The Discrete Fréchet Distance and Applications

Thesis submitted in partial fulfillment
of the requirements for the degree of
"DOCTOR OF PHILOSOPHY"

by

Omrit Filtser

Submitted to the Senate of
Ben-Gurion University of the Negev

March 2019

Beer-Sheva

This work was carried out under the supervision of

Prof. Matthew J. Katz

In the Department of Computer Science
Faculty of Natural Sciences

To my dear parents, my beloved husband,
and to my precious, clever, daughters...

“My mother made me a scientist without ever intending to. Every other Jewish mother in Brooklyn would ask her child after school: So? Did you learn anything today? But not my mother. ”Izzy,” she would say,“did you ask a good question today?” That difference – asking good questions – made me become a scientist.”

– Isidor Isaac Rabi

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Matthew (Matya) Katz, who guided me through both my Master and PhD studies. Matya, thank you for being such a wonderful teacher, for your great ideas and insights, and for your endless care and support. Your calmness and patience are a real blessing, I could not have asked for a better advisor.

I am most grateful to my collaborators: Boris Aronov, Stav Ashur, Rinat Ben Avraham, Daniel Berend, Liat Cohen, Stephane Durocher, Chenglin Fan, Arnold Filtser, Michael Horton, Haim Kaplan, Rachel Saban, Micha Sharir, Khadijeh Sheikhan, Tim Wylie, and Binhai Zhu. I am so happy that I had the chance to work with all of you, it was a pleasure and I have learned a lot.

My sincere thanks must also go to the administrative staff in the computer science department of Ben-Gurion university, for their care, kindness, and help in various bureaucracies. Furthermore, I would like to thank the faculty members of the department for maintaining a friendly and welcoming atmosphere on the one hand, and yet pushing for excellence on the other hand.

I also want to thank all those who encouraged me to continue to Doctoral studies. Especially, I thank my husband Arnold for his contagious enthusiasm for research, and my advisor Matya for constantly suggesting new intriguing problems to solve. I specifically remember one insightful conversation with my aunt Sarah, my mom's sister, who just said to me: "you should continue your studies for as long as you can". So I did, and I am grateful for that.

A special thanks to my family, for their unconditional love and boundless support. I deeply appreciate and thank my parents, Yael and Eli Naftali, for encouraging me to pursue my interests, whatever they were in each stage of my life.

There are no proper words to describe my gratitude and appreciation for my husband, Arnold Filtser, who is walking with me in (almost) the same path since we were in high-school together. Arnold, thank you for the love, care and support, and for being my best friend and an excellent colleague. It was a real pleasure to discuss research ideas in various (unconventional) times and locations.

Finally, I thank with love to my two daughters, Naama and Hadass, for their inspiring curiosity and joy of life.

Table of Contents

Abstract

Polygonal curves play an important role in many applied areas, such as 3D modeling in computer graphics, map matching in GIS, and protein backbone structural alignment and comparison in computational biology. Measuring the similarity of two curves in such applications is a challenging task, and various similarity measures have been suggested and investigated. The Fréchet distance is a useful and well-studied similarity measure that has been applied in many fields of research and applications.

The *Fréchet distance* is often described by an analogy of a man and a dog connected by a leash, each walking along a curve from its starting point to its end point. Both the man and the dog can control their speed but they are not allowed to backtrack. The Fréchet distance between the two curves is the minimum length of a leash that is sufficient for traversing both curves in this manner.

This research focuses on the *discrete Fréchet distance*, where, instead of continuous curves, we are given finite sequences of points, obtained, e.g., by sampling the continuous curves, or corresponding to the vertices of polygonal chains. Now, the man and the dog only hop monotonically along the sequences of points. The discrete Fréchet distance is considered a good approximation of the continuous distance, and is easier to compute. Much research has been done on the Fréchet distance, the majority of which considers only the continuous version. However, in some situations, the discrete Fréchet distance is more appropriate. For example, in the context of computational biology where each vertex of the chain represents an alpha-carbon atom, using the continuous Fréchet distance will result in mapping of arbitrary points, which is biologically meaningless.

This thesis contains two main parts, where in each part we study several problems with a common basic motivation.

In the first part we consider some real-world situations, in which the discrete Fréchet distance might not give a meaningful estimation of the resemblance between two curves. For example, when the input curves contain noise, or when they are not aligned with each other, the Fréchet distance may be much larger than the "true" value. Thus, in this part, we study other variants of Fréchet distance that are more meaningful in these situations, specifically, the discrete Fréchet distance with shortcuts, and the discrete Fréchet distance under translation. We also introduce a new variant of the Fréchet distance, which we call the discrete Fréchet gap. We believe that in some situations this new measure (and its variants) better reflects our intuitive notion of similarity.

In the second part, we deal with problems that arise from the constantly growing amounts of data, specifically trajectory data. When the input curves or chains are large, or when our data set contains a huge amount of trajectories, running time becomes a critical issue, and tools that enable fast calculations on the data are needed. First, we consider the nearest neighbor problem and the clustering problem for curves. These are two fundamental problems, where the input contains a large set of polygonal curves that need to be preprocessed or compressed in some way, such that certain information can be calculated efficiently. Then, we consider the simplification problem and the chain pair simplification problem. In these problems we are given only one or two input curves, but the number of points defining them is large. Thus, before we can perform calculations on them or visualize them, we must simplify them, without losing important features.

Chapter 1

Introduction

Polygonal curves play an important role in many applied areas, such as 3D modeling in computer graphics, map matching in GIS, and protein backbone structural alignment and comparison in computational biology. In such applications, the objects of interest are often modeled by their shape, and thus, an important step of the recognition process is to look for known shapes in an image. In many applications, two-dimensional shapes are given by the planar curves forming their boundaries. Consequently, a natural problem in shape comparison and recognition is to measure to what extent two given curves resemble each other. Naturally, the first question to be answered is what distance measure between curves should be used to reflect the intuitive notion of resemblance.

1.1 The Fréchet distance

Many methods are used to compare curves in these applications, and one of the most prevalent is the ***Fréchet distance*** [Fré06]. Other measures, such as the ***Hausdorff distance*** and ***RMSD*** (Root Mean Square Deviation) only take into account the sets of points on both curves but not the order in which they appear along the curves. For example, given two polygonal curves $A : [0, m] \to \mathbb{R}^d$ and $B : [0, n] \to \mathbb{R}^d$, the Hausdorff distance between them is defined as follows:

$$d_H(A, B) = \max\Big\{ \max_{x\in[0,m]}\{ \min_{y\in[0,n]} d(A(x), B(y))\},\ \max_{y\in[0,n]}\{ \min_{x\in[0,m]} d(A(x), B(y))\}\Big\}.$$

We use $d(a, b)$ to denote the Euclidean distance between two points a and b, but, depending on the application, other distance measures may be used. A polygonal curve $A : [0, m] \to \mathbb{R}^d$ consists of m line segments $\overline{a_i a_{i+1}}$ for each $i \in \{0, 1, \ldots, m-1\}$, where $a_i = A(i)$.

In Figure 1.1 we give an example of a pair of non-similar polygonal curves (in the Fréchet sense) such that the Hausdorff distance between them is small.

In order to overcome this discrepancy, one can use the Fréchet distance, which

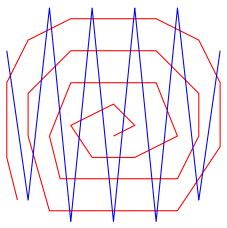
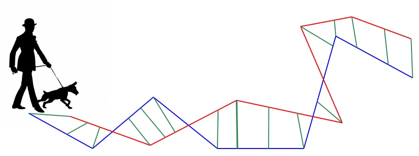


Figure 1.1: A pair of curves that are similar under the Hausdorff distance, since the order along the curves is not taken into account. The curves are not similar under the Fréchet distance.

was first defined by Maurice Fréchet (1878-1973). The Fréchet distance is generally described as follows: Consider a person and a dog connected by a leash, each walking along a curve from its starting point to its end point. Both can control their speed but they are not allowed to backtrack. The Fréchet distance between the two curves A and B, denoted by $d_F(A,B)$, is the minimum length of a leash that is sufficient for traversing both curves in this manner.



More formally, the Fréchet distance is usually defined as:

$$d_F(A,B) = \min_{\substack{\alpha:[0,1]\to[0,m] \\ \beta:[0,1]\to[0,n]}} \max_{t\in[0,1]} \left\{ d\Big(A(\alpha(t)),\, B(\beta(t)) \Big) \right\},$$

where α and β range over all continuous non-decreasing functions with $\alpha(0)=0$, $\alpha(1)=m$, $\beta(0)=0$, $\beta(1)=n$.

The ***discrete Fréchet distance*** (DFD for short) is a simpler variant that arises when one replaces each of the input curves by a sequence of sample points. When the sample is sufficiently dense, the resulting discrete distance is a good approximation of the actual continuous distance. We can view these sequences of points as polygonal curves or chains.

Intuitively, the discrete Fréchet distance replaces the curves by two sequences of points $A=(a_1,...,a_m)$ and $B=(b_1,...,b_n)$, and replaces the person and the dog by two frogs, the A-frog and the B-frog, initially placed at a_1 and b_1, respectively. At each move, the A-frog or the B-frog (or both) jumps from its current point to the next one. The frogs are not allowed to backtrack. We are interested in the minimum length of a leash that connects the frogs and allows the A-frog and the B-frog to get to a_m and b_n, respectively. More formally, for a given length δ of the leash, a jump is allowed only if the distances between the two frogs before and after the jump are both at most δ; the discrete Fréchet distance between A and B, denoted by $d_{dF}(A,B)$, is then the smallest $\delta>0$ for which there exists a sequence of jumps that brings the frogs to a_m and b_n, respectively.

There are several equivalent ways to formally define the discrete Fréchet distance. In each of the following chapters, we prefer a different definition, i.e., the one that is most convenient for our purposes.

Eiter and Mannila [EM94] showed that the discrete and continuous versions of the Fréchet distance relate to each other as follows:

$$d_F(A,B) \leq d_{dF}(A,B) \leq d_F(A,B) + \max\{D(A), D(B)\},$$

where D(A) is the length of the longest edge in A.

The Fréchet distance and the discrete Fréchet distance are used as similarity measures between curves and sampled curves, respectively, in many applications. Among these are speech recognition [KHM+98], signature verification [MP99], matching of time series in databases [KKS05], map-matching of vehicle tracking data [BPSW05, CDG+11, WSP06], and analysis of moving objects [BBG08, BBG+11]. While one can claim that the discrete Fréchet distance is only a good approximation of the continuous one, the use of discrete Fréchet distance, in many situations, makes more sense. For example, in computational biology, the discrete Fréchet distance was applied to protein backbone alignment [JXZ08]. In this application, each vertex represents an alpha-carbon atom. Applying the continuous Fréchet distance will cause mapping of arbitrary points, which is meaningless biologically.

1.2 Background and related work

The Fréchet distance and its variants have been studied extensively in the past two decades. For two polygonal curves, each of length n, Alt and Godau [AG95] showed that the Fréchet distance between them can be computed, using dynamic programming, in $O(n^2 \log n)$ time. Eiter and Mannila [EM94] showed that the discrete Fréchet distance can be computed, also using dynamic programming, in $O(n^2)$ time.

It has been an open problem to compute (exactly) the continuous or discrete Fréchet distance in subquadratic time. A lower bound of $\Omega(n \log n)$ was given for the problem of deciding whether the Fréchet distance between two curves is smaller than or equal to a given value (for both the continuous and discrete variants) [BBK+07]. Alt [Alt09] has conjectured that the decision problem of the (continuous) Fréchet distance problem is 3SUM-hard [GO95]. Buchin et al. [BBMM14] improved the bound of Alt and Godau by showing how to compute the Fréchet distance in $O(n^2(\log n)^{1/2}(\log\log n)^{3/2})$ time on a pointer machine, and in $O(n^2(\log\log n)^2)$ time on a word RAM. Agarwal et al. [AAKS14] showed how to compute the discrete Fréchet distance in $O\left(\frac{n^2 \log\log n}{\log n}\right)$ time. Bringmann [Bri14], and later Bringmann and Mulzer [BM16], presented a conditional lower bound implying that strongly subquadratic algorithms for the (discrete and continuous) Fréchet distance are unlikely to exist, even in the one-dimensional case and even if the solution may be approximated up to a factor of 1.399. Moreover, they present a linear-time greedy algorithm with approximation factor of $2^{O(n)}$, and an α-approximation algorithm that runs in time $O(n \log n + n^2/\alpha)$, for any $\alpha \in [1, n]$. Recently, Chan and Rahmati [CR18] improved this result by presenting an α-approximation algorithm for any $\alpha \in [1, \sqrt{n/\log n}]$ that runs in $O(n \log n + n^2/\alpha^2)$ time.

Given the apparent difficulty of achieving an efficient constant factor approximation algorithm for the Fréchet distance between two arbitrary polygonal curves, a natural direction is to develop algorithms for realistic scenes. Several restricted families of curves were considered in the literature in the context of Fréchet distance. Usually, these are curves that behave "nicely" and are assumed to be the input in practice. Alt et al. [AKW03] showed that for closed convex curves, the Fréchet distance equals the Hausdorff distance and hence the $O(n \log n)$ algorithm for the Hausdorff distance applies. They also showed that for *k-bounded curves* the Fréchet distance is at most $(1 + k)$ times the Hausdorff distance, which implies an $O(n \log n)$ time $(k + 1)$-approximation algorithm for the Fréchet distance. A planar curve P is called k-bounded for some real parameter $k \geq 1$, if for any two points x and y on P, the portion of P between x, y is contained in the union of the disks $D(x, \frac{k}{2}d(x, y))$ and $D(y, \frac{k}{2}d(x, y))$, where $D(p, r)$ is the disk with center at p and radius r. Aronov et al. [AHK$^+$06] have given a $(1 + \varepsilon)$-approximation algorithm for the discrete Fréchet distance between two *backbone curves* that runs in near linear time. Backbone curves are required to have edges with length in some fixed constant range, and a constant lower bound on the minimal distance between any pair of non-consecutive vertices; they model, e.g., the backbone chains of proteins. Driemel et al. [DHW12] studied the Fréchet distance of another family of curves, called *c-packed curves*. A curve P is c-packed if the total length of P inside any circle is bounded by c times the radius of the circle. Given two c-packed curves P and Q with total complexity n, a $(1 + \varepsilon)$-approximation of the Fréchet distance between them can be computed in $O(cn/\varepsilon + cn \log n)$ time.

In the standard Fréchet metric we consider polygonal curves. Rote [Rot07] considered the Fréchet distance between two curves which consists of a sequence of smooth curved pieces that are sufficiently well behaved, such as circular arcs or parabolic arcs. He showed that the Fréchet distance between two such curves can be computed in $O(n^2 \log n)$ time (n is the total size of the curves). The decision version of the problem can be solved in $O(n^2)$ time, which is the best known running time for polygonal curves.

Many variants of the Fréchet distance have been studied in the literature. For example, the ***weak Fréchet distance***, where the dog and its owner are allowed to backtrack. The weak Fréchet distance can be computed in $O(mn \log(mn))$ time

[AG95]. Another well-known variant is the ***Fréchet distance with shortcuts***, where the dog and its owner are allowed to skip parts of their respective polygonal curves. This variant is also used to reduce the impact of outliers, and it will be discussed in more detail in Chapter 2. Other examples are the *Fréchet distance with speed limits* [MSSZ11], where the speed of traversal along each segment of the curves is restricted to some specified range, and the *locally correct Fréchet matchings* [BBMS19] which aims at restricting the set of Fréchet matchings to "natural" matchings.

Another distance measure that is closely related to DFD is ***Dynamic Time Warping*** (DTW), which is defined between sequences of points rather than curves, and mainly used for analyzing time series. Here, instead of taking the smallest maximum distance between the frogs, we take the smallest sum of distances. Efrat et al. [EFV07] adapted the idea of DTW measure to compute an integral or summed version of the continuous Fréchet distance, and the average Fréchet distance was suggested in [BPSW05].

The Fréchet distance was also considered in different settings, for example, the *geodesic Fréchet distance* [IW08], where the curves reside in a space with obstacles, and the distance between two points is the length of the shortest obstacle-avoiding path between them. In the *homotopic Fréchet distance* [CdVE$^+$10], the leash cannot switch discontinuously from one position to another and cannot jump over obstacles. Ahn et al. [AKS$^+$12] considered a setting where the points of the polygonal curves are imprecise, i.e., each point could lie anywhere within a given region.

1.3 Contribution of this thesis

As demonstrated above, there is a growing body of research that is related to the Fréchet distance and its variants. In our research we focused on several problems that arise from real-world applications for curves, and which carry significant importance in facing the needs of the modern world. This thesis has two parts, each dealing with several different problems that share a similar basic motivation.

Part I: In a Search for a Meaningful Distance Measure

Part I aims to address the fact that the Fréchet and discrete Fréchet distances are not perfect measures, and in some real-world situations may not give a meaningful estimation of the extent to which two given curves resemble each other. For example, when the input curves contain noise, or when they are not aligned with respect to each other, the Fréchet distance may be much larger than the "true" value. Thus, in this part, we consider several other variants of Fréchet distance which are more suitable and meaningful in some situations.

1.3.1 The discrete Fréchet distance with shortcuts

In many of the application domains using the Fréchet distance, the curves or the sampled sequences of points are generated by physical sensors, such as GPS devices. These sensors may generate inaccurate measurements, which we refer to as *outliers*. Since the Fréchet distance is a bottleneck (min-max) measure, it is very sensitive to outliers, which may cause the Fréchet distance to be much larger than the distance without the outliers.

Several variants of the Fréchet distance better suited for handling outliers were suggested and studied in the literature, among them is the (continuous) Fréchet distance with shortcuts, where the dog is allowed to skip parts of its polygonal curve. In the continuous version, each skipped subcurve is replaced by a shortcut, i.e. a straight segment that connects its start and end points. The Fréchet distance with shortcuts is the Fréchet distance between the new curve with shortcuts and the other curve. This variant was introduced by Driemel and Har-Peled [DH13], who gave near-linear time approximation algorithm for the problem where shortcuts are allowed only between vertices of the curve, and the given polygonal curves are c-packed[1]. Buchin et al. [BDS14] considered a more general version of the Fréchet distance with shortcuts, where shortcuts are allowed between any pair of points of the noisy curve. They showed that this problem is NP-Hard, and gave a 3-approximation algorithm for the decision version of this problem that runs in $O(n^3 \log n)$ time.

In Chapter 2 we define and study several variants of the *discrete Fréchet distance with shortcuts*, where one of the frogs (or both frogs in another variant) may take shortcuts, i.e., skip points of the noise-containing sequence, which can be considered as outliers. When shortcuts are allowed only in one noise-containing curve, we give a randomized algorithm that runs in $O((m+n)^{6/5+\varepsilon})$ expected time, for any $\varepsilon > 0$. When shortcuts are allowed in both curves, we give an $O((m^{2/3}n^{2/3} + m + n)\log^3(m+n))$-time deterministic algorithm. We also consider the semi-continuous Fréchet distance with one-sided shortcuts, where we have a sequence of m points and a polygonal curve of n edges, and shortcuts are allowed only in the sequence. We show that this problem can be solved in randomized expected time $O((m+n)^{2/3}m^{2/3}n^{1/3}\log(m+n))$.

In contrast to the results regarding the continuous version, our results are somewhat surprising, as they demonstrate that both variants of the discrete Fréchet distance with shortcuts are easier to compute (exactly, with no restriction on the input) than all previously studied variants of the Fréchet distance.

This is a joint work with Rinat Ben Avraham, Haim Kaplan and Micha Sharir, that appeared in the International Symposium on Computational Geometry, 2014 (see [AFK$^+$14]). A full version of the paper appeared in ACM Transactions on Algorithms (see [AFK$^+$15]). In Chapter 2, we only describe the parts in which I was

[1]A curve P is c-packed if the total length of P inside any ball of radius r is at most cr.

involved and to which I have contributed.

1.3.2 The discrete Fréchet distance under translation

Another fundamental problem in many applications of the Fréchet distance, is that the input curves are not necessarily aligned, and one of them must undergo some transformation in order for the distance computation to be meaningful. Thus, an important variant of DFD is the *discrete Fréchet distance under translation*.

Ben Avraham et al. [AKS15] presented an $O(m^3n^2(1+\log(n/m))\log(m+n))$-time algorithm for DFD between two sequences of points of sizes m and n in the plane under translation. Assuming $m \leq n$, their idea is to construct an arrangement of disks of size $O(n^2m^2)$ and traverse its cells while updating reachability in a directed grid graph of size $O(nm)$, in $O(m(1+\log(n/m))$ time per update. Recently, Bringman et al. [BKN19] managed to improve the update time to $\tilde{O}(n^{2/3})$, and thus improved the running time to $\tilde{O}(n^{4.66\cdots})$. Moreover, they provide evidence that constructing the arrangement of size $O(n^2m^2)$ is unavoidable by proving a conditional lower bound of $n^{4-o(1)}$ on the running time of DFD under translation.

In Chapter 3 we consider two variants of DFD, both under translation. For DFD with shortcuts in the plane, we present an $O(m^2n^2\log^2(m+n))$-time algorithm, by presenting a dynamic data structure for reachability queries in the underlying directed graph. This algorithms can be generalized to any constant dimension $d \geq 1$. Notice that the running time of our algorithm for the shortcuts version is very close to the lower bound of the original version. For points in 1D, we show how to avoid the use of parametric search and remove a logarithmic factor from the running time of (the 1D versions of) these algorithms and of an algorithm for the weak discrete Fréchet distance; the resulting running times are thus $O(m^2n(1+\log(n/m)))$, for the discrete Fréchet distance, $O(mn\log(m+n))$, for the shortcuts variant, and $O(mn\log(m+n)(\log\log(m+n))^3)$ for the weak variant.

Our 1D algorithms follow a general scheme introduced by Martello et al. [MPTDW84] for the Balanced Optimization Problem (BOP), which is especially useful when an efficient dynamic version of the feasibility decider is available. We present an alternative scheme for BOP, whose advantage is that it yields efficient algorithms quite easily, without having to devise a specially tailored dynamic version of the feasibility decider. We demonstrate our scheme on the *most uniform path* problem (significantly improving the known bound), and observe that the weak discrete Fréchet distance under translation in 1D is a special case of it.

This work appeared in the Scandinavian Symposium and Workshops on Algorithm Theory, 2018 (see [FK18]).

1.3.3 The discrete Fréchet gap

In Chapter 4 we introduce the *(discrete) Fréchet gap* and its variants as an alternative measure of similarity between polygonal curves of size n. Referring to the frogs analogy, the discrete Fréchet gap is the minimum difference between the longest and shortest positions of the leash needed for the frogs to traverse their point sequences. For handling outliers, we suggest the *one-sided discrete Fréchet gap with shortcuts* variant, where the frog can skip points of its chain. We believe that in some situations this new measure (and its variants) better reflects our intuitive notion of similarity, since the familiar (discrete) Fréchet distance (and its variants) is indifferent to (matched) pairs of points that are relatively close to each other.

We show an interesting connection between the discrete Fréchet gap and DFD under translation, studied in Chapter 3. More precisely, the shortcuts and the weak versions of DFD, both in 1D under translation, are in some sense analogous to their respective gap variants (in d dimensions and no translation): we can use (almost) similar algorithms to compute them. Notice that the number of potential values for the discrete Fréchet gap is $O(n^4)$, while it is only $O(n^2)$ for the discrete Fréchet distance. Yet our algorithms for the gap variants are much faster, and run in $O(m^2n(1+\log(n/m)))$ for the discrete Fréchet gap, $O(mn\log(m+n))$, for the shortcuts variant, and $O(mn\log(m+n)(\log\log(m+n))^3)$ for the weak variant.

This work (partially) appears in a manuscript published on ArXiv (see [FK15]), and in the Scandinavian Symposium and Workshops on Algorithm Theory, 2018 (see [FK18]).

Part II: Dealing with Big (Trajectory) Data

Part II deals with problems that arise from the constantly growing amounts of data, specifically trajectory data. When the input curves or chains are large, or when our data set contains a huge amount of trajectories, running time becomes a critical issue, and we have to develop tools that allow fast calculations on the data. In this part we consider several different problems that are motivated by the need to handle big data. In Chapters 5 and 6, we consider two fundamental problems where the input contains a large set of polygonal curves that need to be preprocessed or compressed in some way such that certain information can be calculated efficiently. In Chapters 7 and 8, we consider problems where there are only one or two input curves, but the number of points defining them is large. In these cases running time becomes critical, and visualizing or applying calculations on just one curve without losing valuable properties is a more difficult task.

1.3.4 Nearest neighbor search and clustering for curves

Nearest neighbor search is a fundamental problem in computer science, and significant progress on this problem has been in the past couple of decades. This important task also arises in applications where the recorded instances are trajectories or polygonal curves, however, most research has focused on sets of points. In the *nearest neighbor problem for curves*, the goal is to construct a compact data structure over a set $\mathcal{C}$ of n input curves, each of length at most m, such that given a query curve Q of length m, one can efficiently find the curve from $\mathcal{C}$ closest to Q.

Dreimel and Silvestri [DS17] show that unless the orthogonal vectors hypothesis fails, there exists no data structure for nearest neighbor under the (discrete or continuous) Fréchet distance that can be built in $O(n^{2-\varepsilon} poly(m))$ time and has query time in $O(n^{1-\varepsilon} poly(m))$, for any $\varepsilon > 0$. Thus, we look for more relaxed variants of the problem that can be solved efficiently. Our first direction is to investigate the approximate nearest neighbor problem under the discrete Fréchet distance (and also other closely related measures). Several methods were used in previous research of the problem, each leading to not very satisfactory results [Ind02, DS17, EP18]. The most recent result was presented by Emiris and Psarros [EP18], providing approximation factor of $(1+\varepsilon)$, with space complexity in $\tilde{O}(n) \cdot (2 + \frac{d}{\log m})^{O(m^{1/\varepsilon} \cdot d \log(1/\varepsilon))}$ and query time in $O(d \cdot 2^{2m} \log n)$ (for curves in d dimensions). In Chapter 5, we present an algorithm based on a discretization of the space, which is simple and deterministic. Yet, somewhat surprisingly, our algorithm is more efficient than all previous results: we still give an approximation factor of $(1+\varepsilon)$, but with space complexity in $n \cdot O(\frac{1}{\varepsilon})^{md}$ and query time in $O(md \log(\frac{mnd}{\varepsilon}))$.

However, its was shown in [IM04, DKS16] that unless the strong exponential time hypothesis fails, nearest neighbor problem under DFD is hard to approximate within a factor of $c < 3$, with a data structure requiring $O(n^{2-\varepsilon} \operatorname{polylog} m)$ preprocessing and $O(n^{1-\varepsilon} \operatorname{polylog} m)$ query time for $\varepsilon > 0$. Our approximation data structure uses space and query time exponential in m, which makes it impractical for large curves. Therefore, in our second direction (presented in Chapter 6), we identify two important cases for which it is possible to obtain practical bounds for the nearest neighbor problem, even when m and n are large. In these cases, either Q is a line segment or $\mathcal{C}$ consists of line segments, and the bounds on the size of the data structure and query time are nearly linear in the size of the input and query curve, respectively. The returned answer is either exact under L_∞, or approximated to within a factor of $1+\varepsilon$ under L_2. We also consider the variants in which the location of the input curves is only fixed up to translation, and obtain similar bounds, under L_∞.

Clustering is another fundamental problem in data analysis that aims to partition an input collection of curves $\mathcal{C}$ into clusters where the curves within each cluster are similar in some sense. In the center problem, the goal is to find a curve Q, such that the maximum distance between Q and the curves in $\mathcal{C}$ is minimized. Driemel et al. [DKS16] introduced the (k,ℓ)-Center problem, where the k desired center curves are limited to at most ℓ vertices each. In the case of the (k,ℓ)-Center problem under the discrete Fréchet distance, Driemel et al. showed that the problem is NP-hard to approximate within a factor of $2-\varepsilon$ when k is part of the input, even if $\ell = 2$ and $d = 1$. Furthermore, the problem is NP-hard to approximate within a factor $2-\varepsilon$ when ℓ is part of the input, even if $k = 2$ and $d = 1$, and when $d = 2$ the inapproximability bound is $3\sin\pi/3 \approx 2.598$ [BDG+19]. Again, the above results imply that algorithms for the (k,ℓ)-Center problem that achieve efficient running times are not realistic. Thus, in Chapter 6 we focus on a specific important settings, where the center is a line segment, i.e., we seek the line segment that represents the given set as well as possible. We present near-linear time exact algorithms under L_∞, even when the location of the input curves is only fixed up to translation. Under L_2, we present a roughly $O(n^2m^3)$-time exact algorithm.

The results presented in Chapter 5 were obtained in a joint work with Arnold Filtser, and appeared in ArXiv, 2019 (see [FFK19]). The results presented in Chapter 5 are a joint work with Boris Aronov, Michael Horton, and Khadijeh Sheikhan.

1.3.5 Simplifying chains under the discrete Fréchet distance

Many real-world applications have to deal with very large chains, which makes the processing time a critical issue. A natural approach is to process another simpler chain, that is a good approximation of the original one. For instance, many GPS applications use trajectories that are represented by sequences of densely sampled points, which we want to simplify in order to perform efficient calculations.

In Chapter 7, we discuss the simplification problem. Here, given some chain A of length n, the goal is to find a smaller chain A' which is as similar as possible to the original chain A. First we present an $O(n^2 \log n)$-time algorithm for the, so called, Min-δ Fitting with k-Chain Simplification problem, presented in [BJW+08], improving their $O(n^3)$ time algorithm. Then we show how to adapt the techniques of [DH13] to achieve an approximate simplification under the discrete Fréchet distance. Following [DH13], we present a collection of data structures for discrete Fréchet distance queries, and then show how to use it to preprocess a chain in near-linear time and space, such that given a number k, one can compute a simplification in $O(k)$ time which has $K = 2k-1$ vertices (of the original chain) and is optimal up to a constant factor with respect to the discrete Fréchet distance, compared to any chain of k arbitrary vertices.

This work appeared in Information Processing Letters, 2018 (see [Fil18]).

When polygonal chains are large, it is difficult to efficiently compute and visualize the structural resemblance between them. Simplifying two aligned chains independently does not necessarily preserve the resemblance between the chains. This problem in the context of protein backbone comparison has led Bereg et al. [BJW+08] to pose the Chain Pair Simplification problem (CPS). In this problem, the goal is to simplify both chains simultaneously, so that the discrete Fréchet distance between the resulting simplifications is bounded. More precisely, given two chains A and B, one needs to find two simplifications A',B' with vertices from A,B, respectively, such that the discrete Fréchet distance between A, A', B, B', and A', B' is small.

When the chains are simplified using the Hausdorff distance instead of DFD, the problem becomes **NP**-complete. However, the complexity of the version that uses DFD has been open since 2008. In Chapter 8 we introduce the weighted chain pair simplification problem and prove that the weighted version using DFD is weakly **NP**-complete. Then, we resolve the question concerning the complexity of CPS under the discrete Fréchet distance by proving that it is polynomially solvable, contrary to what was believed. Moreover, we devise a sophisticated $O(m^2n^2 \min\{m, n\})$-time dynamic programming algorithm for the minimization version of the problem. Besides being interesting from a theoretical point of view, only after developing (and implementing) this algorithm, were we able to apply the minimization problem to datasets from the Protein Data Bank (PDB). In addition, we study several less rigid variants of the problem.

Next, we consider for the first time the problem where the vertices of the simplifications A', B' may be arbitrary points, i.e., they are not necessarily from A, B, respectively. Since this problem is more general, we call it General CPS, or GCPS for short. Our main contribution, is a (relatively) efficient polynomial-time algorithm for GCPS, and a more efficient 2-approximation algorithm for the problem. We also investigated GCPS under the Hausdorff distance, showing that it is **NP**-complete and presented an approximation algorithm for the problem.

These results led to two papers: the first is a joint work with Chenglin Fan, Tim Wylie and Binhai Zhu, which appeared in the International Symposium on Algorithms and Data Structures, 2015 (see [FFK+15]), and the second is a joint work with Chenglin Fan and Binhai Zhu, which appeared in the International Symposium on Mathematical Foundations of Computer Science, 2016 (see [FFKZ16]).

In Search for a Meaningful Distance Measure

Chapter 2

The Discrete Fréchet Distance with Shortcuts

2.1 Introduction

In many of the application domains using the Fréchet distance, the curves or the sampled sequences of points are generated by physical sensors, such as GPS. These sensors may generate inaccurate measurements, which we refer to as *outliers*. The Fréchet distance and the discrete Fréchet distance are bottleneck (min-max) measures, and are therefore sensitive to outliers, and may fail to capture the similarity between the curves when there are outliers, because the large distance from an outlier to the other curve might determine the Fréchet distance, making it much larger than the distance without the outliers.

In order to handle outliers, Driemel and Har-Peled [DH13] introduced the (continuous) Fréchet distance with shortcuts. They considered piecewise linear curves and allowed (only) the dog to take shortcuts by walking from a vertex v to any succeeding vertex w along the straight segment connecting v and w. This "one-sided" variant allows one to "ignore" subcurves of one (noisy) curve that substantially deviate from the other (more reliable) curve. Driemel and Har-Peled gave efficient approximation algorithms for the Fréchet distance in such scenarios; these are reviewed in more detail later on.

Driven by the same motivation of reducing sensitivity to outliers, we define two variants of the discrete Fréchet distance with shortcuts. In the one-sided variant, we allow the A-frog to jump to any point that comes later in its sequence, rather than just to the next point. The B frog has to visit all the B points in order, as in the standard discrete Fréchet distance problem. However, we add the restriction that only a single frog is allowed to jump in each move (see below for more details). As in the standard discrete Fréchet distance, for a leash of length δ such a jump is allowed only if the distances between the two frogs before and after the jump are both at most δ. The ***one-sided discrete Fréchet distance with shortcuts***, denoted as

$d^{-}_{dF}(A, B)$, is the smallest $\delta > 0$ for which there exists such a sequence of jumps that brings the frogs to a_m and b_n, respectively.

We also define the ***two-sided discrete Fréchet distance with shortcuts***, denoted as $d^{+}_{dF}(A, B)$, to be the smallest $\delta > 0$ for which there exists a sequence of jumps, where both frogs are allowed to skip points as long as the distances between the two frogs before and after the jump are both at most δ. Here too, we allow only one of the frogs to jump at each move.

In the (standard) discrete Fréchet distance, the frogs can make simultaneous jumps, each to its next point. Here though simultaneous jumps make the problem degenerate as it is possible for the frogs to jump from a_1 and b_1 straight to a_m and b_n (in the two-sided scenario). The one-sided version can easily be extended to the case where simultaneous jumps are allowed, but, to keep the description simple, we describe here only the case where such simultaneous jumps are not allowed.

Our results. In a joint work with Rinat Ben Avraham, Haim Kaplan and Micha Sharir (See [AFK$^+$14]), we give efficient algorithms for computing the discrete Fréchet distance with one-sided and two-sided shortcuts. The structure of the one-sided problem allows us to decide whether the distance is no larger than a given δ, in $O(n + m)$ time, and the challenge is to search for the optimum, using this fast decision procedure, with a small overhead. The naive approach would be to use the $O((m^{2/3}n^{2/3} + m + n) \log(m + n))$-time distance selection procedure of [KS97], which would make the running time $\Omega((m^{2/3}n^{2/3} + m + n) \log(m + n))$, much higher than the linear cost of the decision procedure.

To tighten this gap, we develop an algorithm that, given an interval $(\alpha, \beta]$ and a parameter L, decides, with high probability and in $O((m+n)^{4/3+\varepsilon}/L^{1/3}+m+n)$ time, whether the number of pairs in $A \times B$ of distance in $(\alpha, \beta]$ is at most L. Furthermore, if this number is larger than L, our algorithm provides a sample of these pairs, of logarithmic size, that contains, with high probability, a pair at approximate median distance (in the middle three quarters of the distances in $(\alpha, \beta]$). We combine this algorithm with a binary search to obtain a procedure that produces an interval that contains the optimal distance as well as at most L other distances. Finally we use the decision procedure in order to find the optimal value among these L remaining distances in $O(L(m+n))$ time. As L increases, the first stage becomes faster and the second stage becomes slower. Choosing L to balance the two gives us an algorithm for the one-sided version that runs in $O((m + n)^{5/4+\varepsilon})$ time for any $\varepsilon > 0$.

In [AFK$^+$14] a more sophisticated technique is given in addition, that again uses the decision procedure in order to find the optimal value among these L remaining distances in $O((m + n)L^{1/2} \log(m + n))$ time. Choosing the optimal L yields an algorithm that runs in $O((m + n)^{6/5+\varepsilon})$ time for any $\varepsilon > 0$.

We also use the above algorithm to solve the semi-continuous version of the one-sided Fréchet distance with shortcuts in a similar manner. In this problem A is a sequence of m points and $f \subseteq \mathbb{R}^2$ is a polygonal curve of n edges. A frog has to jump over the points in A, connected by a leash to a person who walks on f. The frog can make shortcuts and skip points, but the person must traverse f continuously. The frog and the person cannot backtrack. We want to compute the minimum length of a leash that allows the frog and the person to get to their final positions in such a scenario. In Section 2.6 we give an overview of an algorithm that runs in $O((m+n)^{2/3}m^{2/3}n^{1/3}\log(m+n))$ expected time for this problem. While less efficient than the fully discrete version, it is still significantly subquadratic.

For the two-sided version we take a different approach. More specifically, we implement the decision procedure by using an implicit compact representation of all pairs in $A \times B$ at distance at most δ as the disjoint union of complete bipartite cliques [KS97]. This representation allows us to maintain the pairs reachable by the frogs with a leash of length at most δ implicitly and efficiently. The cost of the decision procedure is $O((m^{2/3}n^{2/3}+m+n)\log^2(m+n))$, which is comparable with the cost of the distance selection procedure of [KS97], as mentioned above. We can then run a binary search for the optimal distance, using this distance selection procedure. The resulting algorithm runs in $O((m^{2/3}n^{2/3}+m+n)\log^3(m+n))$ time and requires $O((m^{2/3}n^{2/3}+m+n)\log(m+n))$ space.

Interestingly, the algorithms developed for these variants of the discrete Fréchet distance problem are sublinear in the size of $A \times B$ and way below the slightly subquadratic bound for the discrete Fréchet distance, obtained in [AAKS14].

In principle, the algorithm for the one-sided Fréchet distance with shortcuts can be generalized to work in higher dimensions. More details are given in the full version of the paper [AFK$^+$14].

Related work. As already noted, the (one-sided) continuous Fréchet distance with shortcuts was first studied by Driemel and Har-Peled [DH13]. They considered the problem where shortcuts are allowed only between *vertices* of the noise-containing curve, in the manner outlined above, and gave approximation algorithms for solving two variants of this problem. In the first variant, any number of shortcuts is allowed, and in the second variant, the number of allowed shortcuts is at most k, for some $k \in \mathbb{N}$. Their algorithms work efficiently only for *c-packed* polygonal curves. Both algorithms compute a $(3+\varepsilon)$-approximation of the Fréchet distance with shortcuts between two c-packed polygonal curves and both run in near-linear time (ignoring the dependence on ε). Buchin et al. [BDS14] consider a more general version of the (one-sided) continuous Fréchet distance with shortcuts, where shortcuts are allowed between any pair of points of the noise-containing curve. They show that this problem is NP-Hard. They also give a 3-approximation algorithm for the decision version of this problem that runs in $O(n^3 \log n)$ time.

In contrast with the results just reviewed, our results are somewhat surprising, as they demonstrate that both variants of the discrete Fréchet distance with shortcuts are easier to compute (exactly, with no restriction on the input) than all previously studied variants of the Fréchet distance.

We also note that there have been several other works that treat outliers in different ways. One such result is of Buchin et al. [BBW09], who considered the partial Fréchet similarity problem, where one is given two curves f and g, and a distance threshold δ, and the goal is to maximize the total length of the portions of f and g that are matched (using the Fréchet distance paradigm) with L_p-distance smaller than δ. They gave an algorithm that solves this problem in $O(mn(m+n)\log(mn))$ time, under the L_1 or L_∞ norm. The definition of the partial Fréchet similarity aims at situations where the extent of a pre-required similarity is known (and given by the distance threshold δ), and we wish to know how much (and which parts) of the curves are similar to this extent. The definition of the (one-sided and two-sided) Fréchet with shortcuts is practically used in cases where we have a pre-assumption that the curves are similar, up to the existence of (not too many) outliers, and we want to estimate the magnitude of this similarity, eliminating the outliers. Since we assume that the points are sampled along curves that we want to match, our algorithms are applicable to any scenario in which the continuous Fréchet with shortcuts is applicable. Practical implementations of Fréchet distance algorithms that are made for experiments on real data in map matching applications, remove outliers from the data set [CDG+11, WSP06]. In another map matching application, Brakatsoulas et al. [BPSW05] define the notion of integral Fréchet distance to deal with outliers. This distance measure averages over certain distances instead of taking the maximum. Bereg et al. [BJW+08] and then Wylie and Zho [WZ13] considered the discrete Fréchet distance in biological context, for protein (backbone) structure alignment and comparison. They use pair simplification of the protein backbones, that can be interpreted as making shortcuts while comparing them under the discrete Fréchet distance.

2.2 Preliminaries

A formal definition of the discrete Fréchet distance was given in Section 1.1. However, in this chapter we prefer to use a an equivalent graph-based formal definition of the discrete Fréchet distance and its variants with shortcuts.

Let $A = (a_1, \ldots, a_m)$ and $B = (b_1, \ldots, b_n)$ be two sequences of m and n points, respectively, in the plane. Let $G(V, E)$ denote a graph whose vertex set is V and edge set is E, and let $\|\cdot\|$ denote the Euclidean norm. Fix a distance $\delta > 0$, and define the following three directed graphs $G_\delta = G(A \times B, E_\delta)$, $G_\delta^- = G(A \times B, E_\delta^-)$, and $G_\delta^+ = G(A \times B, E_\delta^+)$, where

$$\begin{aligned}
E_\delta = &\Big\{ \Big((a_i, b_j), (a_{i+1}, b_j) \Big) \Bigm| \|a_i - b_j\|,\ \|a_{i+1} - b_j\| \le \delta \Big\} \bigcup \\
&\Big\{ \Big((a_i, b_j), (a_i, b_{j+1}) \Big) \Bigm| \|a_i - b_j\|,\ \|a_i - b_{j+1}\| \le \delta \Big\}, \\
E_\delta^- = &\Big\{ \Big((a_i, b_j), (a_k, b_j) \Big) \Bigm| k > i,\ \|a_i - b_j\|,\ \|a_k - b_j\| \le \delta \Big\} \bigcup \\
&\Big\{ \Big((a_i, b_j), (a_i, b_{j+1}) \Big) \Bigm| \|a_i - b_j\|,\ \|a_i - b_{j+1}\| \le \delta \Big\}, \\
E_\delta^+ = &\Big\{ \Big((a_i, b_j), (a_k, b_j) \Big) \Bigm| k > i,\ \|a_i - b_j\|,\ \|a_k - b_j\| \le \delta \Big\} \bigcup \\
&\Big\{ \Big((a_i, b_j), (a_i, b_l) \Big) \Bigm| l > j,\ \|a_i - b_j\|,\ \|a_i - b_l\| \le \delta \Big\}.
\end{aligned}$$

For each of these graphs we say that a position (a_i, b_j) is a *reachable* position if (a_i, b_j) is reachable from (a_1, b_1) in the respective graph.

Then the discrete Fréchet distance $d_{dF}(A, B)$ is the smallest $\delta \ge 0$ for which (a_m, b_n) is a reachable position in G_δ.

Similarly, the one-sided Fréchet distance with shortcuts (one-sided DFDS for short) $d_{dF}^-(A, B)$ is the smallest $\delta \ge 0$ for which (a_m, b_n) is a reachable position in G_δ^-, and the two-sided Fréchet distance with shortcuts (two-sided DFDS for short) $d_{dF}^+(A, B)$ is the smallest $\delta > 0$ for which (a_m, b_n) is a reachable position in G_δ^+.

2.3 Decision algorithm for the one-sided DFDS

We first consider the corresponding decision problem. That is, given a value $\delta \ge 0$, we wish to decide whether $d_{dF}^-(A, B) \le \delta$ (we ignore the issue of discrimination between the cases of strict inequality and equality, in the decision procedures of both the one-sided variant and the two-sided variant, since this will be handled in the optimization procedures, described later).

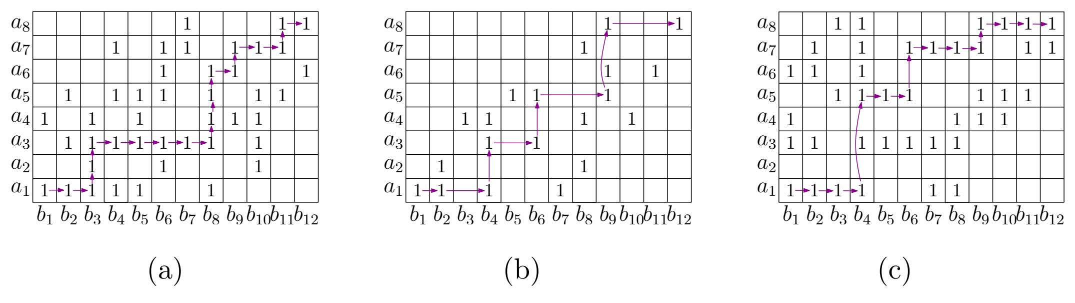

a_8 a_7 a_6 a_5 a_4 a_3 a_2 a_1
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12}
(a)
(b)
(c)

Figure 2.1: (a) A right-upward staircase (for DFD with no simultaneous jumps). (b) A semi-sparse staircase (for the one-sided DFDS). (c) A sparse staircase (for the two-sided DFDS).

Let M be the matrix whose rows correspond to the elements of A and whose columns correspond to the elements of B, and $M_{i,j} = 1$ if $\|a_i - b_j\| \le \delta$, and $M_{i,j} = 0$ otherwise. Consider first the DFD variant (no shortcuts allowed), in which, at each move, exactly one of the frogs has to jump to the next point. Suppose that (a_i, b_j)

is a reachable position of the frogs. Then, necessarily, $M_{i,j} = 1$. If $M_{i+1,j} = 1$ then the next move can be an ***upward move*** in which the A-frog moves from a_i to a_{i+1}, and if $M_{i,j+1} = 1$ then the next move can be a ***right move*** in which the B-frog moves from b_j to b_{j+1}. It follows that to determine whether $d_{dF}(A,B) \le \delta$, we need to determine whether there is a ***right-upward staircase*** of ones in M that starts at $M_{1,1}$, ends at $M_{m,n}$, and consists of a sequence of interweaving upward moves and right moves (see Figure 2.1(a)).

In the one-sided version of DFDS, given a reachable position (a_i, b_j) of the frogs, the A-frog can move to any point $a_k, k > i$, for which $M_{k,j} = 1$; this is a ***skipping upward move*** in M which starts at $M_{i,j} = 1$, skips over $M_{i+1,j}, \ldots, M_{k-1,j}$ (some of which may be 0), and reaches $M_{k,j} = 1$. However, in this variant, as in the DFD variant, the B-frog can only make a consecutive right move from b_j to b_{j+1}, provided that $M_{i,j+1} = 1$ (otherwise no move of the B-frog is possible at this position). Determining whether $d^-_{dF}(A,B) \le \delta$ corresponds to deciding whether there is a ***semi-sparse staircase*** of ones in M that starts at $M_{1,1}$, ends at $M_{m,n}$, and consists of an interweaving sequence of skipping upward moves and (consecutive) right moves (see Figure 2.1(b)).

Assume that $M_{1,1} = 1$ and $M_{m,n} = 1$; otherwise, we can immediately conclude that $d^-_{dF}(A,B) > \delta$ and terminate the decision procedure. From now on, whenever we refer to a semi-sparse staircase, we mean a semi-sparse staircase of ones in M starting at $M_{1,1}$, as defined above, but without the requirement that it ends at $M_{m,n}$.

Algorithm 2.1 Decision procedure for the one-sided discrete Fréchet distance with shortcuts.

```
• $S \leftarrow \langle M_{1,1} \rangle$
• $i \leftarrow 1$, $j \leftarrow 1$
• While ($i < m$ or $j < n$) do
    – If (a right move is possible) then
        ∗ Make a right move and add position $M_{i,j+1}$ to $S$
        ∗ $j \leftarrow j + 1$
    – Else
        ∗ If (a skipping upward move is possible) then
            · Move upwards to the first (i.e., lowest) position $M_{k,j}$, with $k > i$ and $M_{k,j} = 1$,
              and add $M_{k,j}$ to $S$
            · $i \leftarrow k$
        ∗ Else
            · Return $d^-_{dF}(A,B) > \delta$
• Return $d^-_{dF}(A,B) \le \delta$
```

Algorithm 2.1 constructs a semi-sparse staircase S by always making a right move if possible. The correctness of the decision procedure is established by the following lemma.

Lemma 2.1. *If there exists a semi-sparse staircase that ends at $M_{m,n}$, then S also ends at $M_{m,n}$. Hence S ends at $M_{m,n}$ if and only if $d^-_{dF}(A,B) \le \delta$.*

Proof. Let S' be a semi-sparse staircase that ends at $M_{m,n}$. We think of S' as a sequence of possible positions (i.e., 1-entries) in M. Note that S' has at least one position in each column of M, since skipping is not allowed when moving rightwards. We claim that for each position $M_{k,j}$ in S', there exists a position $M_{i,j}$ in S, such that $i \le k$. This, in particular, implies that S reaches the last column. If S reaches the last column, we can continue it and reach $M_{m,n}$ by a sequence of skipping upward moves (or just by one such move), so the lemma follows.

We prove the claim by induction on j. It clearly holds for $j = 1$ as both S and S' start at $M_{1,1}$. We assume then that the claim holds for $j = \ell - 1$, and establish it for ℓ. That is, assume that if S' contains an entry $M_{k,\ell-1}$, then S contains $M_{i,\ell-1}$ for some $i \le k$. Let $M_{k',\ell}$ be the lowest position of S' in column ℓ; clearly, $k' \ge k$. We must have $M_{k',\ell-1} = 1$ (as the only way to move from a column to the next is by a right move). If $M_{i,\ell} = 1$ then $M_{i,\ell}$ is added to S by making a right move, and $i \le k \le k'$ as required. Otherwise, S is extended by a sequence of skipping upward moves in column $\ell - 1$ followed by a right move between $M_{i',\ell-1}$ and $M_{i',\ell}$ where i' is the smallest index $\ge i$ for which both $M_{i',\ell-1}$ and $M_{i',\ell}$ are one. But since $i \le k'$ and $M_{k',\ell-1}$ and $M_{k',\ell}$ are both 1, we get that $i' \le k'$, as required. □

Running time. The entries of M that the decision procedure tests form a row- and column-monotone path, with an additional entry to the right for each upward turn of the path. (This also takes into account the 0-entries of M that are inspected during a skipping upward move.) Therefore it runs in $O(m + n)$ time.

2.4 One-sided DFDS via approximate distance counting and selection

We now show how to use the decision procedure of Algorithm 2.1 to solve the optimization problem of the one-sided discrete Fréchet distance with shortcuts. This is based on the algorithm provided in Lemma 2.2 given below.

First note that if we increase δ continuously, the set of 1-entries of M can only grow, and this can only happen when δ is a distance between a point of A and a point of B. Performing a binary search over the $O(mn)$ pairwise distances of pairs in $A \times B$ can be done using the distance selection algorithm of [KS97]. This will be the method of choice for the two-sided DFDS problem, treated in Section 2.5. Here however, this procedure, which takes $O(m^{2/3}n^{2/3}\log^3(m + n))$ time is rather excessive when compared to the linear cost of the decision procedure. While solving the optimization problem in close to linear time is still a challenging open problem, we manage to improve the running time considerably, to $O((m + n)^{5/4+\varepsilon})$, for any $\varepsilon > 0$.

Lemma 2.2. *Given a set A of m points and a set B of n points in the plane, an interval $(\alpha, \beta] \subset \mathbb{R}$, and parameters $0 < L \leq mn$ and $\varepsilon > 0$, we can determine, with high probability, whether $(\alpha, \beta]$ contains at most L distances between pairs in $A \times B$. If $(\alpha, \beta]$ contains more than L such distances, we return a sample of $O(\log(m+n))$ pairs, so that, with high probability, at least one of these pairs determines an approximate median (in the middle three quarters) of the pairwise distances that lie in $(\alpha, \beta]$. Our algorithm runs in $O((m+n)^{4/3+\varepsilon}/L^{1/3} + m + n)$ time and uses $O((m+n)^{4/3+\varepsilon}/L^{1/3} + m + n)$ space.*

The proof of Lemma 2.2 can be found in [AFK$^+$14]. We believe that this technique is of independent interest, beyond the scope of computing the one-sided Fréchet distance with shortcuts, and that it may be applicable to other optimization problems over pairwise distances.

The way it is described, the algorithm does not verify that the samples that it returns satisfy the desired properties, nor does it verify that the number of distances in $(\alpha, \beta]$ is indeed at most L, when it makes this assertion. As such, the running time is deterministic, and the algorithm succeeds with high probability (which can be calibrated by the choice of the constants c_1, c_2). See below for another comment regarding this issue.

We use the procedure provided by Lemma 2.2 to find an interval $(\alpha, \beta]$ that contains at most L distances between pairs of $A \times B$, including $d^-_{dF}(A, B)$. We find this interval using binary search, starting with $(\alpha, \beta] = (0, \infty)$, say. In each step of the search, we run the algorithm of Lemma 2.2. If it determines that the number of critical distances in $(\alpha, \beta]$ is at most L we stop. (The concrete choice of L that we will use is given later.) Otherwise, the algorithm returns a random sample R that contains, with high probability, an approximate median (in the middle three quarters) of the distances in $(\alpha, \beta]$. We then find two consecutive distances α', β' in R such that $d^-_{dF}(A, B) \in (\alpha', \beta']$, using the decision procedure (see Algorithm 2.1). $(\alpha', \beta']$ is a subinterval of $(\alpha, \beta]$ that contains, with high probability, at most $7/8$ of the distances in $(\alpha, \beta]$. We then proceed to the next step of the binary search, applying again the algorithm of Lemma 2.2 to the new interval $(\alpha', \beta']$. The resulting algorithm runs in $O((m+n)^{4/3+\varepsilon}/L^{1/3} + (m+n)\log(m+n))$ time, for any $\varepsilon > 0$.

Once we have narrowed down the interval $(\alpha, \beta]$, so that it now contains at most L distances between pairs of $A \times B$, including $d^-_{dF}(A, B)$, we can find $d^-_{dF}(A, B)$ by simulating the execution of the decision procedure at the unknown $d^-_{dF}(A, B)$. A simple way of doing this is as follows. To determine whether $M_{i,j} = 1$ at $d^-_{dF}(A, B)$, we compute the critical distance $r' = \|a_i - b_j\|$ at which $M_{i,j}$ becomes 1. If $r' \leq \alpha$ then $M_{i,j} = 0$, and if $r' \geq \beta$ then $M_{i,j} = 1$. Otherwise, $\alpha < r' < \beta$ is one of the at most L distances in $(\alpha, \beta]$. In this case we run the decision procedure at r' to determine $M_{i,j}$. Since there are at most L distances in $(\alpha, \beta]$, the total running time is $O(L(m+n))$. By picking $L = (m+n)^{1/4+\varepsilon}$ for another, but still arbitrarily small

$\varepsilon > 0$, we balance the bounds of $O((m+n)^{4/3+\varepsilon}/L^{1/3} + (m+n)\log(m+n))$ and $O(L(m+n))$, and obtain the bound of $O((m+n)^{5/4+\varepsilon})$, for any $\varepsilon > 0$, on the overall running time.

Although this significantly improves the naive implementation mentioned earlier, it suffers from the weakness that it has to run the decision procedure separately for each distance in $(\alpha, \beta]$ that we encounter during the simulation. In [AFK+14] we show how to accumulate several unknown distances and resolve them all using a binary search that is guided by the decision procedure. This allows us to find $d^-_{dF}(A,B)$ within the interval $(\alpha,\beta]$ more efficiently, in $O((m+n)L^{1/2}\log(m+n))$ time. Choosing the optimal L yields an algorithm that runs in $O((m+n)^{6/5+\varepsilon})$ time for any $\varepsilon > 0$. Details can be found in the full version of the paper [AFK+14].

Theorem 2.3. *Given a set A of m points and a set B of n points in the plane, and a parameter $\varepsilon > 0$, we can compute the one-sided discrete Fréchet distance $d^-_{dF}(A,B)$ with shortcuts in randomized expected time $O((m+n)^{6/5+\varepsilon})$ using $O((m+n)^{6/5+\varepsilon})$ space.*

2.5 The two-sided DFDS

We first consider the corresponding decision problem. That is, given $\delta > 0$, we wish to decide whether $d^+_{dF}(A,B) \le \delta$.

Consider the matrix M as defined in Section 2.3. In the two-sided version of DFDS, given a reachable position (a_i, b_j) of the frogs, the A-frog can make a skipping upward move, as in the one-sided variant, to any point $a_k, k > i$, for which $M_{k,j} = 1$. Alternatively, the B-frog can jump to any point $b_l, l > j$, for which $M_{i,l} = 1$; this is a ***skipping right move*** in M from $M_{i,j} = 1$ to $M_{i,l} = 1$, defined analogously. Determining whether $d^+_{dF}(A,B) \le \delta$ corresponds to deciding whether there exists a ***sparse staircase*** of ones in M that starts at $M_{1,1}$, ends at $M_{m,n}$, and consists of an interweaving sequence of skipping upward moves and skipping right moves (see Figure 2.1(c)).

Katz and Sharir [KS97] showed that the set $S = \{(a_i,b_j) \mid \|a_i - b_j\| \le \delta\} = \{(a_i,b_j) \mid M_{i,j} = 1\}$ can be computed, in $O((m^{2/3}n^{2/3} + m + n)\log n)$ time and space, as the union of the edge sets of a collection $\Gamma = \{A_t \times B_t \mid A_t \subseteq A,\ B_t \subseteq B\}$ of edge-disjoint complete bipartite graphs. The number of graphs in Γ is $O(m^{2/3}n^{2/3}+m+n)$, and the overall sizes of their vertex sets are

$$\sum_t |A_t|, \sum_t |B_t| = O((m^{2/3}n^{2/3} + m + n)\log n).$$

We store each graph $A_t \times B_t \in \Gamma$ as a pair of sorted linked lists L_{A_t} and L_{B_t} over the points of A_t and of B_t, respectively. For each graph $A_t \times B_t \in \Gamma$, there is 1 in each entry $M_{i,j}$ such that $(a_i,b_j) \in A_t \times B_t$. That is, $A_t \times B_t$ corresponds to a submatrix

$M^{(t)}$ of ones in M (whose rows and columns are not necessarily consecutive). See Figure 2.2(a).

Note that if $(a_i, b_j) \in A_t \times B_t$ is a reachable position of the frogs, then every pair in the set $\{(a_k, b_l) \in A_t \times B_t \mid k \geq i, l \geq j\}$ is also a reachable position. (In other words, the positions in the upper-right submatrix of $M^{(t)}$ whose lower-left entry is $M_{i,j}$ are all reachable; see Figure 2.2(b)).

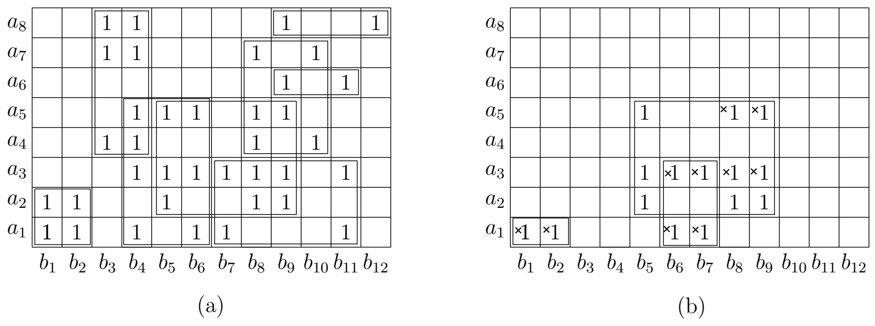

a_8
a_7
a_6
a_5
a_4
a_3
a_2
a_1
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8 b_9 b_{10} b_{11} b_{12}
(a)
(b)

Figure 2.2: (a) A possible representation of the matrix M as a collection of submatrices of ones, corresponding to the complete bipartite graphs $\{a_1, a_2\} \times \{b_1, b_2\}, \{a_1, a_3, a_5\} \times \{b_4, b_6\}, \{a_1, a_3\} \times \{b_7, b_{11}\}, \{a_2, a_3, a_5\} \times \{b_5, b_8, b_9\}, \{a_4, a_7, a_8\} \times \{b_3, b_4\}, \{a_4, a_7\} \times \{b_8, b_{10}\}, \{a_6\} \times \{b_9, b_{11}\}, \{a_8\} \times \{b_9, b_{12}\}$. (b) Another matrix M, similarly decomposed, where the reachable positions are marked with an x.

We say that a graph $A_t \times B_t \in \Gamma$ *intersects* a row i (resp., a column j) in M if $a_i \in A_t$ (resp., $b_j \in B_t$). We denote the subset of graphs of Γ that intersect the ith row of M by Γ_i^r and those that intersect the jth column by Γ_j^c. The sets Γ_i^r are easily constructed from the lists L_{A_t} of the graphs in Γ, and are maintained as linked lists. Similarly, the sets Γ_j^c are constructed from the lists L_{B_t}, and are maintained as doubly-linked lists, so as to facilitate deletions of elements from them. We have $\sum_i |\Gamma_i^r| = \sum_t |A_t| = O((m^{2/3}n^{2/3} + m + n) \log n)$ and $\sum_j |\Gamma_j^c| = \sum_t |B_t| = O((m^{2/3}n^{2/3} + m + n) \log n)$.

We define a 1-entry (a_k, b_j) to be *reachable from below row* i, if $k \geq i$ and there exists an entry (a_ℓ, b_j), $\ell < i$, which is reachable. We process the rows of M in increasing order and for each graph $A_t \times B_t \in \Gamma$ maintain a reachability variable v_t, which is initially set to ∞. We maintain the invariant that when we start processing row i, if $A_t \times B_t$ intersects at least one row that is not below the ith row, then v_t stores the smallest index j for which there exists an entry $(a_k, b_j) \in A_t \times B_t$ that is reachable from below row i.

Before we start processing the rows of M, we verify that $M_{1,1} = 1$ and $M_{m,n} = 1$, and abort the computation if this is not the case, determining that $d_{dF}^{+}(A, B) > \delta$.

Assuming that $M_{1,1} = 1$, each position in $P_1 = \{(a_1, b_l) \mid M_{1,l} = 1\}$ is a reachable position. It follows that for each graph $A_t \times B_t \in \Gamma$, v_t should be set to $\min\{l \mid A_t \times B_t \in \Gamma_l^c \text{ and } (a_1, b_l) \in P_1\}$. Note that graphs $A_t \times B_t$ in this set are not necessarily in Γ_1^r. We update the v_t's using this rule, as follows. We first compute P_1, the set of pairs, each consisting of a_1 and an element of the union of the lists L_{B_t}, for $A_t \times B_t \in \Gamma_1^r$. Then, for each $(a_1, b_l) \in P_1$, we set, for each graph $A_u \times B_u \in \Gamma_l^c$, $v_u \leftarrow \min\{v_u, l\}$.

In principle, this step should now be repeated for each row i. That is, we should compute $y_i = \min\{v_t \mid A_t \times B_t \in \Gamma_i^r\}$; this is the index of the leftmost entry of row i that is reachable from below row i. Next, we should compute $P_i = \{(a_i, b_l) \mid M_{i,l} = 1 \text{ and } l \geq y_i\}$ as the union of those pairs that consist of a_i and an element of

$$\{b_j \mid b_j \in L_{B_t} \text{ for } A_t \times B_t \in \Gamma_i^r \text{ and } j \geq y_i\}.$$

The set P_i is the set of reachable positions in row i. Then we should set for each $(a_1, b_l) \in P_i$ and for each graph $A_u \times B_u \in \Gamma_l^c$, $v_u \leftarrow \min\{v_u, l\}$. This however is too expensive, because it may make us construct explicitly all the 1-entries of M.

To reduce the cost of this step, we note that, for any graph $A_t \times B_t$, as soon as v_t is set to some column l at some point during processing, we can remove b_l from L_{B_t} because its presence in this list has no effect on further updates of the v_t's. Hence, at each step in which we examine a graph $A_t \times B_t \in \Gamma_l^c$, for some column l, we remove b_l from L_{B_t}. This removes b_l from any further consideration in rows with index greater than i and, in particular, Γ_l^c will not be accessed anymore. This is done also when processing the first row.

Specifically, we process the rows in increasing order and when we process row i, we first compute $y_i = \min\{v_t \mid A_t \times B_t \in \Gamma_i^r\}$, in a straightforward manner. (If $i = 1$, then we simply set $y_1 = 1$.) Then we construct a set $P_i' \subseteq P_i$ of the "relevant" (i.e., reachable) 1-entries in the i-th row as follows. For each graph $A_t \times B_t \in \Gamma_i^r$ we traverse (the current) L_{B_t} *backwards*, and for each $b_j \in L_{B_t}$ such that $j \geq y_i$ we add (a_i, b_j) to P_i'. Then, for each $(a_i, b_l) \in P_i'$, we go over all graphs $A_u \times B_u \in \Gamma_l^c$, and set $v_u \leftarrow \min\{v_u, l\}$. After doing so, we remove b_l from all the corresponding lists L_{B_u}.

When we process row m (the last row of M), we set $y_m = \min\{v_t \mid A_t \times B_t \in \Gamma_m^r\}$. If $y_m < \infty$, we conclude that $d_{dF}^+(A, B) \leq \delta$ (recalling that we already know that $M_{m,n} = 1$). Otherwise, we conclude that $d_{dF}^+(A, B) > \delta$.

Correctness. We need to show that $d_{dF}^+(A, B) \leq \delta$ if and only if $y_m < \infty$ (when we start processing row m). To this end, we establish in Lemma 2.4 that the invariant stated above regarding v_t indeed holds. Hence, if $y_m < \infty$, then the position (a_m, b_{y_m}) is reachable from below row m, implying that (a_m, b_n) is also a reachable position and thus $d_{dF}^+(A, B) \leq \delta$. Conversely, if $d_{dF}^+(A, B) \leq \delta$ then (a_m, b_n) is a reachable position. So, either (a_m, b_n) is reachable from below row m, or there exists a position (a_m, b_j), $j < n$, that is reachable from below row m (or both). In either case there exists a graph $A_t \times B_t$ in Γ_m^r such that $v_t \leq n$ and thus $y_m < \infty$. We next show that the reachability variables v_t of the graphs in Γ are maintained correctly.

Lemma 2.4. *For each $i = 1, \ldots, m$, the following property holds. Let $A_t \times B_t$ be a graph in Γ_i^r, and let j denote the smallest index for which $(a_i, b_j) \in A_t \times B_t$ and (a_i, b_j) is reachable from below row i. Then, when we start processing row i, we have $v_t = j$.*

Proof. We prove this claim by induction on i. For $i = 1$, this claim holds trivially. We assume then that $i > 1$ and that the claim is true for each row $i' < i$, and show that it also holds for row i.

Let $A_t \times B_t$ be a graph in Γ_i^r, and let j denote the smallest index for which there exists a position $(a_i, b_j) \in A_t \times B_t$ that is reachable from below row i. We need to show that $v_t = j$ when we start processing row i.

Since (a_i, b_j) is reachable from below row i, there exists a position (a_k, b_j), with $k < i$, that is reachable, and we let k_0 denote the smallest index for which (a_{k_0}, b_j) is reachable. Let $A_o \times B_o$ be the graph containing (a_{k_0}, b_j). We first claim that when we start processing row k_0, b_j was not yet deleted from L_{B_o} (nor from the corresponding list of any other graph in Γ_j^c). Assume to the contrary that b_j was deleted from L_{B_o} before processing row k_0. Then there exists a row $z < k_0$ such that $(a_z, b_j) \in P'_z$ and we deleted b_j from L_{B_o} when we processed row z. By the last assumption, (a_z, b_j) is a reachable position. This is a contradiction to k_0 being the smallest index for which (a_{k_0}, b_j) is reachable. (The same argument applies for any other graph, instead of $A_o \times B_o$.)

We next show that $v_t \le j$. Since $(a_{k_0}, b_j) \in A_o \times B_o$, $A_o \times B_o \in \Gamma_{k_0}^r \cap \Gamma_j^c$. Since k_0 is the smallest index for which (a_{k_0}, b_j) is reachable, there exists an index j_0, such that $j_0 < j$ and (a_{k_0}, b_{j_0}) is reachable from below row k_0. (If $k_0 = 1$, we use instead the starting placement (a_1, b_1).) It follows from the induction hypothesis that $y_{k_0} \le j_0 < j$. Thus, when we processed row k_0 and we went over L_{B_o}, we encountered b_j (as just argued, b_j was still in that list), and we consequently updated the reachability variables v_u of each graph in Γ_j^c, including our graph $A_t \times B_t$ to be at most j.

(Note that if there is no position in $A_t \times B_t$ that is reachable from below row i (i.e., $j = \infty$), we trivially have $v_t \le \infty$.)

Finally, we show that $v_t = j$. Assume to the contrary that $v_t = j_1 < j$ when we start processing row i. Then we have updated v_t to hold j_1 when we processed b_{j_1} at some row $k_1 < i$. So, by the induction hypothesis, $y_{k_1} \le j_1$, and thus (a_{k_1}, b_{j_1}) is a reachable position. Moreover, $A_t \times B_t \in \Gamma_{j_1}^c$, since v_t has been updated to hold j_1 when we processed b_{j_1}. It follows that $(a_i, b_{j_1}) \in A_t \times B_t$. Hence, (a_i, b_{j_1}) is reachable from below row i. This is a contradiction to j being the smallest index such that (a_i, b_j) is reachable from below row i. This establishes the induction step and thus completes the proof of the lemma. □

Running Time. We first analyze the initialization cost of the data structure, and then the cost of traversal of the rows for maintaining the variables v_t.

- Initialization: Constructing Γ takes $O((m^{2/3}n^{2/3} + m + n)\log(m + n))$ time. Sorting the lists L_{A_t} (resp., L_{B_t}) of each $A_t \times B_t \in \Gamma$ takes $O((m^{2/3}n^{2/3} + m + n)\log^2(m+n))$ time. Constructing the lists Γ_i^r (resp., Γ_j^c) for each $a_i \in A$ (resp., $b_j \in B$) takes time linear in the sum of the sizes of the A_t's and the B_t's, which is $O((m^{2/3}n^{2/3} + m + n)\log(m + n))$.

- Traversing the rows: When we process row i we first compute y_i by scanning Γ_i^r. This takes a total of $O\left(\sum_i |\Gamma_i^r|\right) = O((m^{2/3}n^{2/3} + m + n)\log n)$ for all rows. Since the lists L_{B_t} are sorted, the computation of P_i' is linear in the size of P_i'. This is so because, once we have added a pair (a_i, b_j) to P_i', we remove b_j from all lists that contain it, so we will not encounter it again when scanning other lists $L_{B_{t'}}$. For each pair $(a_i, b_\ell) \in P_i'$ we scan Γ_ℓ^c, which must contain at least one graph $A_t \times B_t \in \Gamma$ such that $a_i \in A_t$ (and $b_j \in B_t$). For each element $A_t \times B_t \in \Gamma_\ell^c$ we spend constant time updating v_t and removing b_ℓ from L_{B_t}. It follows that the total time, over all rows, of computing P_i' and scanning the lists Γ_ℓ^c is $O\left(\sum_l |\Gamma_l^c|\right) = O((m^{2/3}n^{2/3} + m + n)\log n)$.

We conclude that the total running time is $O((m^{2/3}n^{2/3} + m + n)\log^2(m + n))$.

The optimization procedure. We use the above decision procedure for finding the optimum $d_{dF}^{+}(A, B)$, as follows. Note that if we increase δ continuously, the set of 1-entries of M can only grow, and this can only happen at a distance between a point of A and a point of B. We thus perform a binary search over the mn pairwise distances between the pairs of $A \times B$. In each step of the search we need to determine the kth smallest pairwise distance r_k in $A \times B$, for some value of k. We do so by using the distance selection algorithm of Katz and Sharir [KS97], which can easily be adapted to work for this bichromatic scenario. We then run the decision procedure on r_k, using its output to guide the binary search. At the end of this search, we obtain two consecutive critical distances δ_1, δ_2 such that $\delta_1 < d_{dF}^{+}(A, B) \le \delta_2$, and we can therefore conclude that $d_{dF}^{+}(A, B) = \delta_2$. The running time of the distance selection algorithm of [KS97] is $O((m^{2/3}n^{2/3} + m + n)\log^2(m + n))$, which also holds for the bipartite version that we use. We thus obtain the following main result of this section.

Theorem 2.5. *Given a set A of m points and a set B of n points in the plane, we can compute the two-sided discrete Fréchet distance with shortcuts $d_{dF}^{+}(A, B)$, in time $O((m^{2/3}n^{2/3} + m + n)\log^3(m + n))$, using $O((m^{2/3}n^{2/3} + m + n)\log(m + n))$ space.*

2.6 Semi-continuous Fréchet distance with shortcuts

Let $f \subseteq \mathbb{R}^2$ denote a polygonal curve with n edges $e_1, \ldots, e_n$ and $n+1$ vertices $p_0, p_1, \ldots, p_n$, and let $A = (a_1, \ldots, a_m)$ denote a sequence of m points in the plane. Consider a person that is walking along f from its starting endpoint to its final endpoint, and a frog that is jumping along the sequence A of stones. The frog is allowed to make shortcuts (i.e., skip stones) as long as it traverses A in the right (increasing) direction, but the person must trace the complete curve f (see Figure 2.3(a)). Assuming that the person holds the frog by a leash, our goal is to compute the minimal length $d^s_{dF}(A, f)$ of a leash that is required in order to traverse f and (parts of) A in this manner, taking the frog and the person from (a_1, p_0) to (a_m, p_n).

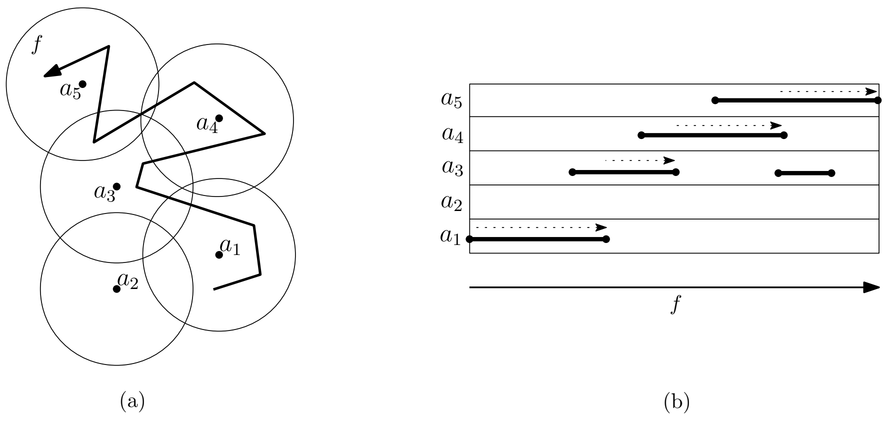


Figure 2.3: (a) A curve f and a sequence of points $A = (a_1, \ldots, a_5)$. (b) Thinking of f as a continuous mapping from $[0,1]$ to $\mathbb{R}^2$, the ith row depicts the set $\{t \in [0,1] \mid f(t) \in D_\delta(a_i)\}$. The dotted subintervals and their connecting upward moves (not drawn) constitute the lowest semi-sparse staircase between the starting and final positions.

We next very briefly review our algorithm. Details can be found in the full version of the paper [AFK+14]. Consider the decision version of this problem, where, given a parameter $\delta > 0$, we wish to decide whether the person and the frog can traverse f and (parts of) A using a leash of length δ. This problem can be solved using the algorithm for solving the one-sided DFDS, with a slight modification that takes into account the continuous nature of f. Specifically, for a point $p \in \mathbb{R}^2$, let $D_\delta(p)$ denote the disk of radius δ centered at p. Now, consider a vector M whose entries correspond to the points of A. For each $i = 1, \ldots, m$, the ith entry of M is

$$M_i = M(a_i) = f \cap D_\delta(a_i)$$

(see Figure 2.3(b)). Each M_i is a finite union of connected subintervals of f. We do not compute M explicitly, because the overall "description complexity" of its entries might be too large. Specifically, the number of connected subsegments of the edges of f that comprise the elements of M can be mn in the worst case.

Instead, we assume availability of (efficient implementations of) the following two primitives.

(i) **NextEndpoint(x, a_i):** Given a point $x \in f$ and a point a_i of A, such that $x \in D_\delta(a_i)$, return the forward endpoint of the connected component of $f \cap D_\delta(a_i)$ that contains x.

(ii) **NextDisk(x, a_i):** Given x and a_i, as in (i), find the smallest $j > i$ such that $x \in D_\delta(a_j)$, or report that no such index exists (return $j = \infty$).

Both primitives admit efficient implementations. For our purposes it is sufficient to implement Primitive (i) by traversing the edges of f one by one, starting from the edge containing x, and checking for each such edge e_j of f whether the forward endpoint p_j of e_j belongs to $D_\delta(a_i)$. For the first e_j for which this test fails, we return the forward endpoint of the interval $e_j \cap D_\delta(a_i)$. It is also sufficient to implement Primitive (ii) by checking for each $j > i$ in increasing order, whether $x \in D_\delta(a_j)$, and return the first j for which this holds. To solve the decision problem, we proceed as in the decision procedure of the one-sided DFDS (see Algorithm 2.1), except that when we move "right", we move along f as long as we can within the current disk (using Primitive (i)), and when we move "up", we move to the first following disk that contains the current point of f (using Primitive (ii)).

The correctness of the decision procedure is proved similarly to the correctness of the decision procedure of the one-sided DFDS (Algorithm 2.1). More specifically, here a ***semi-continuous semi-sparse staircase*** is an interweaving sequence of ***discrete skipping upward moves*** and ***continuous right moves***, where a discrete skipping upward move is a move from a reachable position (a_i, p) of the frog and the person to another position (a_j, p) such that $j > i$ and $p \in D_\delta(a_j)$. A continuous right move is a move from a reachable position (a_i, p) of the frog and the person to another position (a_i, p') where p', and the entire portion of f between p and p', are contained in $D_\delta(a_i)$. Then there exists a semi-continuous semi-sparse staircase that reaches the position (a_m, p_n) if and only if $\delta_F^s(A, f) \le \delta$.

Concerning correctness, we prove that if there exists a semi-continuous semi-sparse staircase S' that reaches position (a_m, p_n), then the decision procedure maintains a partial semi-continuous semi-sparse staircase S that is always "below" S' (in terms of the corresponding indices of the positions of the frog), and therefore S reaches a position where the person is at p_n (and the frog can then jump directly to a_m). Intuitively, this holds since the decision procedure can at any point join the plot of S' using a discrete skipping upward move. The running time of this decision procedure is $O(n + m)$ since we advance along f at each step of Primitive (i), and we advance along A at each step of Primitive (ii), so our naive implementations of these primitives never back up along the path and sequence, and consequently take $O(m + n)$ time in total.

We then present an algorithm that leads, in combination with the decision procedure, to an algorithm for the optimization problem that runs in $O((m+n)^{2/3}m^{2/3}n^{1/3}\log(m+n))$ randomized expected time. This algorithm is analogous to the algorithm of Lemma 2.2 of the discrete case. This demonstrates that the general framework of the optimization algorithm of Section 2.4 can be applied (with twists) in other algorithms.

Chapter 3

The Discrete Fréchet Distance under Translation

3.1 Introduction

In many applications of the Fréchet distance, the input curves are not necessarily aligned, and one of them needs to be adjusted (i.e., undergo some transformation) for the distance computation to be meaningful. In this chapter we consider the discrete Fréchet distance under translation, in which we are given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, and wish to find a translation t that minimizes the discrete Fréchet distance between A and $B + t$.

For points in the plane, Alt et al. [AKW01] gave an $O(m^3n^3(m+n)^2 \log(m+n))$-time algorithm for computing the continuous Fréchet distance under translation, and an algorithm computing a $(1+\varepsilon)$-approximation in $O(\varepsilon^{-2}mn)$ time. In 3D, Wenk [Wen03] showed that the continuous Fréchet distance under any reasonable family of transformations, can be computed in $O((m+n)^{3f+2} \log(m+n))$ time, where f is the number of degrees of freedom for moving one sequence w.r.t. the other. Thus, for translations only ($f = 3$), the continuous Fréchet distance in $\mathbb{R}^3$ can be computed in $O((m+n)^{11} \log(m+n))$ time.

In the discrete case, the situation is a little better. For points in the plane, Jiang et al. [JXZ08] gave an $O(m^3n^3 \log(m+n))$-time algorithm for DFD under translation, and an $O(m^4n^4 \log(m+n))$-time algorithm when both translations and rotations are allowed. Mosig et al. [MC05] presented an approximation algorithm for DFD under translation, rotation and scaling in the plane, with approximation factor close to 2 and running time $O(m^2n^2)$. Finally, Ben Avraham et al. [AKS15] presented an $O(m^3n^2(1 + \log(n/m)) \log(m+n))$-time algorithm for DFD under translation. Their decision algorithm (deciding whether the distance is smaller than a given δ) is based on a dynamic data structure which supports updates and reachability queries in $O(m(1 + \log(n/m))$ time. Given sequences A and B, the basic idea is to maintain the reachability graph G_δ defined in Chapter 2, while traversing a subdivision of the plane of translations. The subdivision is such that when moving from one cell to an adjacent one, only for a single pair of points in $A \times B$ their (Euclidean) distance becomes smaller or larger than δ, thus only a constant number of edges in G_δ need to be updated. Using a more general data structure of Diks and Sankowski [DS07] for dynamic maintenance of reachability in directed planar graphs, one can obtain a slightly less efficient algorithm for the problem.

Another related paper is by de Berg and Cook IV [dBI11], who presented the direction-based Fréchet distance, which is invariant under translations and scalings. This measure optimizes over all parameterizations for a pair of curves, but based on differences between the directions of movement along the curves, rather than on distances between the positions.

In this chapter we consider two variants of DFD, both under translation: the first is discrete Fréchet distance with shortcuts (DFDS), and the second is weak discrete Fréchet distance (WDFD), in which the frogs are allowed to jump also backwards to the previous point in their sequence.

Our results. Our first major result is an efficient algorithm for DFDS under translation. We provide a dynamic data structure which supports updates and reachability queries in $O(\log(m+n))$ time. The data structure is based on Sleator and Tarjan's Link-Cut Trees structure [ST83], and, by plugging it into the optimization algorithm of Ben Avraham et al. [AKS15], we obtain an $O(m^2n^2\log^2(m+n))$-time algorithm for DFDS under translation; an order of magnitude faster than the the algorithm for DFD under translation.

For curves in 1D, the optimization algorithm of [AKS15] yields an $O(m^2n(1+\log(n/m))\log(m+n))$-time algorithm for DFD, using their reachability structure, an $O(mn\log^2(m+n))$-time algorithm for DFDS, using our reachability with shortcuts structure, and an $O(mn\log^2(m+n)(\log\log(m+n))^3)$-time algorithm for WDFD, using the reachability structure of Thorup [Tho00] for undirected general graphs. We describe a simpler optimization algorithm for 1D, which avoids the need for parametric search and yields an $O(m^2n(1+\log(n/m)))$-time algorithm for DFD, an $O(mn\log(m+n))$-time algorithm for DFDS, and an $O(mn\log(m+n)(\log\log(m+n))^3)$-time algorithm for WDFD; i.e., we remove a logarithmic factor from the bounds obtained with the algorithm of Ben Avraham et al.

Our optimization algorithm for 1D follows a general scheme introduced by Martello et al. [MPTDW84] for the Balanced Optimization Problem (BOP). BOP is defined as follows. Let $E = \{e_1, \ldots, e_l\}$ be a set of l elements (where here $l = O(mn)$), $c : E \to \mathbb{R}$ a cost function, and $\mathcal{F}$ a set of feasible subsets of E. Find a feasible subset $S^* \in \mathcal{F}$ that minimizes $\max\{c(e_i) : e_i \in S\} - \min\{c(e_i) : e_i \in S\}$, over all $S \in \mathcal{F}$. Given a feasibility decider that decides whether a subset is feasible or not in $f(l)$ time, the algorithm of [MPTDW84] finds an optimal range in $O(lf(l) + l\log l)$-time.

The scheme of [MPTDW84] is especially useful when an efficient dynamic version of the feasibility decider is available, as in the cases of DFD (where $f(l) = O(m(1 + \log(n/m))))$, DFDS (where $f(l) = O(\log(m + n)))$, and WDFD (where $f(l) = O(\log(m + n)(\log\log(m + n))^3))$[1].

Our second major result is an alternative scheme for BOP. Our optimization scheme does not require a specially tailored dynamic version of the feasibility decider in order to obtain faster algorithms (than the naive $O(lf(l) + l \log l)$ one), rather, whenever the underlying problem has some desirable properties, it produces algorithms with running time $O(f(l) \log^2 l + l \log l)$. Thus, the advantage of our scheme is that it yields efficient algorithms quite easily, without having to devise an efficient dynamic version of the feasibility decider, a task which is often difficult if at all possible.

We demonstrate our scheme on the *most uniform path* problem (MUPP). Given a weighted graph $G = (V, E, w)$ with n vertices and m edges and two vertices $s, t \in V$, the goal is to find a path P^* in G between s and t that minimizes $\max\{w(e) : e \in P\} - \min\{w(e) : e \in P\}$, over all paths P from s to t. This problem was introduced by Hansen et al. [HSV97], who gave an $O(m^2)$-time algorithm for it. By using the dynamic connectivity data structure of Thorup [Tho00], one can reduce the running time to $O(m \log n(\log\log n)^3)$. We apply our scheme to MUPP to obtain a much simpler algorithm with a slightly larger ($O(m \log^2 n)$) running time. Finally, we observe that WDFD under translation in 1D can be viewed as a special case of MUPP, so we immediately obtain a much simpler algorithm than the one based on Thorup's dynamic data structure (see above), at the cost of an additional logarithmic factor.

3.2 Preliminaries

The definition of discrete Fréchet distance that we use in this chapter is almost similar to the graph definition in Chapter 2, but with a little modification that allows us to describe a dynamic data structure on this graph.

Let $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$ be two sequences of points. We define a directed graph $G = G(V = A \times B, E = E_A \cup E_B \cup E_{AB})$, whose vertices are the possible positions of the frogs and whose edges are the possible moves between positions:

$E_A = \{\langle(a_i, b_j), (a_{i+1}, b_j)\rangle\}, E_B = \{\langle(a_i, b_j), (a_i, b_{j+1})\rangle\}, E_{AB} = \{\langle(a_i, b_j), (a_{i+1}, b_{j+1})\rangle\}.$

The set E_A corresponds to moves where only the A-frog jumps forward, the set E_B corresponds to moves where only the B-frog jumps forward, and the set E_{AB}

[1]Actually, the query (decision) time in Thorup's data structure is only $O(\log(m+n)/\log\log\log(m+n))$, but in each step of the search we also have to update the data structure in $O(\log(m + n)(\log\log(m + n))^3)$ time. The question whether logarithmic-time is achievable for both query and update (of connectivity in general graphs) is still open.

corresponds to moves where both frogs jump forward. Notice that any valid sequence of moves (with unlimited leash length) corresponds to a path in G from (a_1, b_1) to (a_n, b_m), and vice versa.

It is likely that not all positions in $A \times B$ are ***valid***; for example, when the leash is short. We thus assume that we are given an ***indicator*** function $\sigma : A \times B \to \{0, 1\}$, which determines for each position whether it is valid or not. Now, we say that a position (a_i, b_j) is a ***reachable position*** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), consisting of only valid positions, i.e., for each position $(a_k, b_l) \in P$, we have $\sigma(a_k, b_l) = 1$.

Let $d(a_i, b_j)$ denote the Euclidean distance between a_i and b_j. For any distance $\delta \geq 0$, the function σ_δ is defined as follows: $\sigma_\delta(a_i, b_j) = \begin{cases} 1, & d(a_i, b_j) \leq \delta \\ 0, & \text{otherwise} \end{cases}$.

The ***discrete Fréchet distance*** $d_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a reachable position w.r.t. σ_δ.

One-sided shortcuts. Let σ be an indicator function. We say that a position (a_i, b_j) is an ***s-reachable position*** (w.r.t. σ), if there exists a path P in G from (a_1, b_1) to (a_i, b_j), such that $\sigma(a_1, b_1) = 1$, $\sigma(a_i, b_j) = 1$, and for each b_l, $1 < l < j$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma(a_k, b_l) = 1$). We call such a path an ***s-path***. In general, an s-path consists of both valid and non-valid positions. Consider the sequence S of positions that is obtained from P by deleting the non-valid positions. Then S corresponds to a sequence of moves, where the A-frog is allowed to skip points, and the leash satisfies σ. Since in any path in G the two indices (of the A-points and of the B-points) are monotonically non-decreasing, it follows that in S the B-frog visits each of the points $b_1, \ldots, b_j$, in order, while the A-frog visits only a subset of the points $a_1, \ldots, a_i$ (including a_1 and a_i), in order.

The ***discrete Fréchet distance with shortcuts*** $d^s_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is an s-reachable position w.r.t. σ_δ.

Weak Fréchet distance. Let $G_w = G(V = A \times B, E_w)$, where $E_w = \{(u, v) | \langle u, v\rangle \in E_A \cup E_B \cup E_{AB}\}$. That is, G_w is an undirected graph obtained from the graph G of the 'strong' version, which contains directed edges, by removing the directions from the edges. Let σ be an indicator function. We say that a position (a_i, b_j) is a ***w-reachable position*** (w.r.t. σ), if there exists a path P in G_w from (a_1, b_1) to (a_i, b_j) consisting of only valid positions. Such a path corresponds to a sequence of moves of the frogs, with a leash satisfying σ, where backtracking is allowed.

The ***weak discrete Fréchet distance*** $d^w_{dF}(A, B)$ is the smallest $\delta \geq 0$ for which (a_n, b_m) is a w-reachable position w.r.t. σ_δ.

The translation problem. Given two sequences of points $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_m)$, we wish to find a translation t^* that minimizes $d_{dF}(A, B + t)$ (similarly, $d^s_{dF}(A, B + t)$ and $d^w_{dF}(A, B + t)$), over all translations t. Denote

$$\begin{aligned}
\widehat{d_{dF}}(A, B) &= \min_t \{d_{dF}(A, B + t)\}, \\
\widehat{d^s_{dF}}(A, B) &= \min_t \{d^s_{dF}(A, B + t)\}, \text{ and} \\
\widehat{d^w_{dF}}(A, B) &= \min_t \{d^w_{dF}(A, B + t)\}.
\end{aligned}$$

3.3 DFDS under translation

The discrete Fréchet distance (and its shortcuts variant) between A and B is determined by two points, one from A and one from B. Consider the decision version of the translation problem: given a distance δ, decide whether $\widehat{d_{dF}}(A, B) \le \delta$ (or $\widehat{d^s_{dF}}(A, B) \le \delta$).

Ben Avaraham et al. [AKS15] described a subdivision of the plane of translations: given two points $a \in A$ and $b \in B$, consider the disk $D_\delta(a - b)$ of radius δ centered at $a-b$, and notice that $t \in D_\delta(a-b)$ if and only if $d(a-b, t) \le \delta$ (or $d(a, b+t) \le \delta$). That is, $D_\delta(a-b)$ is precisely the set of translations t for which $b+t$ is at distance at most δ from a. They construct the arrangement A_δ of the disks in $\{D_\delta(a-b) \mid (a, b) \in A \times B\}$, which consists of $O(m^2n^2)$ cells. Then, they initialize their dynamic data structure for (discrete Fréchet) reachability queries, and traverse the cells of A_δ such that, when moving from one cell to its neighbor, the dynamic data structure is updated and queried a constant number of times in $O(m(1 + \log(n/m))$ time. Finally, they use parametric search in order to find an optimal translation, which adds only a $O(\log(m + n))$ factor to the running time.

In this section we present a dynamic data structure for s-reachability queries, which allows updates and queries in $O(\log(m + n))$ time. We observe that the same parametric search can be used in the shortcuts variant, since the critical values are the same. Thus, by combining our dynamic data structure with the parametric search of [AKS15], we obtain an $O(m^2n^2 \log^2(m + n))$-time algorithm for DFDS under translation.

We now describe the dynamic data structure for DFDS. Consider the decision version of the problem: given a distance δ, we would like to determine whether $d^s_{dF}(A, B) \le \delta$, i.e., whether (a_n, b_m) is an s-reachable position w.r.t. σ_δ. In Chapter 2, we presented a linear time algorithm for this decision problem. Informally, the decision algorithm on the graph G is as follows: starting at (a_1, b_1), the B-frog jumps forward (one point at a time) as long as possible, while the A-frog stays in place, then the A-frog makes the smallest forward jump needed to allow the B-frog to continue. They continue advancing in this way, until they either reach (a_n, b_m) or get stuck.

Consider the (directed) graph[2] $G_\delta = G(V = A \times B, E = E'_A \cup E'_B)$, where

$$E'_A = \{\langle (a_i, b_j), (a_{i+1}, b_j)\rangle \mid \sigma_\delta(a_i, b_j) = 0,\ 1 \le i \le n-1,\ 1 \le j \le m\}\text{, and}$$
$$E'_B = \{\langle (a_i, b_j), (a_i, b_{j+1})\rangle \mid \sigma_\delta(a_i, b_j) = 1,\ 1 \le i \le n,\ 1 \le j \le m-1\}\,.$$

In G_δ, if the current position of the frogs is valid, only the B-frog may jump forward and the A-frog stays in place. And, if the current position is non-valid, the B-frog stays in place and only the A-frog may jump forward. Let M_δ be an

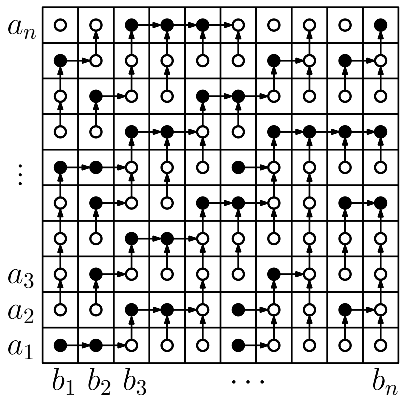


Figure 3.1: The graph G_δ on the matrix M_δ. The black vertices are valid and the white ones are non-valid.

$n \times m$ matrix such that $M_{i,j} = \sigma_\delta(a_i, b_j)$. Each vertex in G_δ corresponds to a cell of the matrix. The directed edges of G_δ correspond to right-moves (the B-frog jumps forward) and upward-moves (the A-frog jumps forward) in the matrix. Any right-move is an edge originating at a valid vertex, and any upward-move is an edge originating at a non-valid vertex (see Figure 3.1).

Observation 3.1. *G_δ is a set of rooted binary trees, where a root is a vertex of out-degree 0.*

Proof. Clearly, G is a directed acyclic graph, and G_δ is a subgraph of G. In G_δ, each vertex has at most one outgoing edge. It is easy to see (by induction on the number of vertices) that such a graph is a set of rooted trees. □

We call a path P in G from (a_i, b_j) to $(a_{i'}, b_{j'})$, $i \le i'$, $j \le j'$, a ***partial s-path***, if for each b_l, $j \le l < j'$, there exists a position $(a_k, b_l) \in P$ that is valid (i.e., $\sigma_\delta(a_k, b_l) = 1$).

Observation 3.2. *All the paths in G_δ are partial s-paths.*

Proof. Let P be a path from (a_i, b_j) to $(a_{i'}, b_{j'})$ in G_δ. Each right-move in P advances the B-frog by one step forward. If $j = j'$ then the claim is vacuously true. Else, P must contain a right-move for each b_l, $j \le l < j'$. Any right-move is an edge

[2]Note that this definition of G_δ is different from the one we used in Chapter 2

originating at a valid vertex, thus for any $j \leq l < j'$ there exists a position $(a_k, b_l) \in P$ such that $\sigma_\delta(a_k, b_l) = 1$. □

Denote by $r(a_i, b_j)$ the root of (a_i, b_j) in G_δ.

Lemma 3.3. *(a_n, b_m) is an s-reachable position in G w.r.t. σ_δ, if and only if $\sigma_\delta(a_1, b_1) = 1$, $\sigma_\delta(a_n, b_m) = 1$, and $r(a_1, b_1) = (a_i, b_m)$ for some $1 \leq i \leq n$.*

Proof. Assume that $\sigma_\delta(a_1, b_1) = 1$, $\sigma_\delta(a_n, b_m) = 1$, and $r(a_1, b_1) = (a_i, b_m)$ for some $1 \leq i \leq n$. Then by Observation 3.2 there is a partial s-path from (a_1, b_1) to (a_i, b_m) in G_δ, and since $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$ we have an s-path from (a_1, b_1) to (a_n, b_m).

Now assume that (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ. Then, in particular, $\sigma_\delta(a_1, b_1) = 1$ and $\sigma_\delta(a_n, b_m) = 1$, and there exists an s-path P in G from (a_1, b_1) to (a_n, b_m). Let P' be the path in G_δ from (a_1, b_1) to $r(a_1, b_1)$. Informally, we claim that P' is always not above P. More precisely, we prove that if a position (a_i, b_j) is an s-reachable position in G, then there exists a position $(a_{i'}, b_j) \in P'$, $i' \leq i$, such that $\sigma_\delta(a_{i'}, b_j) = 1$. In particular, since (a_n, b_m) is an s-reachable position in G, there exists a position $(a_{i'}, b_m) \in P'$, $i' \leq n$, such that $\sigma_\delta(a_{i'}, b_m) = 1$, and thus $r(a_1, b_1) = (a_{i''}, b_m)$ for some $i' \leq i'' \leq n$.

We prove this claim by induction on j. The base case where $j = 1$ is trivial, since $(a_1, b_1) \in P \cap P'$ and $\sigma_\delta(a_1, b_1) = 1$. Let P be an s-path from (a_1, b_1) to (a_i, b_{j+1}), then $\sigma_\delta(a_i, b_{j+1}) = 1$. Let (a_k, b_j), $k \leq i$, be a position in P such that $\sigma_\delta(a_k, b_j) = 1$. (a_k, b_j) is an s-reachable position in G, so by the induction hypothesis there exists a vertex $(a_{k'}, b_j) \in P'$, $k' \leq k$, such that $\sigma_\delta(a_{k'}, b_j) = 1$. By the construction of G_δ, there is an edge $\langle (a_{k'}, b_j), (a_{k'}, b_{j+1}) \rangle$, and we have $(a_{k'}, b_{j+1}) \in P'$. Now, let $k' \leq i' \leq i$ be the smallest index such that $\sigma_\delta(a_{i'}, b_{j+1}) = 1$. Since there are no right-moves in P' before reaching $(a_{i'}, b_{j+1})$, we have $(a_{i'}, b_{j+1}) \in P'$. □

We represent G_δ using the Link-Cut tree data structure, which was developed by Sleator and Tarjan [ST83]. The data structure stores a set of rooted trees and supports the following operations in $O(\log n)$ amortized time:

- *Link*(v, u) — connect a root node v to another node u as its child.
- *Cut*(v) — disconnect the subtree rooted at v from the tree to which it belong.
- *FindRoot*(v) — find the root of the tree to which v belongs.

Now, in order to maintain the representation of G_δ following a single change in σ_δ (i.e., when switching one position (a_i, b_j) from valid to non-valid or vice versa), one edge should be removed and one edge should be added to the structure. We update our structure as follows: Let T be the tree containing (a_i, b_j).

- When switching (a_i, b_j) from valid to non-valid, we first need to remove the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$, if $j < m$, by disconnecting (a_i, b_j) (and its subtree) from T ($Cut(a_i, b_j)$). Then, if $i < n$, we add the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$ by connecting (a_i, b_j) (which is now the root of its tree) to (a_{i+1}, b_j) as its child ($Link((a_i, b_j), (a_{i+1}, b_j))$).

- When switching a position from non-valid to valid, we need to remove the edge $\langle (a_i, b_j), (a_{i+1}, b_j) \rangle$, if $i < n$, by disconnecting (a_i, b_j) (and its subtree) from T ($Cut(a_i, b_j)$). Then, if $j < m$, we add the edge $\langle (a_i, b_j), (a_i, b_{j+1}) \rangle$ by connecting (a_i, b_j) (which is now the root of its tree) to (a_i, b_{j+1}) as its child ($Link((a_i, b_j), (a_i, b_{j+1}))$).

Assume $\sigma_\delta(a_1, b_1) = \sigma_\delta(a_n, b_m) = 1$. By Lemma 3.3, in the Link-Cut tree data structure representing G_δ, $FindRoot(a_1, b_1)$ is (a_i, b_m) for some $1 \le i \le n$ if and only if (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ. We thus obtain the following theorem.

Theorem 3.4. *Given sequences A and B and an indicator function σ_δ, one can construct a dynamic data structure in $O(mn \log(m+n))$ time, which supports the following operations in $O(\log(m+n))$ time: (i) change a single value of σ_δ, and (ii) check whether (a_n, b_m) is an s-reachable position in G w.r.t. σ_δ.*

Theorem 3.5. *Given sequences A and B with n and m points respectively in the plane, $\widehat{d^s_{dF}}(A, B)$ can be computed in $O(m^2 n^2 \log^2(m+n))$-time.*

3.4 Translation in 1D

The algorithm of [AKS15] can be generalized to any constant dimension $d \ge 1$; only the size of the arrangement of balls, A_δ, changes to $O(m^d n^d)$. The running time of the algorithm for two sequences of points in $\mathbb{R}^d$ is therefore $O(m^{d+1} n^d (1 + \log(n/m)) \log(m+n))$, for DFD, $O(m^d n^d \log^2(m+n))$ for DFDS, and $O(m^d n^d \log^2(m+n)(\log\log(m+n))^3)$ for WDFD; see relevant paragraph in Section 3.1.

When considering the translation problem in 1D, we can improve the bounds above by a logarithmic factor, by avoiding the use of parametric search and applying a direct approach instead. We thus obtain an $O(m^2 n (1 + \log(n/m)))$-time algorithm, for DFD, an $O(mn \log(m+n))$-time algorithm for DFDS, and $O(mn \log(m+n)(\log\log(m+n))^3)$-time algorithm for WDFD.

Let $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_m)$ be two sequences of points in $\mathbb{R}^d$. Consider the set $\mathcal{D} = \{a_i - b_j \mid a_i \in A, b_j \in B\}$. Then, each vertex $v = (a_i, b_j)$ of the graph G has a corresponding point $\overline{v} = (a_i - b_j)$ in $\mathcal{D}$. Given a path P in G from (a_1, b_1) to (a_n, b_m), denote by $\overline{V(P)}$ the set of points of $\mathcal{D}$ corresponding to the vertices $V(P)$ of P. Denote by $S(o,r)$ the sphere with center o and radius r. We define a new indicator function: $\sigma_{S(o,r)}(a_i,b_j) = \begin{cases} 1, & d(a_i-b_j,o) \le r \\ 0, & \text{otherwise} \end{cases}$.

Lemma 3.6. *Let $S = S(t^*,\delta)$ be a smallest sphere for which (a_n,b_m) is a reachable position w.r.t. σ_S. Then, t^* is a translation that minimizes $d_{dF}(A,B+t)$, over all translations t, and $d_{dF}(A,B+t^*) = \delta$.*

Proof. Let t be a translation such that $d_{dF}(A,B+t)=\delta'$, and denote $S'=S(t,\delta')$. Thus, there exist a path P from (a_1,b_1) to (a_n,b_m) in G such that for each vertex (a,b) of P, $d(a,b+t) \le \delta'$. But $d(a,b+t)=d(a-b,t)$, so for each vertex (a,b) of P, $d(a-b,t) \le \delta'$, and thus (a_n,b_m) is a reachable position w.r.t. $\sigma_{S'}$. Since S is the smallest sphere for which (a_n,b_m) is a reachable position w.r.t. σ_S, we get that $\delta' \ge \delta$.

Now, since (a_n,b_m) is a reachable position w.r.t. σ_S, there exists a path P from (a_1,b_1) to (a_n,b_m), such that for each vertex (a,b) of P, $d(a-b,t^*) \le \delta$. But again $d(a-b,t^*) = d(a,b+t^*)$, and thus $d_{dF}(A,B+t^*) \le \delta$. □

Notice that the above lemma is true for the shortcuts and the weak variants as well, by letting (a_n,b_m) be an s-reachable or a w-reachable position, respectively.

Thus, our goal is to find the smallest sphere S for which (a_n,b_m) is a reachable position w.r.t. σ_S. We can perform an exhaustive search: check for each sphere S defined by $d+1$ points of $\mathcal{D}$ whether (a_n,b_m) is a reachable position w.r.t. σ_S. There are $O(m^{d+1}n^{d+1})$ such spheres, and checking whether (a_n,b_m) is a reachable position in G takes $O(mn)$ time. This yields an $O(m^{d+2}n^{d+2})$-time algorithm.

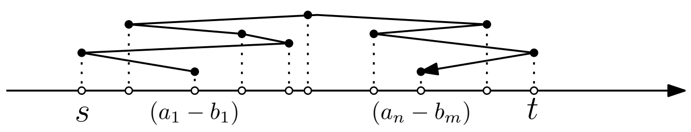


Figure 3.2: The points of $\overline{V(P)}$.

When considering the problem on the line, the goal is to find a path P from (a_1,b_1) to (a_n,b_m), such that the one-dimensional distance between the leftmost point in $\overline{V(P)}$ and the rightmost point in $\overline{V(P)}$ is minimum (see Figure 3.2). In other words, our indicator function is now defined for a given range $[s,t]$: $\sigma_{[s,t]}(a_i,b_j) = \begin{cases} 1, & s \le a_i-b_j \le t \\ 0, & \text{otherwise} \end{cases}$.

We say that a range $[s,t]$ is a ***feasible range*** if (a_n,b_m) is a reachable position in G w.r.t $\sigma_{[s,t]}$. Now, we need to find the smallest feasible range delimited by two points of $\mathcal{D}$.

Consider the following search procedure: Sort the values in $\mathcal{D} = \{d_1,\ldots,d_l\}$ such that $d_1 < d_2 < \cdots < d_l$, where $l = mn$. Set $p \leftarrow 1, q \leftarrow 1$. While $q \le l$, if (a_n,b_m) is a reachable position in G w.r.t. $\sigma_{[d_p,d_q]}$, set $p \leftarrow p+1$, else set $q \leftarrow q+1$. Return the translation corresponding to the smallest feasible range $[d_p, d_q]$ that was found during the while loop.

We use the data structure of [AKS15] for the decision queries, and update it in $O(m(1+\log(n/m))$ time in each step of the algorithm. For DFDS we use our data structure, where the cost of a decision query or an update is $O(\log(m+n))$, and for WDFD we use the data structure of [Tho00], where the cost of a decision query is $O(\log(m+n)/\log\log\log(m+n))$ and an update is $O(\log(m+n)(\log\log(m+n))^3)$.

Theorem 3.7. *Let A and B be two sequences of n and m points ($m \le n$), respectively, on the line. Then, $\widehat{d_{dF}}(A,B)$ can be computed in $O(m^2 n(1+\log(n/m)))$ time, $\widehat{d^s_{dF}}(A,B)$ in $O(mn\log(m+n))$ time, and $\widehat{d^w_{dF}}(A,B)$ in $O(mn\log(m+n)(\log\log(m+n))^3)$.*

3.5 A general scheme for BOP

In the previous section we showed that DFD, DFDS, and WDFD, all under translation and in 1D, can be viewed as BOP. In this section, we present a general scheme for BOP, which yields efficient algorithms quite easily, without having to devise an efficient dynamic version of the feasibility decider.

BOP's definition (see **??**) is especially suited for graphs, where, naturally, E is the set of weighted edges of the graph, and $\mathcal{F}$ is a family of well-defined structures, such as matchings, paths, spanning trees, cut-sets, edge covers, etc.

Let $G=(V,E,w)$ be a weighted graph, where V is a set of n vertices, E is a set of m edges, and $w: E \to \mathbb{R}$ is a weight function. Let $\mathcal{F}$ be a set of feasible subsets of E. For a subset $S \subseteq E$, let $S_{max} = \max\{w(e) : e \in S\}$ and $S_{min} = \min\{w(e) : e \in S\}$. The Balanced Optimization Problem on Graphs (BOPG) is to find a feasible subset $S^* \in \mathcal{F}$ which minimizes $S_{max} - S_{min}$ over all $S \in \mathcal{F}$. A range $[l,u]$ is a ***feasible range*** if there exists a feasible subset $S \in \mathcal{F}$ such that $w(e) \in [l,u]$ for each $e \in S$. A ***feasibility decider*** is an algorithm that decides whether a given range is feasible.

We assume for simplicity that each edge has a unique weight. Our goal is to find the smallest feasible range. First, we sort the m edges by their weights, and let $e_1, e_2, \ldots, e_m$ be the resulting sequence. Let $w_1 = w(e_1) < w_2 = w(e_2) < \cdots < w_m = w(e_m)$.

Let M be the matrix whose rows correspond to $w_1, w_2, \ldots, w_m$ and whose columns correspond to $w_1, w_2, \ldots, w_m$ (see Figure 3.3(a)). A cell $M_{i,j}$ of the matrix corresponds to the range $[w_i, w_j]$. Notice that some of the cells of M correspond to invalid ranges: when $i > j$, we have $w_i > w_j$ and thus $[w_i, w_j]$ is not a valid range.

M is sorted in the sense that range $M_{i,j}$ contains all the ranges $M_{i',j'}$ with $i \le i' \le j' \le j$. Thus, we can perform a binary search in the middle row to find the smallest feasible range $M_{\frac{m}{2},j} = [w_{\frac{m}{2}}, w_j]$ among the ranges in this row. $M_{\frac{m}{2},j}$ induces

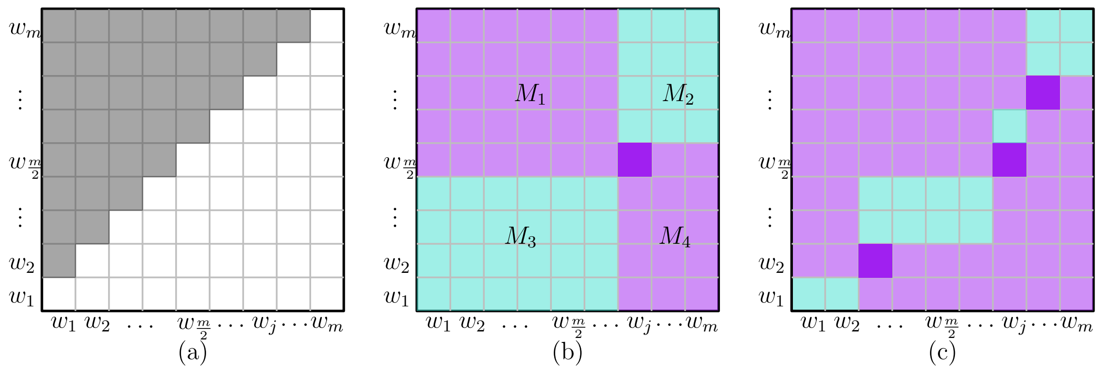


Figure 3.3: The matrix of possible ranges. (a) The shaded cells are invalid ranges. (b) The cell $M_{\frac{m}{2},j}$ induces a partition of M into 4 submatrices: M_1, M_2, M_3, M_4. (c) The four submatrices at the end of the second level of the recursion tree.

a partition of M into 4 submatrices: M_1, M_2, M_3, M_4 (see Figure 3.3(b)). Each of the ranges in M_1 is contained in a range of the middle row which is not a feasible range, hence none of the ranges in M_1 is a feasible range. Each of the ranges in M_4 contains $M_{\frac{m}{2},j}$ and hence is at least as large as $M_{\frac{m}{2},j}$. Thus, we may ignore M_1 and M_4 and focus only on the ranges in the submatrices M_2 and M_3.

Sketch of the algorithm. We perform a recursive search in the matrix M. The input to a recursive call is a submatrix M' of M and a corresponding graph G'. Let $[w_i, w_j]$ be a range in M'. The feasibility decider can decide whether $[w_i, w_j]$ is a feasible range or not by consulting the graph G'. In each recursive call, we perform a binary search in the middle row of M' to find the smallest feasible range in this row, using the corresponding graph G'. Then, we construct two new graphs for the two submatrices of M' in which we still need to search in the next level of the recursion.

The number of potential feasible ranges is equal to the number of cells in M, which is $O(m^2)$. But, since we are looking for the smallest feasible range, we do not need to generate all of them. We only use M to illustrate the search algorithm, its cells correspond to the potential feasible ranges, but do not contain any values. We thus represent M and its submatrices by the indices of the sorted list of weights that correspond to the rows and columns of M. For example, we represent M by $M([1,m] \times [1,m])$, M_2 by $M([\frac{m}{2}+1, m] \times [j, m])$, and M_3 by $M([1, \frac{m}{2}-1] \times [1, j-1])$. We define the *size* of a submatrix of M by the sum of its number of rows and number of columns, for example, M is of size $2m$, $|M_2| = \frac{3m}{2} - j + 1$, and $|M_3| = \frac{m}{2} + j - 2$.

Each recursive call is associated with a range of rows $[l, l']$ and a range of columns $[u', u]$ (the submatrix $M([l, l'] \times [u', u])$), and a corresponding input graph $G' = G([l, l'] \times [u', u])$. The scheme does not state which edges should be in G' or how to construct it, but it does require the followings properties:

1. The number of edges in G' should be $O(|M'|)$.

2. Given G', the feasibility decider can answer a feasibility query for any range in M', in $O(f(|G'|))$ time.

3. The construction of the graphs for the next level should take $O(|G'|)$ time.

The optimization scheme is given in Algorithm 3.1; its initial input is $G = G([1, m] \times [1, m])$.

Algorithm 3.1 Balance($G([l, l'] \times [u', u])$)

```
1. Set $i = \frac{l+l'}{2}$
2. Perform a binary search on the ranges $[i, j]$, $u' \leq j \leq u$, to find the smallest feasible
   range, using the feasibility decider with the graph $G([l, l'] \times [u', u])$ as input.
3. If there is no feasible range, then:
   (a) If $l = l'$, return $\infty$.
   (b) Else, construct $G_1 = G([l, i-1] \times [u', u])$ and return Balance($G_1$).
4. Else, let $[w_i, w_j]$ be the smallest feasible range found in the binary search.
   (a) If $l = l'$, return $(w_j - w_i)$.
   (b) Else, construct two new graphs, $G_1 = G([i+1, l'] \times [j, u])$ and $G_2 = G([l, i-1] \times$
       $[u', j-1])$, and return $\min\{(w_j - w_i), \text{Balance}(G_1), \text{Balance}(G_2)\}$.
```

Correctness. Let g be a bivariate real function with the property that for any four values of the weight function $c \leq a \leq b \leq d$, it holds that $g(a, b) \leq g(c, d)$. In our case, $g(a, b) = b - a$. We prove a somewhat more general theorem – that our scheme applies to any such monotone function g; for example, $g(a, b) = b/a$ (assuming the edge weights are positive numbers).

Theorem 3.8. *Algorithm 3.1 returns the minimum value $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$.*

Proof. We claim that given a graph $G' = G([l, l'] \times [u', u])$ as input, Algorithm 3.1 returns the minimal $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$, such that $S_{min} \in [l, l']$ and $S_{max} \in [u', u]$. Let $M' = M([l, l'] \times [u', u])$ be the corresponding matrix. The proof is by induction on the number of rows in M'.

First, notice that the algorithm runs the feasibility decider only on ranges from M'. The base case is when M' contains a single row, i.e. $l = l'$. In this case the algorithm returns the minimal feasible range $[w_l, w_j]$ such that $j \in [u', u]$, or returns ∞ if there is no such range. Else, M' has more than one row. Assume that there is no feasible range in the middle row of M'. In other words, there is no $j \in [u', u]$ such that $[w_i, w_j]$ is a feasible range. Trivially, for any $i' > i$ we have $w_{i'} > w_i$, and therefore for any $j \in [u', u]$, $[w_{i'}, w_j]$ is not a feasible range, and the algorithm continues recursively with $G_1 = G([l, i-1] \times [u', u])$. Now assume that $[w_i, w_j]$ is the minimal feasible range in the middle row. We can partition the ranges in M' to four types (submatrices):

1. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [i+1, l']$ and $j' \in [j, u]$.
2. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [l, i-1]$ and $j' \in [u', j-1]$.
3. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [i, l']$ and $j' \in [u', j-1]$. For any such valid range ($j' > i'$), we have $[w_{i'}, w_{j'}] \subseteq [w_i, w_j]$, so it is not a feasible range (otherwise, the result of the binary search would be $[w_i, w_{j'}]$).
4. All the ranges $[w_{i'}, w_{j'}]$ where $i' \in [l, i]$ and $j' \in [j, u]$. Since $j \geq i$, all these ranges are valid. For any such range, we have $w_{i'} \leq w_i \leq w_j \leq w_{j'}$, therefore, all these ranges are feasible, but since $g(w_i, w_j) \leq g(w_{i'}, w_{j'})$, there is no need to check them.

Indeed, the algorithm continues recursively with G_1 and G_2 (corresponding to ranges of type 1 and 2, respectively), which may contain smaller feasible ranges. By the induction hypothesis, the recursive calls return the minimal $g(S_{min}, S_{max})$ over all feasible subsets $S \in \mathcal{F}$, such that $S_{min} \in [i+1, l']$ and $S_{max} \in [j, u]$ or $S_{min} \in [l, i-1]$ and $S_{max} \in [u', j-1]$. Finally, the algorithm returns the minimum over all the feasible ranges in M'. □

Lemma 3.9. *The total size of the matrices in each level of the recursion tree is at most* $2m$.

Proof. By induction on the level. The only matrix in level 0 is M, and $|M| = 2m$. Let $M' = M([l, l'] \times [u', u])$ be a matrix in level $i-1$. The size of M' is $l' - l + u - u' + 2$ (it has $l' - l + 1$ rows and $u - u' + 1$ columns). In level i we perform a binary search in the middle row of M' to find the smallest feasible range $[w_{\frac{l+l'}{2}}, w_j]$ in this row. It is easy to see that the resulting two submatrices are of sizes $l' - \frac{l+l'}{2} + u - j + 1$ and $\frac{l+l'}{2} - l + j - u'$, respectively, which sums to $l' - l + u - u' + 1$. □

Running time. Consider the recursion tree. It consists of $O(\log m)$ levels, where the i'th level is associated with 2^i disjoint submatrices of M. Level 0 is associated with the matrix $M_0 = M$, level 1 is associated with the submatrices M_2 and M_3 of M (see Figure 3.3), etc.

In the i'th level we apply Algorithm 3.1 to each of the 2^i submatrices associated with this level. Let $\{M_k^i\}_{k=1}^{2^i}$ be the submatrices associated with the i'th level. Let G_k^i be the graph corresponding to M_k^i. The size of G_k^i is linear in the size of M_k^i. The feasibility decider runs in $O(f(|M_k^i|))$ time, and thus the binary search in M_k^i

runs in $O(f(|M_k^i|)\log|M_k^i|)$ time. Constructing the graphs for the next level takes $O(|M_k^i|)$ time. By Lemma 3.9, the total time spent on the i'th level is

$$\begin{aligned}O(\sum_{k=1}^{2^i}(|M_k^i| + f(|M_k^i|)\log|M_k^i|)) &\le O(\sum_{k=1}^{2^i}|M_k^i| + \sum_{k=1}^{2^i} f(|M_k^i|)\log m)\\ &= O(m + \log m\sum_{k=1}^{2^i} f(|M_k^i|)).\end{aligned}$$

Finally, the running time of the entire algorithm is

$$O(m\log m + \sum_{i=1}^{\log m}(m + \log m\sum_{k=1}^{2^i} f(|M_k^i|))) = O(m\log m + \log m\sum_{i=1}^{\log m}\sum_{k=1}^{2^i} f(|M_k^i|)).$$

Notice that the number of potential ranges is $O(m^2)$, while the number of weights is only $O(m)$. Nevertheless, whenever $f(|M'|)$ is a linear function, our optimization scheme runs in $O(m\log^2 m)$ time. More generally, whenever $f(|M'|)$ is a function for which $f(x_1) + \cdots + f(x_k) = O(f(x_1 + \cdots + x_k))$, for any $x_1, \ldots, x_k$, our scheme runs in $O(m\log m + f(2m)\log^2 m)$ time.

3.6 MUPP and WDFD under translation in 1D

In Section 3.4 we described an algorithm for WDFD under translation in 1D, which uses a dynamic data structure due to Thorup [Tho00]. In this section we present a much simpler algorithm for the problem, which avoids heavy tools and has roughly the same running time.

As shown in Section 3.4, WDFD under translation in 1D can be viewed as BOP. More precisely, we say that a range $[s,t]$ is a feasible range if (a_n, b_m) is a w-reachable position in G_w w.r.t. $\sigma_{[s,t]}$. Now, our goal is to find a feasible range of minimum size.

Consider the following weighted graph $\tilde{G}_w = (\tilde{V}_w, \tilde{E}_w, \omega)$, where $\tilde{V}_w = (A \times B) \cup \{v_e \mid e \in E_w\}$, $\tilde{E}_w = \{(u, v_e), (v_e, v) \mid e = (u,v) \in E_w\}$, and $\omega(((a_i, b_j), v_e)) = a_i - b_j$. In other words, $\tilde{G}_w$ is obtained from G_w by adding, for each edge $e = (u,v)$ of G_w, a new vertex v_e, which splits the edge into two new edges, $(u, v_e), (v_e, v)$, whose weight is the value associated with their original vertex (i.e., either u or v).

Now (a_n, b_m) is a w-reachable position in G_w w.r.t. $\sigma_{[s,t]}$, if and only if there exists a path P between (a_1, b_1) and (a_n, b_m) in G_w such that for each vertex $v \in P$, $\overline{v} \in [s,t]$, if and only if there exists a path $\tilde{P}$ between (a_1, b_1) and (a_n, b_m) in $\tilde{G}_w$ such that for each edge $e \in \tilde{P}$, $\omega(e) \in [s,t]$. Thus, we have reduced our problem to a special case of the Most Uniform Path Problem (MUPP).

Note that the technique used in Section 3.4 can also be applied to MUPP: Search in the sorted sequence of edge weights and use the reachability data structure of Thorup [Tho00] to obtain an $O(m\log n(\log\log n)^3)$-time algorithm. Below we show

how to apply our BOP scheme to MUPP, with a linear-time feasibility decider, to obtain a much simpler but slightly slower $O(m \log^2 n)$-time algorithm.

Here $\mathcal{F}$ is the set of paths in graph G between vertices s and t. The matrix for the initial call is M and G is its associated graph. Consider a recursive call, and let M' be the submatrix and G' the graph associated with it. Throughout the execution of the algorithm, we maintain the following properties:

1. The number of edges and vertices in G' is at most $O(|M'|)$, and
2. Given a range $[w_p, w_q]$ in M', there exists a path between s and t in G' with edges in the range $[w_p, w_q]$ if and only if such a path exists in G.

Construction of the graphs for the next level. Given the input graph G' and a submatrix $M'' = M([p, p'] \times [q', q])$ of M', we construct the corresponding graph G'' as follows: First, we remove from G' all the edges e such that $w(e) \notin [w_p, w_q]$. Then, we contract edges with weights in the range $(w_{p'}, w_{q'})$, and finally we remove all the isolated vertices. Notice that G'' is a graph minor of G', and, clearly, all the properties hold.

The feasibility decider. Let $[w_p, w_q]$ be a range from M'. Run a BFS in G', beginning from s, while ignoring edges with weights outside the range $[w_p, w_q]$. If the BFS finds t, return "yes", otherwise return "no". The algorithm returns "yes" if and only if there exists a path between s and t in G' with edges in the range $[w_p, w_q]$, i.e., if and only if such a path exists in G. The running time of the decider is $O(|G'|) = O(|M'|)$.

3.7 More applications

We have introduced an alternative optimization scheme for BOP and demonstrated its power. It would be interesting to find additional applications of this scheme. For example, consider the following problems:

Most uniform spanning tree. Given a graph G, find a spanning tree T^* of G, which minimizes $(\max\{w(e) : e \in T\} - \min\{w(e) : e \in T\})$ over all spanning trees T of G.

In 1986, Camerini et al. [CMMT86] presented an $O(mn)$-time algorithm for the problem. Later, by using an involved dynamic data structure, Galil and Schieber [GS88] showed how to reduce the running time to $O(m \log n)$.

Using our optimization scheme, in a quite straightforward manner, we obtain an $O(m \log^2 n)$ time algorithm. Although slower by a factor of $\log n$, our algorithm does not require any special data structures, and its description is easy and much shorter using the general optimization scheme.

In this case, $\mathcal{F}$ is the set of all spanning trees of G. The construction of the graphs for the recursive calls is similar to the construction in MUPP. The feasibility decider just has to check that G' has a connected spanning subgraph with edges in the given range. This can be done using a BFS or DFS algorithm, ignoring edges outside the range, in $O(|G'|) = O(|M'|)$ time.

A generalization of MUPP. Given a constant number of pairs of vertices $\{(s_i, t_i)\}_{i=1}^{k}$, find a minimum range $[l, u]$ such that for each $1 \leq i \leq k$, G contains a path between s_i and t_i with all edge weights in the range $[l, u]$. The algorithm above can be easily adapted for solving the above problem in $O(m \log^2 n)$ time.

Chapter 4

The Discrete Fréchet Gap

4.1 Introduction

We suggest a new variant of the discrete Fréchet distance — the *discrete Fréchet gap* (DFG for short). Returning to the frogs analogy, in the discrete Fréchet gap the leash is elastic and its length is determined by the distance between the frogs. When the frogs are at the same location, the length of the leash is zero. The rules governing the jumps are the same, i.e., traverse all the points in order, no backtracking. We are interested in the minimum *gap* of the leash, i.e., the minimum difference between the longest and shortest positions of the leash needed for the frogs to traverse their corresponding sequences.

We use the graph definition from Chapter 3 to formally define the discrete Fréchet gap, as follows. Given two sequences of points $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$, the ***discrete Fréchet gap*** between them, $d_{dFg}(A, B)$, is the size of a smallest range $[s, t]$, $0 \leq s \leq t$, for which (a_n, b_m) is a reachable position w.r.t. the following indicator function:

$$\hat{\sigma}_{[s,t]}(a_i, b_j) = \begin{cases} 1, & s \leq d(a_i, b_j) \leq t \\ 0, & \text{otherwise} \end{cases} .$$

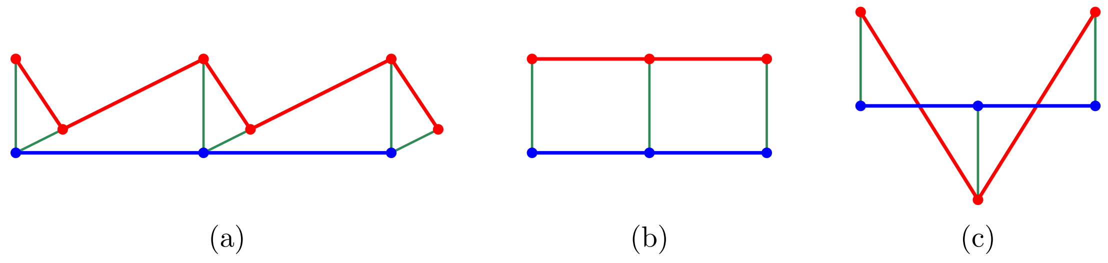


Figure 4.1: (a) Two non-similar curves, with large gap and large distance. (b) Two similar curves. The gap is zero while the distance remains the same as in (b). (c) Two non-similar curves with small gap and large distance.

While the discrete Fréchet distance is determined by the (matched) pairs of points that are very far from each other and is indifferent towards (matched) pairs of points that are very close to each other, the discrete Fréchet gap measure is sensitive to both. In some cases (though not always), this sensitivity results in better reflection of reality; see Figure 4.1 for examples.

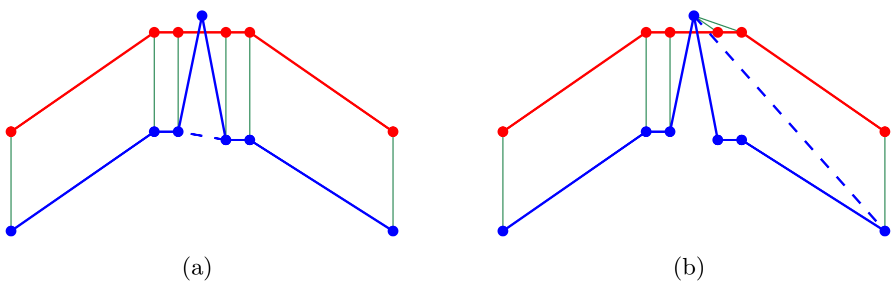


Figure 4.2: (a) The 1-sided Fréchet gap with shortcuts is small and the outlier is ignored. (b) The 1-sided Fréchet distance with shortcuts is large and the outlier is matched.

For handling outliers, we suggest the ***one-sided discrete Fréchet gap with shortcuts*** variant. Comparing to the one-sided discrete Fréchet distance with shortcuts, we believe that the gap variant better reflects the intuitive notion of resemblance between curves in the presence of outliers. Roughly, the gap measure is more suitable for detecting outliers, and by enabling shortcuts one can neutralize them. Figure 4.2 depicts two curves that look similar, except for a single outlier, with small Fréchet gap with shortcuts and large Fréchet distance with shortcuts.

Also notice that the gap variant gives a more "natural" matching of the points, which better captures the similarity between the curves. In general, since the Fréchet distance is determined by the maximum distance between (matched) points, there can be many different Fréchet matchings, not all of them are useful. It has been noted before, and some solutions have been suggested, for example see [BBMS12] and [BBvL$^+$13].

Other variants of the discrete Fréchet distance have corresponding meaningful gap variants. For example, the *weak discrete Fréchet distance* in which the frogs are allowed to jump also backwards to the previous point in their sequence.

Recently, Fan and Raichel [FR17] considered the continuous Fréchet gap. They gave an $O(n^5 \log n)$ time exact algorithm and a more efficient $O(n^2 \log n + \frac{n^2}{\varepsilon} \log \frac{1}{\varepsilon})$-time $(1+\varepsilon)$-approximation algorithm for computing it, where n is the total number of vertices of the input curves.

4.2 DFG and DFD under translation

The following theorem reveals a connection between the discrete Fréchet gap and the discrete Fréchet distance under translation.

Theorem 4.1. *For any two sequences A and B of points in $\mathbb{R}^d$, $\widehat{d_{dF}}(A,B) \geq d_{dFg}(A,B)/2$.*

Proof. Let $d_{dFg}(A,B)$ be determined by the range $[s,t]$, and denote $\delta = \widehat{d_{dF}}(A,B)$. Assume by contradiction that $\delta < (t-s)/2$. Then, by Lemma 3.6, there exists a point o such that (a_n, b_m) is a reachable position w.r.t. $\sigma_{S(o,\delta)}$. In other words, there exists a path P in G from (a_1, b_1) to (a_n, b_m), such that for each vertex (a_i, b_j) in P it holds that $d(a_i - b_j, o) \leq \delta < (t-s)/2$, i.e., $\|(a_i - b_j) - o\| \leq \delta < (t-s)/2$. Thus, by the triangle inequality, $\|o\| - (t-s)/2 < \|a_i - b_j\| < \|o\| + (t-s)/2$, which means that there exists a range $[s', t'] \subset [s,t]$ such that for each vertex (a_i, b_j) in P it holds that $s' \leq d(a_i, b_j) \leq t'$. In other words, (a_n, b_m) is a reachable position w.r.t. $\hat{\sigma}_{[s',t']}$, which contradicts the assumption that $d_{dFg}(A,B) = t - s$. □

Most variants of the (original) Fréchet distance (shortcuts, weak, partial, etc.) have a natural gap counterpart: instead of recording the maximum length of the leash in a walk, we record the difference between the maximum length and the minimum length.

We denote by $d^s_{dFg}(A,B)$ and $d^w_{dFg}(A,B)$ the *discrete Fréchet gap with shortcuts* (DFGS) and *weak discrete Fréchet gap* (WDFG) variants, respectively, between two sequences of points A and B.

It is interesting that DFD, DFDS, and WDFD, all in 1D under translation, are in some sense analogous to their respective gap variants (DFG, DFGS, and WDFG, in d dimensions and no translation). We can use algorithms similar to those presented in Chapter 3 in order to compute them, but with the indicator function $\hat{\sigma}_{[s,t]}$. Observe that since we are interested in the minimum feasible range, we may restrict our attention to ranges whose limits are distances between points of A and points of B. (Otherwise, we can increase the lower limit and decrease the upper limit until they become such ranges.) Thus, we can search for the minimum feasible range with boundaries in the set $\hat{\mathcal{D}} = \{d(a_i, b_j) | a_i \in A, b_j \in B\}$. As in Section 3.4, we can use the search algorithm on $\hat{\mathcal{D}}$ (instead of on $\mathcal{D}$), together with a suitable data structure using the indicator function $\hat{sigma}_{[s,t]}$, in order to solve DFG and its variants. The running times are thus similar to those in Section 3.4.

Theorem 4.2. *Let A and B be two sequences of n and m points ($m \leq n$), respectively. Then, $d_{dFg}(A,B)$ can be computed in $O(m^2 n(1 + \log(n/m)))$ time, $d^s_{dFg}(A,B)$ in $O(mn \log(m+n))$, and $d^w_{dFg}(A,B)$ in $O(mn \log(m+n)(\log\log(m+n))^3)$ time.*

Remark 4.3. Our algorithms can also be used for computing the *discrete Fréchet ratio* (and its variants), in which we are interested in the minimum ratio between the longest and the shortest positions of the leash. More generally, one can replace the gap function with any other function g defined for pairs of distances, provided that it is monotone, i.e., for any four distances $c \leq a \leq b \leq d$, it holds that $g(a,b) \leq g(c,d)$.

Dealing with Big (Trajectory) Data

Chapter 5

Approximate Near-Neighbor for Curves

5.1 Introduction

Nearest neighbor search is a fundamental and well-studied problem that has various applications in machine learning, data analysis, and classification. Such analysis of curves has many practical applications, where the position of an object as it changes over time is recorded as a sequence of readings from a sensor to generate a *trajectory*. For example, the location readings from GPS devices attached to migrating animals [ABB$^+$14], the traces of players during a football match captured by a computer vision system [GH17], or stock market prices [NW13]. In each case, the output is an ordered sequence C of m vertices (i.e., the sensor readings), and by interpolating the location between each pair of vertices as a segment, a polygonal chain is obtained.

Let $\mathcal{C}$ be a set of n curves, each consisting of m points in d dimensions, and let δ be some distance measure for curves. In the *nearest-neighbor* problem for curves, the goal is to construct a data structure for $\mathcal{C}$ that supports nearest-neighbor queries, that is, given a query curve Q of length m, return the curve $C^* \in \mathcal{C}$ closest to Q (according to δ). The approximation version of this problem is the $(1+\varepsilon)$-*approximate nearest-neighbor* problem, where the answer to a query Q is a curve $C \in \mathcal{C}$ with $\delta(Q, C) \leq (1+\varepsilon)\delta(Q, C^*)$. We study a decision version of this approximation problem, which is called the $(1+\varepsilon, r)$-*approximate near-neighbor* problem for curves. Here, if there exists a curve in $\mathcal{C}$ that lies within distance r of the query curve Q, one has to return a curve in $\mathcal{C}$ that lies within distance $(1+\varepsilon)r$ of Q.

Note that there exists a reduction from the $(1+\varepsilon)$-approximate nearest-neighbor problem to the $(1+\varepsilon, r)$-approximate near-neighbor problem [Ind00, SDI06, HPIM12], at the cost of an additional logarithmic factor in the query time and an $O(\log^2 n)$ factor in the storage space.

It was shown in [IM04, DKS16] that unless the strong exponential time hypothesis fails, nearest neighbor under DFD is hard to approximate within a factor of $c < 3$, with a data structure requiring $O(n^{2-\varepsilon} \operatorname{polylog} m)$ preprocessing and $O(n^{1-\varepsilon} \operatorname{polylog} m)$ query time, for $\varepsilon > 0$.

Indyk [Ind02] gave a deterministic near-neighbor data structure for curves under DFD. The data structure achieves an approximation factor of $O((\log m + \log\log n)^{t-1})$ given some trade-off parameter $t > 1$. Its space consumption is very high, $O(m^2|X|)^{tm^{1/t}} \cdot n^{2t}$, where $|X|$ is the size of the domain on which the curves are defined, and the query time is $(m \log n)^{O(t)}$. In Table 5.1 we set $t = 1 + o(1)$ to obtain a constant approximation factor.

Later, Driemel and Silvestri [DS17] presented a locality-sensitive-hashing scheme for curves under DFD, improving the result of Indyk for short curves. Their data structure uses $O(2^{4md}mn \log n + mn)$ space and answers queries in $O(2^{4md}m \log n)$ time with an approximation factor of $O(d^{3/2})$. They also provide a trade-off between approximation quality and computational performance: for a parameter $k \in [m]$, a data structure that uses $O(2^{2k}m^{k-1}n \log n + mn)$ space is constructed that answers queries in $O(2^{2k}m^k \log n)$ time with an approximation factor of $O(d^{3/2}m/k)$. They also show that this result can be applied to DTW, but only for one extreme of the trade-off which gives $O(m)$ approximation.

Recently, Emiris and Psarros [EP18] presented near-neighbor data structures for curves under both DFD and DTW distance. Their algorithm provides approximation factor of $(1 + \varepsilon)$, at the expense of increasing space usage and preprocessing time. The idea is that for a fixed alignment between two curves (i.e. a given sequence of hops of the two frogs), the problem can be reduced to near-neighbor problem on points in ℓ_∞ (in a higher dimension). Their basic idea is to construct a data structure for all possible alignments. Once a query is given, they query all these data structures and return the closest curve found. This approach is responsible for the 2^m factor in their query time. Furthermore, they generalize this approach using randomized projections of l_p-products of Euclidean metrics (for any $p \geq 1$), and define the $\ell_{p,2}$-distance for curves (for $p \geq 1$), which is exactly DFD when $p = \infty$, and DTW distance when $p = 1$ (see Section 5.2). The space used by their data structure is $\tilde{O}(n) \cdot (2 + \frac{d}{\log m})^{O(m^{1/\varepsilon} \cdot d \log(1/\varepsilon))}$ for DFD and $\tilde{O}(n) \cdot O(\frac{1}{\varepsilon})^{md}$ for DTW, while the query time in both cases is $O(d \cdot 2^{2m} \log n)$.

De Berg, Gudmundsson, and Mehrabi [dBGM17] described a dynamic data structure for approximate nearest neighbor for curves (which can also be used for other types of queries such as range reporting), under the (continuous) Fréchet distance. Their data structure uses $n \cdot O\left(\frac{1}{\varepsilon}\right)^{2m}$ space and has $O(m)$ query time, but with an additive error of $\varepsilon \cdot \mathit{reach}(Q)$, where $\mathit{reach}(Q)$ is the maximum distance between the start vertex of the query curve Q and any other vertex of Q. Furthermore, their query procedure might fail when the distance to the nearest neighbor is relatively large.

Afshani and Driemel [AD18] studied (exact) range searching under both the discrete and continuous Fréchet distance. In this problem, the goal is to preprocess $\mathcal{C}$ such that given a query curve Q of length m_q and a radius r, all the curves in $\mathcal{C}$ that are within distance r from Q can be found efficiently. For DFD, their data structure uses $O(n(\log\log n)^{m-1})$ space and has $O(n^{1-\frac{1}{d}} \cdot \log^{O(m)} n \cdot m_q^{O(d)})$ query time, where m_q is limited to $\log^{O(1)} n$. Additionally, they provide a lower bound in the pointer model, stating that every data structure with $Q(n) + O(k)$ query time, where k is the output size, has to use roughly $\Omega\left((n/Q(n))^2\right)$ space in the worst case. Afshani and Driemel conclude their paper by asking whether more efficient data structures might be constructed if one allows approximation.

De Berg, Cook IV and Gudmundsson [dBIG13] considered the following approximation version of range counting for curves under the (continuous) Fréchet distance. Given a collection of polygonal curves $\mathcal{C}$ with a total number of n vertices in the plane, preprocess $\mathcal{C}$ into a data structure such that given a threshold value r and query segment Q of length at least $6r$, returns the number of all the inclusion-minimal subcurves of the curves in $\mathcal{C}$ whose Fréchet distance to Q is at most r, plus possibly additional subcurves whose Fréchet distance to Q is up to $(2+3\sqrt{2})r$. Each subcurve of a curve $C \in \mathcal{C}$ is a connected subset of $\mathcal{C}$, and the endpoints of a subcurve can lie in the interior of one of C's segments. For any parameter $n \leq s \leq n^2$, the space used by the data structure is in $O(s \text{ polylog } n)$, the preprocessing time is $O(n^3 \log n)$, and the queries are answered in $O(\frac{n}{\sqrt{s}} \text{ polylog } n)$ time.

Our results. We present a data structure for the $(1+\varepsilon, r)$-approximate near-neighbor problem using a bucketing method. We construct a relatively small set of curves $\mathcal{I}$ such that given a query curve Q, if there exists some curve in $\mathcal{C}$ within distance r of Q, then one of the curves in $\mathcal{I}$ must be very close to Q. The points of the curves in $\mathcal{I}$ are chosen from a simple discretization of space, thus, while it is not surprising that we get the best query time, it is surprising that we achieve a better space bound. See Table 5.1 for a summary of our results. In the table, we do not state our result for the general $\ell_{p,2}$-distance. Instead, we state our results for the two most important cases, i.e. DFD and DTW, and compare them with previous work. Note that our results substantially improve the current state of the art for any $p \geq 1$. In particular, we remove the exponential dependence on m in the query bounds and significantly improve the space bounds.

We also apply our methods to an approximation version of range counting for curves (for the general $\ell_{p,2}$ distance) and achieve bounds similar to those of our ANNC data structure. Moreover, at the cost of an additional $O(n)$-factor in the space bound, we can also answer approximate range searching queries, thus answering the question of Afshani and Driemel [AD18] (see above), with respect to the discrete Fréchet distance.

Finally, note that our approach with obvious modifications works also in a dynamic setting, that is, we can construct a dynamic data structure for ANNC as well as for other related problems such as range counting and range reporting for curves.

	Space	Query	Approx.	Comments
DFD	$O(m^2\|X\|)^{m^{1-o(1)}} \cdot n^{2-o(1)}$	$(m \log n)^{O(1)}$	$O(1)$	deterministic, [Ind02]
	$O(2^{4md} n \log n + nm)$	$O(2^{4md} m \log n)$	$O(d^{3/2})$	randomized, using LSH [DS17]
	$\tilde{O}(n) \cdot (2 + \frac{d}{\log m})^{O(m^{1/\varepsilon} \cdot d \log(1/\varepsilon))}$	$O(d \cdot 2^{2m} \log n)$	$1+\varepsilon$	randomized, [EP18]
	$n \cdot O(\frac{1}{\varepsilon})^{md}$	$O(md \log(\frac{mnd}{\varepsilon}))$	$1+\varepsilon$	deterministic, Theorem 5.8
DTW	$O(2^{4md} n \log n + nm)$	$O(2^{4md} m \log n)$	$O(m)$	randomized, using LSH [DS17]
	$\tilde{O}(n) \cdot O(\frac{1}{\varepsilon})^{md}$	$O(d \cdot 2^{2m} \log n)$	$1+\varepsilon$	randomized, [EP18]
	$n \cdot O(\frac{1}{\varepsilon})^{md}$	$O(md \log(\frac{mnd}{\varepsilon}))$	$1+\varepsilon$	deterministic, Theorem 5.12

Table 5.1: Our approximate near-neighbor data structure under DFD and DTW compared to the previous results.

Organization. We begin by presenting our data structure for the special case where the distance measure is DFD (Section 5.3), since this case is more intuitive. Then, we apply the same approach to the case where the distance measure is $\ell_{p,2}$-distance, for any $p \geq 1$ (Section 5.4). Surprisingly, we achieve the exact same time and space bounds, without any dependence on p. Finally, we show that a similar data structure can be used in order to solve a version of approximate range counting for curves (Section 5.5).

5.2 Preliminaries

A formal definition of the discrete Fréchet distance was given in Section 1.1, and a different equivalent one was used in Sections 2.2 and 3.2. In this chapter, the definition of DFD is rather different from graph definition, and uses the notion of alignment between curves.

First note that in order to simplify the presentation, we assume throughout the chapter that all the input and query curves have exactly the same size, but this assumption can be easily removed.

Let $\mathcal{C}$ be a set of n curves, each consists of m points in d dimensions, and let δ be some distance measure for curves.

Problem 5.1 ($(1+\varepsilon)$-approximate nearest-neighbor for curves)**.** Given a parameter $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q, returns a curve $C' \in \mathcal{C}$, such that $\delta(Q, C') \leq (1+\varepsilon) \cdot \delta(Q, C)$, where C is the curve in $\mathcal{C}$ closest to Q.

Problem 5.2 ($(1+\varepsilon, r)$-approximate near-neighbor for curves)**.** Given a parameter r and $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q, if there exists a curve $C_i \in \mathcal{C}$ such that $\delta(Q, C_i) \leq r$, returns a curve $C_j \in \mathcal{C}$ such that $\delta(Q, C_j) \leq (1+\varepsilon)r$.

Curve alignment. Given an integer m, let $\tau := \langle (i_1, j_1), \ldots, (i_t, j_t) \rangle$ be a sequence of pairs where $i_1 = j_1 = 1$, $i_t = j_t = m$, and for each $1 < k \leq t$, one of the following properties holds:

(i) $i_k = i_{k-1} + 1$ and $j_k = j_{k-1}$,

(ii) $i_k = i_{k-1}$ and $j_k = j_{k-1} + 1$, or

(iii) $i_k = i_{k-1} + 1$ and $j_k = j_{k-1} + 1$.

We call such a sequence τ an *alignment* of two curves.

Let $P = (p_1, \ldots, p_m)$ and $Q = (q_1, \ldots, q_m)$ be two curves of length m in d dimensions.

Discrete Fréchet distance (DFD). The ***Fréchet cost*** of an alignment τ w.r.t. P and Q is $\sigma_{dF}(\tau) := \max_{(i,j)\in\tau} \|p_i - q_j\|_2$. The discrete Fréchet distance is defined over the set $\mathcal{T}$ of all alignments as

$$d_{dF}(P, Q) = \min_{\tau\in\mathcal{T}} \sigma_{dF}(\tau).$$

Dynamic time wrapping (DTW). The ***time warping cost*** of an alignment τ w.r.t. P and Q is $\sigma_{DTW}(\tau) := \sum_{(i,j)\in\tau} \|p_i - q_j\|_2$. The DTW distance is defined over the set $\mathcal{T}$ of all alignments as

$$d_{DTW}(P, Q) = \min_{\tau\in\mathcal{T}} \sigma_{DTW}(\tau).$$

$\ell_{p,2}$-distance for curves. The ***$\ell_{p,2}$-cost*** of an alignment τ w.r.t. P and Q is $\sigma_{p,2}(\tau) := \left(\sum_{(i,j)\in\tau} \|p_i - q_j\|_2^p\right)^{1/p}$. The $\ell_{p,2}$-distance between P and Q is defined over the set $\mathcal{T}$ of all alignments as

$$d_{p,2}(P, Q) = \min_{\tau\in\mathcal{T}} \sigma_{p,2}(\tau).$$

Notice that $\ell_{p,2}$-distance is a generalization of DFD and DTW, in the sense that $\sigma_{dF} = \sigma_\infty$ and $d_{dF} = d_\infty$, $\sigma_{DTW} = \sigma_1$ and $d_{DTW} = d_1$. Also note that DFD satisfies the triangle inequality, but DTW and $\ell_{p,2}$-distance (for $p \neq \infty$) do not.

Emiris and Psarros [EP18] showed that the total number of all possible alignments between two curves is in $O(m{\cdot}2^{2m})$. We reduce this bound by counting only alignments that can determine the $\ell_{p,2}$-distance between two curves. More formally, let τ be a curve alignment. If there exists a curve alignment τ' such that $\tau' \subset \tau$, then clearly $\sigma_p(\tau') \leq \sigma_p(\tau)$, for any $1 \leq p \leq \infty$ and w.r.t. any two curves. In this case, we say that τ cannot determine the $\ell_{p,2}$-distance between two curves.

Lemma 5.3. *The number of different alignments that can determine the $\ell_{p,2}$-distance between two curves (for any $1 \leq p \leq \infty$) is at most $O(\frac{2^{2m}}{\sqrt{m}})$.*

Proof. Let $\tau = \langle (i_1, j_1), \ldots, (i_t, j_t) \rangle$ be a curve alignment. Notice that $m \leq t \leq 2m-1$. By definition, τ has 3 types of (consecutive) subsequences of length two:

(i) $\langle (i_k, j_k), (i_k + 1, j_k) \rangle$,

(ii) $\langle (i_k, j_k), (i_k, j_k + 1) \rangle$, and

(iii) $\langle (i_k, j_k), (i_k + 1, j_k + 1) \rangle$.

Denote by $\mathcal{T}_1$ the set of all alignments that do not contain any subsequence of type (iii). Then, any $\tau_1 \in \mathcal{T}_1$ is of length exactly $2m - 1$. Moreover, τ_1 contains exactly $2m - 2$ subsequences of length two, of which $m - 1$ are of type (i) and $m - 1$ are of type (ii). Therefore, $|\mathcal{T}_1| = \binom{2m-2}{m-1} = O(\frac{2^{2m}}{\sqrt{m}})$.

Assume that a curve alignment τ contains a subsequence of the form $(i_k, j_k - 1), (i_k, j_k), (i_k + 1, j_k)$, for some $1 < k \leq t - 1$. Notice that removing the pair (i_k, j_k) from τ results in a legal curve alignment τ', such that $\sigma_p(\tau') \leq \sigma_p(\tau)$, for any $1 \leq p \leq \infty$. We call the pair (i_k, j_k) a *redundant pair*. Similarly, if τ contains a subsequence of the form $(i_k - 1, j_k), (i_k, j_k), (i_k, j_k + 1)$, for some $1 < k \leq t - 1$, then the pair (i_k, j_k) is also a redundant pair. Therefore we only care about alignments that do not contain any redundant pairs. Denote by $\mathcal{T}_2$ the set of all alignments that do not contain any redundant pairs, then any $\tau_2 \in \mathcal{T}_2$ contains at least one subsequence of type (iii).

We claim that for any alignment $\tau_2 \in \mathcal{T}_2$, there exists a unique alignment $\tau_1 \in \mathcal{T}_1$. Indeed, if we add the redundant pair $(i_l, j_l + 1)$ between (i_l, j_l) and $(i_l + 1, j_l + 1)$ for each subsequence of type (iii) in τ_2, we obtain an alignment $\tau_1 \in \mathcal{T}_1$. Moreover, since τ_2 does not contain any redundant pairs, the reverse operation on τ_1 results in τ_2. Thus we obtain $|\mathcal{T}_2| \leq |\mathcal{T}_1| = O(\frac{2^{2m}}{\sqrt{m}})$. ☐

Points and balls. Given a point $x \in \mathbb{R}^d$ and a real number $R > 0$, we denote by $B_p^d(x, R)$ the d-dimensional ball under the ℓ_p norm with center x and radius R, i.e., a point $y \in \mathbb{R}^d$ is in $B_p^d(x,R)$ if and only if $\|x-y\|_p \le R$, where $\|x-y\|_p = \left(\sum_{i=1}^d |x_i - y_i|^p\right)^{1/p}$. Let $B_p^d(R) = B_p^d(0,R)$, and let $V_p^d(R)$ be the volume (w.r.t. Lebesgue measure) of $B_p^d(R)$, then

$$V_p^d(R) = \frac{2^d \Gamma(1+1/p)^d}{\Gamma(1+d/p)} R^d,$$

where $\Gamma(\cdot)$ is Euler's Gamma function (an extension of the factorial function). For $p=2$ and $p=1$, we get

$$V_2^d(R) = \frac{\pi^{d/2}}{\Gamma(1+d/2)} R^d \quad \text{and} \quad V_1^d(R) = \frac{2^d}{d!} R^d.$$

Our approach consists of a discretization of the space using lattice points, i.e., points from $\mathbb{Z}^d$.

Lemma 5.4. *The number of lattice points in the d-dimensional ball of radius R under the ℓ_p norm (i.e., in $B_p^d(R)$) is bounded by $V_p^d(R + d^{1/p})$.*

Proof. With each lattice point $z = (z_1, z_2, \dots, z_d)$, $z_i \in \mathbb{Z}$, we match the d-dimensional lattice cube $C(z) = [z_1, z_1+1] \times [z_2, z_2+1] \times \cdots \times [z_d, z_d+1]$. Notice that $z \in C(z)$, and the ℓ_p-diameter of a lattice cube is $d^{1/p}$. Therefore, the number of lattice points in the ℓ_p^d-ball of radius R is bounded by the number of lattice cubes that are contained in a ℓ_p^d-ball with radius $R + d^{1/p}$. This number is bounded by $V_p^d(R + d^{1/p})$ divided by the volume of a lattice cube, which is $1^d = 1$. $\square$

Remark 5.5. In general, when the dimension d is large, i.e. $d \gg \log n$, one can use dimension reduction (using the celebrated Johnson-Lindenstrauss lemma [JL84]) in order to achieve a better running time, at the cost of inserting randomness to the prepossessing and query. However, such an approach can work only against oblivious adversary, as it will necessarily fail for some curves. Recently Narayanan and Nelson [NN18] (improving [EFN17, MMMR18]) proved a terminal version of the JL-lemma. Given a set K of k points in $\mathbb{R}^d$ and $\varepsilon \in (0,1)$, there is a dimension reduction function $f : \mathbb{R}^d \to \mathbb{R}^{O(\frac{\log k}{\varepsilon^2})}$ such that for every $x \in K$ and $y \in \mathbb{R}^d$ it holds that $\|x-y\|_2 \le \|f(x) - f(y)\|_2 \le (1+\varepsilon) \cdot \|x-y\|_2$.

This version of dimension reduction can be used such that the query remains deterministic and always succeeds. The idea is to take all the nm points from all the input curves to be the terminals, and let f be the terminal dimension reduction. We transform each input curve $P = (p_1, \dots, p_m)$ into $f(P) = (f(p_1), \dots, f(p_m))$, a curve in $\mathbb{R}^{O(\frac{\log nm}{\varepsilon^2})}$. Given a query $Q = (q_1, \dots, q_m)$ we transform it to $f(Q) = (f(q_1), \dots, f(q_m))$. Since the pairwise distances between every query point to all input points are preserved, so is the distance between the curves. Specifically, the $\ell_{p,2}$-cost of any alignment τ is preserved up to a $1+\varepsilon$ factor, and therefore we can reliably use the answer received using the transformed curves.

5.3 ANNC under the discrete Fréchet distance

Consider the infinite d-dimensional grid with edge length $\frac{\varepsilon r}{\sqrt{d}}$.Given a point x in $\mathbb{R}^d$, by rounding one can find in $O(d)$ time the grid point closest to x. Let $G(x,R)$ denote the set of grid points that are contained in $B_2^d(x,R)$.

Corollary 5.6. $|G(x,(1+\varepsilon)r)| = O(\frac{1}{\varepsilon})^d$.

Proof. We scale our grid so that the edge length is 1, hence we are looking for the number of lattice points in $B_2^d(x, \frac{1+\varepsilon}{\varepsilon}\sqrt{d})$. By Lemma 5.4 we get that this number is bounded by the volume of the d-dimensional ball of radius $\frac{1+\varepsilon}{\varepsilon}\sqrt{d} + \sqrt{d} \le \frac{3\sqrt{d}}{\varepsilon}$. Using Stirling's formula we conclude,

$$V_2^d\left(\frac{3\sqrt{d}}{\varepsilon}\right) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} \cdot \left(\frac{3\sqrt{d}}{\varepsilon}\right)^d = \left(\frac{\alpha}{\varepsilon}\right)^d$$

where α is a constant (approximately 12.4). □

Denote by p_j^i the j'th point of C_i, and let $G_i = \bigcup_{1\le j\le m} G(p_j^i,(1+\varepsilon)r)$ and $\mathcal{G} = \bigcup_{1\le i\le n} G_i$, then by the above corollary we have $|G_i| = m\cdot O(\frac{1}{\varepsilon})^d$ and $|\mathcal{G}| = mn\cdot O(\frac{1}{\varepsilon})^d$. Let $\mathcal{I}_i$ be the set of all curves $\overline{Q} = (x_1, x_2, \ldots, x_m)$ with points from G_i, such that $d_{dF}(C_i, \overline{Q}) \le (1+\frac{\varepsilon}{2})r$.

Claim 5.7. $|\mathcal{I}_i| = O(\frac{1}{\varepsilon})^{md}$ *and it can be computed in* $O(\frac{1}{\varepsilon})^{md}$ *time.*

Proof. Let $\overline{Q} \in \mathcal{I}_i$ and let τ be an alignment with $\sigma_{dF}(\tau) \le (1+\frac{\varepsilon}{2})r$ w.r.t. C_i and $\overline{Q}$. For each $1 \le k \le m$ let j_k be the smallest index such that $(j_k, k) \in \tau$. In other words, j_k is the smallest index that is matched to k by the alignment τ. Since $d_{dF}(C_i, \overline{Q}) \le (1+\frac{\varepsilon}{2})r$, we have $x_k \in B_2^d(p_{j_k}^i, (1+\frac{\varepsilon}{2})r)$, for $k = 1, \ldots, m$. This means that for any curve $\overline{Q} \in \mathcal{I}_i$ such that $\sigma_{dF}(\tau) \le (1+\frac{\varepsilon}{2})r$ w.r.t. C_i and $\overline{Q}$, we have $x_k \in G(p_{j_k}^i, (1+\frac{\varepsilon}{2})r)$, for $k = 1, \ldots, m$. By Corollary 5.6, the number of ways to choose a grid point x_k from $G(p_{j_k}^i, (1+\frac{\varepsilon}{2})r)$ is bounded by $O(\frac{1}{\varepsilon})^d$.

We conclude that given an alignment τ, the number of curves $\overline{Q}$ with m points from G_i such that $\sigma_{dF}(\tau) \le (1+\frac{\varepsilon}{2})r$ w.r.t. C_i and $\overline{Q}$ is bounded by $O(\frac{1}{\varepsilon})^{md}$. Finally, by Lemma 5.3, the total number of curves in $\mathcal{I}_i$ is bounded by $2^{2m}\cdot O(\frac{1}{\varepsilon})^{md} = O(\frac{1}{\varepsilon})^{md}$. □

The data structure. Denote $\mathcal{I} = \bigcup_{1\le i\le n} \mathcal{I}_i$, so $|\mathcal{I}| = n\cdot O(\frac{1}{\varepsilon})^{md}$. We construct a prefix tree $\mathcal{T}$ for the curves in $\mathcal{I}$, as follows. For each $1 \le i \le n$ and curve $\overline{Q} \in \mathcal{I}_i$, if $\overline{Q} \notin \mathcal{T}$, insert $\overline{Q}$ to $\mathcal{T}$, and set $C(\overline{Q}) \leftarrow C_i$.

Each node $v \in \mathcal{T}$ corresponds to a grid point from $\mathcal{G}$. Denote the set of v's children by $N(v)$. We store with v a multilevel search tree on $N(v)$, with a level for each coordinate. The points in $\mathcal{G}$ are the grid points contained in nm balls of radius $(1+\varepsilon)r$. Thus when projecting these points to a single dimension, the number of 1-dimensional points is at most $nm \cdot \frac{\sqrt{d}(1+\varepsilon)r}{\varepsilon r} = O(\frac{nm\sqrt{d}}{\varepsilon})$. So in each level of the search tree on $N(v)$ we have $O(\frac{nm\sqrt{d}}{\varepsilon})$ 1-dimensional points, so the query time is $O(d\log(\frac{nmd}{\varepsilon}))$.

Inserting a curve of length m to the tree $\mathcal{T}$ takes $O(md\log(\frac{nmd}{\varepsilon}))$ time. Since $\mathcal{T}$ is a compact representation of $|\mathcal{I}| = n \cdot O(\frac{1}{\varepsilon})^{dm}$ curves of length m, the number of nodes in $\mathcal{T}$ is $m \cdot |\mathcal{I}| = nm \cdot O(\frac{1}{\varepsilon})^{dm}$. Each node $v \in \mathcal{T}$ contains a search tree for its children of size $O(d \cdot |N(v)|)$, and $\sum_{v\in\mathcal{T}} |N(v)| = nm \cdot O(\frac{1}{\varepsilon})^{dm}$ so the total space complexity is $O(nmd) \cdot O(\frac{1}{\varepsilon})^{md} = n \cdot O(\frac{1}{\varepsilon})^{md}$. Constructing $\mathcal{T}$ takes $O(|\mathcal{I}| \cdot md\log(nmd/\varepsilon)) = n\log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ time.

The query algorithm. Let $Q = (q_1, \dots, q_m)$ be the query curve. The query algorithm is as follows: For each $1 \le k \le n$ find the grid point q'_k (not necessarily from $\mathcal{G}$) closest to q_k. This can be done in $O(md)$ time by rounding. Then, search for the curve $Q' = (q'_1, \dots, q'_m)$ in the prefix tree $\mathcal{T}$. If Q' is in $\mathcal{T}$, return $C(Q')$, otherwise, return NO. The total query time is then $O(md\log(\frac{nmd}{\varepsilon}))$.

Correctness. Consider a query curve $Q = (q_1, \dots, q_m)$. Assume that there exists a curve $C_i \in \mathcal{C}$ such that $d_{dF}(C_i, Q) \le r$. We show that the query algorithm returns a curve C^* with $d_{dF}(C^*, Q) \le (1+\varepsilon)r$.

Consider a point $q_k \in Q$. Denote by $q'_k \in \mathcal{G}$ the grid point closest to q_k, and let $Q' = (q'_1, \dots, q'_m)$.

We have $\|q_k - q'_k\|_2 \le \frac{\varepsilon r}{2}$, so $d_{dF}(Q, Q') \le \frac{\varepsilon r}{2}$. By the triangle inequality,

$$d_{dF}(C_i, Q') \le d_{dF}(C_i, Q) + d_{dF}(Q, Q') \le r + \frac{\varepsilon r}{2} = (1 + \frac{\varepsilon}{2})r,$$

so Q' is in $\mathcal{I}_i \subseteq \mathcal{I}$. This means that $\mathcal{T}$ contains Q' with a curve $C(Q') \in \mathcal{C}$ such that $d_{dF}(C(Q'), Q') \le (1 + \frac{\varepsilon}{2})r$, and the query algorithm returns $C(Q')$. Now, again by the triangle inequality,

$$d_{dF}(C(Q'), Q) \le d_{dF}(C(Q'), Q') + d_{dF}(Q', Q) \le (1 + \frac{\varepsilon}{2})r + \frac{\varepsilon r}{2} = (1+\varepsilon)r.$$

We obtain the following theorem.

Theorem 5.8. *There exists a data structure for the* $(1+\varepsilon, r)$*-ANNC under DFD, with* $n \cdot O(\frac{1}{\varepsilon})^{dm}$ *space,* $n \cdot \log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ *preprocessing time, and* $O(md\log(\frac{nmd}{\varepsilon}))$ *query time.*

m	Reference	Space	Query	Approx.
$\log n$	[DS17]	$O(n^{4d+1}\log n)$	$O(n^{4d}\log^2 n)$	$d\sqrt{d}$
	[EP18]	$\Omega(n^{O(d\log n)})$	$O(dn^2\log n)$	$1+\varepsilon$
	Theorem 5.8	$n^{O(d)}$	$O(d\log^2 n)$	$1+\varepsilon$
$O(1)$	[DS17]	$2^{O(d)}n\log n$	$2^{O(d)}\cdot\log n$	$d\sqrt{d}$
	[EP18]	$d^{O(d)}\cdot\tilde{O}(n)$	$O(d\log n)$	$1+\varepsilon$
	Theorem 5.8	$2^{O(d)}n$	$O(d\log(nd))$	$1+\varepsilon$

Table 5.2: Comparing our near-neighbor data structure to previous results, for a fixed ε (say $\varepsilon = 1/2$).

5.4 $\ell_{p,2}$-distance of polygonal curves

For the near-neighbor problem under the $\ell_{p,2}$-distance, we use the same basic approach as in the previous section, but with two small modifications. The first is that we set the grid's edge length to $\frac{\varepsilon r}{m^{1/p}\sqrt{d}}$, and redefine $G(x,R)$, G_i, and $\mathcal{G}$, as in the previous section but with respect to the new edge length of our grid. The second modification is that we redefine $\mathcal{I}_i$ to be the set of all curves $\overline{Q} = (x_1, x_2, \ldots, x_m)$ with points from $\mathcal{G}$, such that $d_p(C_i, \overline{Q}) \le 1 + \frac{\varepsilon}{2}$.

We assume without loss of generality from now and to the end of this section that $r = 1$ (we can simply scale the entire space by $1/r$), so the grid's edge length is $\frac{\varepsilon}{m^{1/p}\sqrt{d}}$. The following corollary is respective to Corollary 5.6.

Corollary 5.9. $|G_p(x,R)| = O\left(1 + \frac{m^{1/p}}{\varepsilon}R\right)^d$.

Proof. We scale our grid so that the edge length is 1, hence we are looking for the number of lattice points in $B_2^d(x, \frac{m^{1/p}\sqrt{d}}{\varepsilon}R)$. By Lemma 5.4 we get that this number is bounded by the volume of the d-dimensional ball of radius $(1 + \frac{m^{1/p}}{\varepsilon}R)\sqrt{d}$. Using Stirling's formula we conclude,

$$V_2^d\left(\left(1 + \frac{m^{1/p}}{\varepsilon}R\right)\sqrt{d}\right) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)} \cdot \left(\left(1 + \frac{m^{1/p}}{\varepsilon}R\right)\sqrt{d}\right)^d = \alpha^d \cdot \left(1 + \frac{m^{1/p}}{\varepsilon}R\right)^d$$

where α is a constant (approximately 4.13). □

In the following claim we bound the size of $\mathcal{I}_i$, which, surprisingly, is independent of p.

Claim 5.10. *$|\mathcal{I}_i| = O(\frac{1}{\varepsilon})^{md}$ and it can be computed in $O(\frac{1}{\varepsilon})^{md}$ time.*

Proof. Let $\overline{Q} = (x_1, x_2, \ldots, x_m) \in \mathcal{I}_i$, and let τ be an alignment with $\sigma_p(\tau) \le (1+\frac{\varepsilon}{2})r$ w.r.t. C_i and $\overline{Q}$. For each $1 \le k \le m$ let j_k be the smallest index such that $(j_k, k) \in \tau$. In other words, j_k is the smallest index that is matched to k by the alignment τ.

Set $R_k = \|x_k - p^i_{j_k}\|_2$. Since $d_p(C_i, \overline{Q}) \le 1 + \frac{\varepsilon}{2}$, we have $\|(R_1, \ldots, R_m)\|_p \le 1 + \frac{\varepsilon}{2}$. Let α_k be the smallest integer such that $R_k \le \alpha_k \frac{\varepsilon}{m^{1/p}}$, then $\alpha_k \le \frac{m^{1/p}}{\varepsilon} R_k + 1$, and by triangle inequality,

$$\begin{aligned}\|(\alpha_1, \alpha_2, \ldots, \alpha_m)\|_p &\le \frac{m^{1/p}}{\varepsilon}\|(R_1, R_2, \ldots, R_m)\|_p + m^{1/p} \\ &\le \frac{m^{1/p}}{\varepsilon}\left(1 + \frac{\varepsilon}{2}\right) + m^{1/p} < \left(2 + \frac{1}{\varepsilon}\right) m^{1/p}.\end{aligned}$$

Clearly, $x_k \in B^d_2(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$.

We conclude that for each curve $\overline{Q} = (x_1, x_2, \ldots, x_m) \in \mathcal{I}_i$ there exists an alignment τ such that $\sigma_p(\tau) \le 1 + \frac{\varepsilon}{2}$ w.r.t. C_i and $\overline{Q}$, and a sequence of integers $(\alpha_1, \ldots, \alpha_m)$ such that $\|(\alpha_1, \alpha_2, \ldots, \alpha_m)\|_p \le (2 + \frac{1}{\varepsilon})m^{1/p}$ and $x_k \in B^d_2(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$, for $k = 1, \ldots, m$. Therefore, the number of curves in $\mathcal{I}_i$ is bounded by the multiplication of three numbers:

1. The number of alignments that can determine the distance, which is at most 2^{2m} by Lemma 5.3.

2. The number of ways to choose a sequence of m positive integers $\alpha_1, \ldots, \alpha_m$ such that $\|(\alpha_1, \alpha_2, \ldots, \alpha_m)\|_p \le (2 + \frac{1}{\varepsilon})m^{1/p}$, which is bounded by the number of lattice points in $B^d_p((2+\frac{1}{\varepsilon})m^{1/p})$ (the m-dimensional ℓ_p-ball of radius $(2+\frac{1}{\varepsilon})m^{1/p}$). By Lemma 5.4, this number is bounded by

$$V^m_p((2+\frac{1}{\varepsilon})m^{1/p}+m^{1/p}) \le V^m_p(\frac{4m^{1/p}}{\varepsilon}) = \frac{2^m\Gamma(1+1/p)^m}{\Gamma(1+m/p)}\left(\frac{4m^{1/p}}{\varepsilon}\right)^m = O(\frac{1}{\varepsilon})^m\,,$$

   where the last equality follows as $\frac{m^{m/p}}{\Gamma(1+m/p)} = O(1)^m$.

3. The number of ways to choose a curve $(x_1, x_2, \ldots, x_m)$, such that $x_k \in G_p(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$, for $k = 1, \ldots, m$. By Corollary 5.9, the number of grid points in $G_p(p^i_{j_k}, \alpha_k \frac{\varepsilon}{m^{1/p}})$ is $O(1 + \alpha_k)^d$, so the number of ways to choose $(x_1, x_2, \ldots, x_m)$ is at most $\Pi^m_{k=1} O(1 + \alpha_k)^d = O(1)^{md}\left(\Pi^m_{k=1}(1 + \alpha_k)\right)^d$. By the inequality of arithmetic and geometric means we have

$$\begin{aligned}(\Pi^m_{k=1}(1 + \alpha_k)^p)^{1/p} &\le \left(\frac{\sum^m_{k=1}(1 + \alpha_k)^p}{m}\right)^{m/p} \\ &= \left(\frac{\|(1 + \alpha_1, \ldots, 1 + \alpha_m)\|_p}{m^{1/p}}\right)^m \\ &\le \left(\frac{\|1\|_p + \|(\alpha_1, \ldots, \alpha_m)\|_p}{m^{1/p}}\right)^m \\ &\le \left(\frac{m^{1/p} + (2 + \frac{1}{\varepsilon})m^{1/p}}{m^{1/p}}\right)^m = O(\frac{1}{\varepsilon})^m,\end{aligned}$$

   so $\Pi^m_{k=1} O(1 + \alpha_k)^d = O(1)^{md} O(\frac{1}{\varepsilon})^{md} = O(\frac{1}{\varepsilon})^{md}$.

□

The data structure and query algorithm are exactly the same as we described for DFD, but the analysis of space complexity and query time are different.

Space complexity and query time. The size of I_i and $\mathcal{I}$ are the same as in Section 5.3, so the total number of curves stored in the tree $\mathcal{T}$ is the same in our case. We only need to show that the upper bound on the size and query time of the search tree associated with a given node v of the tree $\mathcal{T}$ remains as in Section 5.3.

The grid points corresponding to the nodes in $N(v)$ are from n sets of m balls with radius $(1+\varepsilon)$. When projecting the grid points in one of the ball to a single dimension, the number of 1-dimensional points is at most $\frac{m^{1/p}\sqrt{d}}{\varepsilon} \cdot (1+\varepsilon)$, so the total number of projected points is at most $\frac{nm^{1+\frac{1}{p}}\sqrt{d}}{\varepsilon} \cdot (1+\varepsilon)$.

Thus in each level of the search tree of v we have $O(\frac{nm^2\sqrt{d}}{\varepsilon})$ 1-dimensional points, so the query time is $O(d\log(\frac{nmd}{\varepsilon}))$, and inserting a curve of length m to the tree $\mathcal{T}$ takes $O(md\log(\frac{nmd}{\varepsilon}))$ time. Note that the size of the search tree of v remains $O(d \cdot |N(v)|)$

We conclude that the total space complexity is $O(\frac{nm^2\sqrt{d}}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{dm} = n \cdot O(\frac{1}{\varepsilon})^{dm}$, constructing $\mathcal{T}$ takes $O(|\mathcal{I}| \cdot md\log(nmd/\varepsilon)) = n\log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ time, and the total query time is $O(md\log(\frac{nmd}{\varepsilon}))$.

Correctness. Consider a query curve $Q = (q_1, \dots, q_m)$. Assume that there exists a curve $C_i \in \mathcal{C}$ such that $d_p(C_i, Q) \leq 1$. We will show that the query algorithm returns a curve C^* with $d_p(C^*, Q) \leq 1+\varepsilon$.

Consider a point $q_k \in Q$. Denote by $q'_k \in \mathcal{G}$ the grid point closest to q_k, and let $Q' = (q'_1, \dots, q'_m)$.

We have $\|q_k - q'_k\|_2 \leq \frac{\varepsilon}{2m^{1/p}}$. Let τ be an alignment such that the $\ell_{p,2}$-cost of τ w.r.t. C_i and Q is at most 1. Unlike the Fréchet distance, $\ell_{p,2}$-distance for curves does not satisfy the triangle inequality. However, by the triangle inequality under ℓ_2 and ℓ_p, we get that the $\ell_{p,2}$-cost of τ w.r.t. C_i and Q' is

$$\begin{aligned}
\sigma_p(\tau) &= \left(\sum_{(j,t)\in\tau} (\|p^i_j - q'_t\|_2)^p \right)^{1/p} \\
&\leq \left(\sum_{(j,t)\in\tau} (\|p^i_j - q_t\|_2 + \|q_t - q'_t\|_2)^p \right)^{1/p} \\
&\leq \left(\sum_{(j,t)\in\tau} (\|p^i_j - q_t\|_2)^p \right)^{1/p} + \left(\sum_{(j,t)\in\tau} (\|q_t - q'_t\|_2)^p \right)^{1/p} \\
&\leq 1 + \left(m \left(\frac{\varepsilon}{2m^{1/p}} \right)^p \right)^{1/p} \leq 1 + \frac{\varepsilon}{2}.
\end{aligned}$$

So $d_p(C_i, Q') \leq 1+\frac{\varepsilon}{2}$, and thus Q' is in $I_i \subseteq \mathcal{I}$. This means that $\mathcal{T}$ contains Q' with a curve $C(Q') \in \mathcal{C}$ such that $d_p(C(Q'), Q') \leq 1 + \frac{\varepsilon}{2}$, and the query algorithm returns $C(Q')$. Now, again by the same argument (using an alignment with $\ell_{p,2}$-cost at most $1+\frac{\varepsilon}{2}$ w.r.t. $C(Q')$ and Q'), we get that $d_p(C(Q'), Q) \leq 1+\frac{\varepsilon}{2}+\left(m(\frac{\varepsilon}{2m^{1/p}})^p\right)^{1/p} = 1+\varepsilon$.

We obtain the following theorem.

Theorem 5.11. *There exists a data structure for the $(1+\varepsilon, r)$-ANNC under $\ell_{p,2}$-distance, with $n \cdot O(\frac{1}{\varepsilon})^{dm}$ space, $n\log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ preprocessing time, and $O(md\log(\frac{nmd}{\varepsilon}))$ query time.*

As mentioned in the preliminaries section, the DTW distance between two curves equals to their $\ell_{1,2}$-distance, and therefore we obtain the following theorem.

Theorem 5.12. *There exists a data structure for the $(1+\varepsilon, r)$-ANNC under DTW, with $n \cdot O(\frac{1}{\varepsilon})^{dm}$ space, $n\log(\frac{n}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md}$ preprocessing time, and $O(md\log(\frac{nmd}{\varepsilon}))$ query time.*

5.5 Approximate range counting

In the range counting problem for curves, we are given a set $\mathcal{C}$ of n curves, each consisting of m points in d dimensions, and a distance measure for curves δ. The goal is to preprocess $\mathcal{C}$ into a data structure that given a query curve Q and a threshold value r, returns the number of curves that are within distance r from Q.

In this section we consider the following approximation version of range counting for curves, in which r is part of the input (see Remark 5.15). Note that by storing pointers to curves instead of just counters, we can obtain a data structure for the approximate range searching problem (at the cost of an additional $O(n)$-factor to the storage space).

Problem 5.13 ($(1+\varepsilon, r)$-approximate range-counting for curves)**.** Given a parameter r and $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure that given a query curve Q, returns the number of all the input curves whose distance to Q is at most r plus possibly additional input curves whose distance to Q is greater than r but at most $(1+\varepsilon)r$.

We construct the prefix tree $\mathcal{T}$ for the curves in $\mathcal{I}$ as in Section 5.4, as follows. For each $1 \leq i \leq n$ and curve $\overline{Q} \in \mathcal{I}_i$, if $\overline{Q}$ is not in $\mathcal{T}$, insert it into $\mathcal{T}$ and initialize $C(\overline{Q}) \leftarrow 1$. Otherwise, if $\overline{Q}$ is in $\mathcal{T}$, update $C(\overline{Q}) \leftarrow C(\overline{Q}) + 1$. Notice that $C(\overline{Q})$ holds the number of curves from $\mathcal{C}$ that are within distance $(1+\frac{\varepsilon}{2})r$ to $\overline{Q}$. Given a query curve Q, we compute Q' as in Section 5.4. If Q' is in $\mathcal{T}$, we return $C(Q')$, otherwise, we return 0.

Clearly, the storage space, preprocessing time, and query time are similar to those in Section 5.4. We claim that the query algorithm returns the number of curves from $\mathcal{C}$ that are within distance r to Q plus possibly additional input curves whose distance to Q is greater than r but at most $(1+\varepsilon)r$. Indeed, let C_i be a curve such that $d_{dF}(C_i,Q) \leq r$. As shown in Section 5.4 we get $d_p(C_i,Q') \leq (1+\frac{\varepsilon}{2})r$, so Q' is in $\mathcal{I}_i$ and C_i is counted in $C(Q')$. Now let C_i be a curve such that $d_p(C_i,Q) > (1+\varepsilon)r$. If $d_p(C_i,Q') \leq (1+\frac{\varepsilon}{2})r$, then by a similar argument (switching the rolls of Q and Q') we get that $d_p(C_i,Q') \leq (1+\varepsilon)r$, a contradiction. So $d_p(C_i,Q') > (1+\frac{\varepsilon}{2})r$, and thus C_i is not counted in $C(Q')$.

We obtain the following theorem.

Theorem 5.14. *There exists a data structure for the* $(1+\varepsilon,r)$*-approximate range-counting for curves under* $\ell_{p,2}$*-distance, with* $n\cdot O(\frac{1}{\varepsilon})^{dm}$ *space,* $n\log(\frac{n}{\varepsilon})\cdot O(\frac{1}{\varepsilon})^{md}$ *preprocessing time, and* $O(md\log(\frac{nmd}{\varepsilon}))$ *query time.*

Remark 5.15. When the threshold parameter r is part of the query, we call the problem the $(1+\varepsilon)$*-approximate range-counting problem.* Note that the reduction from $(1+\varepsilon)$-approximate nearest-neighbor to $(1+\varepsilon,r)$-approximate near-neighbor can be easily adapted to a reduction from $(1+\varepsilon)$-approximate range-counting to $(1+\varepsilon,r)$-approximate range-counting.

Chapter 6

Nearest Neighbor and Clustering for Curves and Segments

6.1 Introduction

We consider efficient algorithms for two fundamental problems for sets of polygonal curves in the plane: nearest-neighbor query and clustering. Both of these problems have been studied extensively and bounds on the running time and storage consumption have been obtained. In general, these bounds suggest that the existence of algorithms that can efficiently process large datasets of curves of high complexity is unlikely. Therefore we study special cases of the problems where some curves are assumed to be directed line segments (henceforth referred to as segments), and the distance metric is the discrete Fréchet distance.

Given a collection $\mathcal{C}$ of n curves, a natural question to ask is whether it is possible to preprocess $\mathcal{C}$ into a data structure so that the nearest curve in the collection to a query curve Q can be determined efficiently. This is the (exact) *nearest-neighbor* problem for curves (NNC).

In Chapter 5, we study the approximation version of the nearest-neighbor problem for curves, and give a survey of the literature regarding this version of the problem.

A close-related problem is range searching (or counting) for curves. In this problem, the goal is to preprocess $\mathcal{C}$ such that given a query curve Q of length m_q and a radius r, all the curves in $\mathcal{C}$ that are within distance r from Q can be found efficiently. As mentioned in Chapter 5, Afshani and Driemel [AD18] studied (exact) range searching under both the discrete and continuous Fréchet distance. For the discrete Fréchet distance in the plane, their data structure uses space in $O(n(\log\log n)^{m-1})$ and has query time in $O(\sqrt{n} \cdot \log^{O(m)} n \cdot m_q^{O(1)})$, assuming $m_q = \log^{O(1)} n$. They also show that any data structure in the pointer model that achieves $Q(n) + O(k)$ query time, where k is the output size, has to use roughly $\Omega(n/Q(n))^2)$ space in the worst case, even if $m_q = 1$!

Clustering is another fundamental problem in data analysis that aims to partition an input collection of curves into clusters where the curves within each cluster are similar in some sense, and a variety of formulations have been proposed [ACMLM03, CL07, DKS16]. The k-Center problem [Gon85, AP02, HN79] is a classical problem in which a point set in a metric space is clustered. The problem is defined as follows: given a set $\mathcal{P}$ of n points, find a set $\mathcal{G}$ of k center points, such that the maximum distance from a point in $\mathcal{P}$ to a nearest point in $\mathcal{G}$ is minimized.

Given an appropriate metric for curves, such as the discrete Fréchet distance, one can define a metric space on the space of curves and then use a known algorithm for point clustering. The clustering obtained by the k-Center problem is useful in that it groups similar curves together, thus uncovering a structure in the collection, and furthermore the center curves are of value as each can be viewed as a representative or exemplar of its cluster, and so the center curves are a compact summary of the collection. However, an issue with this formulation, when applied to curves, is that the optimal center curves may be *noisy*, i.e., the size of such a curve may be linear in the total number of vertices in its cluster, see [DKS16] for a detailed description. This can significantly reduce the utility of the centers as a method of summarizing the collection, as the centers should ideally be of low complexity. To address this issue, Driemel et al. [DKS16] introduced the (k, ℓ)-Center problem, where the k desired center curves are limited to at most ℓ vertices each.

Several hardness of approximation results for both the NNC and (k, ℓ)-Center problems are known. For the NNC problem under the discrete Fréchet distance, no data structure exists requiring $O(n^{2-\varepsilon} \operatorname{polylog} m)$ preprocessing and $O(n^{1-\varepsilon} \operatorname{polylog} m)$ query time for $\varepsilon > 0$, and achieving an approximation factor of $c < 3$, unless the strong exponential time hypothesis fails [IM04, DKS16]. Dreimel and Silvestri [DS17] show that unless the orthogonal vectors hypothesis fails, there exists no data structure for range searching or nearest neighbor searching under the (discrete or continuous) Fréchet distance that can be built in $O(n^{2-\varepsilon} poly(m))$ time and achieves query time in $O(n^{1-\varepsilon} poly(m))$ for any $\varepsilon > 0$. In the case of the (k, ℓ)-Center problem under the discrete Fréchet distance, Driemel et al. showed that the problem is NP-hard to approximate within a factor of $2 - \varepsilon$ when k is part of the input, even if $\ell = 2$ and $d = 1$. Furthermore, the problem is NP-hard to approximate within a factor $2 - \varepsilon$ when ℓ is part of the input, even if $k = 2$ and $d = 1$, and when $d = 2$ the inapproximability bound is $3 \sin \pi/3 \approx 2.598$ [BDG+19].

However, we are interested in algorithms that can process large inputs, i.e., where n and/or m are large, which suggests that the processing time ought to be near-linear in nm and the query time for NNC queries should be near-linear in m only. The above results imply that algorithms for the NNC and (k, ℓ)-Center problems that achieve such running times are not realistic. Moreover, given that strongly subquadratic algorithms for computing the discrete Fréchet distance are unlikely to exist, an algorithm that must compute pairwise distances explicitly will incur a roughly $O(m^2)$ running time. To circumvent these constraints, we focus on specific important settings: for the NNC problem, either the query curve is assumed to be a segment or the input curves are segments; and for the (k, ℓ)-Center problem the center is a segment and $k = 1$, i.e., we focus on the $(1, 2)$-Center problem.

While these restricted settings are of theoretical interest, they also have a practical motivation when the inputs are trajectories of objects moving through space, such as migrating birds. A segment ab can be considered a trip from a starting point a to a destination b. Given a set of trajectories that travel from point to point in a noisy manner, we may wish to find the trajectory that most closely follows a direct path from a to b, which is the NNC problem with a segment query. Conversely, given an input of (directed) segments and a query trajectory, the NNC problem would identify the segment (the simplest possible trajectory, in a sense) that the query trajectory most closely resembles. In the case of the $(1, 2)$-Center problem, the obtained segment center for an input of trajectories would similarly represent the summary direction of the input, and the radius r^* of the solution would be a measure of the maximum deviation from that direction for the collection.

Our results. We present algorithms for a variety of settings (summarized in the table below) that achieve the desired running time and storage bounds. Under the L_∞ metric, we give exact algorithms for the NNC and $(1, 2)$-Center problems, including under translation, that achieve the roughly linear bounds. For the L_2 metric, $(1+\varepsilon)$-approximation algorithms with near-linear running times are given for the NNC problem, and for the $(1, 2)$-Center problem, an exact algorithm is given whose running time is roughly $O(n^2m^3)$ and whose space requirement is quadratic. (Parentheses point to results under translation)

	Input/query: m**-curves/segment**	**Input/query: segments/**m**-curve**	**Input: (1,2)-center**
L_∞	Section 6.3.1 (Section 6.5.1)	Section 6.3.2 (Section 6.5.2)	Section 6.6.1 (Section 6.6.2)
L_2	Section 6.4.1	Section 6.4.2	Section 6.6.3

6.2 Preliminaries

The discrete Fréchet distance is a measure of similarity between two curves, defined as follows. Consider the curves $C = (p_1, \ldots, p_m)$ and $C' = (q_1, \ldots, q_{m'})$, viewed as sequences of vertices. A (monotone) *alignment* of the two curves is a sequence $\tau := \langle (p_{i_1}, q_{j_1}), \ldots, (p_{i_v}, q_{j_v}) \rangle$ of pairs of vertices, one from each curve, with $(i_1, j_1) = (1, 1)$ and $(i_v, j_v) = (m, m')$. Moreover, for each pair (i_u, j_u), $1 < u \le v$, one of the following holds: (i) $i_u = i_{u-1}$ and $j_u = j_{u-1} + 1$, (ii) $i_u = i_{u-1} + 1$ and $j_u = j_{u-1}$, or (iii) $i_u = i_{u-1} + 1$ and $j_u = j_{u-1} + 1$. The discrete Fréchet distance is defined as

$$d_{dF}^{d}(C, C') = \min_{\tau \in \mathcal{T}} \max_{(i,j) \in \tau} d(p_i, q_j),$$

with the minimum taken over the set $\mathcal{T}$ of all such alignments τ, and where d denotes the metric used for measuring interpoint distances.

We now give two alternative, equivalent definitions of the discrete Fréchet distance between a segment $s = ab$ and a polygonal curve $C = (p_1, \ldots, p_m)$ (we will drop the point metric d from the notation, where it is clear from the context). Let $C[i,j] := \{p_i, \ldots, p_j\}$.

Denote by $B(p, r)$ the ball of radius r centered at p, in metric d. The discrete Fréchet distance between s and C is at most r, if and only if there exists a partition of C into a prefix $C[1,i]$ and a suffix $C[i+1, m]$, such that $B(a, r)$ contains $C[1,i]$ and $B(b, r)$ contains $C[i+1, m]$.

A second equivalent definition is as follows. Consider the intersections of balls around the points of C. Set $I_i(r) = B(p_1, r) \cap \cdots \cap B(p_i, r)$ and $\overline{I}_i(r) = B(p_{i+1}, r) \cap \cdots \cap B(p_m, r)$, for $i = 1, \ldots, m-1$. Then, the discrete Fréchet distance between s and C is at most r, if and only if there exists an index $1 \le i \le m-1$ such that $a \in I_i(r)$ and $b \in \overline{I}_i(r)$.

Given a set $\mathcal{C} = \{C_1, \ldots, C_n\}$ of n polygonal curves in the plane, the nearest-neighbor problem for curves is formulated as follows:

Problem 6.1 (NNC)**.** Preprocess $\mathcal{C}$ into a data structure, which, given a query curve Q, returns a curve $C \in \mathcal{C}$ with $d_{dF}(Q, C) = \min_{C_i \in \mathcal{C}} d_{dF}(Q, C_i)$.

We consider two variants of Problem 6.1: (i) when the query curve Q is a segment, and (ii) when the input $\mathcal{C}$ is a set of segments.

Secondly, we consider a particular case of the (k, ℓ)-Center problem for curves [DKS16].

Problem 6.2 ($(1,2)$-Center)**.** Find a segment s^* that minimizes $\max_{C_i \in \mathcal{C}} d_{dF}(s, C_i)$, over all segments s.

6.3 NNC and L_∞ metric

When d is the L_∞ metric, each ball $B(p_i, r)$ is a square. Denote by $S(p, d)$ the axis-parallel square of radius d centered at p.

Given a curve $C = (p_1, \ldots, p_m)$, let d_i, for $i = 1, \ldots, m-1$, be the smallest radius such that $S(p_1, d_i) \cap \cdots \cap S(p_i, d_i) \neq \emptyset$. In other words, d_i is the radius of the smallest enclosing square of $C[1,i]$. Similarly, let $\overline{d}_i$, for $i = 1, \ldots, m-1$, be the smallest radius such that $S(p_{i+1}, \overline{d}_i) \cap \cdots \cap S(p_m, \overline{d}_i) \neq \emptyset$.

For any $d > d_i$, $S(p_1, d) \cap \cdots \cap S(p_i, d)$ is a rectangle, $R_i = R_i(d)$, defined by four sides of the squares $S(p_1, d), \ldots, S(p_i, d)$, see Figure 6.1. These sides are fixed and do not depend on the specific value of d. Furthermore, the left, right, bottom and top sides of $R_i(d)$ are provided by the sides corresponding to the right-, left-, top- and bottom-most vertices in $C[1, i]$, respectively, i.e., the sides corresponding to the vertices defining the bounding box of $C[1, i]$.

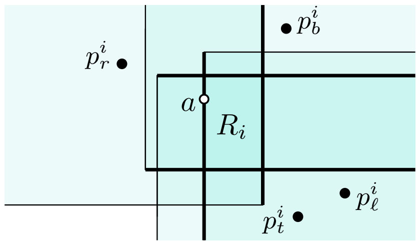


Figure 6.1: The rectangle $R_i(d)$ and the vertices of the ith prefix of C that define it.

Denote by p_ℓ^i the vertex in the ith prefix of C that contributes the left side to $R_i(d)$, i.e., the left side of $S(p_\ell^i, d)$ defines the left side of $R_i(d)$. Furthermore, denote by p_r^i, p_b^i, and p_t^i the vertices of the ith prefix of C that contribute the right, bottom, and top sides to $R_i(d)$, respectively. Similarly, for any $d > \overline{d}_i$, we denote the four vertices of the ith suffix of C that contribute the four sides of the rectangle $\overline{R}_i(d) = S(p_{i+1}, d) \cap \cdots \cap S(p_m, d)$ by $\overline{p}_\ell^i$, $\overline{p}_r^i$, $\overline{p}_b^i$, and $\overline{p}_t^i$, respectively.

Finally, we use the notation $R_i^j = R_i^j(d)$ ($\overline{R}_i^j = \overline{R}_i^j(d)$) to refer to the rectangle $R_i = R_i(d)$ ($\overline{R}_i = \overline{R}_i(d)$) of curve C_j.

Observation 6.3. *Let $s = ab$ be a segment, C be a curve, and let $d > 0$. Then, $d_{dF}(s, C) \le d$ if and only if there exists i, $1 \le i \le m - 1$, such that $a \in R_i(d)$ and $b \in \overline{R}_i(d)$.*

6.3.1 Query is a segment

Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the input curves, each of size m. Given a query segment $s = ab$, the task is to find a curve $C \in \mathcal{C}$ such that $d_{dF}(s, C) = \min_{C' \in \mathcal{C}} d_{dF}(s, C')$.

The data structure. The data structure is an eight-level search tree. The first level of the data structure is a search tree for the x-coordinates of the vertices p_ℓ^i, over all curves $C \in \mathcal{C}$, corresponding to the nm left sides of the nm rectangles $R_i(d)$. The second level corresponds to the nm right sides of the rectangles $R_i(d)$, over all curves $C \in \mathcal{C}$. That is, for each node u in the first level, we construct a search tree for the subset of x-coordinates of vertices p_r^i which corresponds to the canonical set of u. Levels three and four of the data structure correspond to the bottom and top sides, respectively, of the rectangles $R_i(d)$, over all curves $C \in \mathcal{C}$, and they are constructed using the y-coordinates of the vertices p_b^i and the y-coordinates of the vertices p^i_t, respectively. The fifth level is constructed as follows. For each node u in the fourth level, we construct a search tree for the subset of x-coordinates of vertices $\overline{p}^i_\ell$ which corresponds to the canonical set of u; that is, if the y-coordinate of p^j_t is in u's canonical subset, then the x-coordinate of $\overline{p}^j_\ell$ is in the subset corresponding to u's canonical set. The bottom four levels correspond to the four sides of the rectangles $\overline{R}_i(d)$ and are built using the x-coordinates of the vertices $\overline{p}^i_\ell$, the x-coordinates of the vertices $\overline{p}^i_r$, the y-coordinates of the vertices $\overline{p}^i_b$, and the y-coordinates of the vertices $\overline{p}^i_t$, respectively.

The query algorithm. Given a segment $s = ab$ and a distance $d > 0$, we can use our data structure to determine whether there exists a curve $C \in \mathcal{C}$, such that $d_{dF}(s, C) \leq d$. The search in the first and second levels of the data structure is done with $a.x$, the x-coordinate of a, in the third and fourth levels with $a.y$, in the fifth and sixth levels with $b.x$ and in the last two levels with $b.y$. When searching in the first level, instead of performing a comparison between $a.x$ and the value v that is stored in the current node (which is an x-coordinate of some vertex p^i_ℓ), we determine whether $a.x \geq v - d$. Similarly, when searching in the second level, at each node that we visit we determine whether $a.x \leq v + d$, where v is the value that is stored in the node, etc.

Notice that if we store the list of curves that are represented in the canonical subset of each node in the bottom (i.e., eighth) level of the structure, then curves whose distance from s is at most d may also be reported in additional time roughly linear in their number.

Finding the closest curve. Let $s = ab$ be a segment, let C be the curve in $\mathcal{C}$ that is closest to s and set $d^* = d_{dF}(s, C)$. Then, there exists $1 \leq i \leq m - 1$, such that $a \in R_i(d^*)$ and $b \in \overline{R}_i(d^*)$. Moreover, one of the endpoints a or b lies on the boundary of its rectangle, since, otherwise, we could shrink the rectangles without 'losing' the endpoints. Assume without loss of generality that a lies on the left side of $R_i(d^*)$. Then, the difference between the x-coordinate of the vertex p^i_ℓ and $a.x$ is exactly d^*. This implies that we can find d^* by performing a binary search in the set of all x-coordinates of vertices of curves in $\mathcal{C}$. In each step of the binary search, we need to determine whether $d \geq d^*$, where $d = v - a.x$ and v is the current x-coordinate, and our goal is to find the smallest such d for which the answer is still yes. We resolve a comparison by calling our data structure with the appropriate distance d. Since we do not know which of the two endpoints, a or b, lies on the boundary of its rectangle and on which of its sides, we perform 8 binary searches, where each search returns a candidate distance. Finally, the smallest among these 8 candidate distances is the desired d^*.

In other words, we perform 4 binary searches in the set of all x-coordinates of vertices of curves in $\mathcal{C}$. In the first we search for the smallest distance among the distances $d_\ell = v - a.x$ for which there exists a curve at distance at most d_ℓ from s; in the second we search for the smallest distance $d_r = a.x - v$ for which there exists a curve at distance at most d_r from s; in the third we search for the smallest distance $\overline{d}_\ell = v - b.x$ for which there exists a curve at distance at most $\overline{d}_\ell$ from s; and in the fourth we search for the smallest distance $\overline{d}_r = b.x - v$ for which there exists a curve at distance at most $\overline{d}_r$ from s. We also perform 4 binary searches in the set of all y-coordinates of vertices of curves in $\mathcal{C}$, obtaining the candidates d_b, d_t, $\overline{d}_b$, and $\overline{d}_t$. We then return the distance $d^* = \min\{d_\ell, d_r, \overline{d}_\ell, \overline{d}_r, d_b, d_u, \overline{d}_b, \overline{d}_u\}$.

Theorem 6.4. *Given a set $\mathcal{C}$ of n curves, each of size m, one can construct a search structure of size $O(nm\log^7(nm))$ for segment nearest-curve queries. Given a query segment s, one can find in $O(\log^8(nm))$ time the curve $C \in \mathcal{C}$ and distance d^* such that $d_{dF}(s,C) = d^*$ and $d^* \leq d_{dF}(s,C')$ for all $C' \in \mathcal{C}$, under the L_∞ metric.*

6.3.2 Input is a set of segments

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be the input set of segments. Given a query curve $Q = (p_1, \ldots, p_m)$, the task is to find a segment $s = ab \in \mathcal{S}$ such that $d_{dF}(Q,s) = \min_{s' \in \mathcal{S}} d_{dF}(Q,s')$, after suitably preprocessing $\mathcal{S}$. We use an overall approach similar to that used in Section 6.3.1, however the details of the implementation of the data structure and algorithm differ.

The data structure. Preprocess the input $\mathcal{S}$ into a four-level search structure $\mathcal{T}$ consisting of a two-dimensional range tree containing the endpoints a, and where the associated structure for each node in the second level of the tree is another two-dimensional range tree containing the endpoints b corresponding to the points in the canonical subset of the node.

This structure answers queries consisting of a pair of two-dimensional ranges (i.e., rectangles) $(R, \overline{R})$ and returns all segments $s = ab$ such that $a \in R$ and $b \in \overline{R}$. The preprocessing time for the structure is $O(n\log^4 n)$, and the storage is $O(n\log^3 n)$. Querying the structure with two rectangles requires $O(\log^3 n)$ time, by applying fractional cascading [WL85].

The query algorithm. Consider the decision version of the problem where, given a query curve Q and a distance d, the objective is to determine if there exists a segment $s \in \mathcal{S}$ with $d_{dF}(s,Q) \leq d$. Observation 6.3 implies that it is sufficient to query the search structure $\mathcal{T}$ with the pair of rectangles $(R_i(d), \overline{R}_i(d))$ of the curve Q, for all $1 \leq i \leq m-1$. If $\mathcal{T}$ returns at least one segment for any of the partitions, then this segment is within distance d of Q.

As we traverse the curve Q left-to-right, the bounding box of $Q[1,i]$ can be computed at constant incremental cost. For a fixed $d > 0$, each rectangle $R_i(d)$ can be constructed from the corresponding bounding box in constant time. Rectangle $\overline{R}_i(d)$ can be handled similarly by a reverse traversal. Hence all the rectangles can be computed in time $O(m)$, for a fixed d. Each pair of rectangles requires a query in $\mathcal{T}$, and thus the time required to answer the decision problem is $O(m \log^3 n)$.

Finding the closest segment. In order to determine the nearest segment s to Q, we claim, using an argument similar to that in Section 6.3.1, for a segment $s = ab$ of distance d^* from Q that either a lies on the boundary of $R_i(d^*)$ or b lies on the boundary of $\overline{R}_i(d^*)$ for some $1 \leq i < m$.

Thus, in order to determine the value of d^* it suffices to search over all $8m$ pairs of rectangles where either a or b lies on one of the eight sides of the obtained query rectangles.

The sorted list of candidate values of d for each side can be computed in $O(n)$ time from a sorted list of the corresponding x- or y-coordinates of a or b. The smallest value of d for each side is then obtained by a binary search of the sorted list of candidate values. For each of the $O(\log n)$ evaluated values d, a call to $\mathcal{T}$ decides on the existence of a segment within d of Q.

Theorem 6.5. *Given an input $\mathcal{S}$ of n segments, a search structure can be preprocessed in $O(n \log^4 n)$ time and requiring $O(n \log^3 n)$ storage that can answer the following. For a query curve Q of m vertices, find the segment $s^* \in \mathcal{S}$ and distance d^* such that $d_{dF}(Q, s^*) = d^*$ and $d_{dF}(Q, s) \geq d^*$ for all $s \in \mathcal{S}$ under the L_∞ metric. The time to answer the query is $O(m \log^4 n)$.*

6.4 NNC and L_2 metric

In this section, we present algorithms for approximate nearest-neighbor search under the discrete Fréchet distance using L_2. Notice that the algorithms from Section 6.3 for the L_∞ version of the problem, already give $\sqrt{2}$-approximation algorithms for the L_2 version. Next, we provide $(1+\varepsilon)$-approximation algorithms.

6.4.1 Query is a segment

Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be a set of n polygonal curves in the plane. The $(1+\varepsilon)$-approximate nearest-neighbor problem is defined as follows: Given $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure supporting queries of the following type: given a query segment s, return a curve $C' \in \mathcal{C}$, such that $d_{dF}(s, C') \leq (1+\varepsilon) d_{dF}(s, C)$, where C is the curve in $\mathcal{C}$ closest to s.

Here we provide a data structure for the $(1+\varepsilon, r)$-approximate nearest-neighbor problem, defined as: Given a parameter r and $0 < \varepsilon \leq 1$, preprocess $\mathcal{C}$ into a data structure supporting queries of the following type: given a query segment s, if there exists a curve $C_i \in \mathcal{C}$ such that $d_{dF}(s, C_i) \le r$, then return a curve $C_j \in \mathcal{C}$ such that $d_{dF}(s, C_j) \le (1+\varepsilon)r$.

There exists a reduction from the $(1+\varepsilon)$-approximate nearest-neighbor problem to the $(1+\varepsilon, r)$-approximate nearest-neighbor problem [Ind00], at the cost of an additional logarithmic factor in the query time.

An exponential grid. Given a point $p \in \mathbb{R}^2$, a parameter $0 < \varepsilon \le 1$, and an interval $[\alpha, \beta] \subseteq \mathbb{R}$, we can construct the following exponential grid $G(p)$ around p, which is a slightly different version of the exponential grid presented in [Dri13]:

Consider the series of axis-parallel squares S_i centered at p and of side lengths $\lambda_i = 2^i\alpha$, for $i = 1, \dots, \lceil \log(\beta/\alpha) \rceil$. Inside each region $S_i \setminus S_{i-1}$ (for $i > 1$), construct a grid G_i of side length $\frac{\varepsilon\lambda_i}{2\sqrt{2}}$. The total number of grid cells is at most

$$1 + \sum_{i=2}^{\lceil \log(\beta/\alpha) \rceil} \left(\lambda_i / \frac{\varepsilon\lambda_i}{2\sqrt{2}}\right)^2 = O((1/\varepsilon)^2 \lceil \log(\beta/\alpha) \rceil).$$

Given a point $q \in \mathbb{R}^2$ such that $\alpha \le \|q - p\| \le \beta$, let i be the smallest index such that $q \in S_i$. If q is in S_1, then $\|q - p\| \le \sqrt{2}\alpha$. Else, we have $i > 1$. Let g be the grid cell of G_i that contains q, and denote by c_g the center point of g. So we have $\|q - c_g\| \le \frac{\sqrt{2}}{2}\frac{\varepsilon\lambda_i}{2\sqrt{2}} = \frac{\varepsilon}{2}2^{i-1}\alpha \le \frac{\varepsilon}{2}2^{\log(\beta/\alpha)}\alpha = \frac{\varepsilon\beta}{2}$.

A data structure for $(1+\varepsilon, r)$**-ANNC.** For each curve $C_i = (p^i_1, \dots, p^i_m) \in \mathcal{C}$, we construct two exponential grids: $G(p^i_1)$ around p^i_1 and $G(p^i_m)$ around p^i_m, both with the range $[\frac{\varepsilon r}{2\sqrt{2}}, r]$, as described above. Now, for each pair of grid cells $(g, h) \in G(p^i_1) \times G(p^i_m)$, let $C(g, h) = C \in \mathcal{C}$ be the curve such that $d_{dF}(c_g c_h, C) = \min_j\{d_{dF}(c_g c_h, C_j)\}$. In other words, $C(g, h)$ is the closest input curve to the segment $c_g c_h$.

Let $\mathcal{G}_1$ be the union of the grids $G(p^1_1), G(p^2_1), \dots, G(p^n_1)$, and $\mathcal{G}_m$ the union of the grids $G(p^1_m), G(p^2_m), \dots, G(p^n_m)$. The number of grid cells in each grid is $O((1/\varepsilon)^2 \lceil \log(r / \frac{\varepsilon r}{2\sqrt{2}}) \rceil) = O(\frac{1}{\varepsilon^2}\log(1/\varepsilon))$. The number of grid cells in $\mathcal{G}_1$ and $\mathcal{G}_m$ is thus $O(n\frac{1}{\varepsilon^2}\log(1/\varepsilon))$.

The data structure is a four-level segment tree, where each grid cell is represented in the structure by its bottom- and left-edges. The first level is a segment tree for the horizontal edges of the cells of $\mathcal{G}_1$. The second level corresponds to the vertical edges of the cells of $\mathcal{G}_1$: for each node u in the first level, a segment tree is constructed for the set of vertical edges that correspond to the horizontal edges in the canonical subset of u. That is, if some horizontal edge of a cell in $G(p^i_1)$ is in u's canonical subset, then the vertical edge of the same cell is in the segment tree of the second level associated with u. Levels three and four of the data structure correspond to the horizontal and vertical edges, respectively, of the cells in $\mathcal{G}_m$.

The third level is constructed as follows. For each node u in the second level, we construct a segment tree for the subset of horizontal edges of cells in $\mathcal{G}_m$ which corresponds to the canonical set of u; that is, if a vertical edge of $G(p_1^i)$ is in u's canonical subset, then all the horizontal edges of $G(p_m^i)$ are in the subset corresponding to u's canonical set. Thus, the size of the third-level subset is $O(\frac{1}{\varepsilon^2}\log(1/\varepsilon))$ times the size of the second-level subset.

Each node of the forth level corresponds to a subset of pairs of grid cells from the set $\bigcup_{i=1}^{n}(G(p_1^i) \times G(p_m^i))$. In each such node u we store the curve $C(g,h)$ such that (g,h) is the pair in u's corresponding set for which $d_{dF}(c_g c_h, C(g,h))$ is minimum.

Given a query segment $s = ab$, we can obtain all pairs of grid cells $(g,h) \in \bigcup_{i=1}^{n}(G(p_1^i) \times G(p_m^i))$, such that $a \in g$ and $b \in h$, as a collection of $O(\log^4(\frac{n}{\varepsilon}))$ canonical sets in $O(\log^4(\frac{n}{\varepsilon}))$ time. Then, we can find, within the same time bound, the pair of cells g,h among them for which $d_{dF}(c_g c_h, C(g,h))$ is minimum. The space required is $O(n\frac{1}{\varepsilon^4}\log^4(\frac{n}{\varepsilon}))$.

The query algorithm. Given a query segment $s = ab$, let p,q be the pair of cell center points returned when querying the data structure with s, and let $C_j \in \mathcal{C}$ be the closest curve to pq. We show that if there exists a curve $C_i \in \mathcal{C}$ with $d_{dF}(ab, C_i) \leq r$, then $d_{dF}(ab, C_j) \leq (1+\varepsilon)r$.

Since $d_{dF}(ab, C_i) \leq r$, it holds that $d_{dF}(ab, p_1^i p_m^i) \leq r$, and thus there exists a pair of grid cells $g \in G(p_1^i)$ and $h \in G(p_m^i)$ such that $a \in g$ and $b \in h$. The data structure returns p,q, so we have $d_{dF}(pq, C_j) \leq d_{dF}(c_g c_h, C_i)$ (1). The properties of the exponential grids $G(p_1^i)$ and $G(p_m^i)$ guarantee that $\|a - c_g\|, \|b - c_h\| \leq \max\{\sqrt{2}\alpha, \frac{\varepsilon\beta}{2}\} = \frac{\varepsilon}{2}r$. Therefore, $d_{dF}(c_g c_h, ab) \leq \frac{\varepsilon}{2}r$ (2), and, similarly, $d_{dF}(pq, ab) \leq \frac{\varepsilon}{2}r$ (3). By the triangle inequality and Equation (2), $d_{dF}(c_g c_h, C_i) \leq d_{dF}(c_g c_h, ab) + d_{dF}(ab, C_i) \leq (1+\frac{\varepsilon}{2})r$ (4). Finally, by the triangle inequality and Equations (1), (3) and (4),

$$\begin{aligned} d_{dF}(ab, C_j) \leq d_{dF}(ab, pq) + d_{dF}(pq, C_j) &\leq d_{dF}(ab, pq) + d_{dF}(c_g c_h, C_i) \\ &\leq \frac{\varepsilon}{2}r + (1+\frac{\varepsilon}{2})r = (1+\varepsilon)r\,. \end{aligned}$$

Theorem 6.6. *Given a set $\mathcal{C}$ of n curves, each of size m, and $0 < \varepsilon \leq 1$, one can construct a search structure of size $O(\frac{n}{\varepsilon^4}\log^4(\frac{n}{\varepsilon}))$ for approximate segment nearest-neighbor queries. Given a query segment s, one can find in $O(\log^5(\frac{n}{\varepsilon}))$ time a curve $C' \in \mathcal{C}$ such that $d_{dF}(s, C') \leq (1+\varepsilon)d_{dF}(s, C)$, under the L_2 metric, where C is the curve in $\mathcal{C}$ closest to s.*

6.4.2 Input is a set of segments

In Section 6.3.2, we presented an exact algorithm for the problem under L_∞, in which we compute the intersections of the squares of radius d around the vertices of the query curve, and use a two-level data structure for rectangle-pair queries.

To achieve an approximation factor of $(1+\varepsilon)$ for the problem under L_2, we can use the same approach, except that instead of squares we use regular k-gons. Given a query curve $Q = (p_1, \ldots, p_m)$, the intersections of the regular k-gons of radius d around the vertices of Q are polygons with at most k edges, defined by at most k sides of the regular k-gons. The orientations of the edges of the intersections are fixed, and thus we can construct a two-level data structure for k-gon-pair queries, where each level consists of k inner levels, one for each possible orientation. The size of such a data structure is thus $O(n \log^{2k} n)$.

Given a parameter ε, we pick $k = O(\frac{1}{\sqrt{\varepsilon}})$, so that the approximation factor is $(1+\varepsilon)$, the space complexity is $O(n \log^{O(\frac{1}{\sqrt{\varepsilon}})} n)$ and the query time is $O(m \log^{O(\frac{1}{\sqrt{\varepsilon}})} n)$.

Theorem 6.7. *Given an input $\mathcal{S}$ of n segments, and $0 < \varepsilon \leq 1$, one can construct a search structure of size $O(n \log^{O(\frac{1}{\sqrt{\varepsilon}})} n)$ for approximate segment nearest-neighbor queries. Given a query curve Q of size m, one can find in $O(m \log^{O(\frac{1}{\sqrt{\varepsilon}})} n)$ time a segment $s' \in \mathcal{S}$ such that $d_{dF}(s', Q) \leq (1+\varepsilon) d_{dF}(s, Q)$, under the L_2 metric, where s is the segment in $\mathcal{S}$ closest to Q.*

6.5 NNC under translation and L_∞ metric

An analogous approach yields algorithms with similar running times for the problems under translation.

For a curve C and a translation t, let C_t be the curve obtained by translating C by t, i.e., by translating each of the vertices of C by t. In this section we study the two problems studied in Section 6.3, assuming the input curves are given up to translation. That is, the distance between the query curve Q and an input curve C is now $\min_t d_{dF}(Q, C_t)$, where the discrete Fréchet distance is computed using the L_∞ metric.

6.5.1 Query is a segment

Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of input curves, each of size m. We need to preprocess $\mathcal{C}$ for segment nearest-neighbor queries under translation, that is, given a query segment $s = ab$, find the curve $C \in C$ that minimizes $\min_t d_{dF}(s, C'_t) = \min_t d_{dF}(s_t, C')$, where s_t and C_t are the images of s and C, respectively, under the translation t. Let t^* be the translation that minimizes $d_{dF}(s_t, C)$, and set $d^* = d_{dF}(s_{t^*}, C)$. Consider the partition of $C = (p_1, \ldots, p_m)$ into prefix $C[1, i]$ and suffix $C[i+1,m]$, such that $a_{t^*} \in R_i(d^*)$ and $b_{t^*} \in \overline{R}_i(d^*)$. The following trivial observation allows us to construct a set of values to which d^* must belong.

Observation 6.8. *One of the following statements holds:*

1. *a_{t^*} lies on the left side of $R_i(d^*)$ and b_{t^*} lies on the right side of $\overline{R}_i(d^*)$, or vice versa, i.e., a_{t^*} lies on the right side of $R_i(d^*)$ and b_{t^*} lies on the left side of $\overline{R}_i(d^*)$.*

2. *a_{t^*} lies on the bottom side of $R_i(d^*)$ and b_{t^*} lies on the top side of $\overline{R}_i(d^*)$, or vice versa.*

Assume without loss of generality that $a.x < b.x$ and $a.y < b.y$ and that the first statement holds. Let $\delta_x = b.x - a.x$ denote the x-span of s, and let δ_y denote the y-span of s. Then, either (i) $(\overline{p}^i_r.x + d^*) - (p^i_l.x - d^*) = \delta_x$, or (ii) $(\overline{p}^i_l.x - d^*) - (p^i_r.x + d^*) = \delta_x$, where as before p^i_l (p^i_r) is the vertex of C which determines the left (right) side of R_i and $\overline{p}^i_l$ ($\overline{p}^i_r$) is the vertex of C which determines the left (right) side of $\overline{R}_i$. That is, either (i) $d^* = \frac{\delta_x - (\overline{p}^i_r.x - p^i_l.x)}{2}$, or (ii) $d^* = \frac{(\overline{p}^i_l.x - p^i_r.x) - \delta_x}{2}$.

The data structure. Consider the decision version of the problem: Given d, is there a curve in $\mathcal{C}$ whose distance from s under translation is at most d? We now present a five-level data structure to answer such decision queries. We continue to assume that $a.x < b.x$ and $a.y < b.y$. For a curve C_j, let d^j_i ($\overline{d}^j_i$) be the smallest radius such that R^j_i ($\overline{R}^j_i$) is non-empty, and set $r^j_i = \max\{d^j_i, \overline{d}^j_i\}$. The top level of the structure is simply a binary search tree on the $n(m-1)$ values r^j_i; it serves to locate the pairs $(R^j_i(d), \overline{R}^j_i(d))$ in which both rectangles are non-empty. The role of the remaining four levels is to filter the set of relevant pairs, so that at the bottom level we remain with those pairs for which s can be translated so that a is in the first rectangle and b is in the second.

For each node v in the top level tree, we construct a search tree over the values $\overline{p}^i_r.x - p^i_l.x$ corresponding to the pairs in the canonical subset of v. These trees constitute the second level of the structure. The third-level trees are search trees over the values $\overline{p}^i_l.x - p^i_r.x$, the fourth-level ones — over the values $\overline{p}^i_t.y - p^i_b.y$, and finally the fifth-level ones — over the values $\overline{p}^i_b.y - p^i_t.y$.

The query algorithm. Given a query segment $s = ab$ (with $a.x < b.x$ and $a.y < b.y$) and $d > 0$, we employ our data structure to answer the decision problem. In the top level, we select all pairs $(R^j_i, \overline{R}^j_i)$ satisfying $r^j_i \le d$. Of these pairs, in the second level, we select all pairs satisfying $\overline{p}^i_r.x - p^i_l.x \ge \delta_x - 2d$. In the third level, we select all pairs satisfying $\overline{p}^i_l.x - p^i_r.x \le \delta_x + 2d$. Similarly, in the fourth level, we select all pairs satisfying $\overline{p}^i_t.y - p^i_b.y \ge \delta_y - 2d$, and in the fifth level, we select all pairs satisfying $\overline{p}^i_b.y - p^i_t.y \le \delta_y + 2d$. At this point, if our current set of pairs is non-empty, we return YES, otherwise, we return NO.

To find the nearest curve C and the corresponding distance d^*, we proceed as follows, utilizing the observation above. We perform a binary search over the $O(nm)$ values of the form $\overline{p}^i_r.x - p^i_l.x$ to find the largest value for which the decision algorithm returns YES on $d = \frac{\delta_x - (\overline{p}^i_r.x - p^i_l.x)}{2}$. (We only consider the values $\overline{p}^i_r.x - p^i_l.x$ that are smaller than δ_x.) Similarly, we perform a binary search over the values $\overline{p}^i_t.y - p^i_b.y$ to find the largest value for which the decision algorithm returns YES on $d = \frac{\delta_y - (\overline{p}^i_t.y - p^i_b.y)}{2}$. We perform two more binary searches; one over the values $\overline{p}^i_l.x - p^i_r.x$ to find the smallest value for which the decision algorithms returns YES on $d = \frac{(\overline{p}^i_l.x - p^i_r.x) - \delta_x}{2}$, and one over the values $\overline{p}^i_b.y - p^i_t.y$. Finally, we return the smallest d for which the decision algorithm has returned YES.

Our data structure was designed for the case where b lies to the right and above a. Symmetric data structures for the other three cases are also needed. The following theorem summarizes the main result of this section.

Theorem 6.9. *Given a set $\mathcal{C}$ of n curves, each of size m, one can construct a search structure of size $O(nm\log^4(nm))$, such that, given a query segment s, one can find in $O(\log^6(nm))$ time the curve $C \in C$ nearest to s under translation, that is the curve minimizing $\min_t d_{dF}(s_t, C')$, where the discrete Fréchet distance is computed using the L_∞ metric.*

6.5.2 Input is a set of segments

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be the input set of segments, with $s_i = a_i b_i$. We need to preprocess $\mathcal{S}$ for nearest-neighbor queries under translation, that is, given a query curve $Q = (p_1, \ldots, p_m)$, find the segment $s = ab \in \mathcal{S}$ that minimizes $\min_t d_{dF}(Q, s'_t) = \min_t d_{dF}(Q_t, s')$. Since translations are allowed, without loss of generality we can assume that the first point of all the segments is the origin. In other words, the input is converted to a two-dimensional point set $\mathcal{C} = \{c_i = b_i - a_i \mid a_i b_i \in \mathcal{S}\}$.

The idea is to find the nearest segment corresponding to each of the $m-1$ partitions of the query. Let $s = ab$ be any segment and d some radius. The following observation holds for any partition of Q into $Q[1,i]$ and $Q[i+1,m]$, where $R^{\oplus}_i(d) = (-R_i(d)) \oplus \overline{R}_i(d)$ and $\oplus$ is the Minkowski sum operator, see Figure 6.2.

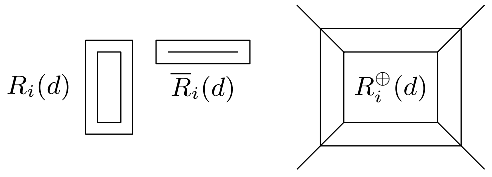


Figure 6.2: The rectangle $R^{\oplus}_i(d)$, as d increases.

Observation 6.10. *There exists a translation t such that $a_t \in R_i(d)$ and $b_t \in \overline{R}_i(d)$ if and only if $c = b - a \in R^{\oplus}_i(d)$.*

Based on this observation segment ab is within distance d of Q under translation, if for some i, $R_i^{\oplus}(d)$ contains the point $c = b - a$, which means translations can be handled implicitly.

The data structure. According to Observation 6.10, a data structure is required to answer the following question: Given a partition of Q into prefix $Q[1, i]$ and suffix $Q[i+1, m]$, what is the smallest radius d^* so that $R_i^{\oplus}(d^*)$ contains some $c_j \in \mathcal{C}$? The smallest radius d' where both $R_i(d')$ and $\overline{R}_i(d')$—and hence $R_i^{\oplus}(d')$—are nonempty can be determined in linear time. This value which depends on i is a lower bound on d^*.

Since $-R_i(d')$ and $\overline{R}_i(d')$ are both axis-aligned rectangles (segments or points in special cases), their Minkowski sum, $R_i^{\oplus}(d')$, is also a possibly degenerate axis-aligned rectangle. If this rectangle contains some point $c_j \in \mathcal{C}$, then s_j is the nearest segment with respect to this partition and the optimal distance is d'. If it contains more than one point from $\mathcal{C}$, then all the corresponding segments are equidistant from the query and each of them can be reported as the nearest neighbor corresponding to this partition. The data structure needed here is a two-dimensional range tree on $\mathcal{C}$.

If $R_i^{\oplus}(d') \cap \mathcal{C}$ is empty, then we need to find the smallest radius d^* so that $R_i^{\oplus}(d^*)$ contains some c_j. For any distance $d > d'$, $R_i^{\oplus}(d)$ is a rectangle concentric with $R_i^{\oplus}(d')$ but whose edges are longer by an additive amount of $4(d - d')$.

As d increases, the four edges of the rectangle sweep through 4 non-overlapping regions in the plane, so any point in the plane that gets covered by $R_i^{\oplus}(d)$, first appears on some edge. We divide this problem into 4 sub-problems based on the edge that the optimal c_j might appear on. Below, we solve the sub-problem for the right edge of the rectangle: Given a partition of Q into prefix $Q[1, i]$ and suffix $Q[i+1, m]$, what is the smallest radius d_r^* so that the right edge of $R_i^{\oplus}(d_r^*)$ contains some c_j? All other sub-problems are solved symmetrically.

Any point c_j that appears on the right edge belongs to the intersection of three half-planes:

1. On or below the line of slope $+1$ passing through the top-right corner of the rectangle $R_i^{\oplus}(d')$.

2. On or above the line of slope -1 passing through the bottom-right corner of $R_i^{\oplus}(d')$.

3. To the right of the line through the right edge of $R_i^{\oplus}(d')$.

The first point in this region swept by the right edge of the growing rectangle $R_i^{\oplus}(d)$ is the one with the smallest x-coordinate. This point can be located using a three-dimensional range tree on $\mathcal{C}$.

The query algorithm. Given a query curve $Q = (p_1, \ldots, p_m)$, the nearest segment under translation can be determined by using the data structure to find the nearest segment—and its distance from Q—for each of the $m-1$ partitions and selecting the segment whose distance is smallest.

As stated in Section 6.3.2, all $O(m)$ bounding boxes can be computed in $O(m)$ total time. For a particular partition, knowing the two bounding boxes, one can determine the smallest radius d' where $R_i^{\oplus}(d')$ is nonempty in constant time. Now the two-dimensional range tree on $\mathcal{C}$ is used to search for points inside $R_i^{\oplus}(d')$. If the data structure returns some point $c \in \mathcal{C}$, then the segment corresponding to c is the nearest segment under translation. Otherwise, one has to do four three-level range searches in the second data structure and compare the results to find the nearest segment. This is the most expensive step which takes $O(\log^2 n)$ time using fractional cascading [WL85]. The following theorem summarizes the main result of this section.

Theorem 6.11. *Given a set $\mathcal{S}$ of n segments, one can construct a search structure of size $O(n \log^2 n)$, so that, given a query curve Q of size m, one can find in $O(m \log^2 n)$ time the segment $s \in \mathcal{S}$ nearest to Q under translation, that is the segment minimizing $\min_t d_{dF}(Q, s'_t)$, where the discrete Fréchet distance is computed using the L_∞ metric.*

6.6 $(1,2)$-Center

The objective of the $(1,2)$-Center problem is to find a segment s such that $\max_{C_i \in \mathcal{C}} d_{dF}(s, C_i)$ is minimized. This can be reformulated equivalenly as: Find a pair of balls $(B, \overline{B})$, such that (i) for each curve $C \in \mathcal{C}$, there exists a partition at $1 \le i < m$ of C into prefix $C[1,i]$ and suffix $C[i+1,m]$, with $C[1,i] \subseteq B$ and $C[i+1,m] \subseteq \overline{B}$, and (ii) the radius of the larger ball is minimized.

6.6.1 $(1,2)$-Center and L_∞ metric

An optimal solution to the $(1,2)$-Center problem under the L_∞ metric is a pair of squares $(S, \overline{S})$, where S contains all the prefix vertices and $\overline{S}$ contains all the suffix vertices. Assume that the optimal radius is r^*, and that it is determined by S, i.e., the radius of S is r^* and the radius of $\overline{S}$ is at most r^*. Then, there must exist two *determining vertices* p, p', belonging to the prefixes of their respective curves, such that p and p' lie on opposite sides of the boundary of S. Clearly, $||p - p'||_\infty = 2r^*$. Let the positive normal direction of the sides be the *determining direction* of the solution.

Let R be the axis-aligned bounding rectangle of $C_1 \cup \cdots \cup C_n$, and denote by e_ℓ, e_r, e_t, and e_b the left, right, top, and bottom edges of R, respectively.

Lemma 6.12. *At least one of p, p' must lie on the boundary of R.*

Proof. Assume that the determining direction is the positive x-direction, and that neither p nor p' lies on the boundary of R. Thus, there must exist a pair of vertices $q, q' \in \overline{S}$ with $q.x < p.x$ and $q'.x > p'.x$, which implies that $||q-q'||_\infty > ||p-p'||_\infty = 2r^*$, contradicting the assumption that p, p' are the determining vertices. □

We say that a corner of S (or $\overline{S}$) *coincides* with a corner of R when the corner points are incident, and they are both of the same type, i.e., top-left, bottom-right, etc.

Lemma 6.13. *There exists an optimal solution $(S, \overline{S})$ where at least one corner of S or $\overline{S}$ coincides with a corner of R.*

Proof. Let $p, p' \in S$ be a pair of determining vertices, and assume, without loss of generality, that p lies on the boundary of R. If p is a corner of R, then the claim trivially holds. Otherwise, p lies in the interior of an edge of R, and assume without loss of generality that it lies on e_ℓ.

If S contains a vertex on e_t, then we can shift S vertically down until its top edge overlaps e_t. Else, if it contains a vertex on e_b, then we can shift S up until its bottom edge overlaps e_b. In both cases, the lemma conclusion holds.

If S does not contain any vertex from e_t or e_b, then clearly $\overline{S}$ must contain vertices $q \in e_t$ and $q' \in e_b$ with $||q-q'||_\infty \le 2r^*$. Therefore, S intersects e_b or e_t (or both), and can be shifted vertically until its boundary overlaps e_b or e_t, as desired.

A symmetric argument can be made when p and p' are suffix vertices, i.e., $p, p' \in \overline{S}$. □

Lemma 6.13 implies that for a given input $\mathcal{C}$ where the determining vertices are in S, there must exist an optimal solution where S is positioned so that one of its corners coincides with a corner of the bounding rectangle, and that one of the determining vertices is on the boundary of R. The optimal solution can thus be found by testing all possible candidate squares that satisfy these properties and returning the valid solution that yields the smallest radius. The algorithm presented in the sequel will compute the radius r^* of an optimal solution $(S^*, \overline{S^*})$ such that r^* is determined by the prefix square S^*, see Figure 6.3. The solution where r^* is determined by $\overline{S^*}$ can be computed in a symmetric manner.

For each corner v of the bounding rectangle R, we sort the $(m-2)n$ vertices in $C_1 \cup \cdots \cup C_n$ that are not endpoints—the initial vertex of each curve must always be contained in the prefix, and the final vertex in the suffix—by their L_∞ distance from v. Each vertex p in this ordering is associated with a square S of radius $||v-p||_\infty/2$, coinciding with R at corner v.

A sequential pass is made over the vertices, and their respective squares S, and for each S we compute the radius of S and $\overline{S}$ using the following data structures. We maintain a balanced binary tree T_C for each curve $C \in \mathcal{C}$, where the leaves of T_C

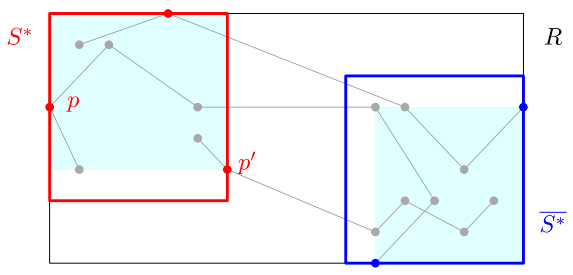


Figure 6.3: The optimal solution is characterized by a pair of points p, p' lying on the boundary of S^*, and a corner of S^* coincides with a corner of R.

correspond to the vertices of C, in order. Each node of the tree contains a single bit: The bit at a leaf node corresponding to vertex p_j indicates whether $p_j \in S$, where S is the current square. The value of the bit at a leaf of T_C can be updated in $O(\log m)$ time. The bit of an internal node is 1 if and only if all the bits in the leaves of its subtree are 1, and thus the longest prefix of C can be determined in $O(\log m)$ time.

At each step in the pass, the radius of $\overline{S}$ must also be computed, and this is obtained by determining the bounding box of the suffix vertices. Thus, two balanced binary trees are maintained: $\overline{T}_x$ contains a leaf for each of the suffix vertices ordered by their x-coordinate; and $\overline{T}_y$ where the leaves are ordered by the y-coordinate. The extremal vertices that determine the bounding box can be determined in $O(\log mn)$ time. Finally, the current optimal squares S^* and $\overline{S^*}$, and the radius r^* of S^* are persisted.

The trees $T_{C_1}, \ldots, T_{C_n}$ are constructed with all bits initialized to 0, except for the bit corresponding to the initial vertex in each tree which is set to 1, taking $O(nm)$ time in total. $\overline{T}_x$ and $\overline{T}_y$ are initialized to contain all non-initial vertices in $O(mn \log mn)$ time. The optimal square S^* containing all the initial vertices is computed, and $\overline{S^*}$ is set to contain the remaining vertices. The optimal radius r^* is the larger of the radii induced by S^* and $\overline{S^*}$.

At the step in the pass for vertex p of curve C_j whose associated square is S, the leaf of T_C corresponding to p is updated from 0 to 1 in $O(\log m)$ time. The index i of the longest prefix covered by S can then be determined, also in $O(\log m)$ time. The vertices from C_j that are now in the prefix must be deleted from $\overline{T}_x$ and $\overline{T}_y$, and although there may be $O(m)$ of them in any iteration, each will be deleted exactly once, and so the total update time over the entire sequential pass is $O(mn \log mn)$. The radius of the square S is $\|v - p\|_\infty / 2$, and the radius of $\overline{S}$ can be computed in $O(\log mn)$ time as half the larger of x- and y-extent of the suffix bounding box. The optimal squares S^*, $\overline{S^*}$, and the cost r^* are updated if the radius of S determines the cost, and the radius of S is less than the existing value of r^*.

Finally, we return the optimal pair of squares $(S^*, \overline{S^*})$ with the minimal cost r^*.

Theorem 6.14. *Given a set of curves $\mathcal{C}$ as input, an optimal solution to the* $(1, 2)$-

Center *problem using the discrete Fréchet distance under the L_∞ metric can be computed in time $O(mn \log mn)$ using $O(mn)$ storage.*

6.6.2 $(1,2)$-Center under translation and L_∞ metric

The $(1,2)$-Center problem under translation and the L_∞ metric can be solved using a similar approach.

The objective is to find a segment s^* that minimizes the maximum discrete Fréchet distance under L_∞ between s^* and the input curves whose locations are fixed only up to translation. A solution will be a pair of squares $(S, \overline{S})$ of equal size and whose radius r^* is minimal, such that, for each $C \in \mathcal{C}$, there exists a translation t and a partition index i where $C_t[1,i] \subset S$ and $C_t[i+i,m] \subset \overline{S}$. Clearly, an optimal solution will not be unique as the curves can be uniformly translated to obtain an equivalent solution, and moreover, in general there is freedom to translate either square in the direction of at least one of the x- or y-axes.

Let $\delta_x(C)$ be the x-extent of the curve C and $\delta_y(C)$ be the y-extent. Let R be the closed rectangle whose bottom-left corner lies at the origin and whose top-right corner is located at (δ_x^*, δ_y^*) where $\delta_x^* := \max_{C \in \mathcal{C}} \delta_x(C)$ and $\delta_y^* := \max_{C \in \mathcal{C}} \delta_y(C)$. Furthermore, let w_ℓ and w_r be the left- and right-most vertices in a curve with x-span δ_x^*, and let w_t and w_b be the top- and bottom-most vertices in a curve with y-span δ_y^*. Clearly, all curves in $\mathcal{C}$ can be translated to be contained within R, and for all such sets of curves under translation, the extremal vertices w_t, w_b, w_ℓ and w_r each must lie on the corresponding side of R. We claim that if a solution exists with radius r^*, then an equivalent solution $(S, \overline{S})$ can be obtained using the same partition of each curve, where S and $\overline{S}$ are placed at opposite corners of R.

Lemma 6.15. *Given a set $\mathcal{C}$ of n curves, if there exists a solution of radius r^* to the problem, then there also exists a solution $(S, \overline{S})$ of radius r^* where a corner of S and a corner of $\overline{S}$ coincide with opposite corners of the rectangle R.*

Proof. Let $(S', \overline{S'})$ be a solution of radius r^* where all the curves under translation are not necessarily contained in R, and the corners of S' and $\overline{S'}$ do not coincide with the corners of R. The proof is constructive: The coordinate system is defined such that prefix square S' is positioned so that its corner coincides with the appropriate corner of R ensuring that $S' \equiv S$, and we define a continuous family of squares $\overline{S}(\lambda)$ parameterized on $\lambda \in [0,1]$ where $\overline{S}(0) = \overline{S'}$ and $\overline{S}(1) = \overline{S}$, such that $\overline{S}$ coincides with the opposite corner of R. This family traces a translation of $\overline{S}(\lambda)$, first in the x-direction and then in the y-direction, and we show that the prefix and suffix of each curve—possibly under translation—remain within S and $\overline{S}(\lambda)$, and thus the solution remains valid.

We prove this for the case where the top-right corner v of S' is *below-left* the top-right corner $\overline{v}$ of $\overline{S'}$, i.e., $v.x \leq \overline{v}.x$ and $v.y \leq \overline{v}.y$. In the sequel we will show that an equivalent solution $(S, \overline{S})$ exists where the bottom-left corner of S lies at the origin and the top-right corner of $\overline{S}$ lies at (δ_x^*, δ_y^*) as required by the claim in the lemma. A symmetric argument exists for the other cases where v's position relative to $\overline{c}$ is *above-left*, *below-right* and *below-left.*

First, observe that $\overline{v}.x \geq \delta_x^*$, as either w_r is a vertex in a prefix of some curve and thus $\delta_x^* \leq v.x \leq \overline{v}.x$, or w_r is a vertex in a suffix and thus $\delta_x^* \leq \overline{v}.x$. A similar argument proves that $\overline{v}.y \geq \delta_y^*$, and thus $\overline{S}(\lambda)$ will move to the left until the x-coordinate of the right edge of $\overline{S}$ is δ_x^* and then down under the continuous translation to $\overline{S}$, i.e., the y-coordinate of the top edge of $\overline{S}$ is δ_y^*.

Consider the validity of the solution $(S, \overline{S}(\lambda))$ as the suffix square moves leftwards. If there are no suffix vertices on the right edge of square $\overline{S}(\lambda)$ then it can be translated to the left and remain a valid solution, until such time as some suffix vertex $\overline{p}$ of curve C lies on the right edge. Subsequently, C is translated together with $\overline{S}(\lambda)$, and thus the suffix vertices of C continue to be contained in $\overline{S}(\lambda)$. For a prefix vertex p of C to move outside S under the translation it must cross the left-side of S, however this would imply that $|\overline{p}.x - p.x| > \overline{p}.x \geq \delta_x^*$, contradicting the fact that δ_x^* is the maximum extent in the x-direction of all curves. The same analysis can be applied to the translation of $\overline{S}(\lambda)$ in the downward direction. This shows that the continuous family of squares $\overline{S}(\lambda)$ imply a family of optimal solutions $(S, \overline{S}(\lambda))$ to the problem, and in particular $(S, \overline{S})$ is a solution. □

Lemma 6.15 implies that an optimal solution of radius r^* exists where S and $\overline{S}$ coincide with opposite corners of R. Next, we consider the properties of such an optimal solution, and show that r^* is determined by two vertices from a *single* curve. Recall that a pair of vertices are *determining vertices* if they lie on opposite sides of one of the squares. Here, we refine the definition with the condition that the pair both belong to the prefix or suffix of the same curve. Furthermore, denote a pair of vertices $(p, \overline{p})$, where p is in the prefix and $\overline{p}$ is in the suffix of the same curve, as *opposing vertices* if they preclude a smaller pair of squares coincident with the same opposing corners of R. Assuming that S coincides with the top-left corner of R and $\overline{S}$ with the bottom-right corner, then p and $\overline{p}$ are opposing vertices if, either: (i) p lies on the right edge of S and $\overline{p}$ lies on the left edge of $\overline{S}$; or (ii) p lies on bottom edge of S and $\overline{p}$ lies on the top edge of $\overline{S}$. Symmetrical conditions exist for the cases where S and $\overline{S}$ are coincident with the other three (ordered) pairs of corners. We claim that the conditions in the following lemma are necessary for a solution.

Lemma 6.16. *Let $(S, \overline{S})$ be an optimal solution of radius r^* such that S and $\overline{S}$ are coincident with opposite corners of R, and let $\mathcal{C}' := \{C_t \mid C \in \mathcal{C}\}$ be the set of curves under translation from which $(S, \overline{S})$ was obtained. At least one of the following conditions must hold for some curve $C_t \in \mathcal{C}'$:*

(i) there must be a pair of determining vertices for either S or $\overline{S}$; or

(ii) there must be a pair of opposing vertices for S and $\overline{S}$.

Proof. Since $(S, \overline{S})$ is a valid solution, then for each translated curve $C_t \in \mathcal{C}'$, there must exist a partition of C_t defined by an index i such that $C_t[1, i] \subset S$ and $C_t[i+1, m] \subset \overline{S}$.

Assume that neither of the conditions stated in the lemma hold. Then the radius of the squares can be decreased to obtain a smaller pair of squares coincident with the same corners of R. If no vertices from the curves in $\mathcal{C}'$ lie on the inner sides of S and $\overline{S}$—that is, the sides that are not collinear with sides of R then the radius can be reduced without translating the curves in $\mathcal{C}'$. If one or more prefix (suffix) vertices of lie on the inner sides of S ($\overline{S}$), then C_t is translated in a direction determined in the following way. For each such vertex p lying on a side s of its assigned square, let $\vec{n}$ be the direction of the inner normal of s. The direction of translation is the direction of the vector obtained by summing the normal vectors. Such a direction would allow all the vertices lying on the sides of their respective squares to remain on the side, unless two vertices lie on opposing sides of the same square, i.e., condition (i) holds, or they lie on the opposing inner sides of different squares, i.e., condition (ii) holds. □

Lemma 6.16 implies that the optimality of a solution will be determined by the partition of a single curve. The minimum radius of a solution for a partition at i of a curve C_j under translation may be computed in constant time by finding the bounding boxes around the prefix and suffix of the curve, and the radius of the solution can then be obtained from the candidate pairs of determining and opposing vertices implied by the bounding boxes. Specifically, the value r_i^j is a lower bound on the optimal radius obtained by the partition at i of curve C_j, and can be computed in constant time, for example, when S is *below-left* of $\overline{S}$:

$$r_i^j := \frac{1}{2} \min \left\{ \begin{array}{l} \delta_x(C_j[1,i]), \\ \delta_x(C_j[i+1,m]), \\ (\delta_x^* - (\min_{v \in C[i+1,m]} v.x - \max_{v \in C[1,i]} v.x))/2, \\ \delta_y(C_j[1,i]), \\ \delta_y(C_j[i+1,m]), \\ (\delta_y^* - (\min_{v \in C[i+1,m]} v.y - \max_{v \in C[1,i]} v.y))/2 \end{array} \right\}.$$

An optimal solution for $\mathcal{C}$ under translation where the squares coincide with a particular pair of opposing corners of R can computed as $r := \max_{j:\, C_j \in \mathcal{C}} \min_{1 \le i \le m} r_i^j$, i.e., the minimum radius of a pair of squares covering the partition of a curve, and then determining the largest such value over all curves. The solutions are evaluated where S and $\overline{S}$ coincide with each of the four ordered pairs of opposite corners of R, and the overall solution is the smallest of these values.

We thus obtain the following result.

Theorem 6.17. *Given a set of curves $\mathcal{C}$ as input, an optimal solution to the* $(1,2)$-CENTER *problem under translation using the discrete Fréchet distance under the L_∞ metric can be computed in $O(nm)$ time and $O(nm)$ space.*

6.6.3 $(1,2)$-CENTER and L_2 metric

For the $(1,2)$-CENTER problem and L_2 we need some more sophisticated arguments, but again we use a similar basic approach.

We first consider the decision problem: Given a value $r > 0$, determine whether there exists a segment s such that $\max_{C_i \in \mathcal{C}} d_{dF}(s, C_i) \leq r$.

For each curve $C \in \mathcal{C}$ and for each vertex p of C, draw a disk of radius r centered at p and denote it by $D(p)$. Let $\mathcal{D}$ denote the resulting set of nm disks and let $\mathcal{A}(\mathcal{D})$ be the arrangement of the disks in $\mathcal{D}$. The combinatorial complexity of $\mathcal{A}(\mathcal{D})$ is $O(n^2m^2)$. Let A be a cell of $\mathcal{A}(\mathcal{D})$. Then, each curve $C = (p_1, \ldots, p_m) \in \mathcal{C}$ induces a bit vector V_C of length m; the ith bit of V_C is 1 if and only if $D(p_i) \supseteq A$. Moreover, if j is the index of the first 0 in V_C, then *the suffix of curve C at cell A* is $C[j, m]$.

We maintain the vectors V_C as we traverse the arrangement $\mathcal{A}(\mathcal{D})$, by constructing a binary tree T_C, for each curve C, as described in the previous section. The leaves of T_C correspond to the vertices of C, and in each node we store a single bit. Here, the bit at a leaf node corresponding to vertex p_i is 1 if and only if $D(p_i) \supseteq A$, where A is the current cell of the arrangement. For an internal node, the bit is 1 if and only if all the bits in the leaves of its subtree are 1. We can determine the current suffix of C in $O(\log m)$ time, and the cost of an update operation is $O(\log m)$. We also maintain the set P, where P is the union of the suffixes of the curves in $\mathcal{C}$, and its corresponding region $X = \cap_{p \in P} D(p)$. Actually, we only need to know whether X is empty or not.

We begin by constructing the trees $T_{C_1}, \ldots, T_{C_n}$ and initializing all bits to 0, which takes $O(mn)$ time. We also construct the data structures for P and X, where initially $P = C_1[1, m] \cup \cdots \cup C_n[1, m]$. This takes $O(nm \log^2(nm))$ time in total. For P we use a standard balanced search tree, and for X we use, e.g., the data structure of Sharir [Sha97], which supports updates to X in $O(\log^2(nm))$ time. We now traverse $\mathcal{A}(\mathcal{D})$ systematically, beginning with the unbounded cell of $\mathcal{A}(\mathcal{D})$, which is not contained in any of the disks of $\mathcal{D}$. Whenever we enter a new cell A from a neighboring cell separated from it by an arrangement edge, then we either enter or exit the unique disk of $\mathcal{D}$ whose boundary contains this edge. We thus first update the corresponding tree T_C accordingly, and redetermine the suffix of C. We now may need to perform $O(m)$ update operations on the data structures for P and X, so that they correspond to the current cell. At this point, if $X \neq \emptyset$, then we halt and return YES (since we know that the minimum enclosing disk of the union of the prefixes is at most r). If, however, $X = \emptyset$, then we continue to the next cell of $\mathcal{A}(\mathcal{D})$, unless there is no such cell in which case we return NO. We conclude that the decision problem can be solved in $O(n^2m^3\log^2(nm))$ time and $O(n^2m^2)$ space.

Notice that the minimum radius r^* for which the decision version returns YES, is determined by three of the nm curve vertices. Thus, we perform a binary search in the (implicit) set of potential radii (whose size is $O(n^3m^3)$) in order to find r^*. Each comparison in this search is resolved by solving the decision problem for the appropriate potential radius. Moreover, after resolving the current comparison, the potential radius for the next comparison can be found in $O(n^2m^2\log^2(nm))$ time, as in the early near-quadratic algorithms for the well-known 2-center problem, see, e.g., [AS94, JK94, KS97].

The following theorem summarizes the main result of this section.

Theorem 6.18. *Given a set of curves $\mathcal{C}$ as input, an optimal solution to the* $(1,2)$-CENTER *problem using the discrete Fréchet distance under the L_2 metric can be computed in $O(n^2m^3\log^3(nm))$ time and $O(n^2m^2)$ space.*

Chapter 7

Simplifying Chains under the Discrete Fréchet Distance

7.1 Introduction

Simplifying polygonal chains is a well-studied topic with many applications in a variety of fields of research and technology. When polygonal chains are large, running time becomes critical. A natural approach is to find a small chain which is a good approximation of the original one. For instance, many GPS applications use trajectories that are represented by sequences of densely sampled points, which we want to simplify in order to perform efficient calculations. In short, given a chain A with n vertices, we want to find a chain A' such that A' is close to A and $|A'| << n$. Curve simplification is used to simplify the representation of rivers, roads, coastlines, and other features when a map at large scale is produced. The simplification process has many advantages, such as removing unnecessary cluttering due to excessive detail, saving disk and memory space, and reducing the rendering time.

Recently, the discrete Fréchet distance has been utilized for protein backbone comparison. Within structural biology, polygonal curve alignment and comparison is a central problem in relation to proteins. Proteins are usually studied with RMSD (Root Mean Square Deviation), but recently the discrete Fréchet distance was used to align and compare protein backbones, which yielded beneficial results over RMSD in many instances [JXZ08, WLZ11]. There may be as many as 500$\sim$600 α-carbon atoms along a protein backbone (which are the nodes of the chain). This makes efficient computation a priority and is one of the reasons simplification was originally considered.

Related work. Bereg et al. [BJW$^+$08] were the first to study simplification problems under the discrete Fréchet distance. They considered two such problems. In the first, the goal is to minimize the number of vertices in the simplification, given a bound on the distance between the original chain and its simplification, and, in the second problem, the goal is to minimize this distance, given a bound k on the number of vertices in the simplification. They presented an $O(n^2)$-time algorithm for the former problem and an $O(n^3)$-time algorithm for the latter problem, both using dynamic programming, for the case where the vertices of the simplification are from the original chain. (For the arbitrary vertices case, they solve the problems in $O(n \log n)$ time and in $O(kn \log n \log(n/k))$ time, respectively.)

Agarwal et al. [AHMW05] considered the problem of approximating an ε-simplification. In this problem a polygonal curve A and an error criterion are given, and we want to find another polygonal curve A' whose vertices are a subset of the vertices of A, with minimal number of vertices, such that the error between A and A' is below a certain threshold. They considered two different error measures - Hausdorff and Fréchet error measures. For both error criteria, they presented near-linear time approximation algorithms. The Fréchet error measure is not similar to the Fréchet distance, and will be reviewed in more detail later on.

Driemel and Har-Peled [DH13] showed how to preprocess a polygonal curve in near-linear time and space, such that, given an integer $k > 0$, one can compute a simplification in $O(k)$ time which has $2k-1$ vertices of the original curve and is optimal up to a constant factor (w.r.t. the continuous Fréchet distance), compared to any curve consisting of k arbitrary vertices.

Our Results. In Section 7.3 we discuss optimal simplification problems considered by Bereg et al. [BJW+08]. We suggest and solve more general versions of these problems. In particular, we improve the result of Bereg et al. [BJW+08] mentioned above for the problem of finding the best simplification of a given length under the discrete Fréchet distance, by presenting a more general $O(n^2 \log n)$-time algorithm (rather than an $O(n^3)$-time algorithm).

In Section 7.4 we discuss approximation algorithms for simplification. First we adapt the techniques and algorithms presented by Driemel and Har-Peled [DH13] to the discrete Fréchet distance, with slightly improved approximation factors. Then we discuss the Fréchet error measure as presented in [AHMW05].

7.2 Preliminaries

In the previous chapter we used the notion of curves alignment to define DFD. Here (and in the following chapter), again, we prefer to use yet another equivalent definition, following [God91], [BJW+08] and [DH13].

Paired walk. Given two chains $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$:

A ***paired walk*** along A and B is a sequence of pairs $W = \{(A_i, B_i)\}_{i=1}^{k}$, such that $A_1, ..., A_k$ and $B_1, ..., B_k$ partition A and B, respectively, into (disjoint) non-empty sub-chains, and for any i it holds that $|A_i| = 1$ or $|B_i| = 1$. The ***cost of a paired walk*** W along A and B is

$$d_{dF}^{W}(A, B) = \max_{i} \max_{(a,b)\in A_i\times B_i} d(a, b).$$

The ***discrete Fréchet distance*** from A to B is $d_{dF}(A, B) = \min_{W} d_{dF}^{W}(A, B)$.

Simplification. Given a chain $P = (p_1, \dots, p_n)$:

A ***simplification*** of P is a chain $P' = (p_{x_1}, \dots, p_{x_k})$ of points from P, where $x_1 < x_2 < \cdots < x_k$. An ***arbitrary simplification*** of P is a chain P' with $|P'| \le |P|$.

The ***error of a simplification*** (arbitrary or non-arbitrary) P' of P is $d_{dF}(P, P')$.

Spine. Given a chain $Z = (z_1, \dots, z_n)$ and a segment pq:

The ***spine*** of Z denoted by $spine(Z)$ is the segment $z_1 z_n$. A ***spine chain*** of Z is a chain $z_{x_1}, \dots, z_{x_k}$ of points from Z, where $1 = x_1 < x_2 < \cdots < x_k = n$.

A ***split point*** of Z with respect to pq is a point z_i for which the cost of the paired walk $\{(p, Z\langle z_1, z_i\rangle), (q, Z\langle z_{i+1}, z_n\rangle)\}$ of Z and pq is $d_{dF}(Z, pq)$.

7.3 The simplification problem

As mentioned in the introduction, Bereg et al. [BJW+08] were the firsts to study the problem of simplifying 3D polygonal chains under the discrete Fréchet distance. We present a more general definition of the problem:

Problem 7.1.
Instance: Given a pair of polygonal chains A and B of lengths m and n, respectively, an integer k, and a real number $\delta > 0$.
Problem: Does there exist a chain A' of at most k vertices, such that the vertices of A' are from A and $d_{dF}(A', B) \le \delta$?

This problem induces two optimization problems (as in [BJW+08]), depending on whether we wish to optimize the length of A' or the distance between A' and B. Below we solve both of them, beginning with the former problem.

7.3.1 Minimizing k given δ

In this problem, we wish to minimize the length of A' without exceeding the allowed error bound.

Problem 7.2. Given two chains $A = (a_1, \dots, a_m)$ and $B = (b_1, \dots, b_n)$ and an error bound $\delta > 0$, find a simplification A' of A of minimum length, such that the vertices of A' are from A and $d_{dF}(A', B) \le \delta$.

For $B = A$, Bereg et al. [BJW$^+$08] presented an $O(n^2)$-time dynamic programming algorithm. (For the case where the vertices of A' are not necessarily from A, they presented an $O(n \log n)$-time greedy algorithm.)

Theorem 7.3. *Problem 7.2 can be solved in $O(mn)$ time and space.*

Proof. We present an $O(mn)$-time dynamic programming algorithm. The algorithm finds the length of an optimal simplification; the actual simplification is constructed by backtracking the algorithm's actions.

Define two $m \times n$ tables, O and X. The cell $O[i, j]$ will store the length of a minimum-length simplification A^i of $A[i \dots m]$ that begins at a_i and such that $d_{dF}(A^i, B[j \dots n]) \le \delta$. The algorithm will return the value $\min_{1 \le i \le m} O[i, 1]$.

We use the table X to assist us in the computation of O. More precisely, we define:

$$X[i, j] = \min_{i' \ge i} O[i', j]\,.$$

Notice that $X[i, j]$ is simply the minimum of $X[i + 1, j]$ and $O[i, j]$.

We compute $O[-, -]$ and $X[-, -]$ simultaneously, where the outer for-loop is governed by (decreasing) i and the inner for-loop by (decreasing) j. First, notice that if $d(a_i, b_j) > \delta$, then there is no simplification fulfilling the required conditions, so we set $O[i, j] = \infty$. Second, the entries (in both tables) where $i = m$ or $j = n$ can be handled easily. In general, if $d(a_i, b_j) \le \delta$, we set

$$O[i, j] = \min\{O[i, j + 1],\, X[i + 1, j + 1] + 1\}\,.$$

We now justify this setting. Let A^i be a minimum-length simplification of $A[i \dots n]$ that begins at a_i and such that $d_{dF}(A^i, B[j \dots n]) \le \delta$. The initial configuration of the joint walk along A^i and $B[j \dots n]$ is (a_i, b_j). The next configuration is either (a_i, b_{j+1}), $(a_{i'}, b_j)$ for some $i' \ge i + 1$, or $(a_{i'}, b_{j+1})$ for some $i' \ge i + 1$. However, clearly $X[i + 1, j + 1] \le X[i + 1, j]$, so we may disregard the middle option. □

7.3.2 Minimizing δ given k

In this problem, we wish to minimize the discrete Fréchet distance between A' and B, without exceeding the allowed length.

Problem 7.4. Given two chains $A = (a_1, \dots, a_m)$ and $B = (b_1, \dots, b_n)$ and a positive integer k, find a simplification A' of A of length at most k, such that the vertices of A' are from A and $d_{dF}(A', B)$ is minimized.

For $B = A$, Bereg et al. [BJW$^+$08] presented an $O(n^3)$-time dynamic programming algorithm. (For the case where the vertices of A' are not necessarily from A, they presented an $O(kn \log n \log(n/k))$-time greedy algorithm.) We give an

$O(mn \log(mn))$-time algorithm for our problem, which yields an $O(n^2 \log n)$-time algorithm for $B = A$, thus significantly improving the result of Bereg et al.

Theorem 7.5. *Problem 7.4 can be solved in $O(mn \log(mn))$ time and $O(mn)$ space.*

Proof. Set $D = \{d(a,b) | a \in A, b \in B\}$. Then, clearly, $d_{dF}(A', B) \in D$, for any simplification A' of A. Thus, we can perform a binary search over D for an optimal simplification of length at most k. Given $\delta \in D$, we apply the algorithm for Problem 7.2 to find (in $O(mn)$ time) a simplification A' of A of minimum length such that $d_{dF}(A', B) \le \delta$. Now, if $|A'| > k$, then we proceed to try a larger bound, and if $|A'| \le k$, then we proceed to try a smaller bound. After $O(\log(mn))$ iterations we reach the optimal bound. □

Remark 7.6. In Problem 7.2 we could require a simplification of maximum length instead of minimum length. In this case, the problem bacomes a discrete one-sided version of the partial Fréchet similarity problem, mentioned in the introduction of Chapter 2. The goal is to match a maximal portion of the points from A to B, while ensuring a certain error bound. This problem aims at situations where the extent of a pre-required similarity is known (and given by δ), and we wish to know how much (and which parts) of A are similar to B in this extent. This problem can be solved in a similar manner using the same dynamic programming algorithm. Also, in Problem 7.4 we could require at least instead of at most k vertices. In this case, again this problem relates to the partial Fréchet similarity problem. However, now the extent of similarity is not given, but at least k vertices should be matched. This aims to a case where B is a library curve and A is a sequence of densely sampled points that should match B, but might contain outliers. We wish to filter the outliers from A (non-outliers might be filtered too) while keeping it close to B.

7.4 Universal vertex permutation for curve simplification

In [DH13], Driemel and Har-Peled presented a collection of data structures for Fréchet distance queries. They used it in order to give an approximation algorithm for the Fréchet distance with shortcuts problem (see Chapter 2), and also for obtaining a universal approximate simplification. This is done by computing a permutation of the vertices of the input curve, in near-linear time and space, such that the approximate simplification of size k is the subcurve defined by the first k vertices in this permutation. We follow their results and apply their techniques to the discrete Fréchet distance, with a slight improvement of the approximation factor.

7.4.1 A segment query to the entire curve

In this section we describe a data-structure that preprocesses a chain $Z = (z_1, ..., z_n)$, and given a query segment pq returns a $(1-\varepsilon)$-approximation of the discrete Fréchet distance $d_{dF}(Z, pq)$, i.e., a value Δ such that $(1-\varepsilon)d_{dF}(Z, pq) \leq \Delta \leq d_{dF}(Z, pq)$.

The data structure

We need the following lemmas:

Lemma 7.7. *[Dri13]Given a point $u \in \mathbb{R}^d$,a parameter $0 < \varepsilon \leq 1$ and an interval $[\alpha, \beta] \subseteq \mathbb{R}$, one can compute in $O(\varepsilon^{-d} \log(\beta/\alpha))$ time and space an exponential grid of points $G(u)$, such that for any point $p \in \mathbb{R}^d$ with $\|p - u\| \in [\alpha, \beta]$, one can compute in constant time a grid point $p' \in G(u)$, with $\|p - p'\| \leq (\varepsilon/2) \|p - u\|$.*

Lemma 7.8. *Let pq be a segment and Z a chain, then*

$$d_{dF}(pq, Z) \geq d_{dF}(spine(Z), Z)/2.$$

Proof. Let $spine(Z) = uv$. Clearly,

$$d_{dF}(pq, Z) \geq \max(\|p - u\|, \|q - v\|) = d_{dF}(spine(Z), pq).$$

By the triangle inequality, we get

$$d_{dF}(spine(Z), Z) \leq d_{dF}(spine(Z), pq) + d_{dF}(pq, Z) \leq 2d_{dF}(pq, Z).$$

□

Preprocessing. Let uv be the spine of Z, and $L = d_{dF}(Z, uv)$. We construct two exponential grids $G(u)$ and $G(v)$ of points around u and v, both with the range $[\varepsilon L/4,\ L/\varepsilon]$ as described in the lemma. We also add u to $G(u)$ and v to $G(v)$. For every pair of points $\langle p', q' >\in G(u) \times G(v)$ we compute $D[p', q'] = d_{dF}(Z, p'q')$. The preprocessing time is $O(n\varepsilon^{-2d} \log^2(1/\varepsilon))$, as we have $O(\varepsilon^{-d} \log(1/\varepsilon))$ points in each grid, and computing the discrete Fréchet distance of a curve to a segment takes $O(n)$ time. The space required is $O(\varepsilon^{-2d} \log^2(1/\varepsilon))$.

Answering a query. Given a query segment pq, we want to return an approximation to the distance $d_{dF}(Z, pq)$.

We compute the distance $r = \max\{\|p - u\|, \|q - v\|\}$.

If $r \leq \varepsilon L/4$, we return $L - r$.

If $r \geq L/\varepsilon$ we return r.

Otherwise, w.l.o.g $r = \|p - u\|$ so by Lemma 7.7 we can find $p' \in G(u)$ such that $\|p - p'\| \leq (\varepsilon/2) \|p - u\| = (\varepsilon/2)r$. If $\|q - v\| \geq \varepsilon L/4$, find a grid point $q' \in G(v)$ such that $\|q - q'\| \leq (\varepsilon/2) \|q - v\| \leq (\varepsilon/2)r$. Else, if $\|q - v\| \leq \varepsilon L/4$, set $q' = v$. Finally, return $D[p', q'] - \max\{\|p - p'\|, \|q - q'\|\}$.

Analysis

Lemma 7.9. *Given a chain Z with n points in $\mathbb{R}^d$ and $0 < \varepsilon < 1$, one can build a data structure in $O(n\varepsilon^{-2d}\log^2(1/\varepsilon))$ time and $O(\varepsilon^{-2d}\log^2(1/\varepsilon))$ space, such that given a query segment pq, one can return in $O(1)$ time a value Δ such that $(1-\varepsilon)d_{dF}(Z,pq) \le \Delta \le d_{dF}(Z,pq)$.*

Proof. As described above, the preprocessing of the data structure takes $O(n\varepsilon^{-2d}\log^2(1/\varepsilon))$ time and the space required is $O(\varepsilon^{-2d}\log^2(1/\varepsilon))$. Given a segment query pq, we can compute $r = \max\{\|p-u\|, \|q-v\|\} = d_{dF}(pq,uv)$ in $O(1)$ time. Let Δ be the returned value, we show that $(1-\varepsilon)d_{dF}(Z,pq) \le \Delta \le d_{dF}(Z,pq)$:

If $r \le \varepsilon L/4$, we return $\Delta = L - r$. By the triangle inequality,

$$\Delta = L - r = d_{dF}(Z,uv) - d_{dF}(pq,uv) \le d_{dF}(Z,pq).$$

$$\begin{aligned} d_{dF}(Z,pq) &\le L + r \le L + \varepsilon L/4 = L + \varepsilon L - \varepsilon L/4 - \varepsilon L/2 \le \\ &\le L + \varepsilon L - r - 2r \le L + \varepsilon L - r - \varepsilon r = \\ &= (1+\varepsilon)(L-r) = (1+\varepsilon)\Delta \le \Delta/(1-\varepsilon). \end{aligned}$$

If $r \ge L/\varepsilon$, we return $\Delta = r$. The values $\|p-u\|, \|q-v\|$ participate in computing $d_{dF}(Z,pq)$, so we have $\Delta = r \le d_{dF}(Z,pq)$. By the triangle inequality

$$d_{dF}(Z,pq) \le L + r \le \varepsilon r + r = (1+\varepsilon)r = (1+\varepsilon)\Delta \le \Delta/(1-\varepsilon).$$

Otherwise, we return

$$\Delta = D[p',q'] - \max\{\|p-p'\|, \|q-q'\|\} = d_{dF}(Z,p'q') - d_{dF}(pq,p'q').$$

First, by the triangle inequality we have

$$\Delta = d_{dF}(Z,p'q') - d_{dF}(pq,p'q') \le d_{dF}(Z,pq),$$

and also

$$d_{dF}(Z,pq) \le d_{dF}(Z,p'q') + d_{dF}(pq,p'q') \le \Delta + 2d_{dF}(pq,p'q'). \tag{7.1}$$

Again, w.l.o.g we assume $r = \|p-u\|$ and we have two cases:

1. If $\|q-v\| \ge \varepsilon L/4$, then q' is also a grid point and

$$d_{dF}(pq,p'q') \le (\varepsilon/2)d_{dF}(pq,uv) \le (\varepsilon/2)d_{dF}(Z,pq).$$

From Equation (7.1):

$$d_{dF}(Z,pq) \le \Delta + 2(\varepsilon/2)d_{dF}(Z,pq) = \Delta + \varepsilon d_{dF}(Z,pq),$$

and we get that $\Delta \ge (1-\varepsilon)d_{dF}(Z,pq)$.

2. Else, $\|q-v\| \le \varepsilon L/4$, then $q' = v$ and

$$\begin{aligned} d_{dF}(pq,p'v) &= \max\{\|p-p'\|, \|q-v\|\} \\ &\le \max\{(\varepsilon/2)\|p-u\|, \varepsilon L/4\} \\ &= (\varepsilon/2)\max\{\|p-u\|, L/2\}. \end{aligned}$$

By Lemma 7.8 we have $L/2 = d_{dF}(Z,uv)/2 \le d_{dF}(Z,pq)$, and from Equation (7.1):

$$d_{dF}(Z,pq) \le \Delta + 2(\varepsilon/2)\max\{\|p-u\|, L/2\} \le \Delta + \varepsilon d_{dF}(Z,pq),$$

or $\Delta \ge (1-\varepsilon)d_{dF}(Z,pq)$.

□

7.4.2 A segment query to a subcurve

In this section we describe a data-structure that preprocesses a sequence Z of n points, and given a query segment pq and a subcurve $Z\langle u,v\rangle$ returns a $(1-\varepsilon)$ approximation of the Discrete Fréchet distance $d_{dF}(Z\langle u,v\rangle,pq)$, i.e., a value Δ such that

$$(1-\varepsilon)d_{dF}(Z\langle u,v\rangle,pq) \le \Delta \le d_{dF}(Z\langle u,v\rangle,pq).$$

The data structure

First notice that the Discrete Fréchet distance of a chain Z to a segment pq is actualy a partition of Z into two subchains, Z_1 and Z_2, such that $Z = Z_1Z_2$, and

$$d_{dF}(Z,pq) = \max\{\max_{z\in Z_1}\|z-p\|, \max_{z\in Z_2}\|z-q\|\}.$$

The last point of Z_1 is a *split* point of Z with respect to pq. We need the following lemma:

Lemma 7.10. *Let Z be a chain, and pq a segment. Let $Z_1,...,Z_k$ be a partition of Z into k subchains such that $Z = Z_1Z_2...Z_k$. Let*

$$\delta_i = \max\{\max_{j<i} d_{dF}(Z_j,p),\ d_{dF}(Z_i,pq),\ \max_{j>i} d_{dF}(Z_j,q)\},$$

$1 \le i \le k$, and set $\alpha = \min_i \delta_i$. Let

$$\delta_i' = \max\{\max_{j \le i} d_{dF}(Z_j, p),\ \max_{j > i} d_{dF}(Z_j, q)\},$$

$1 \le i \le k$, and set $\beta = \min_i \delta_i'$. Then $d_{dF}(Z, pq) = \min_i\{\alpha, \beta\}$.

Proof. Let $\delta = d_{dF}(Z, pq)$. First notice that $d_{dF}(Z_j, p) = \max_{z \in Z_j} \|z - p\|$, and thus $\max_{j<i} d_{dF}(Z_j, p) = \max_{z \in Z_j, j<i} \|z - p\|$. Symmetrically, $\max_{j>i} d_{dF}(Z_j, p) = \max_{z \in Z_j, j>i} \|z - p\|$. Let i be the index such that $\delta_i = \alpha$. The split point of Z_i with respect to pq defines a partition of the entire sequence Z into two subchains, and α is the weight of a Fréchet walk of Z and pq with respect to that split point. A similar claim is true for $\delta_i' = \beta$ and the last point of Z_i as the split point. Thus we have $\alpha \ge \delta$ and $\beta \ge \delta$. Now let $z \in Z$ be the split vertex of Z with respect to pq, and let Z_l be the subchain containing z. If z is not the end point of Z_l, then we have

$$\delta = \max\{\max_{j<l} d_{dF}(Z_j, p),\ d_{dF}(Z_l, pq),\ \max_{j>l} d_{dF}(Z_j, q)\} = \delta_l \ge \alpha,$$

and if z is the end point of Z_l, then we have

$$\delta = \max\{\max_{j \le l} d_{dF}(Z_j, p),\ \max_{j>l} d_{dF}(Z_j, q)\} = \delta_l' \ge \beta.$$

We conclude that $\delta = \min_i\{\alpha, \beta\}$. □

Preprocessing. Similarly to the construction by Driemel and Har-Peled, we build a balanced binary tree T on the points of Z. Every node v of T corresponds to a subchain of Z, denoted by *seq*(v). For every node v we build the data structure $ES(v)$ of Lemma 7.9.

Answering a query. Given a query segment pq, and two points u, v on Z, we want to return an approximation to the distance $d_{dF}(Z\langle u, v\rangle, pq)$. First, compute $k = O(\log n)$ nodes $v_1, ..., v_k$ of T, such that $Z\langle u, v\rangle = seq(v_1)seq(v_2)...seq(v_k)$. Let ϕ_i^{pq} be the $(1-\varepsilon)$-approximation of $d_{dF}(seq(v_i), pq)$ computed by $ES(v_i)$. Let ϕ_i^p and ϕ_i^q be the $(1-\varepsilon)$-approximation of $d_{dF}(seq(v_i), p)$ and $d_{dF}(seq(v_i), q)$ respectively, also computed by $ES(v_i)$. Now we can compute for every i in increasing order the value $\max_{j<i} \phi_j^p$, and in decreasing order the value $\max_{j>i} \phi_j^q$, in $O(\log n)$ time. Finally, we return

$$\min\{\min_i\{\max\{\max_{j<i} \phi_i^p, \max_{j>i} \phi_i^q, \phi_i^{pq}\}\}, \min_i\{\max\{\max_{j \le i} \phi_i^p, \max_{j>i} \phi_i^q\}\}\}$$

as a $(1-\varepsilon)$-approximation of the distance $d_{dF}(Z\langle u, v\rangle, pq)$.

Analysis.

Lemma 7.11. *Given a polygonal curve Z with n vertices in $\mathbb{R}^d$ and $0 < \varepsilon < 1$, one can build a data structure in $O(n \log n \varepsilon^{-2d} \log^2(1/\varepsilon))$ time and $O(n\varepsilon^{-2d} \log^2(1/\varepsilon))$ space, such that given a query segment and two points u, v on Z, one can $(1-\varepsilon)-$approximate $d_{dF}(Z\langle u,v\rangle, pq)$ in $O(\log n)$ time.*

Proof. As described above, the preprocessing of the data structure takes $O(n\varepsilon^{-2d} \log^2(1/\varepsilon))$ time in each level of the tree T, and $O(n \log n\varepsilon^{-2d} \log^2(1/\varepsilon))$ time overall. The space required is $O(\varepsilon^{-2d} \log^2(1/\varepsilon))$ for each node, and $O(n\varepsilon^{-2d} \log^2(1/\varepsilon))$ for the entire tree. Given a segment query pq and two points u, v on Z, we can compute $k = O(\log n)$ nodes $v_1, ..., v_k$ of T, and return

$$\min\{\min_i\{\max\{\max_{j<i}\phi_j^p, \phi_i^{pq}, \max_{j>i}\phi_j^q\}\}, \min_i\{\max\{\max_{j\leq i}\phi_j^p, \max_{j>i}\phi_j^q\}\}\}$$

in $O(\log n)$ time, as described above.

Let Δ be the returned value, we show that $(1-\varepsilon)d_{dF}(Z\langle u,v\rangle, pq) \leq \Delta \leq d_{dF}(Z\langle u,v\rangle, pq)$:

We have by Lemma 7.9:

$$\begin{aligned}(1-\varepsilon)d_{dF}(seq(v_i), p) &\leq \phi_i^p \leq d_{dF}(seq(v_i), p)\\ (1-\varepsilon)d_{dF}(seq(v_i), q) &\leq \phi_i^q \leq d_{dF}(seq(v_i), q)\\ (1-\varepsilon)d_{dF}(seq(v_i), pq) &\leq \phi_i^{pq} \leq d_{dF}(seq(v_i), pq).\end{aligned}$$

Using Lemma 7.10, we get that $\Delta \leq d_{dF}(Z\langle u,v\rangle, pq)$ and $\Delta \geq (1-\varepsilon)d_{dF}(Z\langle u,v\rangle, pq)$, by replacing ϕ_j^p, ϕ_j^q and ϕ_i^{pq} by $d_{dF}(seq(v_j), p)$, $d_{dF}(seq(v_j), q)$ and $d_{dF}(seq(v_i), pq)$, respectively. □

7.4.3 Universal simplification

Given a sequence Z, our goal is to find a permutation $\pi(Z)$ of the points of Z, such that for any k, $\pi(Z)\langle 1, k\rangle$ is a good approximation to the optimal simplification of Z with k points (not necessarily from Z).

We build a new data-structure using the one described above.

Construction of the permutation. We use the same idea of the algorithm shown by Dreimel and Har-Peled. The idea is to compute for each point of the sequence the error caused by removing it from the sequence, and then remove the point with the lowest error. Then, update the values of its neighbours with respect to the remaining points, and continue until all the points (except the two endpoints) are removed.

Let $Z =< z_1, ..., z_n >$ be the sequence of n points, given by a doubly-linked list. We build for Z the data structure of Lemma 7.11 with $\varepsilon = \frac{1}{10}$.

For each internal point z of Z, let z^+ and z^- be its successor and predecessor on Z respectively, and let $\phi(z)$ be a (9/10)-approximation of $d_{dF}(Z\langle z^-, z^+\rangle, z^-z^+)$. Insert z with weight $\phi(z)$ to a minimum heap H. Finally, insert z_1 and z_n to H with weight $+\infty$.

Repeat until H is empty: extract the point z with minimum $\phi(z)$ from H. Let z^+ and z^- be its successor and predecessor on Z_H respectively, where Z_H is a spine sequence of Z containing only the points of H. Compute the new weights for z^+ and z^- (their successor and predecessor are with respect to Z_H after removing z from H, but the approximated distance is to a subchain of the *original* sequence Z).

Reverse the order of the points extracted from the heap, and return the permutation $\pi = < v_1, v_2, ..., v_n >$ (v_1 and v_2 are the endpoints of Z).

Now given a parameter k, we want to find the spine sequence Z_{π_k}, where π_k is the set of the first k points of π. We store $O(\log n)$ spine sequences of Z: for $i = 1... \lfloor \log n \rfloor$, we compute $Z_{\pi_{2^i}}$ by removing from $Z_{\pi_{2^{i+1}}}$ all the points that are not in π_{2^i}. This construction can be done in linear time and space. Given a query k, we copy the sequence $Z_{\pi_{2^i}}$ such that $2^i \geq k \geq 2^{i-1}$, and remove all the points that are not in π_k. This can be done in $O(k)$ time.

Analysis. We need a few lemmas:

Lemma 7.12. *Let Z be a chain, and p, q two points from Z. Then $d_{dF}(spine(Z), Z) \geq d_{dF}(pq, Z\langle p, q\rangle)/2$.*

Proof. Denote by u and w the end points of Z ($spine(Z) = uw$). Let $\delta = d_{dF}(uw, Z)$, and let $B(u, \delta)$, $B(w, \delta)$ the disks with radius δ around u and w respectively. Observe that the union of the disks covers all the points of Z. Let v be the last vertex that is matched to u, and v' the first vertex that is matched to w in a Fréchet walk with weight δ. We have two cases to consider:

1. If p and q are both between u and v, then $B(p, 2\delta)$ covers the entire disk $B(u, \delta)$ and thus the entire subchain $Z\langle u, v\rangle$ that includes $Z\langle p, q\rangle$. We can conclude that $d_{dF}(pq, Z\langle p, q\rangle) \leq 2\delta$. Symmetrically, the same argument holds when p and q are both between v' and w.

2. If p is between u and v and q is between v' and w, then $B(p, 2\delta)$ covers the disk $B(u, \delta)$ that covers the entire subchain $Z\langle u, v\rangle$, and $B(q, 2\delta)$ is covering $Z\langle v', w\rangle$. The Fréchet walk that matches all the vertices of $Z\langle p, v\rangle$ to p and all the vertices of $Z\langle v', q >$ to q gives us $d_{dF}(pq, Z\langle p, q\rangle) \leq 2\delta$.

□

Let $\pi = < v_1, v_2, ..., v_n >$ be the permutation returned by the preprocessing algorithm, π_k be the first k vertices of π, and $Z_{\pi_k} =< u_1, ..., u_k >$ be the first k

vertices of π by their ordering along Z. Denote by $\phi(v_i)$ the weight of v_i at the time of extraction.

Lemma 7.13. *Given a parameter k, $d_{FD}(Z, Z_{\pi_k}) \leq \max\limits_{1 \leq i < k} d_{dF}(Z\langle u_i, u_{i+1}\rangle, u_i u_{i+1})$.*

Proof. Let z_i be the split vertex of $d_{dF}(Z\langle u_i, u_{i+1}\rangle, u_i u_{i+1})$ for every $1 \leq i < k$. Consider the walk $W = \{(u_1, Z\langle u_1, z_1\rangle)\} \cup \{(u_i, Z\langle z_{i-1}^+, z_i\rangle)\}_{i=2}^{k-1} \cup \{(u_k, Z\langle z_{k-1}^+, u_k\rangle)\}$ (See Figure 7.1). Clearly, $\phi(W) = \max\limits_{1 \leq i < k} d_{dF}(Z\langle u_i, u_{i+1}\rangle, u_i u_{i+1})$. It also holds that $d_{FD}(Z, Z_{\pi_K}) \leq \phi(W)$ because W is some Fréchet walk of Z and Z_{π_k}. $\square$

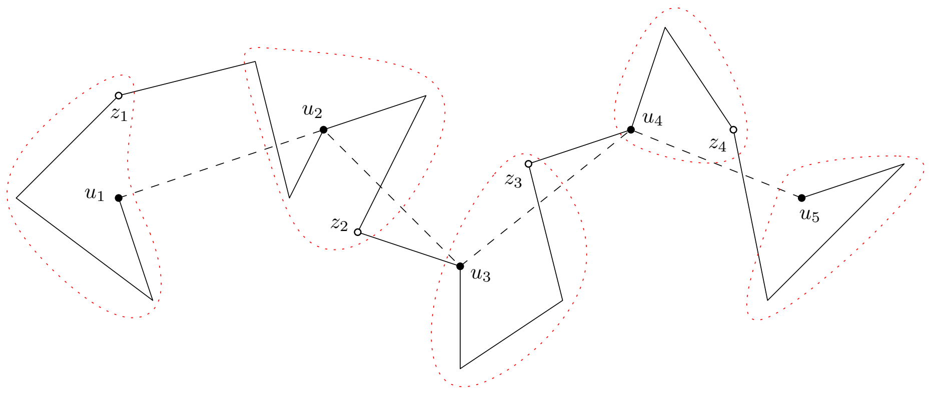


Figure 7.1: The walk W that was contracted using the split points z_i obtained from computing the distance $d_{dF}(Z\langle u_i, u_{i+1}\rangle, u_i u_{i+1})$ for every $1 \leq i < k$.

Lemma 7.14. *Consider the permutation π. Then, for every $1 \leq i \leq n$ and $i \leq j \leq n$, $\phi(v_j) \leq 2 \cdot \frac{10}{9}\phi(v_i)$.*

Proof. Let $\phi_j(v_i)$ be the weight of v_i at the time of extracting v_j. Clearly, we have $\phi(v_j) \leq \phi_j(v_i)$ because the algorithm chooses to extract the point with the minimum weight. Notice that the weight of v_i at the time of extraction, $\phi(v_i)$, is a $\frac{9}{10}$-approximation to $d_{dF}(Z\langle u^i, w^i\rangle, u^i w^i)$ for some points u^i, w^i, and the weight of v_i at the time of extracting v_j, $\phi_j(v_i)$, is a $\frac{9}{10}$-approximation to $d_{dF}(Z\langle u^j, w^j\rangle, u^j w^j)$ for some points u^j, w^j, such that $Z\langle u^j, w^j\rangle$ is a subchain of $Z\langle u^i, w^i\rangle$. The reason for that is that the subchain that determines the weight is always expanding, because we only remove possible end points. By Lemma 7.12 we get

$$\phi_j(v_i) \leq d_{dF}(Z\langle u^j, w^j\rangle, u^j w^j) \leq 2 d_{dF}(Z\langle u^i, w^i\rangle, u^i w^i) \leq 2 \cdot \frac{10}{9}\phi(v_i)$$

$\square$

Lemma 7.15. *For any $3 \leq i \leq n-1$, $d_{dF}(Z, Z_{\pi_i}) \leq 2 \cdot \left(\frac{10}{9}\right)^2 \phi(v_{i+1})$.*

Proof. By Lemma 7.13 we have $d_{FD}(Z, Z_{\pi_i}) \leq \max\limits_{1 \leq j < i} d_{dF}(Z\langle u_j, u_{j+1}\rangle, u_j u_{j+1})$. If u_{j+1} is the successor of u_j on Z, then $d_{dF}(Z\langle u_j, u_{j+1}\rangle, u_j u_{j+1}) = 0$. Else, there must be a point from $\pi \backslash \pi_i =< v_{i+1}, ..., v_n >$ that is between u_j and u_{j+1}. Let v_k be such a point with minimal index, meaning it was the last such point to be extracted, then in the time of its extraction it holds that $d_{dF}(Z\langle u_j, u_{j+1}\rangle, u_j u_{j+1}) \leq \frac{10}{9}\phi(v_k)$. Now we have $\max\limits_{1\leq j<i} d_{dF}(Z\langle u_j, u_{j+1}\rangle, u_j u_{j+1}) \leq \frac{10}{9} \max\limits_{i+1\leq j\leq n} \phi(v_j)$, and by Lemma 7.14 $d_{dF}(Z, Z_{\pi_i}) \leq \frac{10}{9} \max\limits_{i+1\leq j\leq n} \phi(v_j) \leq 2\cdot\left(\frac{10}{9}\right)^2 \phi(v_{i+1})$. □

Lemma 7.16. *Given a parameter $2 \leq k \leq \frac{n}{2} - 1$, let Y_k be a sequence with k points (not necessarily from Z) and with the smallest Fréchet distance from Z. Then $d_{dF}(Z, Y_k) \geq \phi(v_{K+1})/2$, where $K = 2k - 1$.*

Proof. Let $Y_k = (w_1, ..., w_k)$ be a sequence with the smallest discrete Fréchet distance from Z. Let $\delta = d_{dF}(Z, Y_k)$, and $W = \{(Z^i, Y^i)\}$ a Fréchet walk of Z and Y_k with weight δ. W.l.o.g, we can assume that $|Y^i| = 1$ for all i, otherwise, we can build such a sequence with k points and distance δ (See Remark 7.17). Now we can declare a matching function f:

$$f(w_i) = \begin{cases} z^i \in Z^i & , 2 \leq i \leq k-1 \\ z_1 & , i = 1 \\ z_n & , i = k \end{cases}$$

where z^i is some representative point from Z^i (see Figure 7.2). Denote the image of f by $f(Y_k)$. The points of $f(Y_k)$ partition Z into $k-1$ subchains. There are $2k-1 > 2(k-1)$ points in π_K, so by the pigeon hole principle there must be three consecutive points u_i, u_{i+1}, u_{i+2} of Z_{π_K} between two consecutive points $f(w_j)$ and $f(w_{j+1})$ (not including $f(w_{j+1})$, see Figure 7.3). We have $Z\langle u_i, u_{i+2}\rangle \subseteq Z\langle f(w_j), f(w_{j+1})\rangle$, so by Lemma 7.8:

$$\begin{aligned} d_{dF}(Z, Y_k) &\geq d_{dF}(Z\langle f(w_j), f(w_{j+1})\rangle, w_j w_{j+1}) \\ &\geq \min \begin{cases} d_{dF}(Z\langle u_i, u_{i+2}\rangle, w_i w_{i+1}), \\ d_{dF}(Z\langle u_i, u_{i+2}\rangle, w_i), \\ d_{dF}(Z\langle u_i, u_{i+2}\rangle, w_{i+1}) \end{cases} \\ &\geq d_{dF}(Z\langle u_i, u_{i+2}\rangle, u_i u_{i+2})/2 \end{aligned}$$

When v_{K+1} was extracted, the three points u_i, u_{i+1}, u_{i+2} were still in H, thus the weight of u_{i+1} at that time was a $\frac{9}{10}$-approximation to $d(Z\langle u_i, u_{i+2}\rangle, u_i u_{i+2})$, resulting

$$\begin{aligned} d_{dF}(Z\langle u_i, u_{i+2}\rangle, u_i u_{i+2})/2 &\geq \phi_{K+1}(u_{i+1})/2 \\ &\geq \phi_{K+1}(v_{K+1})/2 = \phi(v_{K+1})/2 \end{aligned}$$

as the algorithm extract the vertex with minimum weight in each step. □

Remark 7.17. Let $\delta = d_{dF}(Z, Y_k)$, and $W = \{(Z^i, Y^i)\}$ a Fréchet walk of Z and Y_k with weight δ. Assume there exists some Y^i with $|Y^i| > 1$ (and $|Z^i| = 1$). Remove from Y_k (and Y^i) the last point of Y^i. Now $\phi(W) \leq \delta$. We have $k \leq n$, and thus we can find a pair (Z^j, Y^j) with $|Y^j| = 1$ and $|Z^j| > 1$. Add the first point z of Z^j to Y_k, remove it from Z^j, and add a new pair $(\{z\}, \{z\})$ to W. Now Y_k has exactly k points, and W is a Fréchet walk of Z and Y_k with $\phi(W) \leq \delta$. Continue this process until for any i, $|Y^i| = 1$.

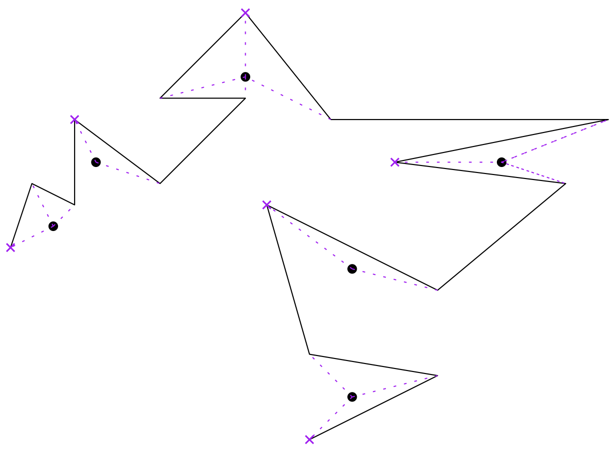


Figure 7.2: The function f. The black points are the points of Y_k and the purple crosses are the image of f.

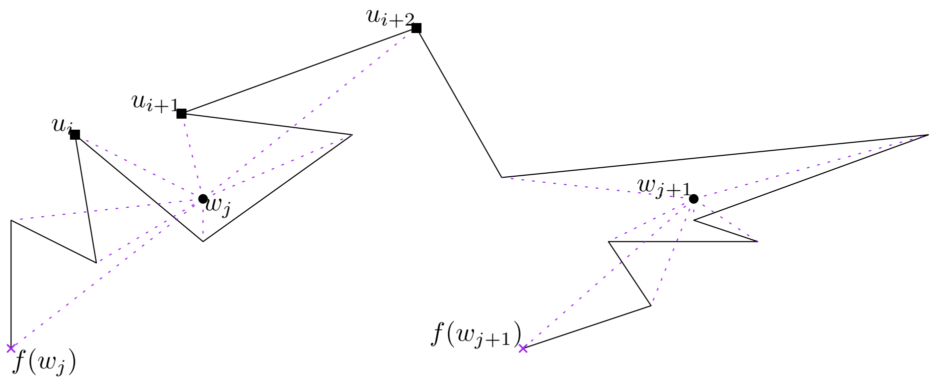


Figure 7.3: Three consecutive points u_i, u_{i+1}, u_{i+2} of Z_{π_K} between two consecutive points $f(w_j), f(w_{j+1})$ of $f(Y_k)$.

Theorem 7.18. *Given a chain Z with n points, we can preprocess it using $O(n)$ space in $O(n \log^2 n)$ time, such that given a parameter $k \in \mathbb{N}$, we can output in $O(k)$ time a $(2k-1)$-spine sequence Z' of Z and a value δ such that*

1. $d_{dF}(Z, Y_k) \geq \frac{1}{2}\delta$, *and*
2. $2 \cdot \left(\frac{10}{9}\right)^2 \delta \geq d_{dF}(Z, Z')$

where Y_k is a sequence with k points and with the smallest Discrete Fréchet distance to Z. The output Z' is a factor 5 approximation to Y_k.

Proof. We use the algorithm described above to obtain a spine sequence $Z' = Z_{V_K}$ for $K = 2k - 1$, and the value $\delta = \phi(v_{K+1})$. Indeed,

$$d_{dF}(Z, Z') \leq 2 \cdot \left(\frac{10}{9}\right)^2 \delta \leq \frac{5}{2}\delta \leq 5 d_{dF}(Z, Y_k)$$

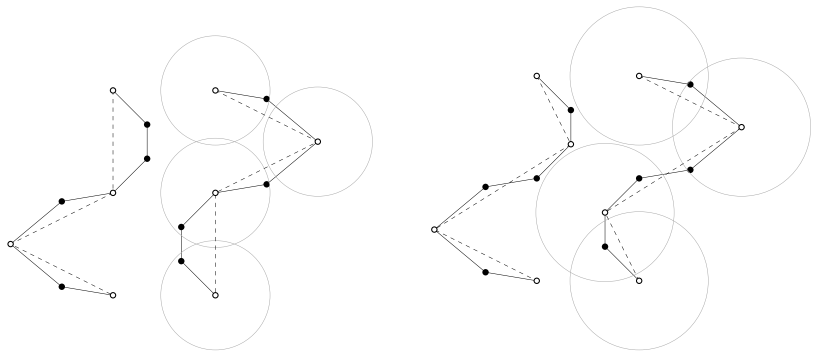
□

Chapter 8

The Chain Pair Simplification Problem

8.1 Introduction

When polygonal chains are large, it is difficult to efficiently compute and visualize the structural resemblance between them. Simplifying two aligned chains independently does not necessarily preserve the resemblance between the chains; see Figure 8.1. Thus, the following question arises: Is it possible to simplify both chains in a way that will retain the resemblance between them?



(a) Simplifying the chains separately does not necessarily preserve the resemblance between them.

(b) A simplification of the chains that preserves their resemblance.

Figure 8.1: Separate simplification vs. simultaneous simplification. The simplification was bounded to 4 vertices chosen from the chain (marked in white). The unit disks are illustrates the Fréchet distance between the right simplifications to their corresponding right chains, and their radius is larger in (b).

This question in the context of protein backbone comparison has led Bereg et al. [BJW$^+$08] to pose the Chain Pair Simplification problem (CPS). In this problem, the goal is to simplify both chains simultaneously, so that the discrete Fréchet distance between the resulting simplifications is bounded. More precisely, given two chains A

and B of lengths m and n, respectively, an integer k and three real numbers δ_1,δ_2,δ_3, one needs to find two chains A',B' with vertices from A,B, respectively, each of length at most k, such that $d_1(A, A') \leq \delta_1$, $d_2(B, B') \leq \delta_2$, $d_{dF}(A', B') \leq \delta_3$ (d_1 and d_2 can be any similarity measures and d_{dF} is the discrete Fréchet distance). When the chains are simplified using the Hausdorff distance, i.e., d_1, d_2 is the Hausdorff distance (CPS-2H), the problem becomes **NP**-complete [BJW+08]. However, the complexity of the version in which d_1, d_2 is the discrete Fréchet distance (CPS-3F) has been open since 2008.

Related work. As mentioned earlier, simplification under the discrete Fréchet distance was first addressed in 2008 when the Chain Pair Simplification (CPS) problem was proposed by Bereg et al. [BJW+08]. They proved that CPS-2H is **NP**-complete, and conjectured that so is CPS-3F. Wylie et al. [WLZ11] gave a heuristic algorithm for CPS-3F, using a greedy method with backtracking, and based on the assumption that the (Euclidean) distance between adjacent α-carbon atoms in a protein backbone is almost fixed. Later, Wylie and Zhu [WZ13] presented an approximation algorithm with approximation ratio 2 for the optimization version of CPS-3F. Their algorithm actually solves the optimization version of a related problem called CPS-3F^+, it uses dynamic programming and its running time is between $O(mn)$ and $O(m^2n^2)$ depending on the input simplification parameters.

The discrete Fréchet with shortcuts problem (studied in Chapter 2) can be interpreted as a special cases of CPS-3F. Taking shortcuts on both of the chains can be interpreted as simplifying both of the chains while preserving the resemblance between them. Unlike CPS-3F, the difference between an original chain and its simplification (in the two-sided variant) can be big, since the sole goal is to minimize the discrete Fréchet distance between the two simplified chains. (For this reason, in the shortcuts problem we do not allow both the man and the dog to move simultaneously, since, otherwise, they would both jump directly to their final points.) Moreover, the length of a simplification is only bounded by the length of the corresponding chain.

Our results. In Section 8.3 we introduce the weighted chain pair simplification problem and prove that weighted CPS-3F is weakly **NP**-complete. Then, in Section 8.4, we resolve the question concerning the complexity of CPS-3F by proving that it is polynomially solvable, contrary to what was believed. We do this by presenting a polynomial-time algorithm for the corresponding optimization problem. We actually prove a stronger statement, implying, for example, that if weights are assigned to the vertices of only one of the chains, then the problem remains polynomially solvable. Since the time complexity of our algorithm is impractical for our motivating biological application, we devise a sophisticated $O(m^2n^2 \min\{m, n\})$-time dynamic programming algorithm for the minimization problem of CPS-3F. Besides being interesting from a theoretical point of view, only after developing (and implementing) this algorithm, were we able to apply the CPS-3F minimization problem to datasets from the Protein Data Bank (PDB), see [FFK$^+$15]. Finally, in this section we also consider the 1-sided version of CPS under DFD. We present simpler and more efficient algorithms for these problems.

We also consider, for the first time, the CPS problem where the vertices of the simplifications A', B' may be arbitrary points, Steiner points, i.e., they are not necessarily from A, B, respectively. Since this problem is more general, we call it General CPS, or GCPS for short.

In Section 8.5, we show that GCPS-3F is polynomially solvable by presenting a (relatively) efficient polynomial-time algorithm for GCPS, or more precisely, for its corresponding optimization problem. As a first step towards devising such an algorithm, we had to characterize the structure of a solution to the problem. This was quite difficult, since on the one hand, we have full freedom in determining the vertices of the simplifications, but, on the other hand, the definition of the problem induces an implicit dependency between the two simplifications. The second challenge in devising such an algorithm, is to reduce its time complexity (which is unavoidably high), by making some non-trivial observations on the combinatorial complexity of an arrangement of complex objects that arises, and by applying some sophisticated tricks. Since the time complexity of our algorithm is still rather high, it makes sense to resort to more realistic approximation algorithms, therefore we give an $O(m+n)^4$-time 2-approximation algorithm for the problem. In addition, we consider the 1-sided version of GCPS.

Finally, in Section 8.6 we investigate GCPS-2H, showing that it is **NP**-complete and presenting an approximation algorithm for the problem.

8.2 Preliminaries

A formal definition of the discrete Fréchet distance was given in Section 1.1, and additional equivalent definitions were used in Sections 2.2, 5.2 and 7.2. In this chapter we refer to the definition from Section 7.2.

Let $A = (a_1, \dots, a_n)$ and $B = (b_1, \dots, b_m)$ be two sequences of points in $\mathbb{R}^d$. We denote by $d(a, b)$ the distance between two points $a, b \in \mathbb{R}^d$. For $1 \le i \le j \le n$, we denote by $A[i, j]$ the subchain $a_i, a_{i+1}, \dots, a_j$ of A.

A ***Fréchet walk*** along A and B is a paired walk W along A and B for which $d^W_{dF}(A, B) = d_{dF}(A, B)$.

A ***δ-simplification*** of A w.r.t. distance d_1, is a sequence of points $A' = (a'_1, \dots, a'_k)$, such that $k \le n$ and $d_1(A, A') \le \delta$. The points of A' can be arbitrary (the *general* case), or a subset of the points in A appearing in the same order as in A, i.e., $A' = (a_{i_1}, \dots, a_{i_k})$ and $i_1 \le \cdots \le i_k$ (the *restricted* case).

The different versions of the chain pair simplification (CPS) problem are formally defined as follows.

Problem 8.1 ((General) Chain Pair Simplification).
Instance: Given a pair of polygonal chains A and B of lengths n and m, respectively, an integer k, and three real numbers $\delta_1, \delta_2, \delta_3 > 0$.
Problem: Does there exist a pair of chains A',B', each of at most k vertices, such that A' is a δ_1-simplification of A w.r.t. d_1 ($d_1(A, A') \leq \delta_1$), B' is a δ_2-simplification of B w.r.t. d_2 ($d_2(B, B') \leq \delta_2$), and $d_{dF}(A', B') \leq \delta_3$?

When the vertices of the simplifications are from A and B (restricted simplifications), the problem is called CPS, and when the vertices of the simplifications are not necessarily from A and B (arbitrary simplifications), we call the problem GCPS. For each problem, we distinguish between two versions:

1. When $d_1 = d_2 = d_H$, the problems are called CPS-2H and GCPS-2H, respectively.

2. When $d_1 = d_2 = d_{dF}$, the problems are called CPS-3F and GCPS-3F, respectively.

Remark 8.2. We sometimes say that a set D of disks of radius δ covers a chain C. By this we mean that there exists a partition of C into consecutive subchains $C_1, \dots, C_t = C$, such that for each $1 \leq i \leq t$ there exists a disk in D that contains all the points of C_i.

8.3 Weighted chain pair simplification

We first introduce and consider a more general version of CPS-3F, namely, Weighted CPS-3F. In the weighted version of the chain pair simplification problem, the vertices of the chains A and B are assigned arbitrary weights, and, instead of limiting the length of the simplifications, one limits their weights. That is, the total weight of each simplification must not exceed a given value. The problem is formally defined as follows.

Problem 8.3 (Weighted Chain Pair Simplification).
Instance: Given a pair of 3D chains A and B, with lengths m and n, respectively, an integer k, three real numbers $\delta_1, \delta_2, \delta_3 > 0$, and a weight function $C : \{a_1, \dots, a_m, b_1, \dots, b_n\} \to \mathbb{R}^+$.
Problem: Does there exist a pair of chains A',B' with $C(A'), C(B') \leq k$, such that the vertices of A',B' are from A, B respectively, $d_1(A, A') \leq \delta_1$, $d_2(B, B') \leq \delta_2$, and $d_{dF}(A', B') \leq \delta_3$?

When $d_1 = d_2 = d_{dF}$, the problem is called WCPS-3F. When $d_1 = d_2 = d_H$, the problem is **NP**-complete, since the non-weighted version (i.e., CPS-2H) is already **NP**-complete [BJW+08].

We prove that WCPS-3F is weakly **NP**-complete via a reduction from the *set partition* problem: Given a set of positive integers $S = \{s_1, \ldots, s_n\}$, find two sets $P_1, P_2 \subset S$ such that $P_1 \cap P_2 = \emptyset$, $P_1 \cup P_2 = S$, and the sum of the numbers in P_1 equals the sum of the numbers in P_2. This is a weakly **NP**-complete special case of the classic subset-sum problem.

Our reduction builds two curves with weights reflecting the values in S. We think of the two curves as the subsets of the partition of S. Although our problem requires positive weights, we also allow zero weights in our reduction for clarity. Later, we show how to remove these weights by slightly modifying the construction.

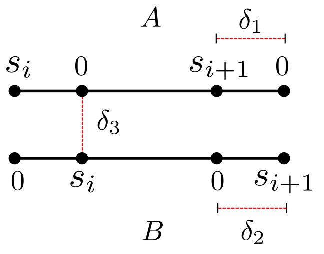


Figure 8.2: The reduction for the weighted chain pair simplification problem under the discrete Fréchet distance.

Theorem 8.4. *The weighted chain pair simplification problem under the discrete Fréchet distance is weakly **NP**-complete.*

Proof. Given the set of positive integers $S = \{s_1, \ldots, s_n\}$, we construct two curves A and B in the plane, each of length $2n$. We denote the weight of a vertex x_i by $w(x_i)$. A is constructed as follows. The i'th odd vertex of A has weight s_i, i.e. $w(a_{2i-1}) = s_i$, and coordinates $a_{2i-1} = (i, 1)$. The i'th even vertex of A has coordinates $a_{2i} = (i + 0.2, 1)$ and weight zero. Similarly, the i'th odd vertex of B has weight zero and coordinates $b_{2i-1} = (i, 0)$, and the i'th even vertex of B has coordinates $b_{2i} = (i + 0.2, 0)$ and weight s_i, i.e. $w(b_{2i}) = s_i$. Figure 8.2 depicts the vertices $a_{2i-1}, a_{2i}, a_{2(i+1)-1}, a_{2(i+1)}$ of A and $b_{2i-1}, b_{2i}, b_{2(i+1)-1}, b_{2(i+1)}$ of B. Finally, we set $\delta_1 = \delta_2 = 0.2$, $\delta_3 = 1$, and $k = \mathfrak{S}$, where $\mathfrak{S}$ denotes the sum of the elements of S (i.e., $\mathfrak{S} = \sum_{j=1}^{n} s_j$).

We claim that S can be partitioned into two subsets, each of sum $\mathfrak{S}/2$, if and only if A and B can be simplified with the constraints $\delta_1 = \delta_2 = 0.2$, $\delta_3 = 1$ and $k = \mathfrak{S}/2$, i.e., $C(A'), C(B') \leq \mathfrak{S}/2$.

First, assume that S can be partitioned into sets S_A and S_B, such that $\sum_{s \in S_A} s = \sum_{s \in S_B} s = \mathfrak{S}/2$. We construct simplifications of A and of B as follows.

$$A' = \{a_{2i-1} \mid s_i \in S_A\} \cup \{a_{2i} \mid s_i \notin S_A\} \text{ and } B' = \{b_{2i} \mid s_i \in S_B\} \cup \{b_{2i-1} \mid s_i \notin S_B\}.$$

It is easy to see that $C(A'), C(B') \leq \mathfrak{S}/2$. Also, since $\{S_A, S_B\}$ is a partition of S, exactly one of the following holds, for any $1 \leq i \leq n$:

1. $a_{2i-1} \in A', b_{2i-1} \in B'$ and $a_{2i} \notin A', b_{2i} \notin B'$.
2. $a_{2i-1} \notin A', b_{2i-1} \notin B'$ and $a_{2i} \in A', b_{2i} \in B'$.

This implies that $d_{dF}(A, A') \leq 0.2 = \delta_1$, $d_{dF}(B, B') \leq 0.2 = \delta_2$ and $d_{dF}(A', B') \leq 1 = \delta_3$.

Now, assume there exist simplifications A', B' of A, B, such that $d_{dF}(A, A') \leq \delta_1 = 0.2$, $d_{dF}(B, B') \leq \delta_2 = 0.2$, $d_{dF}(A', B') \leq \delta_3 = 1$, and $C(A'), C(B') \leq k = \mathfrak{S}/2$. Since $\delta_1 = \delta_2 = 0.2$, for any $1 \leq i \leq n$, the simplification A' must contain one of a_{2i-1}, a_{2i}, and the simplification B' must contain one of b_{2i-1}, b_{2i}. Since $\delta_3 = 1$, for any i, at least one of the following two conditions holds: $a_{2i-1} \in A'$ and $b_{2i-1} \in B'$ or $a_{2i} \in A'$ and $b_{2i} \in B'$. Therefore, for any i, either $a_{2i-1} \in A$ or $b_{2i} \in B$, implying that s_i participates in either $C(A')$ or $C(B')$. However, since $C(A'), C(B') \leq \mathfrak{S}/2$, s_i cannot participate in both $C(A')$ and $C(B')$. It follows that $C(A') = C(B') = \mathfrak{S}/2$, and we get a partition of S into two sets, each of sum $\mathfrak{S}/2$.

Finally, we note that WCPS-3F is in **NP**. For an instance I with chains A, B, given simplifications A', B', we can verify in polynomial time that $d_{dF}(A, A') \leq \delta_1$, $d_{dF}(B, B') \leq \delta_2$, $d_{dF}(A', B') \leq \delta_3$, and $C(A'), C(B') \leq k$. $\square$

Although our construction of A' and B' uses zero weights, a simple modification enables us to prove that the problem is weakly **NP**-complete also when only positive integral weights are allowed. Increase all the weights by 1, that is, $w(a_{2i-1}) = w(b_{2i}) = s_i + 1$ and $w(a_{2i}) = w(b_{2i-1}) = 1$, for $1 \leq i \leq n$, and set $k = \mathfrak{S}/2 + n$. It is easy to verify that our reduction still works. Finally, notice that we could overlay the two curves choosing $\delta_3 = 0$ and prove that the problem is still weakly **NP**-complete in one dimension.

8.4 CPS under DFD

We now turn our attention to CPS-3F, which is the special case of WCPS-3F where each vertex has weight one.

8.4.1 CPS-3F is in P

We present an algorithm for the minimization version of CPS-3F. That is, we compute the minimum integer k^*, such that there exists a "walk", as above, in which each of the dogs makes at most k^* hops. The answer to the decision problem is "yes" if and only if $k^* < k$.

Returning to the analogy of the man and the dog, we can extend it as follows. Consider a man and his dog connected by a leash of length δ_1, and a woman and her dog connected by a leash of length δ_2. The two dogs are also connected to each other by a leash of length δ_3. The man and his dog are walking on the points of a chain A and the woman and her dog are walking on the points of a chain B. The dogs may skip points. The problem is to determine whether there exists a "walk" of the man and his dog on A and the woman and her dog on B, such that each of the dogs steps on at most k points.

Overview of the algorithm. We say that (a_i, a_p, b_j, b_q) is a *possible* configuration of the man, woman and the two dogs on the paths A and B, if $d(a_i, a_p) \leq \delta_1$, $d(b_j, b_q) \leq \delta_2$ and $d(a_p, b_q) \leq \delta_3$. Notice that there are at most m^2n^2 such configurations. Now, let G be the DAG whose vertices are the possible configurations, such that there exists a (directed) edge from vertex $u = (a_i, a_p, b_j, b_q)$ to vertex $v = (a_{i'}, a_{p'}, b_{j'}, b_{q'})$ if and only if our gang can move from configuration u to configuration v. That is, if and only if $i \leq i' \leq i+1$, $p \leq p'$, $j \leq j' \leq j+1$, and $q \leq q'$. Notice that there are no cycles in G because backtracking is forbidden. For simplicity, we assume that the first and last points of A' (resp., of B') are a_1 and a_m (resp., b_1 and b_n), so the initial and final configurations are $s = (a_1, a_1, b_1, b_1)$ and $t = (a_m, a_m, b_n, b_n)$, respectively. (It is easy, however, to adapt the algorithm below to the case where the initial and final points of A' and B' are not specified, see remark below.) Our goal is to find a path from s to t in G. However, we want each of our dogs to step on at most k points, so, instead of searching for any path from s to t, we search for a path that minimizes the value $max\{|A'|, |B'|\}$, and then check if this value is at most k.

For each edge $e = (u, v)$, we assign two weights, $w_A(e), w_B(e) \in \{0, 1\}$, in order to compute the number of hops in A' and in B', respectively. $w_A(u, v) = 1$ if and only if the first dog jumps to a new point between configurations u and v (i.e., $p < p'$), and, similarly, $w_B(u, v) = 1$ if and only if the second dog jumps to a new point between u and v (i.e., $q < q'$). Thus, our goal is to find a path P from s to t in G, such that $max\{\sum_{e \in P} w_A(e), \sum_{e \in P} w_B(e)\}$ is minimized.

Assume w.l.o.g. that $m \leq n$. Since $|A'| \leq m$ and $|B'| \leq n$, we maintain, for each vertex v of G, an array $X(v)$ of size m, where $X(v)[r]$ is the minimum number z such that v can be reached from s with (at most) r hops of the first dog and z hops of the second dog. We can construct these arrays by processing the vertices of G in topological order (i.e., a vertex is processed only after all its predecessors have been processed). This yields an algorithm of running time $O(m^3n^3 \min\{m, n\})$, as described in Algorithm 8.1.

Running time. The number of vertices in G is $|V| = O(m^2n^2)$. By the construction of the graph, for any vertex (a_i, a_p, b_j, b_q) the maximum number of outgoing edges is $O(mn)$. So we have $|E| = O(|V|mn) = O(m^3n^3)$. Thus, constructing the graph G in Step 1 takes $O(n^3m^3)$ time. Step 2 takes $O(|E|)$ time, while Step 3 takes $O(m)$

Algorithm 8.1 CPS-3F

```
1. Create a directed graph $G = (V, E)$ with two weight functions $w_A$, $w_B$, such that:
    • $V$ is the set of all configurations $(a_i, a_p, b_j, b_q)$ with $d(a_i, a_p) \leq \delta_1$, $d(b_j, b_q) \leq \delta_2$, and $d(a_p, b_q) \leq \delta_3$.
    • $E = \{((a_i, a_p, b_j, b_q), (a_{i'}, a_{p'}, b_{j'}, b_{q'})) \mid i \leq i' \leq i+1,\ p \leq p',\ j \leq j' \leq j+1,\ q \leq q'\}$.
    • For each $((a_i, a_p, b_j, b_q), (a_{i'}, a_{p'}, b_{j'}, b_{q'})) \in E$, set
        – $w_A((a_i, a_p, b_j, b_q), (a_{i'}, a_{p'}, b_{j'}, b_{q'})) = \begin{cases} 1, & p < p' \\ 0, & \text{otherwise} \end{cases}$
        – $w_B((a_i, a_p, b_j, b_q), (a_{i'}, a_{p'}, b_{j'}, b_{q'})) = \begin{cases} 1, & q < q' \\ 0, & \text{otherwise} \end{cases}$
2. Sort $V$ topologically.
3. Initialize the array $X(s)$ (i.e., set $X(s)[r] = 0$, for $r = 0, \ldots, m-1$).
4. For each $v \in V \setminus \{s\}$ (advancing from left to right in the sorted sequence) do:
    (a) Initialize the array $X(v)$ (i.e., set $X(v)[r] = \infty$, for $r = 0, \ldots, m-1$).
    (b) For each $r$ between 0 and $m-1$, compute $X(v)[r]$:
        $X(v)[r] = \min_{(u,v) \in E} \begin{cases} X(u)[r] + w_B(u,v), & w_A(u,v) = 0 \\ X(u)[r-1] + w_B(u,v), & w_A(u,v) = 1 \end{cases}$
5. Return $k^* = \min_r \max\{r,\ X(t)[r]\}$.
```

time. In Step 4, for each vertex v and for each index r, we consider all configurations that can directly precede v. So each edge of G participates in exactly m minimum computations, implying that Step 4 takes $O(|E|m)$ time. Step 5 takes $O(m)$ time. Thus, the total running time of the algorithm is $O(m^4n^3)$.

Theorem 8.5. *The chain pair simplification problem under the discrete Fréchet distance (CPS-3F) is polynomial, i.e., CPS-3F* $\in$ ***P***.

Remark 8.6. As mentioned, we have assumed that the first and last points of A' (resp., B') are a_1 and a_m (resp., b_1 and b_n), so we have a single initial configuration (i.e., $s = (a_1, a_1, b_1, b_1)$) and a single final configuration (i.e., $t = (a_m, a_m, b_n, b_n)$). However, it is easy to adapt our algorithm to the case where the first and last points of the chains A' and B' are not specified. In this case, any possible configuration of the form (a_1, a_p, b_1, b_q) is considered a potential initial configuration, and any possible configuration of the form (a_m, a_p, b_n, b_q) is considered a potential final configuration, where $1 \leq p \leq m$ and $1 \leq q \leq n$. Let S and T be the sets of potential initial and final configurations, respectively. (Then, $|S| = O(mn)$ and $|T| = O(mn)$.) We thus remove from G all edges entering a potential initial configuration, so that each such configuration becomes a "root" in the (topologically) sorted sequence. Now, in Step 3 we initialize the arrays of each $s \in S$ in total time $O(m^2n)$, and in Step 4 we only process the vertices that are not in S. The value $X(v)[r]$ for such a vertex v is now the minimum number z such that v can be reached from s with r hops of the first dog and z hops of the second dog, over *all* potential initial configurations $s \in S$. In the final step of the algorithm, we calculate the value k^* in $O(m)$ time, for each potential final configuration $t \in T$. The smallest value obtained is then the desired value. Since the number of potential final configurations is only $O(mn)$, the total running time of the final step of the algorithm is only $O(m^2n)$, and the running time of the entire algorithm remains $O(m^4n^3)$.

The weighted version

Weighted CPS-3F, which was shown to be weakly **NP**-complete in the previous section, can be solved in a similar manner, albeit with running time that depends on the number of different point weights in chain A (alternatively, B). We now explain how to adapt our algorithm to the weighted case. We first redefine the weight functions w_A and w_B (where $C(x)$ is the weight of point x):

- $w_A((a_i, a_p, b_j, b_q), (a_{i'}, a_{p'}, b_{j'}, b_{q'})) = \begin{cases} C(a_{p'}), & p < p' \\ 0, & otherwise \end{cases}$

- $w_B = ((a_i, a_p, b_j, b_q), (a_{i'}, a_{p'}, b_{j'}, b_{q'})) \begin{cases} C(b_{q'}), & q < q' \\ 0, & otherwise \end{cases}$

Next, we increase the size of the arrays $X(v)$ from m to the number of different weights that can be obtained by a subset of A (alternatively, B). (For example, if $|A| = 3$ and $C(a_1) = 2$, $C(a_2) = 2$, and $C(a_3) = 4$, then the weights that can be obtained are $2, 4, 2+4 = 6, 2+2+4 = 8$, so the size of the arrays would be 4.) Let $c[r]$ be the r'th largest such weight. Then $X(v)[r]$ is the minimum number z, such that v can be reached from s with hops of total weight (at most) $c[r]$ of the first dog and hops of total weight z of the second dog. $X(v)[r]$ is calculated as follows:

$$X(v)[r] = \min_{(u,v) \in E} \begin{cases} X(u)[r] + w_B(u,v), & w_A(u,v) = 0 \\ X(u)[r'] + w_B(u,v), & w_A(u,v) > 0 \end{cases},$$

where $c[r'] = c[r] - w_A(u, v)$. If the number of different weights that can be obtained by a subset of A (alternatively, B) is $f(A)$ (resp., $f(B)$), then the running time is $O(m^3n^3f(A))$ (resp., $O(m^3n^3f(B))$), since the only change that affects the running time is the size of the arrays $X(v)$. We thus have

Theorem 8.7. *The weighted chain pair simplification problem under the discrete Fréchet distance (Weighted CPS-3F) (and its corresponding minimization problem) can be solved in $O(m^3n^3 \min\{f(A), f(B)\})$ time, where $f(A)$ (resp., $f(B)$) is the number of different weights that can be obtained by a subset of A (resp., B). In* *particular, if only one of the chains, say B, has points with non-unit weight, then $f(A) = O(m)$, and the running time is polynomial; more precisely, it is $O(m^4 n^3)$.*

Remark 8.8. We presented an algorithm that minimizes $\max\{|A'|, |B'|\}$ given the error parameters $\delta_1, \delta_2, \delta_3$. Another optimization version of CPS-3F is to minimize, e.g., δ_3 (while obeying the requirements specified by δ_1, δ_2 and k). It is easy to see that Algorithm 8.1 can be adapted to solve this version within roughly the same time bound.

8.4.2 An efficient implementation for CPS-3F

The time and space complexity of Algorithm 8.1 (which is $O(m^3 n^3 \min\{m, n\})$ and $O(m^3 n^3)$, respectively) makes it impractical for our motivating biological application (as m, n could be 500$\sim$600). In this section, we show how to reduce the time and space bounds by a factor of mn, using dynamic programming.

We generate all configurations of the form (a_i, a_p, b_j, b_q), where the outermost for-loop is governed by i, the next level loop by j, then p, and finally q. When a new configuration $v = (a_i, a_p, b_j, b_q)$ is generated, we first check whether it is *possible*. If it is not possible, we set $X(v)[r] = \infty$, for $1 \le r \le m$, and if it is, we compute $X(v)[r]$, for $1 \le r \le m$.

We also maintain for each pair of indices i and j, three tables $C_{i,j}$, $R_{i,j}$, $T_{i,j}$ that assist us in the computation of the values $X(v)[r]$:

$$\begin{aligned}
C_{i,j}[p,q,r] &= \min_{1\le p'\le p} X(a_i, a_{p'}, b_j, b_q)[r] \\
R_{i,j}[p,q,r] &= \min_{1\le q'\le q} X(a_i, a_p, b_j, b_{q'})[r] \\
T_{i,j}[p,q,r] &= \min_{\substack{1\le p'\le p\\ 1\le q'\le q}} X(a_i, a_{p'}, b_j, b_{q'})[r]
\end{aligned}$$

Notice that the value of cell $[p, q, r]$ is determined by the value of one or two previously-determined cells and $X(a_i, a_p, b_j, b_q)[r]$ as follows:

$$\begin{aligned}
C_{i,j}[p,q,r] &= \min\{C_{i,j}[p-1,q,r], X(a_i, a_p, b_j, b_q)[r]\} \\
R_{i,j}[p,q,r] &= \min\{R_{i,j}[p,q-1,r], X(a_i, a_p, b_j, b_q)[r]\} \\
T_{i,j}[p,q,r] &= \min\{T_{i,j}[p-1,q,r], T_{i,j}[p,q-1,r], X(a_i, a_p, b_j, b_q)[r]\}
\end{aligned}$$

Observe that in any configuration that can immediately precede the current configuration (a_i, a_p, b_j, b_q), the man is either at a_{i-1} or at a_i and the woman is either at b_{j-1} or at b_j (and the dogs are at $a_{p'}$, $p' \le p$, and $b_{q'}, q' \le q$, respectively). The "saving" is achieved, since now we only need to access a constant number of table entries in order to compute the value $X(a_i, a_p, b_j, b_q)[r]$.

One can illustrate the algorithm using the matrix in Figure 8.3. There are

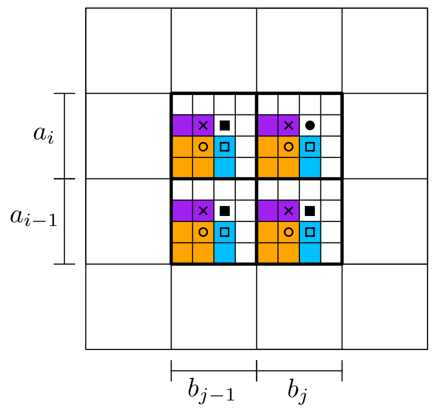


Figure 8.3: Illustration of Algorithm 8.2.

mn large cells, each of them containing a matrix of size mn. The large cells correspond to the positions of the man and the woman. The inner matrices correspond to the positions of the two dogs (for given positions of the man and woman). Consider an optimal "walk" of the gang that ends at cell (a_i, a_p, b_j, b_q) (marked by a full circle), such that the first dog has visited r points. The previous cell in this "walk" must be in one of the 4 large cells (a_i, b_j),(a_{i-1}, b_j),(a_i, b_{j-1}),(a_{i-1}, b_{j-1}). Assume, for example, that it is in (a_{i-1}, b_j). Then, if it is in the blue area, then $X(a_i, a_p, b_j, b_q)[r] = C_{i-1,j}[p-1, q, r-1]$ (marked by an empty square), since only the position of the first dog has changed when the gang moved to (a_i, a_p, b_j, b_q). If it is in the purple area, then $X(a_i, a_p, b_j, b_q)[r] = R_{i-1,j}[p, q-1, r] + 1$ (marked by a x), since only the position of the second dog has changed. If it is in the orange area, then $X(a_i, a_p, b_j, b_q)[r] = T_{i-1,j}[p-1, q-1, r-1] + 1$ (marked by an empty circle), since the positions of both dogs have changed. Finally, if it is the cell marked by the full square, then simply $X(a_i, a_p, b_j, b_q)[r] = X(a_{i-1}, a_p, b_j, b_q)[r]$, since both dogs have not moved. The other three cases, in which the previous cell is in one of the 3 large cells (a_i, b_j),(a_i, b_{j-1}),(a_{i-1}, b_{j-1}), are handled similarly.

We are ready to present the dynamic programming algorithm. The initial configurations correspond to cells in the large cell (a_1, b_1). For each initial configuration (a_1, a_p, b_1, b_q), we set $X(a_1, a_p, b_1, b_q)[1] = 1$.

Theorem 8.9. *The minimization version of the chain pair simplification problem under the discrete Fréchet distance (CPS-3F) can be solved in* $O(m^2n^2 \min\{m, n\})$ *time.*

8.4.3 1-sided CPS

Sometimes, one of the two input chains, say B, is much shorter than the other, possibly because it has already been simplified. In these cases, we only want to simplify A, in a way that maintains the resemblance between the two input chains. We thus define the 1-sided chain pair simplification problem.

Algorithm 8.2 CPS-3F using dynamic programming

```
for $i = 1$ to $m$
    for $j = 1$ to $n$
        for $p = 1$ to $m$
            for $q = 1$ to $n$
                for $r = 1$ to $m$
                    $X_{(-1,0)} = \min \begin{cases} C_{i-1,j}[p-1,q,r-1] \\ R_{i-1,j}[p,q-1,r]+1 \\ T_{i-1,j}[p-1,q-1,r-1]+1 \\ X(a_{i-1},a_p,b_j,b_q)[r] \end{cases}$
                    $X_{(0,-1)} = \min \begin{cases} C_{i,j-1}[p-1,q,r-1] \\ R_{i,j-1}[p,q-1,r]+1 \\ T_{i,j-1}[p-1,q-1,r-1]+1 \\ X(a_i,a_p,b_{j-1},b_q)[r] \end{cases}$
                    $X_{(-1,-1)} = \min \begin{cases} C_{i-1,j-1}[p-1,q,r-1] \\ R_{i-1,j-1}[p,q-1,r]+1 \\ T_{i-1,j-1}[p-1,q-1,r-1]+1 \\ X(a_{i-1},a_p,b_{j-1},b_q)[r] \end{cases}$
                    $X_{(0,0)} = \min \begin{cases} C_{i,j}[p-1,q,r-1] \\ R_{i,j}[p,q-1,r]+1 \\ T_{i,j}[p-1,q-1,r-1]+1 \end{cases}$
                    $X(a_i,a_p,b_j,b_q)[r] = \min\{X_{(-1,0)}, X_{(0,-1)}, X_{(-1,-1)}, X_{(0,0)}\}$
                    $C_{i,j}[p,q,r] = \min\{C_{i,j}[p-1,q,r], X(a_i,a_p,b_j,b_q)[r]\}$
                    $R_{i,j}[p,q,r] = \min\{R_{i,j}[p,q-1,r], X(a_i,a_p,b_j,b_q)[r]\}$
                    $T_{i,j}[p,q,r] = \min\{T_{i,j}[p-1,q,r], T_{i,j}[p,q-1,r], X(a_i,a_p,b_j,b_q)[r]\}$
return $\min_{r,p,q} \max\{r, X(a_m,a_p,b_n,b_q)[r]\}$
```

Problem 8.10 (1-Sided Chain Pair Simplification).

Instance: Given a pair of polygonal chains A and B of lengths m and n, respectively, an integer k, and two real numbers $\delta_1, \delta_3 > 0$.

Problem: Does there exist a chain A' of at most k vertices, such that the vertices of A' are from A, $d_{dF}(A,A') \leq \delta_1$, and $d_{dF}(A',B) \leq \delta_3$?

The optimization version of this problem can be solved using similar ideas to those used in the solution of the 2-sided problem. Here a *possible* configuration is a 3-tuple (a_i, a_p, b_j), where $d(a_i,a_p) \leq \delta_1$ and $d(a_p,b_j) \leq \delta_3$. We construct a graph and find a shortest path from one of the starting configurations to one of the final configurations; see Algorithm 8.3. Arguing as for Algorithm 8.1, we get that $|V| = O(m^2n)$ and $|E| = O(|V|m) = O(m^3n)$. Moreover, it is easy to see that the running time of Algorithm 8.3 is $O(m^3n)$, since it does not maintain an array for each vertex.

To reduce the running time we use dynamic programming as in Section 8.4.2. We generate all configurations of the form (a_i, a_p, b_j). When a new configuration $v = (a_i, a_p, b_j)$ is generated, we first check whether it is *possible*. If it is not possible,

Algorithm 8.3 1-sided CPS-3F

```
1. Create a directed graph $G = (V, E)$ with a weight function $w$, such that:
    • $V = \{(a_i, a_p, b_j) \mid d(a_i, a_p) \le \delta_1$ and $d(a_p, b_j) \le \delta_3\}$.
    • $E = \{((a_i, a_p, b_j), (a_{i'}, a_{p'}, b_{j'})) \mid i \le i' \le i+1,\ p \le p',\ j \le j' \le j+1\}$.
    • For each $((a_i, a_p, b_j), (a_{i'}, a_{p'}, b_{j'})) \in E$, set
          $w((a_i, a_p, b_j), (a_{i'}, a_{p'}, b_{j'})) = \begin{cases} 1, & p < p' \\ 0, & otherwise \end{cases}$
    • Let $S$ be the set of starting configurations and let $T$ be the set of final configurations.
2. Sort $V$ topologically.
3. Set $X(s) = 0$, for each $s \in S$.
4. For each $v \in V \setminus S$ (advancing from left to right in the sorted sequence) do:
          $X(v) = \min_{(u,v) \in E} \{X(u) + w(u, v)\}$.
5. Return $k^* = \min_{t \in T} X(t)$.
```

we set $X(v) = \infty$, and if it is, we compute $X(v)$. We also maintain for each pair of indices i and j, a table $A_{i,j}$ that assists us in the computation of the value $X(v)$: $A_{i,j}[p] = \min_{1 \le p' \le p} X(a_i, a_{p'}, b_j)$. Notice that $A_{i,j}[p]$ is the minimum of $A_{i,j}[p-1]$ and $X(a_i, a_p, b_j)$.

We observe once again that in any configuration that can immediately precede the current configuration (a_i, a_p, b_j), the man is either at a_{i-1} or at a_i and the woman is either at b_{j-1} or at b_j (and the dog is at $a_{p'}$, $p' \le p$). The "saving" is achieved, since now we only need to access a constant number of table entries in order to compute the value $X(a_i, a_p, b_j)$. We obtain Algorithm 8.4, a dynamic programming algorithm whose running time is $O(m^2 n)$.

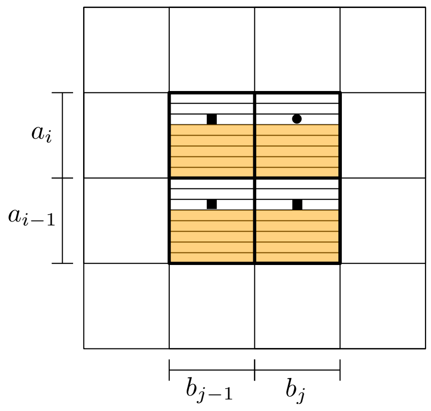


Figure 8.4: The vertex matrix.

We can illustrate the algorithm using the matrix in Figure 8.4. There are mn

Algorithm 8.4 1-sided CPS-3F using dynamic programming

```
for $i = 1$ to $m$
    for $j = 1$ to $n$
        for $p = 1$ to $m$
            $X_{(-1,0)} = \min \begin{cases} A_{i-1,j}[p-1]+1 \\ X(a_{i-1}, a_p, b_j) \end{cases}$
            $X_{(0,-1)} = \min \begin{cases} A_{i,j-1}[p-1]+1 \\ X(a_i, a_p, b_{j-1}) \end{cases}$
            $X_{(-1,-1)} = \min \begin{cases} A_{i-1,j-1}[p-1]+1 \\ X(a_{i-1}, a_p, b_{j-1}) \end{cases}$
            $X_{(0,0)} = A_{i,j}[p-1]+1$
            $X(a_i, a_p, b_j) = \min\{X_{(-1,0)}, X_{(0,-1)}, X_{(-1,-1)}, X_{(0,0)}\}$
            $A_{i,j}[p] = \min\{A_{i,j}[p-1], X(a_i, a_p, b_j)\}$
return $\min_p\{X(a_m, a_p, b_n)\}$
```

large cells, each of them contains an array of size m. The large cells correspond to the positions of the man and the woman. The arrays correspond to the position of the dog. Consider an optimal "walk" of the gang that ends in the position (a_i, a_p, b_j, b_q) (the black circle). The previous position of the gang correspond to a vertex that can be located only in one of the 4 cells (a_i, b_j),(a_{i-1}, b_j),(a_i, b_{j-1}),(a_{i-1}, b_{j-1}). Moreover, it can only be one of the vertices marked in orange or the red circles. If, for example, it is located in the left top orange area, then $C(a_i, a_p, b_j) = A_{i-1,j}[p-1]$, because $A_{i-1,j}[p-1]$ is the minimum number of steps of the dog when position of the man and the woman is (a_{i-1}, b_j). If it is the left top black circle, then it is simply $C(a_{i-1}, a_p, b_j)$ since the dog stayed at the same position. Symmetrically, this is true for the other 3 large cells.

Theorem 8.11. *The 1-sided chain pair simplification problem under the discrete Fréchet distance can be solved in $O(m^2 n)$ time.*

8.5 GCPS under DFD

In this section we consider the general case of the problem, where the points of a simplification are not necessarily from the original chain.

8.5.1 GCPS-3F is in P

In order to solve GCPS-3F, we consider the optimization problem: Given a pair of polygonal chains A and B of lengths n and m, respectively, and three real numbers $\delta_1, \delta_2, \delta_3 > 0$, what is the smallest number k such that there exist a pair of chains A',B', each of at most k (arbitrary) vertices, for which $d_{dF}(A, A') \le \delta_1$, $d_{dF}(B, B') \le \delta_2$, and $d_{dF}(A', B') \le \delta_3$?

We begin by describing some properties that are required from an optimal solution to the problem. Then, based on these properties, we are able to refine our search for the optimal solution.

What does an optimal solution look like?

Let (A', B') be an optimal solution, that is, let A' and B' be two arbitrary simplifications of A and B respectively, such that $d_{dF}(A, A') \leq \delta_1$, $d_{dF}(B, B') \leq \delta_2$, $d_{dF}(A', B') \leq \delta_3$, and $\max\{|A'|, |B'|\}$ is minimum. Moreover, we assume that the shorter of the chains A', B' is as short as possible.

Let $W_{A'B'} = \{(A'_i, B'_i)\}_{i=1}^{t}$ be a Fréchet walk along A' and B'. Notice that, by definition, for any i it holds that $|A'_i| = 1$ or $|B'_i| = 1$.

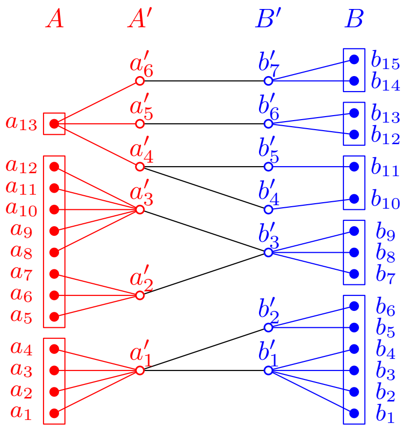


Figure 8.5: How does an optimal solution look like? a composition of pair-components: $W_{A'B'} = \{(\{a'_1\}, \{b'_1, b'_2\}), (\{a'_2, a'_3\}, \{b'_3\}), (\{a'_4\}, \{b'_4, b'_5\}), (\{a'_5\}, \{b'_6\}), (\{a'_6\}, \{b'_7\})\}$ $(A_1 = A[1,4], B_1 = [1,6])$, $(A_2 = A[5,12], B_2 = B[7,9])$, $(A_3 = A[13], B_3 = B[10,11])$, $(A_4 = A[13], B_4 = B[12,13])$, $(A_5 = A[13], B_5 = B[14,15])$.

Let $W_{AA'}$ be a Fréchet walk along A and A'. Notice that unlike in regular (one-sided) simplifications, the pairs in $W_{AA'}$ may match several points from A' to a single point from A, because A' does not depend only on A but also on B' and B. Similarly, let $W_{BB'}$ be a Fréchet walk along B and B' (see Figure 8.5).

With each pair $(A'_i, B'_i) \in W_{A'B'}$, we associate a pair of subchains A_i of A and B_i of B, which we call a *pair component*. Assume $A'_i = A'[p, q]$, then A_i is defined as follows:

1. If $p \neq q$, then each $a'_k \in A'[p, q]$ appears as a singleton in $W_{AA'}$ (since otherwise A' can be shortened). Let A^k be the subchain of A that is matched to a'_k, i.e., $(A^k, a'_k) \in W_{AA'}$, for $k = p, \ldots, q$. Then, we set $A_i = A^p A^{p+1} \cdots A^q$.

2. If $p = q$ and a'_p appears as a singleton in $W_{AA'}$, then we set $A_i = A^p$.

3. If $p = q$ and a'_p belongs to some subchain of A' of length at least two that is matched (in $W_{AA'}$) to a single element $a_l \in A$, we set $A_i = a_l$.

The subchains $B_1, \ldots, B_t$ are defined analogously.

We need two observations. The first one is that A_i and B_i are indeed subchains (consecutive sets of points). This is simply because the matchings of the points from A'_i and B'_i in $W_{AA'}$ and $W_{BB'}$, respectively, are sub-chains, and by definition $A_i = A^p A^{p+1} \cdots A^q$ is also a consecutive set of points. The second observation is that the subchains $A_1, \ldots, A_t$ (resp. $B_1, \ldots, B_t$) are almost-disjoint, in the sense that there can be only one point a_x that belongs to both A_i and A_{i+1}, and in that case $A_i = A_{i+1} = (a_x)$. This is because if there were more than one point in common, or, if one of A_i, A_{i+1} contained more points, then the sets in $W_{AA'}$ (resp. $W_{BB'}$) were not disjoint.

So what does an optimal solution look like? It is composed of such almost-disjoint *pair-components*. A pair-component is a pair of sub-chains, (A_i, B_i), $A_i \subseteq A$, $B_i \subseteq B$, such that the points of A_i (resp. B_i) can be covered by one disk c of radius δ_1 (resp. δ_2), the points of B_i (resp. A_i) can be covered by a set C of disks of radius δ_2 (resp. δ_1), and for any $c' \in C$, the distance between the center of c and c' is at most δ_3.

The idea of the algorithm is to compute all the possible components (and that there are not too many of them), and then use dynamic programming to compute the optimal solution that is composed of pair-components.

The algorithm

For any two sub-chains $A[i,i']$ and $B[j,j']$ there are two possible types of pair-components. In the first type, there is only one disk that covers $A[i,i']$, and in the second type, there is only one disk that covers $B[j,j']$.

We denote by $PC_1[i,i',j,j']$ the size of the minimum-cardinality set C of disks of radius δ_2 needed in order to cover $B[j,j']$, such that there exists a disk c of radius δ_1 that covers $A[i,i']$, and for any $c' \in C$, the distance between the centers of c and c' is at most δ_3. Symmetrically, we define $PC_2[i,i',j,j']$. For any 4-tuple of indices (i,i',j,j') we need to compute $PC_1[i,i',j,j']$ and $PC_2[i,i',j,j']$.

Now, in order to compute an optimal solution, we need to combine pair-components in a way that will result in a simplification of minimum size. We use dynamic programming.

Let $OPT[i,j][r]$ be the minimum number of points in a simplification of $B[1,j]$ in an optimal solution for $A[1,i], B[1,j]$ in which the number of points in the simplification of $A[1,i]$ is at most r. Then we have the following dynamic programming algorithm: $OPT[1,1][r] = 1$ if and only if $||a_1 - b_1|| \le \delta_1 + \delta_2 + \delta_3$, and

$$OPT[1,j][r] = \min_{q \le j}\{OPT[1,q-1][r-1] + PC_1[1,1,q,j]\},$$

$$OPT[i,1][r] = \min_{p \le i}\{OPT[p-1,1][r - PC_2[p,i,1,1]] + 1\},$$

$$\begin{aligned} OPT[i,j][r] = \min_{p\le i, q\le j}\{ & OPT[p-1,q-1][r-1] + PC_1[p,i,q,j], \\ & OPT[i,q-1][r-1] + PC_1[i,i,q,j], \\ & OPT[p-1,q-1][r-PC_2[p,i,q,j]] + 1, \\ & OPT[p-1,j][r-PC_2[p,i,j,j]] + 1\}. \end{aligned}$$

Theorem 8.12. *For any i,j and r, $OPT[i,j][r]$ is the minimum number of points in a simplification of $B[1,j]$ in an optimal solution for $A[1,i], B[1,j]$ in which the number of points in the simplification of $A[1,i]$ is at most r.*

Proof. The proof is by induction on i, j, and r. For $i = 1$ and $j = 1$ the theorem holds by definition. Let A' and B' be an optimal solution for $A[1,i], B[1,j]$, s.t. $|A'| \le r$. Let $[p,i,q,j]$ be the last pair-component in this solution. If $[p,i,q,j]$ is of type 1, i.e. there is one disk that covers $A[p,i]$ and $PC_1[p,i,q,j]$ disks that cover $B[q,j]$, then there are two possibilities: if $p = i$ and the pair-component that came before $[p,i,q,j]$ is $[i,i,q',q-1]$ for some $q' \le q-1$, then

$$OPT[i,j][r] = OPT[i,q-1][r-1] + PC_1[i,i,q,j],$$

else,

$$OPT[i,j][r] = OPT[p-1,q-1][r-1] + PC_1[p,i,q,j].$$

If $[p,i,q,j]$ is of type 2, i.e. there is one point that covers $B[q,j]$ and $PC_2[p,i,q,j]$ points that cover $A[p,i]$, then again we have two possibilities,

$$\begin{aligned} OPT[i,j][r] &= OPT[p-1,j][r-PC_2[p,i,j,j]] + 1, \text{ or} \\ OPT[i,j][r] &= OPT[p-1,q-1][r-PC_2[p,i,q,j]] + 1. \end{aligned}$$

□

Computing the components

Let $D(p,\delta)$ denote the disk centred at p with radius δ.

Recall that $PC_1[i,i',j,j']$ is the size of a minimum-cardinality set C of disks of radius δ_2 needed in order to cover $B[j,j']$, such that there exists a disk c of radius δ_1 that covers $A[i,i']$, and for any $c' \in C$, the distance between the centers of c and c' is at most δ_3.

We show how to find $PC_1[i,i',j,j']$ for all $1 \le i \le i' \le n$ and $1 \le j \le j' \le m$ ($PC_2[i,i',j,j']$ is symmetric). We begin with a few observations to give an intuition for the algorithm.

Consider $PC_1[i,i',j,j']$. First, notice that the center of c is in the region $X_{i,i'} = \bigcap_{i\le k\le i'} D(a_k,\delta_1)$, because the distance from c to any point in $A[i,i']$ is at most δ_1.

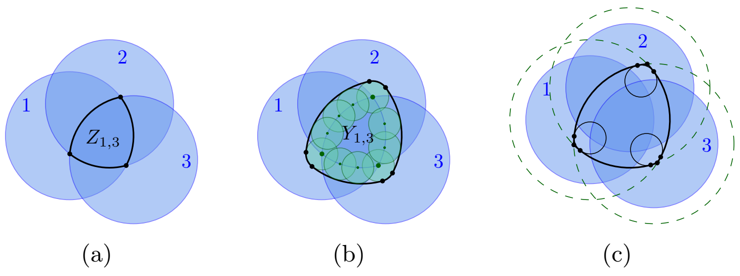


Figure 8.6: The blue filled disks represent $D(b_j, \delta_2)$ and the empty dashed green disks represent $D(b_j, \delta_2 + \delta_3)$. The small disks has radius δ_3.

Any $c' \in C$ is covering a consecutive subchain of $B[j, j']$. Thus, for any $j \leq t \leq t' \leq j'$, the center of a disk c' that covers the subsequence $B[t, t']$ (if exists) is in the region $Z_{t,t'} = \bigcap_{t \leq k \leq t'} D(b_k, \delta_2)$ (see Figure 8.6(a)). There are $O((j' - j)^2) = O(m^2)$ such regions.

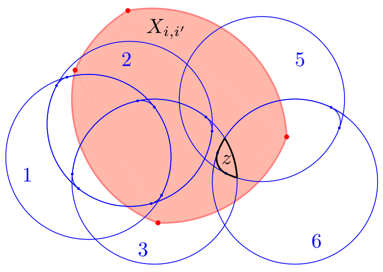


Figure 8.7: The arrangement obtained by the intersection of $X_{i,i'}$ and the arrangement of $\{Y_{t,t'} \mid j \leq t \leq t' \leq j'\}$.

Each such region is convex and composed of linear number of arcs. Any point placed inside $Z_{t,t'}$ can cover $B[t, t']$, and we need a point with distance at most δ_3 to the center of c. For each $Z_{t,t'}$, consider the Minkowski sum $Y_{t,t'} = Z_{t,t'} \oplus \delta_3$ (see Figure 8.6(b)).

Now, consider the arrangement obtained by the intersection of $X_{i,i'}$ and the arrangement of $\{Y_{t,t'} \mid j \leq t \leq t' \leq j'\}$ (see Figure 8.7). Each cell in this arrangement corresponds to a set of $Y_{t,t'}$'s, each has some point with distance at most δ_3 to the same points in $X_{i,i'}$. Each cell corresponds to a possible choice of the center of c, or, in other words, a possible pair-component of type 1.

We now describe an algorithm for computing $PC_1[i, i', j, j']$ for all $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$. The algorithm is quite complex and has several sub-procedures.

Let $X = \{X_{i,i'} = \bigcap_{i \leq k \leq i'} D(a_k, \delta_1) \mid 1 \leq i \leq i' \leq n\}$. The number of shapes in X is $O(n^2)$.

Let $Y = \{Y_{j,j'} \mid 1 \leq j \leq j' \leq m,\ Z_{j,j'} \neq \emptyset\}$. The number of shapes in Y is $O(m^2)$, each shape is of complexity $O(m)$.

Consider the arrangement $\mathcal{A}(Y)$ of the shapes in Y.

Lemma 8.13. *The number of cells in $\mathcal{A}(Y)$ is $O(m^4)$.*

Proof. Let P be the set of intersection points between the disks in $\{D(b_j, \delta_2) \mid 1 \leq j \leq m\}$. Consider the following set of disks: $D = \{D(b_i, \delta_2 + \delta_3) \mid 1 \leq i \leq m\} \cup \{D(p, \delta_3) \mid p \in P\}$. Notice that the arcs and vertices of $\mathcal{A}(Y)$ are a subset of the arcs and vertices of $\mathcal{A}(D)$ (see Figure 8.6(c)). Since the number of points in P is $O(m^2)$, we get that the number of disks in $\mathcal{A}(D)$ is $O(m^2)$, and thus the complexity of $\mathcal{A}(D)$ is $O(m^4)$. □

Notice that for any shape $Y_{j,j'} \in Y$ and a cell $z \in \mathcal{A}(Y)$ it holds that $Y_{j,j'} \cap z \neq \emptyset$ if and only if $z \subseteq Y_{j,j'}$. For each cell $z \in \mathcal{A}(Y)$, let Y_z be the set of $O(m^2)$ shapes from Y that contain z. The algorithm has two main steps:

1. For each cell $z \in \mathcal{A}(Y)$, and for any two indices $1 \leq j \leq j' \leq m$, compute $Size_B(z, j, j')$ – the minimum number of shapes from Y_z needed in order to cover the points of $B[j, j']$. Recall that a shape $Y_{t,t'} \in Y_z$ covers the subsequence $B[t, t']$, in other words, there exists a point q in $Y_{t,t'}$ s.t. $d(q, b_k) \leq \delta_2$ for any $t \leq k \leq t'$.

2. For each shape $X_{i,i'} \in X$, and for any two indices $1 \leq j \leq j' \leq m$, compute $Size_A(X_{i,i'}, j, j') = \min_{z \cap X_{i,i'} \neq \emptyset} Size_B(z, j, j')$.

Note that $Size_A(X_{i,i'}, j, j') = PC_1[i, i', j, j']$.

Step 1

First we have to find the set Y_z for each cell $z \in \mathcal{A}(Y)$. We start by computing Y: for any j, j' we check whether $\bigcap_{j \leq k \leq j'} D(b_k, \delta_2) \neq \emptyset$. If yes, we add $Y_{j,j'}$ to Y. This can be done in $O(m^3)$ time. Then we compute the arrangement $\mathcal{A}(Y)$, while maintaining the lists Y_z for any cell $z \in \mathcal{A}(Y)$. This can be done in $O(m^4)$ as the complexity of $\mathcal{A}(Y)$ is $O(m^4)$.

Now, for each cell $z \in \mathcal{A}(Y)$ we compute $Size_B(z, j, j')$ for all $1 \leq j \leq j' \leq m$ as follows: Notice that the problem of finding a minimum cover to $B[j, j']$ from a set of subsequences, is actually an interval-cover problem. We refer to the shapes in Y_z as intervals (between 1 and m), and the goal is to find the minimum number of intervals from Y_z needed in order to cover the interval $[j, j']$.

First, for every $1 \leq j \leq n$ we find $max(j)$ - the largest interval from Y_z that starts at j. This can be done simply in $O(m^2 \log m)$ time, by sorting the intervals first by their lower bound and then by their upper bound.

Next, for an interval $Y_{t,t'} \in Y_z$, consider the intervals in Y_z whose lower bound lies in $[t, t']$ and whose upper bound is greater than t'. Let $next(Y_{t,t'})$ be the largest upper bound among the upper bounds of these intervals. We can find $next(Y_{t,t'})$, for each $Y_{t,t'} \in Y_z$, in total time $O(m^2 \log m)$, using a segment tree as follows: Let

$S = \{s_1, \dots, s_n\}$ be a set of line segments on the x-axis, $s_i = [a_i, b_i]$. Construct a segment tree for the set S. With each vertex v of the tree, store a variable r_v, whose initial value is $-\infty$. Query the tree with each of the left endpoints. When querying with a_i, in each visited vertex v with non-empty list of segments do: if $b_i > r_v$, then set r_v to b_i. Finally, for each segment s, let $next(s)$ be the maximum over the values r_v of the vertices storing s.

After computing $next(Y_{t,t'})$ for all $Y_{t,t'} \in Y_z$ (assume $next(Y_{t,t'}) = -\infty$ for $Y_{t,t'} \notin Y_z$), we use Algorithm 8.5 to compute $Size_B(z, j, j')$ for all $1 \le j \le j' \le m$. The running time of Algorithm 8.5 is $O(m^2)$. Thus, computing $Size_B(z, j, j')$ for all cells $z \in \mathcal{A}(Y)$ and all indices $1 \le j \le j' \le m$ takes $O(m^6 \log m)$ time.

Algorithm 8.5 $Size_B(Y_z)$

```
For $j$ from 1 to $m$:
    1. Set $counter \leftarrow 1$
    2. Set $j' \leftarrow j$.
    3. Set $p \leftarrow \max\{next(Y_{j,j'}), max(j'+1)\}$.
    4. While $p \neq -\infty$:
        For $k$ from $j'$ to $p$: Set $Size_B(z, j, k) \leftarrow counter$.
        Set $counter \leftarrow counter + 1$
        Set $p \leftarrow \max\{next(Y_{j',k}), max(k+1)\}$.
        Set $j' \leftarrow k$.
```

Step 2

Recall that $\mathcal{A}(Y)$ is the arrangement obtained from the shapes in Y. Let $\mathcal{A}(D_A)$ be the arrangement of the disks $D_A = \{D(a_k, \delta_1) \mid 1 \le k \le n\}$. The number of cells in $\mathcal{A}(D_A)$ is $O(n^2)$.

A trivial algorithm to compute the value $Size_A(X_{i,i'}, j, j')$ is by considering the values $Size_B(z, j, j')$ of $O(m^4)$ cells from $\mathcal{A}(Y)$. Since there are $O(n^2)$ shapes $X_{i,i'} \in X$ and $O(m^2)$ pairs of indices $1 \le j \le j' \le m$, the running time will be $O(n^2 m^6)$. We manage to reduce the running time by a factor of $O(n)$, using some properties of the arrangement of disks.

Let $\mathcal{U}$ be the arrangement of the shapes in Y and the disks in D_A. Notice that $\mathcal{U}$ is a union of the arrangements $\mathcal{A}(D_A)$ and $\mathcal{A}(Y)$.

Lemma 8.14. *The number of cells in $\mathcal{U}$ is $O((m^2 + n)^2)$.*

The proof is similar to the proof of Lemma 8.13.

We make a few quick observations:

Observation 8.15. *For any two cells $w \in \mathcal{U}, x \in \mathcal{A}(D_A)$, $x \cap w \neq \emptyset$ if and only if $w \subseteq x$. Similarly, for any cell $z \in \mathcal{A}(Y)$, $z \cap w \neq \emptyset$ if and only if $w \subseteq z$.*

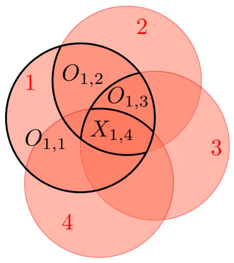


Figure 8.8: The arrangement $\mathcal{A}(D_A)$. After computing $Size_A(X_{1,4}, j, j')$, we know that $Size_A(X_{1,3}, j, j')$ is the minimum between $Size_A(X_{1,4}, j, j')$ and the values of the cells in $O_{1,3}$.

Observation 8.16. *For any cell $x \in \mathcal{A}(D_A)$, if $X_{i,i'} \cap x \neq \emptyset$, then $x \subseteq X_{i,i'}$.*

Observation 8.17. *For any $1 \leq i \leq i' \leq n$ we have $X_{i,i'+1} \subseteq X_{i,i'}$.*

Given $w \in \mathcal{U}$, let z_w be the cell from $\mathcal{A}(Y)$ that contains w. We have $Size_B(w, j, j') = Size_B(z, j, j')$. Let $O_{i,i'}$ be the set of cells $w \in \mathcal{U}$ s.t. $w \subseteq X_{i,i'}$ and $w \nsubseteq X_{i,i'+1}$.

For fixed $1 \leq j \leq j' \leq m$ and $1 \leq i \leq n$, the idea is to compute the values

$$Size_A(X_{i,n}, j, j'), Size_A(X_{i,n-1}, j, j'), \ldots, Size_A(X_{i,i}, j, j')$$

in this order, so we can use the value of $Size_A(X_{i,i'+1}, j, j')$ in order to compute $Size_A(X_{i,i'}, j, j')$, adding only the values of the cells in $O_{i,i'}$ (see Figure 8.8). This way, any cell in $\mathcal{U}$ will be checked only once (for any fixed $1 \leq j \leq j' \leq m$ and $1 \leq i \leq n$), and the running time will be $O(m^2 n(n + m^2)^2)$.

Now we have to show how to find the sets $O_{i,i'}$.

First, for any cell $x \in \mathcal{A}(D_A)$ we find all the cells $w \in \mathcal{U}$ such that $w \subseteq x$. There are $O(n^2)$ cells in $\mathcal{A}(D_A)$, but from Observation 8.15 we keep a total of $O((m^2 + n)^2)$ cells from $\mathcal{U}$.

Then, for any shape $X_{i,i'} \in X$ we find the set of cells $P_{i,i'}\{x \in \mathcal{A}(D_A) \mid x \subseteq X_{i,i'}\}$. There are $O(n^2)$ shapes in X, and for each shape we keep $O(n^2)$ cells from $\mathcal{A}(D_A)$.

Now we have $O_{i,i'} = P_{i,i'} \setminus P_{i,i'+1}$. The size of $P_{i,i'}$ is $O(n^2)$, so computing $O_{i,i'}$ for all $1 \leq i \leq i' \leq n$ takes $O(n^4)$ time.

The total running time for all $PC_1[i, i', j, j']$ is $O(m^6 \log m + m^2 n(n + m^2)^2)$

Total running time

For computing $PC_2[i, i', j, j']$ we get symmetrically a total running time of $O(n^6 \log n + n^2 m(m + n^2)^2)$, so the running time for computing all the components is $O((m + n)^6 \min\{m, n\})$. Calculating $OPT[i, j][r]$ takes $O(m^2 n^2 \min\{m, n\})$ time, all together takes $O((m + n)^6 \min\{m, n\})$ time.

8.5.2 An approximation algorithm for GCPS-3F

To approximate GCPS, we use approximated pair-components which are easier to compute.

Let $APC_1[i, i', j, j']$ be the minimum number of disks with radius δ_2 needed in order to cover the points of $B[j, j']$ (in order), and whose centers are located in $X_{i,i'} \oplus \delta_3$. Similarly, let $APC_2[i, i', j, j']$ be the minimum number of disks with radius δ_1 needed in order to cover the points of $A[i, i']$ (in order), and whose centers are located in $Z_{j,j'} \oplus \delta_3$.

Lemma 8.18. *For any* $1 \le i \le i' \le n, 1 \le j \le j' \le m$, $APC_1[i, i', j, j'] \le PC_1[i, i', j, j']$.

Proof. Recall that $PC_1[i, i', j, j']$ is the size of the minimum set C of disks of radius δ_2 that covers $B[j, j']$, and there exists a disk c of radius δ_1 that covers $A[i, i']$, s.t. for any $c' \in C$, the distance between the center of c and c' is at most δ_3. Notice that c is located in $X_{i,i'}$, and thus all the points of C are located in $X_{i,i'} \oplus \delta_3$. It follows that $APC_1[i, i', j, j'] \le |C| = PC_1[i, i', j, j']$. □

Computing the approximated components

We present a greedy algorithm that given $1 \le i \le i' \le n, 1 \le j \le j' \le m$, computes $APC_1[i, i', j, k]$ for all $j \le k \le j'$ (resp. $APC_2[i, k, j, j']$ for all $i \le k \le i'$). The algorithm runs in $O((j' - j)(j' - j + i' - i))$ time (See Algorithm 8.6).

Algorithm 8.6

```
Find $X_{i,i'} = \bigcap_{i \le k \le i'} D(a_k, \delta_1)$.
Set $R \leftarrow \mathbb{R}$.
Set $counter \leftarrow 1$.
Set $k \leftarrow j$.
While $k \le j'$ and $counter \ne \infty$:
    1. Set $R \leftarrow R \cap D(b_k, \delta_2)$.
    2. If $(X_{i,i'} \oplus \delta_3) \cap R \ne \emptyset$, set $APC_1[i, i', j, k] \leftarrow counter$.
    3. Else,
         Set $R \leftarrow D(b_k, \delta_2)$.
         If $(X_{i,i'} \oplus \delta_3) \cap R \ne \emptyset$, set $counter \leftarrow counter + 1$.
         Else, set $counter \leftarrow \infty$.
         Set $APC_1[i, i', j, k] \leftarrow counter$.
    4. Set $k \leftarrow k + 1$.
```

Running time. Finding $X_{i,i'}$ takes $O(i'-i)$ time, and step 1 takes $O(j'-j)$ time. Step 2 takes $O(j'-j+i'-i)$ time, since the complexity of $X_{i,i'} \oplus \delta_3$ is $O(i'-i)$, the complexity of R is $O(j'-j)$, and both regions are convex. The while loop runs $O(j'-j)$ times, so the total running time is $O((j'-j)(j'-j+i'-i))$.

Computing all the approximated pair components using Algorithm 8.6 takes $O(n^2m^2(m+n))$ time. The idea of our algorithm is to compute only a small part of the components, and then approximate the others using the ones that were computed.

Lemma 8.19. *Fix* $1 \le i \le i' \le n, 1 \le j \le j' \le m$, *then for any* $i \le x \le i'$ *and* $j \le y \le j'$*:*

1. $APC_1[i,x,j,j'] \le APC_1[i,i',j,j']$ *and* $APC_1[x,i',j,j'] \le APC_1[i,i',j,j']$.
2. $APC_1[i,i',j,y] + APC_1[i,i',y,j'] \le APC_1[i,i',j,j'] + 1$.
3. $APC_1[i,x,j,y] + APC_1[x,i',y,j'] \le APC_1[i,i',j,j'] + 1$.

Proof. Let $i \le x \le i'$ and $j \le y \le j'$. (1) is clear because the region $X_{i,i'} \oplus \delta_3$ is contained in the regions $X_{i,x} \oplus \delta_3$ and $X_{x,i'} \oplus \delta_3$, and thus a solution to $APC_1[i,i',j,j']$ is also a solution to $APC_1[i,x,j,j']$ and $APC_1[x,i',j,j']$.

Let $C = c_1,\dots,c_t$ be the set of size $t = APC_1[i,i',j,j']$ of disks that covers $B[j,j']$ and whose centers are located in $X_{i,i'} \oplus \delta_3$. Let c_k be the disk that covers b_y. Then the set $c_1,\dots,c_k$ covers $B[j,y]$ and the set $c_k,\dots,c_t$ covers $B[y,j']$. We have $APC_1[i,i',j,y] \le k$ and $APC_1[i,i',y,j'] \le t-(k-1) = APC_1[i,i',j,j']-k+1$, which proves (2).

From (1) we have $APC_1[i,x,j,y]+APC_1[x,i',y,j'] \le APC_1[i,i',j,y]+APC_1[i,i',y,j']$. From (2), $APC_1[i,i',j,y] + APC_1[i,i',y,j'] \le APC_1[i,i',j,j'] + 1$, which proves (3). □

We only compute $APC_1[i,i,j,j']$,$APC_2[i,i,j,j']$ for all $1 \le i \le n$ and $1 \le j \le j' \le m$, and $APC_1[i,i',j,j]$,$APC_2[i,i',j,j]$ for all $1 \le i \le i' \le n$ and $1 \le j \le m$. This takes $O(nm^3 + n^2m^2)$ time using Algorithm 8.6.

Composing the approximated solution

Let $AAPC_1[i,i',j,j'] = APC_1[i,i,j,j'] + APC_1[i,i',j',j']$. By Lemma 8.19(3), choosing $x = i$ and $y = j'$, we have $APC_1[i,i,j,j'] + APC_1[i,i',j',j'] \le APC_1[i,i',j,j'] + 1$, and by Lemma 8.18 we have $AAPC_1[i,i',j,j'] \le PC_1[i,i',j,j'] + 1$.

Now let $APX[i,j]$ be the approximate solution for $A[1,i]$ and $B[1,j]$. We set

$$APX[i,j] = \min_{p<i,q<j} APX[p,q] + \min\{AAPC_1[p+1,i,q+1,j], AAPC_2[p+1,i,q+1,j]\}$$

Obviously, given the values of $AAPC_1$ and $AAPC_2$, $APX[n,m]$ can be computed in $O(m^2n^2)$ time.

Lemma 8.20. *Let OPT be the size of an optimal solution, i.e. OPT is the smallest number such that there exists a pair of chains A',B' each of at most OPT (arbitrary) vertices, such that $d_1(A, A') \leq \delta_1$, $d_2(B, B') \leq \delta_2$, and $d_{dF}(A', B') \leq \delta_3$. Then $APX[n, m] \leq 2 \cdot OPT$.*

Proof. Let A' and B' be a pair of chains, each of at most OPT (arbitrary) vertices, such that $d_1(A, A') \leq \delta_1$, $d_2(B, B') \leq \delta_2$, and $d_{dF}(A', B') \leq \delta_3$.

Let $W_{A'B'} = \{(A'_i, B'_i)\}_{i=1}^{t}$ be a Fréchet walk along A' and B'. The pairs (A'_i, B'_i) represents the pair components that are composing an optimal solution. Let A_i and B_i be the pair of subchains of A and B, respectively, that we associated with the pair (A'_i, B'_i) in the beginning of Section 8.5.

With each pair (A'_i, B'_i), we associate a value C_i as follows: Let $A_i = A[p, p']$ and $B_i = B[q, q']$, then $C_i = \min\{AAPC_1[p, p', q, q'], AAPC_2[p, p', q, q']\}$. Notice that C_i is the number of points in one side of the approximated component. From Lemma 8.19, we have $C_i \leq \min\{PC_1[p, p', q, q'] + 1, PC_2[p, p', q, q'] + 1\} = \max\{|A'_i|, |B'_i|\} + 1$.

Thus, there exists a solution that uses the approximated components, of size:

$$\sum_{i=1}^{t} C_i \leq \sum_{i=1}^{t} (\max\{|A'_i|, |B'_i|\} + 1) = |A'| + |B'| \leq 2 \cdot \max\{|A'|, |B'|\} = 2 \cdot OPT.$$

□

Thus we have the following theorem:

Theorem 8.21. *A 2-approximation for GCPS can be computed in $O(nm^3 + n^2m^2 + n^3m)$ time.*

Remark 8.22. Notice that we do not have to actually compute a solution to GCPS, just to return the minimum k. A solution of size $2 \cdot OPT$ can be computed as follows: for each approximated component $APC_1[i, i', j, j']$ (or $APC_2[i, i', j, j']$) keep the set C_1 of centers of disks that are located in $X_{i,i'} \oplus \delta_3$. For each such center $c_1 \in C_1$, find a point c_2 in $X_{i,i'}$ s.t. $d(c_1, c_2) \leq \delta_3$, and put c_2 in a new set C_2. If our solution $APX[n, m]$ uses the approximated component $APC_1[i, i', j, j']$, then the points of C_1 will be used to cover $B[j, j']$ and the points of C_2 will be used to cover $A[i, i']$.

8.5.3 1-sided GCPS

In this variant of the problem, we can imagine there are two dogs, one is walking on a path A and the other on a path B, and a man has to walk both of them, one with a leash of length δ_1 and the other with a leash of length δ_2. We have to find a minimum-size polygonal path for the man, such that he can walk both dogs together.

Problem 8.23 (1-Sided General Chain Pair Simplification)**.**
Instance: Given a pair of polygonal chains A and B of lengths n and m, respectively,

an integer k, and two real numbers $\delta_1, \delta_2 > 0$.
Problem: Does there exist a chain C of at most k (arbitrary) vertices, such that $d_{dF}(A, C) \leq \delta_1$ and $d_{dF}(B, C) \leq \delta_2$?

Denote $X_{i,i'} = \bigcap\limits_{i \leq k \leq i'} D(a_k, \delta_1)$ and $Z_{j,j'} = \bigcap\limits_{j \leq k \leq j'} D(b_k, \delta_2)$ as before.

For any $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$, let $I[i, i', j, j'] = \begin{cases} 1, & X_{i,i'} \cap Z_{j,j'} \neq \emptyset \\ 0, & otherwise \end{cases}$.
Notice that $I[i, i', j, j'] = 1$ if and only if there exists one point that covers both $A[i, i']$ and $B[j, j']$. The values of $I[i, i', j, j']$ can be computed in $O((n+m)^4)$ time, using Algorithm 8.7.

Algorithm 8.7 Given i, j, compute $I[i, p, j, q]$ for all $i \leq p \leq n,\ j \leq q \leq m$.

```
Set $q \leftarrow m$

For $p = i$ to $n$:

     Set $I[i, p, j, s] \leftarrow 0$ for all $q < s \leq m$.
     While $q \geq j$:
          If $X_{i,p} \cap Z_{j,q} \neq \emptyset$,
               Set $I[i, p, j, s] \leftarrow 1$ for all $j \leq s \leq q$.
          Else,
               Set $I[i, p, j, q] \leftarrow 0$.
               Set $q \leftarrow q - 1$.
```

Notice that if $I[i, p, j, q] = 0$, than $I[i, p', j, q] = 0$ for any $p' > p$. The running time of Algorithm 8.7 is $O((m+n)^2)$: testing whether $X_{i,p} \cap Z_{j,q} \neq \emptyset$ takes $O(m+n)$ time. The number of such tests is $O(m+n)$, because p, q are monotonically increasing.

Thus we can compute $I[i, i', j, j']$ for any $1 \leq i \leq i' \leq n,\ 1 \leq j \leq j' \leq m$ in $O(mn(m+n)^2)$ time by running Algorithm 8.7 for all i, j.

Now we use a dynamic programming algorithm as follows: Let $OPT[i, j]$ be the length of the minimum-length sequence C such that $d_{dF}(A[1, i], C) \leq \delta_1$ and $d_{dF}(B[1, j], C) \leq \delta_2$. Fix $i, j > 1$, we have $OPT[i, j] = \min\limits_{p,q: I[p,i,q,j]=1} \{OPT[p-1, q-1] + 1\}$.

Running time The values of $I[i, i', j, j']$ can be computed in $O((n+m)^4)$ time. For each $i, j > 1$, we have $O(mn)$ values to check. Thus, the running time is $O((m+n)^4)$.

8.6 GCPS under the Hausdorff distance

The Hausdorff distance between two sets of points A and B is defined as follows:

$$d_H(A, B) = \max\{\max_{a \in A} \min_{b \in B} d(a, b),\ \max_{b \in B} \min_{a \in A} d(a, b)\}.$$

As mentioned above, the chain pair simplification under the Hausdorff distance (CPS-2H) is **NP**-complete. In this section we investigate the general version of this problem. We prove that it is also **NP**-complete, and give an approximation algorithm for the problem.

8.6.1 GCPS-2H is NP-complete

We show that GCPS under Hausdorff distance (GCPS-2H) is **NP**-complete, we use a simple reduction from geometric set cover: Given a set P of n points, and a radius δ, find the minimum number of disks with radius δ that cover P.

Let the sequence A be the points of P in some order (the order does not matter), and the sequence B be one point b with distance 2δ from P. Let $\delta_1 = \delta_2 = \delta$ and $\delta_3 = 4\delta + diam(P)$. Now a simplification for B is just one point anywhere in $D(b, \delta)$, and finding a simplification for A is equivalent to finding the minimum-cardinality set of disks that covers P.

Theorem 8.24. *GCPS-2H is **NP**-complete.*

8.6.2 An approximation algorithm for GCPS-2H

Consider the variant of GCPS-2H where $d_1 = d_2 = d_H$ and the distance between the simplifications A' and B' is measured with Hausdorff distance and not Fréchet distance (i.e. $d_H(A', B') \leq \delta_3$ instead of $d_{dF}(A', B') \leq \delta_3$). We call this variant GCPS-3H. Next, we show that GCPS-3H=GCPS-2H.

Lemma 8.25. *Given two sets of points A and B, if $d_H(A, B) \leq \delta$, then there exist an ordering A' of the points in A and an ordering B' of the points in B, such that $d_{dF}(A', B') \leq \delta$.*

Proof. We construct a bipartite graph $G(V = A \cup B, E)$, where $E = \{(a, b) \mid a \in A,\ b \in B,\ d(a, b) \leq \delta\}$. Notice that since $d_H(A, B) \leq \delta$, there are no isolated vertices. Now, while there exists a path with three edges in the graph, delete the middle edge. The maximal path in the resulting graph G' has at most two edges, and there are still no isolated vertices (because we only delete the middle edge). Let $C_1, \ldots, C_t$ be the connected components of G'. Notice that each C_i has exactly one point from A or exactly one point from B. Let A' be the sequence of points $C_1 \cap A, \ldots, C_t \cap A$, and B' be the sequence $C_1 \cap B, \ldots, C_t \cap B$. We get that $C_1, \ldots, C_t$ are a paired walk along A' and B' with cost at most δ. □

Since we can choose the order of points in the simplifications A' and B' in the GCPS-2H problem, we get by the above lemma that any solution for GCPS-3H is also a solution for GCPS-2H. Also, since for any two sequence P, Q we have $d_H(P, Q) \leq d_{dF}(P, Q)$, we get that any solution for GCPS-2H is also a solution for GCPS-3H.

Lemma 8.26. *Given two sets of points A and B, if $d_H(A,B) \leq \delta$, then there exist an ordering A' of the points in A and an ordering B' of the points in B, such that $d_{dF}(A',B') \leq \delta$.*

Proof. We construct a bipartite graph $G(V = A \cup B, E)$, where $E = \{(a,b) \mid a \in A,\ b \in B,\ d(a,b) \leq \delta\}$. Notice that since $d_H(A,B) \leq \delta$, there are no isolated vertices. Now, while there exists a path with three edges in the graph, delete the middle edge. The maximal path in the resulting graph G' has at most two edges, and there are still no isolated vertices (because we only delete the middle edge). Let $C_1, \dots, C_t$ be the connected components of G'. Notice that each C_i has exactly one point from A or exactly one point from B. Let A' be the sequence of points $C_1 \cap A, \dots, C_t \cap A$, and B' be the sequence $C_1 \cap B, \dots, C_t \cap B$. We get that $C_1, \dots, C_t$ are a paired walk along A' and B' with cost at most δ. □

Since we can choose the order of points in the simplifications A' and B' in the GCPS-2H problem, we get by the above lemma that any solution for GCPS-3H is also a solution for GCPS-2H. Now, since for any two sequence P, Q we have $d_H(P,Q) \leq d_{dF}(P,Q)$, we get that any solution for GCPS-2H is also a solution for GCPS-3H.

Let $S_1 = \{p_1, \dots, p_k\}$ be the smallest set of points such that for each $a_i \in A$ there exists some $p_j \in S_1$ s.t. $d(a_i, p_j) \leq \delta_1$ and for each $p_j \in S_1$ there exists some $b_i \in B$ s.t. $d(p_j, b_i) \leq \delta_2 + \delta_3$. Notice that since S_1 is minimum, we also know that for each $p_j \in S_1$ there exists some $a_i \in A$ s.t. $d(a_i, p_j) \leq \delta_1$ (or, we can just delete the points of S_1 that do not cover any points from A).

We can find a c-approximation for S_1, using a c-approximation algorithm for discrete unit disk cover (DUDC). The DUDC problem is defined as follows: Given a set P of t points and a set D of k unit disks on a 2-dimensional plane, find a minimum-cardinality subset $D' \subseteq D$ such that the unit disks in D' cover all the points in P. We denote by $T_c(k,t)$ the running time for a c-approximation algorithm for the DUDC problem with k unit disks and t points.

Lemma 8.27. *Given a c-approximation algorithm for the DUDC problem that runs in $T_c(k,t)$ time, we can find a c-approximation for S_1 in $T_c(n,(m+n)^2) + O((m+n)^2)$ time.*

Proof. Compute the arrangement of $\{D(a_i, \delta_1)\}_{1 \leq i \leq m} \cup \{D(b_j, \delta_2 + \delta_3)\}_{1 \leq j \leq n}$ (there are $(m+n)^2$ disjoint cells in the arrangement). Clearly, it is enough to choose one candidate from each cell. Now we can use the c-approximation algorithm for the DUDC problem. □

Symmetrically, let $S_2 = \{q_1, \dots, q_l\}$ be the smallest set of points such that for each $b_i \in B$ there exists some $q_j \in S_2$ s.t. $d(b_i, q_j) \leq \delta_2$ and for each $q_j \in S_2$ there exists some $a_i \in A$ s.t. $d(q_j, a_i) \leq \delta_1 + \delta_3$.

For each point $p_j \in S_1$ there exists some $b_i \in B$ s.t. $d(p_j, b_i) \leq \delta_2 + \delta_3$, so we can find a point p'_j such that $d(p'_j, b_i) \leq \delta_2$ and $d(p'_j, p_j) \leq \delta_3$. Denote $S'_1 = \{p'_1, \ldots, p'_k\}$. We do the same for the points of S_2, and find a set $S'_2 = \{q'_1, \ldots, q'_k\}$ such that for any $q'_j \in S'_2$,$d(q'_j, q_j) \leq \delta_3$ and there exists some $a_i \in A$ s.t. $d(q'_j, a_i) \leq \delta_1$.

Now, we know that for each $a_i \in A$ there exists some $p \in S_1 \cup S'_2$ s.t. $d(a_i, p) \leq \delta_1$, and, on the other hand, for each $p \in S_1 \cup S'_2$ there exists some $a_i \in A$ s.t. $d(a_i, p) \leq \delta_1$. So we have $d_H(A, S_1 \cup S'_2) \leq \delta_1$. Similarly, we have $d_H(B, S_2 \cup S'_1) \leq \delta_2$. We also know that for each $p_j \in S_1$ we have a point $p'_j \in S'_1$ s.t. $d(p'_j, p_j) \leq \delta_3$, and for each $q'_j \in S'_2$ we have a point $q_j \in S_2$ s.t. $d(q'_j, q_j) \leq \delta_3$. So we also have $d_H(S_1 \cup S'_2, S_2 \cup S'_1) \leq \delta_3$, and since CPS-2H=CPS-3H, we get that $S_1 \cup S'_2$ and $S_2 \cup S'_1$ is a possible solution for CPS-2H.

The size of the optimal solution OPT is at least $\max\{|S_1|, |S_2|\}$. Using a c-approximation algorithm for finding S_1 and S_2, the size of the approximate solution will be $c(|S_1| + |S_2|) \leq 2c \max\{|S_1| + |S_2|\} = 2c \cdot OPT$.

Theorem 8.28. *Given a c-approximation algorithm for the DUDC problem that runs in $T_c(k, t)$ time, our algorithm gives a 2c-approximation for the GCPS-2H problem, and runs in $T_c(n, (m+n)^2) + T_c(m, (m+n)^2) + O((m+n)^2)$ time.*

Conclusion and Open Problems

In the first part of this thesis, we investigated several variants of the discrete Fréchet distance that make more sense in some realistic scenarios. Specifically, we considered situations were the input curves contain noise, or when they are not aligned with respect to each other.

First, we described efficient algorithms for three variants of the discrete Fréchet distance with shortcuts (DFDS). Previously, only continuous variants of Fréchet distance with shortcuts were considered in the literature, some of which were proven to be NP-hard. We showed that the discrete variants are much easier to compute, even in the semi-continuous case. Moreover, given two curves of lengths $m \leq n$, respectively, we presented a linear time algorithm for the decision version of 1-sided DFDS, and an $O((m+n)^{6/5+\varepsilon})$ expected time algorithm for the optimization version. This gap between the decision and the optimization versions is due to the number of possible values that can determine the distance between the curves. It is an interesting open problem to either close this gap by presenting a near linear time algorithm, or to prove a lower bound stating that no algorithm exists for 1-sided DFDS whose running time is $O(n^{1+\delta})$ for some $\delta < 1/5$. Surprisingly, 1-sided DFDS is even easier to compute than the classic DFD: It was shown that under some computational assumption, there is no algorithm with running time in $O(n^{2-\varepsilon})$ for DFD. It would be interesting to find other variants of Fréchet distance that are meaningful but also easier to compute.

Next, we study another important variant of DFD — the discrete Fréchet distance under translation. We consider several variants of the translation problem. For DFD with shortcuts in the plane, we present an $O(m^2n^2\log^2(m+n))$-time algorithm. The running time of our algorithm is very close to the lower bound of $n^{4-o(1)}$ recently presented in [BKN19], for DFD under translation. It would be interesting to see if a similar bound applies for the shortcuts variant. When the points of the curves are in 1D, we present an $O(m^2n(1+\log(n/m)))$ time algorithm for DFD, $O(mn\log(m+n))$ time algorithm for the shortcuts variant, and $O(mn\log(m+n)(\log\log(m+n))^3)$ time algorithm for the weak variant, all under translation. In contrast to the lower bound of $O(n^{2-\varepsilon})$ for computing DFD (with no translation), which also applies when the points are in 1D, our results show that the translation problem becomes easier in 1D. Another interesting open question is whether lower bounds can be proven for the problem in 1D. Furthermore, in Chapter 3 we presented an alternative scheme for BOP, and demonstrated its advantage when applied to the most uniform path problem, the most uniform spanning tree, and the weak DFD under translation in 1D. It would be interesting to see if there are other problems that could benefit from using our scheme.

Finally, in the last chapter of this part, we presented the discrete Fréchet gap (DFG) as an alternative distance measure for curves. We showed that there is an interesting connection between DFG and DFD under translation in 1D: We can use (almost) similar algorithms to compute them. An open question is are there other connections between these variants, and in which situations one can establish that the gap version and its variants are preferable over the classic DFD.

In the second part of the thesis, we dealt with problems that arise in the context of big data, i.e., when our input is huge and thus its processing must be super efficient. In some of these problems, the input is a large set of polygonal curves or trajectories, and we need to preprocess or compress it such that certain information can be retrieved efficiently. In other problems, we are given one or two protein chains that we need to visualize or manipulate without losing some valuable features.

We first considered the nearest neighbor problem for curves (NNC), which is a fundamental problem in machine learning. We presented a simple and deterministic algorithm for the approximation version of the problem (ANNC), which is more efficient than all previous ones. However, our approximation data structure still uses space and query time exponential in m, which makes it impractical for large curves. Thus, we also identified several important cases in which it is possible to obtain near linear bounds for the problem. In these cases, either the query is a line segment or the set of input curves consists of line segments. There are many questions that remain open regarding the nearest neighbor problem. First, it would be interesting to see how our algorithms generalize to the case of 3-vertex curves, and whether we can achieve near linear bounds for this case as well. Secondly, can we improve the query time of our ANNC data structure? can we find a trade-off between the query time and space complexity? Furthermore, is it possible to use our data structures in order to solve the range searching problem, without increasing the space consumption?

Next, we studied several cases of the (k, ℓ)-center problem for curves. Since this problem is NP-hard when k or ℓ are part of the input, we studied the case where $k = 1$ and $\ell = 2$, i.e., the center curve is a segment. We presented near-linear time exact algorithms under L_∞, even when the location of the input curves is only fixed up to translation. Under L_2, we presented a roughly $O(n^2m^3)$-time exact algorithm. In a very recent result, Buchin et. al. [BDS19] give a polynomial time exact algorithm for (k, ℓ)-center under DFD (with L_2), when k and ℓ are constants. Plugging $k = 1$ and $\ell = 2$ in their bound, one gets a running time of $O(m^5n^4)$. Therefore, an obvious open question is can we generalize our algorithm to the case where k and ℓ are some constants? Another question is what other cases of the center problem can be solved in polynomial time? And also, is there a different definition of the center problem for curves, which is meaningful and also easier to compute? For example, instead of minimizing the distance to the centers, we can minimize their length ℓ or number k, for a given radius r. The problem with this variant is that a solution may not exist if r is too small.

Finally, in the last two chapters of this part, we discussed the simplification problem for polygonal curves or chains. We presented a collection of data structures for DFD queries, and then showed how to use them to preprocess a chain for k-simplification queries. Then we considered the chain pair simplification problem (CPS), which aims at simplifying two chains simultaneously, so that the distance between the resulting simplifications is bounded. When the chains are simplified using the Hausdorff distance (CPS-2H), the problem becomes **NP**-complete. However, the complexity of the version that uses DFD (CPS-3F) has been open since 2008. We introduced the weighted version of the problem (WCPS) and proved that WCPS-3F is weakly **NP**-complete. Then, we resolved the question concerning the complexity of CPS-3F by proving that it is polynomially solvable, contrary to what was believed. Moreover, we devised a sophisticated $O(m^2n^2 \min\{m, n\})$-time dynamic programming algorithm for the minimization version of the problem. We also considered a more general version of the problem (GCPS) where the vertices of the simplifications may be arbitrary points, and presented a (relatively) efficient polynomial-time algorithm for the problem, and a more efficient 2-approximation algorithm. We also investigated GCPS under the Hausdorff distance, showing that it is **NP**-complete and presented an approximation algorithm for the problem. The running times of our algorithms is rather high, and since CPS-3F has several applications that require an efficient running time, an obvious question is whether it is possible to reduce the running time of the algorithm for CPS-3F? Also, this problem was considered only for general curves, is it possible to improve the running time for more "realistic" curves, for example, c-packed or backbone curves? In addition, it would be interesting to consider the case where we want to simplify more than two curves simultaneously.

To wrap up, the Fréchet distance and its variants have been widely studied in many different settings during the last few decades. Nevertheless, many problems are still open, and many new intriguing questions are born with each problem that is settled. In this thesis, we have tried to contribute to the developing theory dealing with the Fréchet distance, by addressing a collection of fundamental problems. We hope that our work will turn out to be useful and that it will stimulate further work in this fascinating domain.

Bibliography

[AAKS14] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM Journal on Computing*, 43(2):429–449, January 2014.

[ABB$^+$14] Sander P. A. Alewijnse, Kevin Buchin, Maike Buchin, Andrea Kölzsch, Helmut Kruckenberg, and Michel A. Westenberg. A framework for trajectory segmentation by stable criteria. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM Press, 2014.

[ACMLM03] C. Abraham, P. A. Cornillon, E. Matzner-Lober, and N. Molinari. Unsupervised curve clustering using b-splines. *Scandinavian Journal of Statistics*, 30(3):581–595, September 2003.

[AD18] Peyman Afshani and Anne Driemel. On the complexity of range searching among curves. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 898–917, 2018.

[AFK$^+$14] Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete Fréchet distance with shortcuts via approximate distance counting and selection. In *Proceedings of the 30th Annual ACM Sympos. on Computational Geometry, SoCG*, page 377, 2014.

[AFK$^+$15] Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Transactions on Algorithms*, 11(4):29, 2015.

[AG95] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05(01n02):75–91, 1995.

[AHK$^+$06] Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *Proceedings of* *the 14th Annual European Sympos. on Algorithms, ESA*, pages 52–63, 2006.

[AHMW05] Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

[AKS$^+$12] Hee-Kap Ahn, Christian Knauer, Marc Scherfenberg, Lena Schlipf, and Antoine Vigneron. Computing the discrete Fréchet distance with imprecise input. *Int. J. Comput. Geometry Appl.*, 22(1):27–44, 2012.

[AKS15] R. Ben Avraham, H. Kaplan, and M. Sharir. A faster algorithm for the discrete Fréchet distance under translation. *CoRR*, abs/1501.03724, 2015.

[AKW01] Helmut Alt, Christian Knauer, and Carola Wenk. Matching polygonal curves with respect to the Fréchet distance. In *Proceedings of the 18th Annual Sympos. on Theoretical Aspects of Computer Science*, pages 63–74, 2001.

[AKW03] Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[Alt09] Helmut Alt. The computational geometry of comparing shapes. In *Efficient Algorithms, Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 235–248, 2009.

[AP02] P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, June 2002.

[AS94] Pankaj K. Agarwal and Micha Sharir. Planar geometric location problems. *Algorithmica*, 11(2):185–195, 1994.

[BBG08] Kevin Buchin, Maike Buchin, and Joachim Gudmundsson. Detecting single file movement. In *Proceedings of the 16th ACM SIGSPATIAL Internat. Sympos. on Advances in Geographic Information Systems, ACM-GIS*, page 33, 2008.

[BBG$^+$11] Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *Int. J. Comput. Geometry Appl.*, 21(3):253–282, 2011.

[BBK$^+$07] Kevin Buchin, Maike Buchin, Christian Knauer, Günter Rote, and Carola Wenk. How difficult is it to walk the dog? In *IProceedings of the 23rd European Workshop on Computational Geometry*, pages 170–173, 2007.

[BBMM14] Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog — with an application to Alt's conjecture. In *Proceedings of the 25th ACM-SIAM Sympos. Discrete Algorithms*, pages 1399–1413, 2014.

[BBMS12] K. Buchin, M. Buchin, W. Meulemans, and B. Speckmann. Locally correct Fréchet matchings. In *Proceedings of the 20th European Symposium Algorithms*, pages 229–240, 2012.

[BBMS19] Kevin Buchin, Maike Buchin, Wouter Meulemans, and Bettina Speckmann. Locally correct fréchet matchings. *Comput. Geom.*, 76:1–18, 2019.

[BBvL$^+$13] K. Buchin, M. Buchin, R. van Leusden, W. Meulemans, and W. Mulzer. Computing the Fréchet distance with a retractable leash. In *Proceedings of the 21st European Sympos. Algorithms*, pages 241–252, 2013.

[BBW09] Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the 20th ACM-SIAM Sympos. Discrete Algorithms*, pages 645–654, 2009.

[BDG$^+$19] Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating (k, l)-center clustering for curves. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2922–2938, 2019.

[BDS14] Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the 30th Sympos. Comput. Geom.*, page 367, 2014.

[BDS19] Kevin Buchin, Anne Driemel, and Martijn Struijs. On the hardness of computing an average curve. *CoRR*, abs/1902.08053, 2019.

[BJW$^+$08] Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proceedings of the 8th Latin American Theoretical Informatics Sympos., LATIN*, pages 630–641, 2008.

[BKN19] Karl Bringmann, Marvin Künnemann, and André Nusser. Fréchet distance under translation: Conditional hardness and an algorithm via offline dynamic grid reachability. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2902–2921, 2019.

[BM16] Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal on Computational Geometry*, 7(2):46–76, 2016.

[BPSW05] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st Internat. Conf. Very Large Data Bases*, pages 853–864, 2005.

[Bri14] Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science*, Philadelphia, PA, USA, October 2014. IEEE.

[CDG$^+$11] Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the 13th Workshop on Algorithm Engineering and Experiments, ALENEX*, pages 75–83, 2011.

[CdVE$^+$10] Erin W. Chambers, Éric Colin de Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Comput. Geom.*, 43(3):295–311, 2010.

[CL07] Jeng-Min Chiou and Pai-Ling Li. Functional clustering and identifying substructures of longitudinal data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(4):679–699, September 2007.

[CMMT86] Paolo M. Camerini, Francesco Maffioli, Silvano Martello, and Paolo Toth. Most and least uniform spanning trees. *Discrete Applied Mathematics*, 15(2-3):181–197, 1986.

[CR18] Timothy M. Chan and Zahed Rahmati. An improved approximation algorithm for the discrete fréchet distance. *Inf. Process. Lett.*, 138:72–74, 2018.

[dBGM17] Mark de Berg, Joachim Gudmundsson, and Ali D Mehrabi. A dynamic data structure for approximate proximity queries in trajectory data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 48. ACM, 2017.

[dBI11] Mark de Berg and Atlas F. Cook IV. Go with the flow: The direction-based Fréchet distance of polygonal curves. In *Proceedings of the International Conference on Theory and Practice of Algorithms in (Computer) Systems*, pages 81–91, 2011.

[dBIG13] Mark de Berg, Atlas F. Cook IV, and Joachim Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, August 2013.

[DH13] A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Computing*, 42(5):1830–1866, 2013.

[DHW12] Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

[DKS16] Anne Driemel, Amer Krivošija, and Christian Sohler. Clustering time series under the Fréchet distance. In *Proceedings of the 27th ACM-SIAM Symposium on Discrete Algorithms*, pages 766–785, Arlington, VA, USA, January 2016. Society for Industrial and Applied Mathematics.

[Dri13] Anne Driemel. *Realistic Analysis for Algorithmic Problems on Geographical Data*. PhD thesis, Utrecht University, 2013.

[DS07] Krzysztof Diks and Piotr Sankowski. Dynamic plane transitive closure. In *Proceedings of the 15th European Sympos. Algorithms*, pages 594–604, 2007.

[DS17] Anne Driemel and Francesco Silvestri. Locality-Sensitive Hashing of Curves. In *Proceedings of the 33rd International Symposium on Computational Geometry*, volume 77, pages 37:1–37:16, Brisbane, Australia, July 2017. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[EFN17] Michael Elkin, Arnold Filtser, and Ofer Neiman. Terminal embeddings. *Theor. Comput. Sci.*, 697:1–36, 2017.

[EFV07] A. Efrat, Q. Fan, and S. Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *J. Mathematical Imaging and Vision*, 27(3):203–216, 2007.

[EM94] Thomas Eiter and Heikki Mannila. Computing discrete fréchet distance. Technical report, Citeseer, 1994.

[EP18] Ioannis Z. Emiris and Ioannis Psarros. Products of euclidean metrics and applications to proximity questions among curves. In *Proceedings of the 34th International Symposium on Computational Geometry, SoCG*, pages 37:1–37:13, 2018.

[FFK^{+}15] Chenglin Fan, Omrit Filtser, Matthew J. Katz, Tim Wylie, and Binhai Zhu. On the chain pair simplification problem. In *Proceedings of the 14th Internat. Symp. on Algorithms and Data Structures WADS*, pages 351–362, 2015.

[FFK19] Arnold Filtser, Omrit Filtser, and Matthew J. Katz. Approximate nearest neighbor for curves - simple, efficient, and deterministic. *CoRR*, abs/1902.07562, 2019.

[FFKZ16] Chenglin Fan, Omrit Filtser, Matthew J. Katz, and Binhai Zhu. On the general chain pair simplification problem. In *Proceedings of the 41st International Symposium on Mathematical Foundations of Computer Science, MFCS*, pages 37:1–37:14, 2016.

[Fil18] Omrit Filtser. Universal approximate simplification under the discrete fréchet distance. *Inf. Process. Lett.*, 132:22–27, 2018.

[FK15] Omrit Filtser and Matthew J. Katz. The discrete Fréchet gap. *CoRR*, abs/1506.04861, 2015.

[FK18] Omrit Filtser and Matthew J. Katz. Algorithms for the discrete fréchet distance under translation. In *Proceedings of the 16th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT*, pages 20:1–20:14, 2018.

[FR17] Chenglin Fan and Benjamin Raichel. Computing the Fréchet gap distance. In *Proceedings of the 33rd Sympos. Comput. Geom.*, pages 42:1–42:16, 2017.

[Fré06] M. Maurice Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo*, 22(1):1–72, 1906.

[GH17] Joachim Gudmundsson and Michael Horton. Spatio-temporal analysis of team sports. *ACM Computing Surveys*, 50(2):1–34, April 2017.

[GO95] Anka Gajentaan and Mark H. Overmars. On a class of o(n2) problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

[God91] Michael Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *Proceedings of the 8th Annual Sympos. on Theoretical Aspects of Computer Science STACS*, pages 127–136, 1991.

[Gon85] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[GS88] Zvi Galil and Baruch Schieber. On finding most uniform spanning trees. *Discrete Applied Mathematics*, 20(2):173–175, 1988.

[HN79] Wen-Lian Hsu and George L. Nemhauser. Easy and hard bottleneck location problems. *Discrete Applied Mathematics*, 1(3):209–215, November 1979.

[HPIM12] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[HSV97] Pierre Hansen, Giovanni Storchi, and Tsevi Vovor. Paths with minimum range and ratio of arc lengths. *Discrete Applied Mathematics*, 78(1-3):89–102, 1997.

[IM04] Piotr Indyk and Jiří Matoušek. Low-distortion embeddings of finite metric spaces. In *Handbook of Discrete and Computational Geometry, Second Edition*. Chapman and Hall/CRC, April 2004.

[Ind00] Piotr Indyk. *High-dimensional computational geometry*. PhD thesis, Stanford University, 2000.

[Ind02] Piotr Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proceedings of the 8th Symposium on Computational Geometry*, pages 102–106, Barcelona, Spain, June 2002. ACM Press.

[IW08] Atlas F. Cook IV and Carola Wenk. Geodesic Fréchet distance inside a simple polygon. In *Proceedings of the 25th Annual Sympos. on Theoretical Aspects of Computer Science, STACS*, pages 193–204, 2008.

[JK94] Jerzy W. Jaromczyk and Miroslaw Kowaluk. An efficient algorithm for the Euclidean two-center problem. In *Proceedings of the 10th Symposium on Computational Geometry*, pages 303–311, Stony Brook, NY, USA, June 1994. ACM Press.

[JL84] William Johnson and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1984.

[JXZ08] Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure-structure alignment with discrete Fréchet distance. *J. Bioinformatics and Computational Biology*, 6(1):51–64, 2008.

[KHM+98] Sam Kwong, Qianhua He, Kim-Fung Man, Chak-Wai Chau, and Kit-Sang Tang. Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition. *IJPRAI*, 12(4):573–594, 1998.

[KKS05] Man-Soon Kim, Sang-Wook Kim, and Miyoung Shin. Optimization of subsequence matching under time warping in time-series databases. In *Proceedings of the ACM Symposium on Applied Computing (SAC)*, pages 581–586, 2005.

[KS97] Matthew J. Katz and Micha Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26(5):1384–1408, 1997.

[MC05] Axel Mosig and Michael Clausen. Approximately matching polygonal curves with respect to the Fréchet distance. *Comput. Geom.*, 30(2):113–127, 2005.

[MMMR18] Sepideh Mahabadi, Konstantin Makarychev, Yury Makarychev, and Ilya P. Razenshteyn. Nonlinear dimension reduction via outer bi-lipschitz extensions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC*, pages 1088–1101, 2018.

[MP99] Mario E. Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *ICCV*, pages 108–115, 1999.

[MPTDW84] Silvano Martello, WR Pulleyblank, Paolo Toth, and Dominique De Werra. Balanced optimization problems. *Operations Research Letters*, 3(5):275–278, 1984.

[MSSZ11] Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Comput. Geom.*, 44(2):110–120, 2011.

[NN18] Shyam Narayanan and Jelani Nelson. Optimal terminal dimensionality reduction in euclidean space. *CoRR*, abs/1810.09250, 2018.

[NW13] Hongli Niu and Jun Wang. Volatility clustering and long memory of financial time series and financial price model. *Digital Signal Processing*, 23(2):489–498, March 2013.

[Rot07] Günter Rote. Computing the Fréchet distance between piecewise smooth curves. *Comput. Geom.*, 37(3):162–174, 2007.

[SDI06] Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. *Nearest-neighbor methods in learning and vision: theory and practice (neural information processing)*. The MIT press, 2006.

[Sha97] Micha Sharir. A near-linear algorithm for the planar 2-center problem. *Discrete & Computational Geometry*, 18(2):125–134, 1997.

[ST83] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[Tho00] Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 343–350. ACM, 2000.

[Wen03] Carola Wenk. *Shape matching in higher dimensions.* PhD thesis, Free University of Berlin, Dahlem, Germany, 2003.

[WL85] Dan E. Willard and George S. Lueker. Adding range restriction capability to dynamic data structures. *J. ACM*, 32(3):597–617, 1985.

[WLZ11] Tim Wylie, Jun Luo, and Binhai Zhu. A practical solution for aligning and simplifying pairs of protein backbones under the discrete Fréchet distance. In *Proceedings of the Internat. Conf. Computational Science and Its Applications, ICCSA*, pages 74–83, 2011.

[WSP06] Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: Localizing globalb curve-matching algorithms. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management, SSDBM*, pages 379–388, 2006.

[WZ13] Tim Wylie and Binhai Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 10(6):1372–1383, 2013.