition.* In *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday.* S. Albers, H. Alt, and S. Näher, editors. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pages 321–340. ISBN: 978-3-642-03456-5. DOI: 10.1007/978-3-642-03456-5_22. URL: https://doi.org/10.1007/978-3-642-03456-5_22.

[147] D. F. Silva and G. E. A. P. A. Batista. Speeding up all-pairwise dynamic time warping matrix calculation. In S. C. Venkatasubramanian and W. M. Jr., editors, *Proceedings of the 2016 SIAM International Conference on Data Mining, Miami, Florida, USA, May 5-7, 2016*, pages 837–845. SIAM, 2016. DOI: 10.1137/1.9781611974348.94. URL: https://doi.org/10.1137/1.9781611974348.94.

[148] D. F. Silva, R. Giusti, E. J. Keogh, and G. E. A. P. A. Batista. Speeding up similarity search under dynamic time warping by pruning unpromising alignments. *Data Min. Knowl. Discov.*, 32(4):988–1016, 2018. DOI: 10.1007/s10618-018-0557-y. URL: https://doi.org/10.1007/s10618-018-0557-y.

[149] S. Subramanian. A fully dynamic data structure for reachability in planar digraphs. In *Proc. 1st European Symposium on Algorithms (ESA'93)*, pages 372–383, 1993.

[150] Y. Tao, A. Both, R. I. Silveira, K. Buchin, S. Sijben, R. S. Purves, P. Laube, D. Peng, K. Toohey, and M. Duckham. A comparative analysis of trajectory similarity measures. *GIScience & Remote Sensing*, 58(5):643–669, 2021. DOI: 10.1080/15481603.2021.1908927. eprint: https://doi.org/10.1080/15481603.2021.1908927. URL: https://doi.org/10.1080/15481603.2021.1908927.

[151] The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 5.3 edition, 2021. URL: https://doc.cgal.org/5.3/Manual/packages.html.

[152] V. Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proc. ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

[153] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[154] M. Vlachos, D. Gunopulos, and G. Kollios. Discovering similar multidimensional trajectories. In R. Agrawal and K. R. Dittrich, editors, *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 673–684. IEEE Computer Society, 2002. DOI: 10.1109/ICDE.2002.994784. URL: https://doi.org/10.1109/ICDE.2002.994784.

[155] H. Wei, R. Fellegara, Y. Wang, L. De Floriani, and H. Samet. Multi-level filtering to retrieve similar trajectories under the Fréchet distance. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '18, pages 600–603, Seattle, Washington. ACM, 2018. ISBN: 978-1-4503-5889-7. DOI: 10.1145/3274895.3274978. URL: http://doi.acm.org/10.1145/3274895.3274978.

[156] R. Wein, E. Berberich, E. Fogel, D. Halperin, M. Hemmer, O. Salzman, and B. Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0.2 edition, 2020. URL: https://doc.cgal.org/5.0.2/Manual/packages.html#PkgArrangementOnSurface2.

[157] C. Wenk. *Shape matching in higher dimensions*. PhD thesis, Freie Universität Berlin, 2002. PhD Thesis.

[158] M. Werner and D. Oliver. ACM SIGSPATIAL GIS Cup 2017: range queries under Fréchet distance. *SIGSPATIAL Special*, 10(1):24–27, June 2018. ISSN: 1946-7729. DOI: 10.1145/3231541.3231549. URL: http://doi.acm.org/10.1145/3231541.3231549.

[159] A. Wigderson. *Mathematics and Computation: A Theory Revolutionizing Technology and Science*. Princeton University Press, 2019. ISBN: 10.1515_9780691192543. DOI: doi:10.1515/9780691192543. URL: https://doi.org/10.1515/9780691192543.

[160] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. DOI: 10.1016/j.tcs.2005.09.023. URL: https://doi.org/10.1016/j.tcs.2005.09.023.

[161] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. DOI: 10.1109/FOCS.2010.67. URL: https://doi.org/10.1109/FOCS.2010.67.

[162] L. Wolsey. *Integer Programming*. Wiley Series in Discrete Mathematics and Optimization. Wiley, 1998. ISBN: 9780471283669.

[163] T. Wylie and B. Zhu. Intermittent map matching with the discrete Fréchet distance. *arXiv preprint arXiv:1409.2456*, 2014.

[164] J. Zheng, X. Gao, E. Zhan, and Z. Huang. Algorithm of on-line handwriting signature verification based on discrete Fréchet distance. In L. Kang, Z. Cai, X. Yan, and Y. Liu, editors, *Advances in Computation and Intelligence*, pages 461–469, Berlin, Heidelberg. Springer Berlin Heidelberg, 2008. ISBN: 978-3-540-92137-0.

[165] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: a collaborative social networking service among user, location and trajectory. *IEEE Data(base) Engineering Bulletin*, 2010. URL: https://www.microsoft.com/en-us/research/publication/geolife-a-collaborative-social-networking-service-among-user-location-and-trajectory/.

[166] Y. Zheng, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In proceedings of international conference on world wide web 2009 edition, 2009. URL: https://www.microsoft.com/en-us/research/publication/mining-interesting-locations-and-travel-sequences-from-gps-trajectories/. WWW 2009.

[167] Y. Zheng, X. Xie, and W.-Y. Ma. Understanding mobility based on gps data. In proceedings of the 10th acm conference on ubiquitous computing (ubicomp 2008) edition, 2008. URL: https://www.microsoft.com/en-us/research/publication/understanding-mobility-based-on-gps-data/.