A novel algorithm for computing the Fréchet distance

van Leusden, R.

Award date:
2013

Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Den Dolech 2, 5612 AZ Eindhoven
P.O. Box 513, 5600 MB Eindhoven
The Netherlands
www.tue.nl

Supervisor
dr. Kevin Buchin (TU/e)

Section
Algorithms and Visualization

Date
May 8, 2013

A Novel Algorithm for Computing the Fréchet Distance

Master's Thesis

Rolf van Leusden

Abstract

Finding the similarity between curves is a topic of research in many areas, such as computer science, geometric information systems and biology. To define the similarity between curves, we are interested in how close these curves are to each other or how much their shapes match. The Fréchet distance is a metric for curves and surfaces which is used to measure the similarity. Earlier algorithms to compute the Fréchet distance between curves are based on a decision problem, that is, given two curves of n segments return whether the Fréchet distance is smaller than or equal to some candidate value. Using this decision problem we can compute the Fréchet distance in $O(n^2 \log n)$ time. We propose a new algorithm which computes the Fréchet distance between curves without using this decision problem. For the Fréchet distance under the Euclidean distance we get a running time of $O(n^2 \log^2 n)$ which is asymptotically slower, however our novel algorithm is easier to implement. Furthermore we can use different distance metrics in our algorithm and get a faster running time. In particular, we can compute the Fréchet distance under the L_1 and L_∞ metric in $O(n^2)$ time. By defining a distance metric which approximates the Euclidean metric, we can compute a $(1+\varepsilon)$-approximation of the Fréchet distance under the Euclidean distance in $O(n^2/\sqrt{\varepsilon})$ time. By using the decision problem as a subroutine, we can improve this running time further to $O(n^2 \log(1/\varepsilon))$.

Preface

This thesis is the result of my graduation project on the master of Computer Science and Engineering at the Eindhoven University of Technology. Completing both the thesis and the preceding education would not have been possible without the endless support of my parents and sisters.

I would like to thank Kevin Buchin for supervising my graduation. Without his guidance, on both the theoretical part as well as on the writing, I would not have been able to present this thesis. For the theoretical part he discussed the topic with others as well, and I would like to thank them too for (in)directly helping me with this thesis. Furthermore I would like to thank the assessment committee with Mark de Berg and Huub van de Wetering for completing my graduation. Although many have read parts of this thesis, I furthermore thank Pauline van Leusden and Twan Vermeulen for selflessly reading the entire thesis and giving valuable comments.

During the graduation project, I have discussed related topics, and less related topics, with everybody working or graduating in the Algorithms group. These discussions were a valuable addition to the results I present in this thesis, or at least a nice break, and made the graduation highly enjoyable.

Rolf van Leusden
Eindhoven, May 8, 2013

Contents

1 Introduction

This thesis is about computing the similarity between curves. Intuitively you obtain a curve by placing a pencil on paper and drawing without lifting up the pencil. The curve has a start point where you first put the pencil down, and it has an end point where you stop drawing. An example is shown in Figure 1.1.

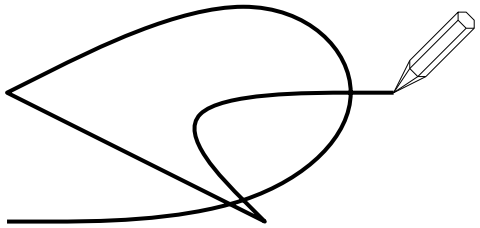


Figure 1.1: An example of a curve, constructed by drawing with a pencil.

Before making the notion of a curve more formal, let us first consider an example for curve similarity. Imagine we have a database of persons and their handwritten signatures, which for simplicity we assume to be curves. Now if someone signs a document, we might want to know which signature in the database is most similar to the signature on the document. With this approach, we can try to find which person just gave his or her signature. This is illustrated in Figure 1.2.

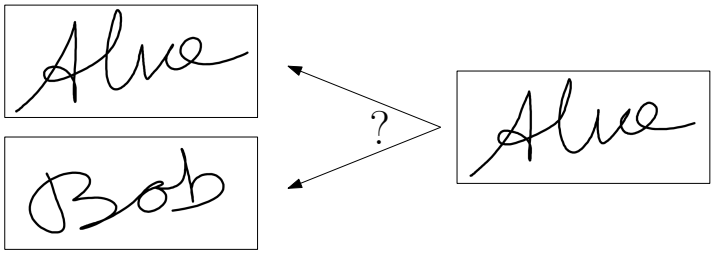


Figure 1.2: We have two handwritten signatures stored, Alice and Bob, and we try to match another signature of Alice to our database. Althouth it is not the same curve, we see that it is very similar to a signature already stored.

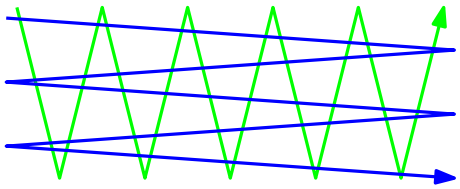


Figure 1.3: A small Hausdorff distance although a large Fréchet distance.

Distance measure for curves

Although the notions above allow us to intuitively argue about curve similarity, we will need to make the notions of a curve and of curve similarity more formal, to handle curve similarity algorithmically. We define a curve π as a continuous mapping with $\pi : [0, 1] \to \mathbb{R}^d$. To illustrate this definition let us consider the example of a curve from Figure 1.1. The point $\pi(0)$ is where we started the curve, the point $\pi(1)$ is where we ended it. Now imagine it took us 1 second to draw the curve, and let $\pi(t)$ is where the pencil was after t seconds, where t is a real number between 0 and 1. Then π indeed maps the unit interval $[0, 1]$ continuously to $\mathbb{R}^2$.

Rather than computing the similarity between two curves, we want to compute a distance between two curves, that is, we compute a measure for dissimilarity. Given any two curves π and σ, the lower the distance, the more similar they are. There are numerous ways to define a distance between curves, some capturing the dissimilarity of curves better than others. A commonly used distance is the Hausdorff distance. It can be calculated by, for all points on one curve, finding the minimum distance to any point on the other curve and vice versa. The maximum of all these minimum distances is the Hausdorff distance. Let us denote $\textsc{dist}(p, q)$ as the distance function between any two points. Mathematically, the Hausdorff distance $\delta_{\mathcal{H}}$ can be defined as a function of two curves π and σ

$$\delta_{\mathcal{H}}(\pi, \sigma) = \max\{\sup_{p\in\pi} \inf_{q\in\sigma} \textsc{dist}(p, q), \sup_{q\in\sigma} \inf_{p\in\pi} \textsc{dist}(p, q)\}.$$

For curves, however, the Hausdorff distance has considerable shortcomings. Consider for example the curves shown in Figure 1.3: Any point on one of the curves has a nearby point on the other curve. Therefore the Hausdorff distance is low. Intuitively these curves are quite dissimilar: while they are close on a point-wise basis, they are not so close if we try to map the curves continuously to each other. A distance measure that captures this intuition is the Fréchet distance. An intuitive illustration of the Fréchet distance is the following: given two curves imagine a man walking over one curve and his leashed dog over the other curve. The man and the dog may vary their speed and may stand still, but they do not move backwards over the curve. Both man and dog begin their walk at the respective starting points of the curves and want to finish their walk at the end points. Now the Fréchet distance between the curves is the minimum length of a leash for such a walk. Intuitively, the difference to the Hausdorff distance is that we require that if we match any point on one curve to any point on the other curve, i.e. p to q with $p \in \pi$ and $q \in \sigma$, we do not allow to match another point p' to q' with $p' \in \pi$ and $q' \in \sigma$ where p' lies after p, yet q' lies before q. The Fréchet distance

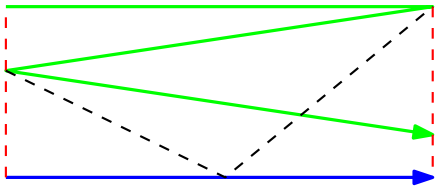


Figure 1.4: Given two curves, we need to enforce the black matching for the Fréchet distance, where we allow a smaller Hausdorff distance with the red lines.

$\delta_{\mathcal{F}}$ between curves is defined as

$$\delta_{\mathcal{F}}(\pi, \sigma) = \inf_{f,g} \max_{t \in [0,1]} \{\text{DIST}(\pi(f(t)), \sigma(g(t)))\},$$

where f and g are continuous strictly increasing functions with $f, g \colon [0,1] \to [0,1]$. The difference between the Hausdorff distance and the Fréchet distance may be arbitrarily large by stretching the example of Figure 1.4 horizontally. In this thesis we are interested in computing the Fréchet distance between curves efficiently. We therefore the discuss existing algorithms for this problem next.

Related work

The seminal algorithm for computing the Fréchet distance between curves by Alt and Godau [1995] has a running time of $O(n^2 \log n)$ for two input curves of each n line segments. The basic idea is to use a decision problem: is the Fréchet distance between two given curves equal to or less than some ε? The decision problem can be solved in quadratic time. To achieve the running time of $O(n^2 \log n)$ they use the parametric search of Megiddo [1983].

Parametric search is widely believed to be not very suitable in practice. Following the result of Alt and Godau, progress has been made on avoiding this parametric search. By randomizing the parametric search, Van Oostrum and Veltkamp [2002] have been able to make the parametric search more practical, however resulting in an expected $O(n^2 \log^2 n)$ time algorithm. Cook and Wenk [2010] presented an alternative to the parametric search and achieved an expected $O(n^2 \log^2 n)$ time algorithm as well. Har-Peled and Raichel [2011] have proposed a randomized algorithm for calculating the Fréchet distance in expected $O(n^2 \log n)$ time. All of these algorithms are based on computing the decision problem, and searching through possible values for the Fréchet distance.

A lower bound of the computation of the Fréchet distance has been given as well, by Buchin et al. [2007]: we need at least $\Omega(n \log n)$ to compute the Fréchet distance. After the initial research in 1995, no subquadratic algorithm has been found, which leads to Alt's conjecture that it might be 3SUM-hard [2009]. 3SUM is a problem which so far can only be computed in $O(n^2)$ time, or in slightly subquadratic time depending on the model of computation. Recent work of Buchin et al. [2012a] shows that this might not be the case. They reduce the computation time to $O(n^2 \sqrt{\log n} (\log \log n)^{3/2})$, which is currently the fastest algorithm to compute the Fréchet distance without making assumptions on the input curves. If we assume that input curves are not arbitrary but have certain reasonable properties, and with that avoiding worst-case input, Driemel et al. [2012] propose an algorithm which calculates the $(1+\varepsilon)$-approximation of the Fréchet distance in $O(cn/\varepsilon + cn \log n)$ time.

The Fréchet distance and its algorithms have been successfully used in various applications, in particular for the signature matching problem [Munich and Perona, 1999] from our introductory example. This approach can be extended to searching in full handwritten documents as shown by Sriraghavendra et al. [2007]. The Fréchet distance has been used in other areas as well. We can map vehicle tracking data to route-maps using the Fréchet distance as shown by [Brakatsoulas et al., 2005], detect patterns in movement data [Buchin et al., 2008, 2011, 2012b] and find protein structure alignment in moleculair biology [Jiang et al., 2008].

Contributions

We present a novel algorithm for computing the Fréchet distance which does not use the decision problem as a subroutine. For this algorithm we present a framework independent of the distance metric that we use between two points. Using the Euclidean distance we achieve a running time of $O(n^2 \log^2 n)$ for two curves of n segments. The strength of our approach can be seen when considering the L_1 or L_∞ distance metric, where we achieve a running time of $O(n^2)$. We can extend this to a $(1+\varepsilon)$-approximation algorithm for the Euclidean distance. By using a combination of our novel algorithm and the decision problem we can achieve a running time of $O(n^2 \log(1/\varepsilon))$ for this approximation.

Overview

In Chapter 2 we explain the earlier algorithm of Alt and Godau and give preliminary information on geometric algorithms and data structures used in this thesis. We then discuss our novel algorithm in Chapter 3. In Chapter 4 we show how to apply it to the Euclidean distance as point-wise distance. Although the Euclidean distance is an intuitive metric for points, we can use polyhedral distance functions as well which we explain in Chapter 5. We conclude our results in Chapter 6.

2 Preliminaries

In this chapter we discuss the concepts that our novel algorithm uses and earlier results. First of all, we shall explain in Section 2.1 the concept of a dynamic upper envelope; a data structure used quite often in geometric algorithms. After that, we shall discuss the algorithm of Alt and Godau [1995] in more detail. Their algorithm is based on a decision problem and in Section 2.2 we explain this decision problem for two line segments which we extend to polygonal curves in Section 2.3. In Section 2.4 we explain how to use the decision problem and conclude their algorithm for the Fréchet distance. Alt and Godau also proposed the Weak Fréchet distance, which we shall briefly discuss in Section 2.5

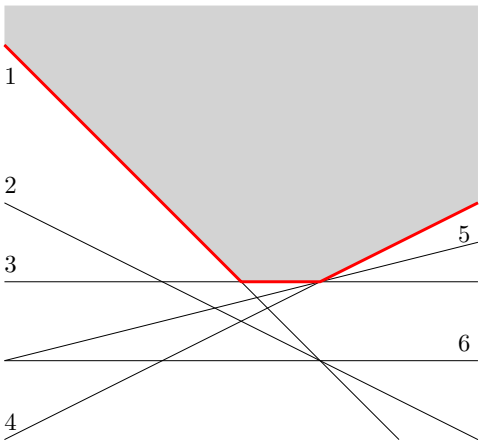
2.1 Dynamic Upper Envelope

For many data structures, we require all data to be known in advance. To calculate the convex hull on a set of points, many algorithms are known and perform up to $O(n \log h)$ by Kirkpatrick and Seidel [1986] and Chan [1996], with n points in the convex set and h points on their exterior, which in the worst case leads to $O(n \log n)$. However, if we add or remove a single point in our data set, these algorithms may need to recalculate the (entire) convex hull from scratch.

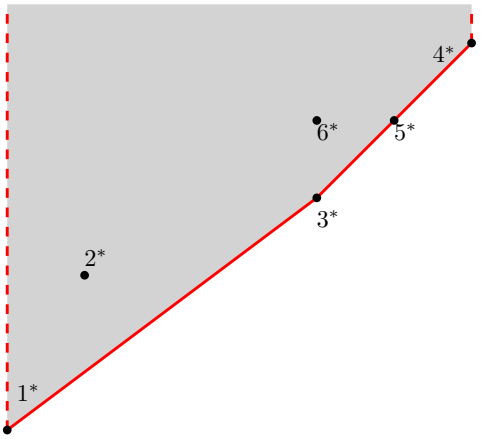
Dynamic data structures allow to maintain a data structure under insertions and deletions. Overmars and Van Leeuwen proposed in 1981 an algorithm to dynamically maintain the convex hull of n points with an insertion and deletion time of $O(\log^2 n)$. Their data structure is a tree, which stores at its root the current convex hull. The convex hull is again stored as a tree, more specifically we can assume that it is stored as a balanced binary search tree.

Using this balanced binary search tree, the following standard queries on the convex hull can be answered in $O(\log n)$ time. An *extreme-point query* returns the vertex which is extreme in a given direction and a *tangent query* asks whether a point lies in the convex hull and if not, the two tangent lines of the convex hull that pass through the given point. A more complicated standard query is a *line-stabbing query* which returns the two edges of the convex hull which the line intersects if such edges exist.

Following the work of Overmars and Van Leeuwen further research has been done on the dynamic convex hulls. Chan [2001] proposed an algorithm for maintaining the dynamic convex hull in which updates take amortized $O(\log^{1+\varepsilon} n)$ time for $\varepsilon > 0$. Many queries can still be done in $O(\log n)$, however a line-stabbing query may take up to $O(\log^2 n)$. By a



(a) The upper envelope on a set of lines.



(b) The lower convex hull of the set of points.

Figure 2.1: An upper envelope and its lower convex hull in the dual plane.

trade-off it is possible to have an amortized $O(\log^{3/2} n)$ update time and $O(\log^{3/2} n)$ query time. A different approach is proposed by Brodal and Jacob [2002] who achieve an optimal amortized $O(\log n)$ insertion and deletion time, which matches the $O(n \log n)$ computation time for the convex hull. However their data structure does not support line-stabbing queries.

In this thesis we actually need to dynamically maintain the upper envelope of lines. We can do so with dynamic convex hulls by using standard point-line duality [De Berg et al., 2008, Section 8.2]. See Figure 2.1 for an illustration.

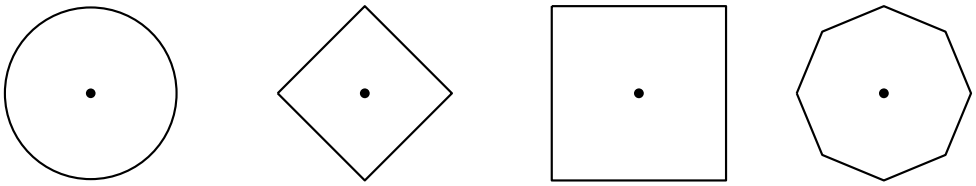
2.2 Free Space Diagram cell

The decision whether two directed line segments have a Fréchet distance which is less than or equal to some ε can be calculated in constant time if we use the Euclidean distance. If ε is larger than the distance between both start points and both end points, then the Fréchet distance is larger than ε. Since we usually have polygonal curves, we shall treat this case of two line segments as a base case for the Free Space Diagram (FSD) that we will later fully introduce. Consider two polygonal curves π and σ of both one segment. We define the FSD cell C_ε with fixed ε as follows.

$$C_\varepsilon = \{(s,t) \in [0,1]^2 \mid \textsc{dist}(\pi(s), \sigma(t)) \leq \varepsilon\}$$

If there exists a strictly increasing path in the C_ε from the lower leftmost point to the upper rightmost point, then this path defines the strictly increasing functions f and g of the definition of the Fréchet distance. Although the Fréchet distance may be lower than ε, finding the Fréchet distance is not part of the decision problem.

To give a more graphical intuition on how C_ε looks, we may draw the cell of the FSD cell of $[0,1]^2$ with every point $(s,t) \in C_\varepsilon$ white and $(s,t) \notin C_\varepsilon$ grey. If there exists a path in C_ε from start to end, then this path lies in an entirely white area. The colouring of C_ε depends on π, σ and the distance function DIST. Let us prove a single property about the C_ε which we require to hold for the rest of the thesis, which is how to colour C_ε. This has been proven before, e.g. in Lemma 3 of Alt and Godau [1995], however we extend it slightly to handle



(a) L_2 (b) L_1 (c) L_∞ (d) k-regular

Figure 2.2: The unit spheres of four different distance functions.

more distance metrics and dimensions.

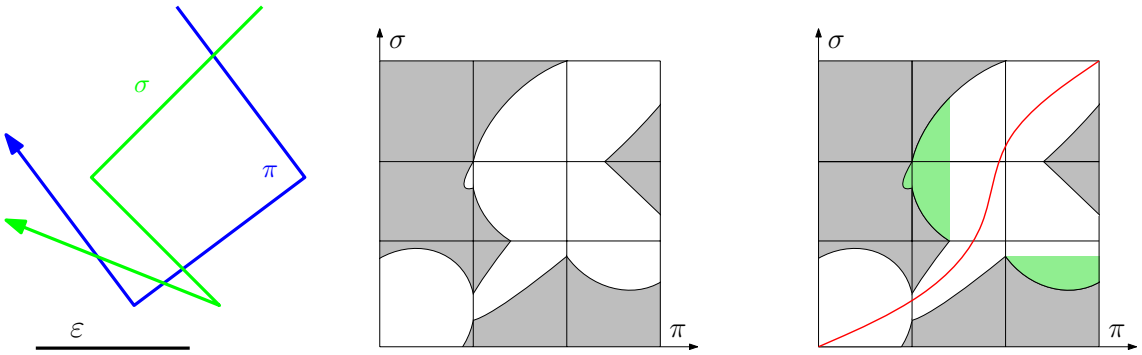
Lemma 1. *Every cell of the Free Space Diagram (FSD), C_ε, is convex using a convex distance function in $\mathbb{R}^d$.*

Proof. Consider line segments $\pi\colon [0,1] \to \mathbb{R}^d$ and $\sigma\colon [0,1] \to \mathbb{R}^d$ that are both affine mappings. A cell C_ε in the FSD represents the distance between π and σ. We extend the functions π and σ from $[0,1]$ to $\mathbb{R}$, such that we have $\pi'\colon \mathbb{R} \to \mathbb{R}^d$ and $\sigma'\colon \mathbb{R} \to \mathbb{R}^d$. We get as a result that π' and σ' are both affine mappings. We take the mapping $f\colon \mathbb{R}^2 \to \mathbb{R}^d$ defined by $f(s,t) = \pi'(s) - \sigma'(t)$, which is by closure of affine mappings, affine too.

Consider the convex set of points $D_\varepsilon^d = \{z \mid z \in \mathbb{R}^d \wedge ||z|| \leq \varepsilon\}$ with $||z||$ as metric of our choice (e.g. the L_2 distance, which implies D_ε^2 is a disk with radius ε). We take the preimage of f with the convex set D_ε^d and have all the points of $(s,t) \in \mathbb{R}^2$ which map into D_ε^d. Since the preimage of an affine mapping is affine too, we can apply an affine mapping to the convex set D_ε^d. An affine mapping to a convex set results in a convex set and thereby we have a convex set of all points (s,t) which fit in D_ε^d. Since we are only interested in $(s,t) \in [0,1]^2$, we take the intersection of the convex $[0,1]^2$ with $f^{-1}(D_\varepsilon^d)$, which is by intersection of two convex sets, convex too. We get that $C_\varepsilon = f^{-1}(D_\varepsilon^d) \cap [0,1]^2$ is convex for all convex distance metrics $||z||$. □

Distance functions can be illustrated with a unit sphere. This is a set of points such that all points with distance 1 are on the unit sphere. The most often used distance function is the Euclidean distance or L_2 as depictured in Figure 2.2a, which has a circle as unit sphere in 2 dimensions and extends to a hypersphere in $\mathbb{R}^d$. Another distance function is the Manhattan distance L_1, as in Figure 2.2b, which extends to cross-polytopes in higher dimensions. The maximum norm L_∞ is another well-known distance function and depictured in Figure 2.2c, which extens to hypercubes. We will discuss how our novel algorithm works without making assumptions on the distance function in Chapter 3. In Chapter 4 we will explain our algorithm for the Euclidean case and in Chapter 5 for polyhedral distance functions such as L_1 and L_∞.

All the distance functions in L_p are convex for $p \geq 1$, however we will only discuss L_1, L_2 and L_∞ for our algorithm. However, we can use other convex distance functions as well based on their unit sphere. Every k-regular polygon is convex, as for example the 8-regular polygon drawn in Figure 2.2d. In Chapter 5 we also explain how we can use a k-regular polygon as a distance function to approximate the Euclidean distance function.



(a) Two curves and ε (b) FSD F_ε (c) RFSD R_ε

Figure 2.3: Two curves and their FSD and RFSD. In the latter, we can see that the upper rightmost point is accessible by the red path, hence the Fréchet distance is less than or equal to ε.

2.3 Free Space Diagram for the decision problem

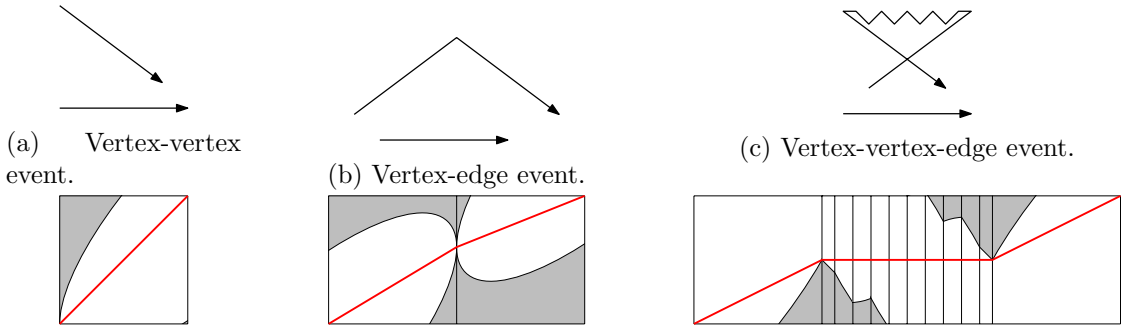
Although in the previous section we have explained how we can construct an FSD cell C_ε, which is on itself is a FSD. We can extend this to an FSD over polygonal curves. Let us have π of n and σ of m line segments. We can compare all these segments and get a grid of $m \times n$, each cell coloured as in Lemma 1. These nm cells together constitute the FSD F_ε. Consider the two curves π and σ as in Figure 2.3a and an ε. We draw the FSD of these lines and ε in Figure 2.3b, where we use the Euclidean distance as metric.

To find whether there exists a path which matches the increasing functions f and g of the definition of the Fréchet distance, we require the notion of the Reachable Free Space Diagram (RFSD) R_ε. We can calculate this iteratively by assuming that if we have calculated R_ε for all cells left or lower in F_ε, we extend this to the current cell. Since in the definition of the Fréchet distance we require increasing functions f and g, we cannot go back and hence a breadth-first computation of R_ε suffice. For each cell, given that we can only come from below or the left to calculate the reachable free space, computing the reachable free space can be done with $O(d)$ computation time per cell for L_1, L_2 and L_∞ in $\mathbb{R}^d$. An example of the resulting R_ε from Figure 2.3b is depicted in Figure 2.3c, where we mark the now inaccessible area green. We observe that the upper rightmost point is reachable over white space with e.g. the red line. As a result, we have that the Fréchet distance $\delta_\mathcal{F}(\pi, \sigma) \leq \varepsilon$.

Since the size of F_ε is nm and calculating the RFSD for this can be done in constant time for each cell given that we have processed earlier cells, we derive that R_ε can be calculated in $O(nmd)$ time. Checking whether the upper rightmost point is in R_ε can be done in constant time and hence the decision problem takes $O(nmd)$ time to compute.

2.4 Critical values

To use the Reachable Free Space Diagram as a decision problem, we need to have a set of candidate values for ε such that at least one of these is the Fréchet distance. These candidate values are critical values which occur by choosing $\varepsilon = 0$ and continuously increase ε. As a result F_ε will become larger. However, the path needs to exist in R_ε rather than only F_ε. By



(a) Vertex-vertex event.

(b) Vertex-edge event.

(c) Vertex-vertex-edge event.

(d) Vertex-vertex event in the FSD.

(e) Vertex-edge event in the FSD.

(f) Vertex-vertex-edge event in the FSD.

Figure 2.4: Three kinds of curves where critical instances of the Fréchet distance is achieved and their corresponding FSD.

increasing ε, the decision whether we can go from one free cell to another can be split into three cases. We name these candidates for the actual Fréchet distance critical values.

First of all, we need to have $(0,0)$ and (n,m) in R_ε and hence in F_ε. This critical distance is a vertex-vertex event (VV-event), every event only occurs between two vertices. Secondly, we are interested in the minimal ε between two neighbouring cells in F_ε, such that decreasing ε closes the possible passage from one cell to the next. These can only occur between every pair of a vertex and an edge and is thereby a vertex-edge event (VE-event). The last case is where we may have a path from one cell to a later cell on that row, but that path is strictly decreasing. The minimal ε such that we can go from a cell to this later cell is again a candidate. For this we need to compare two vertices on one curve to an edge on the other curve. Hence we call it a vertex-vertex-edge event (VVE-event). For the three input curves in Figure 2.4 we have the three different events where the critical distance ε is the Fréchet distance, as explained with their FSD.

We have two VV-events, i.e. the distance both start points and both end points are candidates for the Fréchet distance. For the VE-events, we need to compare all vertices of one curve to all edges on the other curve and vice versa, hence we get $O(nm)$ candidates for the critical distance. The number of VVE-events is even larger since we need to compare all pairs of vertices on one curve to every edge on the other curve, which gives us $O(n^2m)$ critical distances for one way and $O(nm^2)$ the other way. As a result, we have $O(n^2m+nm^2)$ candidates for the critical distances for the VV, VE and VVE-events.

An intuitive solution for finding the Fréchet distance is thereby finding all critical distances, sort them and apply a binary search. Finding these critical distances can be done in $O(1)$ for each event (assuming L_1, L_2 or L_∞). Sorting of critical distances is unfortunately $O((n^2m+nm^2)\log(nm))$, and a binary search on $O(n^2m+nm^2)$ values with the decision problem in $D(n,m)=O(nm)$ is in $O(nm\log(nm))$. The total running time therefore for this algorithm is $O((n^2m+nm^2)\log(nm))$. This can be improved by using a parametric search on the VVE-events. With the parametric search of Megiddo [1983], the running time is $O((D(n,m)+nm)\log(nm))$ running time which leads to well-known result of $O(nm\log(nm))$ running time of Alt and Godau. A disadvantage of this approach is that although the parametric search of Megiddo performs well in theory, in practice this result is not as feasible as one might expect. The main two reasons are that the implementation is hard and the

hidden constants in the big-O-notation are quite large, as explained by e.g. Van Oostrum and Veltkamp [2002].

2.5 Weak Fréchet distance

The Weak Fréchet distance is a weaker notion of the Fréchet distance. Where in the Fréchet distance we require the functions f and g to be strictly increasing, we can drop this requirement for the Weak Fréchet distance; but still enforce that f and g are continuous. Furthermore we add boundary conditions which imply both start points and both end points are mapped to each other. We can define this as

$$\delta_{\mathcal{WF}}(\pi, \sigma) = \inf_{f,g} \max_{t \in [0,1]} \{\text{DIST}(\pi(f(t)), \sigma(g(t)))\},$$

with f and g continuous functions, $f(0) = g(0) = 0$ and $f(1) = g(1) = 1$. Note that with this weaker notion we are allowed to ignore the VVE-events, which makes the computation less complex. A disadvantage to this approach is that we may enter each cell in the FSD also from above or the right when computing the RFSD. Alt and Godau named this distance the Nonmonotone Fréchet-metric and proposed two $O(nm \log(nm))$ time algorithms for solving this [Alt and Godau, 1995, Section 3.1].

The decision problem for calculating the Weak Fréchet Distance is based on the fact that between each two consecutive cells there exists a minimum ε to go from one cell to the other. We create a grid-based planar graph of $O(nm)$ vertices where the weight of the edges is the minimum ε to go from one cell to another. Given an ε as input to the decision problem, we delete all edges where their weight is larger than ε. If there exists a path from $(0, 0)$ to (n, m) for this ε, then $\delta_{\mathcal{WF}}(\pi, \sigma) \leq \varepsilon$. The graph has a size of $O(nm)$ and thereby removing the edges can be done in $O(nm)$ time. To decide if there exists a path from $(0, 0)$ to (n, m) can be computed in $O(nm)$ by a depth-first or breadth-first search over the remaining graph. Since there only exists $O(nm)$ events — VVE-events do not occur — this can be done without parametric search and we derive an $O(nm \log(nm))$ time algorithm.

Alt and Godau's second algorithm for calculating the Weak Fréchet Distance is without decision problem, where we try to find the minimum path from $(0, 0)$ over all edges to (n, m). This is a single-source shortest path problem and Alt and Godau proposed to use Prim's algorithm for calculating this path with start vertex $(0, 0)$ and stop when the minimum spanning tree processed so far contains the vertex (n, m). This approach is however more related to Dijkstra's algorithm than to Prim's algorithm. However in both cases we have the number of vertices $|V| = \Theta(nm)$ and the number of edges $|E| = \Theta(V)$. Hence, their calculation of the Weak Fréchet Distance can be done in $O(nm \log(nm))$. Note that single-source shortest paths can be faster. Since we have a planar graph, using the approach of Henzinger et al. [1997] we can reduce the computation time to $O(nm)$.

3 Algorithm

Recall that we have two polygonal curves π and σ of n and m segments respectively. Existing algorithms that calculate the Fréchet distance between these two curves, e.g. Alt and Godau [1995], do this by testing a subset of possible values for ε against a decision problem. For a given ε they solve the decision problem by computing the reachable part of the Free Space Diagram (FSD) as explained in Chapter 2. In such algorithms, the FSD is traversed multiple times for different candidate values ε for the Fréchet distance. We propose a different algorithm that traverses a grid similar to the FSD only once. However to get the Fréchet distance we need to maintain additional information during this traversal.

We first discuss a weaker notion of the Fréchet distance that we name the Vertex Monotone Fréchet distance and give an algorithm to calculate this. After this, we shall explain how this algorithm needs to be adapted for calculating the Fréchet distance.

3.1 Vertex Monotone Fréchet distance

To explain the conceptual ideas underlying our novel algorithm and the resulting challenges, we first consider a variant of the Fréchet distance. It can be seen as a hybrid metric between the weak Fréchet distance and the Fréchet distance where we do allow ourselves to traverse back, but only in the segment we are considering. We denote the vertices of a curve π as π_i with $i \in \{0, \ldots, n\}$ and for σ similarly. We name this variant the Vertex Monotone Fréchet distance (VMFD) and we can define this distance $\delta_{\mathcal{VMF}}(\pi, \sigma)$ as

$$\delta_{\mathcal{VMF}}(\pi, \sigma) = \inf_{f,g} \max_{t \in [0,1]} \{\text{DIST}(\pi(f(t)), \sigma(g(t)))\},$$

where f and g are continuous functions and if $\pi(f(t))$ is on segment $\pi_{i-1}\pi_i$ with $i \in \{1, \ldots, n\}$, then for $t' > t$ we have that $f(t')$ is not before vertex π_{i-1}, and similar for g. This corresponds in the RFSD to the situation that if we go to a next row or column, we cannot move to a preceding row or column. The RFSD for the VMFD is similar to the RFSD for the Fréchet distance, however with the only change that if we can enter a cell in the reachable free space, we mark the entire cell reachable. The RFSD of the VMFD also similar to the RFSD of the Weak Fréchet distance, however with the main difference that we are only allowed to enter a cell from the left or from below. We actually derive the following order of weakness.

$$\delta_{\mathcal{H}}(\pi, \sigma) \leq \delta_{\mathcal{WF}}(\pi, \sigma) \leq \delta_{\mathcal{VMF}}(\pi, \sigma) \leq \delta_{\mathcal{F}}(\pi, \sigma)$$

This can be proven since for the Hausdorff distance the 'functions' f and g do not need to be continuous. For the Weak Fréchet distance the functions must be continuous and hence this is stronger than the Hausdorff distance. For the VMFD the functions must be continuous and monotonic with respect to the vertices of the curves which is again a stronger requirement on f and g. For the Fréchet distance these functions must be entirely strictly increasing and hence this is an even stronger requirement.

To calculate the VMFD, we consider a grid with the same size as the FSD, i.e. $m \times n$. We dnote a cell in the grid by $C(i, j)$ with $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. We denote the left and lower cell edge of $C(i, j)$ by $L(i, j)$ and $B(i, j)$ respectively. Let $L(i, j)(\alpha)$ be a function that represents the distance between point π_i and segment $\sigma_{j-1}\sigma_j$, that is

$$L(i, j)(\alpha) = \text{DIST}(\pi_i, \sigma_{j-1} + \alpha(\sigma_j - \sigma_{j-1}))$$

for $\alpha \in [0, 1]$. We define $B(i, j)(\alpha)$ analogously.

According to the definition of VMFD we are allowed to traverse backwards on the line segment we are currently considering. We shall construct a graph in a similar way as in Alt and Godau's second algorithm for calculating the Fréchet distance, i.e. we let each cell $C(i, j)$ be a vertex and each cell edge $L(i, j)$ and $B(i, j)$ a *directed* edge in between the vertices, such that we cannot go backwards. The weight of these edges can be defined as the minimum of $L(i, j)(\alpha)$ (or $B(i, j)(\alpha)$) for $\alpha \in [0, 1]$. We assume that we can find this minimum in $O(\mu(D, d))$ time with D a convex distance function and d the number of dimensions. The function $\mu(D, d)$ depends only on the number of dimensions d for L_1, L_2 and L_∞, which implies that for each edge in the graph we have an additional $O(d)$ computation time.

Furthermore, we add two additional vertices $(0, 0)$ and (n, m) in the graph which are the start and end points of the two curves, which we want to match. We only draw a directed edge from $(0, 0)$ to $C(1, 1)$ with as weight $\text{DIST}(\pi_0, \sigma_0)$. Similarly we have one directed edge from $C(n, m)$ to (n, m) with weight $\text{DIST}(\pi_n, \sigma_m)$.

We can enter each cell from either the left or below and in our graph we only have the two matching input edges. We calculate and assign to each vertex a value $C(i, j).\varepsilon^L$ and a value $C(i, j).\varepsilon^B$, which are defined as the minimum ε to enter the free space from respectively the left cell or from the cell below. We define $C(i.j).\varepsilon = \min(C(i.j).\varepsilon^L, C(i.j).\varepsilon^B)$. We get that $C(i, j).\varepsilon^L = \max(C(i-1, j).\varepsilon, \min_{\alpha \in [0,1]}(L(i, j)(\alpha))$ and similarly for ε^B. We do a breadth-first search over all the vertices to calculate the minimum ε to get into the free space of a cell. The two special cases for start and end-point vertices are straightforward. As a result, the ε of (n, m) is the VMFD between the two curves.

The main difference between this algorithm and the approach of Alt and Godau for the Weak Fréchet distance, is that in our case we can only enter the cells from the left or below. This implies that we have directed edges rather than undirected edges, and that we can use a simple breadth-first computation of ε. The implementation is given Algorithm 1, where we implictly assume that cells not existing in the graph are unreachable, i.e. $C(i, 0).\varepsilon = \infty$ and $C(0, j).\varepsilon = \infty$.

We summarize our findings of this section in the following theorem.

Theorem 1. *Given two curves π and σ of n and m segments in $\mathbb{R}^d$, we can in $O(nm \cdot \mu(D, d))$ calculate the VMFD with Algorithm 1, where $\mu(D, d)$ is the time to find the minimum of the functions $L(i, j)$ or $B(i, j)$ which depends on the convex distance metric D and the number of dimensions d.*

Algorithm 1 Algorithm for calculating the VMFD.

VertexMonotoneFréchetDistance(π, σ)

```
1  C(1,1).ε^L ← DIST(π(0), σ(0))
2  C(1,1).ε^B ← DIST(π(0), σ(0))
3  for (i, j) ← (1, 1) to (n, m) by breadth-first
4       do
5          if i > 1
6             then C(i, j).ε^L ← max(C(i − 1, j).ε, min_{α∈[0,1]}(L(i, j)(α))
7          if j > 1
8             then C(i, j).ε^B ← max(C(i, j − 1).ε, min_{α∈[0,1]}(B(i, j)(α))
9          C(i, j).ε ← min(C(i, j).ε^L, C(i, j).ε^B)
10 return max(C(n, m).ε, DIST(π(n), σ(m)))
```

Since we have that $\mu(D, d)$ is $O(d)$ for L_1, L_2 and L_∞, we give a better bound on the VMFD.

Corollary 1. *Given two curves π and σ of respectively n and m segments in $\mathbb{R}^d$, we can in $O(nmd)$ time calculate the VMFD with Algorithm 1 using the L_1, L_2 or L_∞ distance metric.*

3.2 Generic novel Fréchet algorithm

In the previous section we have presented a simple algorithm for the VMFD. The difference between the VMFD and the actual Fréchet distance is that in the latter we require an entirely bimonotonic increasing path from $(0, 0)$ to the free space of $C(i, j)$, where for VMFD we may have decreasing parts between cells $C(h, j)$ and $C(i, j)$ with $h < i$. The main adaptation necessary to calculate the Fréchet distance is the following: when we want to calculate $C(i + 1, j).\varepsilon^L$, we have some path from $(0, 0)$ via $C(h, j)$ to $C(i + 1, j)$ with $h \leq i$. For that, we want to have a monotonically increasing path between $C(h, j)$ and $C(i + 1, j)$ for ε^L.

Although for the previous algorithm it was not really necessary, we prove that calculating $C(i, j).\varepsilon^B$ is symmetric to calculating $C(i, j).\varepsilon^L$.

Lemma 2. *Calculating $C(i, j).\varepsilon^B$ and calculating $C(i, j).\varepsilon^L$ are symmetric if the distance function is symmetric (i.e. $\text{DIST}(a, b) = \text{DIST}(b, a)$).*

Proof. If we swap the input curves π and σ, in the definition of the Fréchet distance the Fréchet distance remains equal. However the FSD and its corresponding RFSD are obtained by a transpose transformation over the diagonal from $C(1, 1)$ to $C(n, m)$. □

Note that although we require that the distance function is symmetic in Lemma 2, without this assumption we are able to calculate the Fréchet distance as well. The main difference is that without this assumption we calculate a directed Fréchet distance, i.e. $\delta_{\mathcal{F}}(\pi, \sigma) \neq \delta_{\mathcal{F}}(\sigma, \pi)$, which is counter-intuitive and will be avoided for the rest of the thesis. For the remainder of this section we shall only discuss how to calculate $C(i, j).\varepsilon^L$, by Lemma 2 $C(i, j).\varepsilon^B$ is calculated in a similar way. However note that when we calculate either one of those, the other is not necessarily known yet.

We first prove that $L(i,j)(\alpha)$ and $B(i,j)(\alpha)$ are unimodal functions for all $i \in \{1,\ldots,n\}$ and $j \in \{1,\ldots,m\}$ given that we use a convex distance function.

Lemma 3. *The functions $L(i,j)(\alpha)$ and $B(i,j)(\alpha)$ are unimodal functions, given that we use a convex distance metric D.*

Proof. For $L(i,j)(\alpha)$ we have the distance between point $p = \pi_i$ and the line segment $l = \sigma_{j-1}\sigma_j$. We assume that p is the origin, otherwise we translate l and p such that p becomes the origin, which does not change the distance between p and l. We apply an affine transformation to the line segment l to get a line, i.e. we have $l\colon [0,1] \to \mathbb{R}^d$ and extend this to $l'\colon \mathbb{R} \to \mathbb{R}^d$.

We define $S(r)$ to be the unit sphere of D around p with radius r. We project $l'(\alpha)$ to $S(r)$ and define this point of projection as $q(\alpha)$ and $l'(\alpha)$ has distance $r(\alpha)$ to the origin p for $\alpha \in \mathbb{R}$. If we extend the unit sphere to radius $r(\alpha)$, then $S(r(\alpha))$ intersects l' at $l'(\alpha) = r(\alpha)q(\alpha)$.

Consider α_m such that for all $\alpha \neq \alpha_m$ we have that $r(\alpha_m) \leq r(\alpha)$, i.e. α_m is a point where the distance between p and l' is minimized. Since $r(\alpha_m)$ is the minimum distance, all other points of l' are not inside $S(r(\alpha_m))$. Since S is convex and l' is continuous, there is only a subset of l' on or inside $S(r)$ for a fixed r. If we decrease r to r', this subset becomes smaller, i.e. points on $S(r)$ are no longer in the set.

We choose r as the maximum distance between p and either endpoint of l, e.g. $r(0)$ or $r(1)$. If we decrease $S(r)$ to $S(r')$, then we decrease the size of the set of points on l', i.e. the points $l'(\alpha)$ with distance r to p are no longer in the set. As a result, for decreasing r to r' we reduce the set of points on l' which have a distance less than or equal to r. This directly implies that the distance between l' and p can be represented as a unimodal function with $r(\alpha_m)$ as minimum. If we take only a part of this unimodal function, e.g. for $\alpha \in [0,1]$, we also have a unimodal function, which is $L(i,j)(\alpha)$. □

For simplicity, if we need to draw the unimodal functions $L(i,j)(\alpha)$ and $B(i,j)(\alpha)$ we will draw them as parabolas. Although this is not the exact unimodal function for most distance functions, it does give a better intuition.

We will use an approach similar to what we used for calculating VMFD, where we want to process all the cells in the grid breadth-first. With this, we can assume that once we process cell $C(i,j)$, all cells $C(1\ldots i, 1\ldots j)$ have been processed, except $C(i,j)$ itself. By Lemma 2 we only need to explain how to calculate $C(i,j).\varepsilon^L$. There exist four different cases for processing $C(i,j)$, which are the following:

1. $C(1,1)$.
2. $C(i,1)$ for $1 < i \leq n$ (and $C(1,j)$ for $1 < j \leq m$ by Lemma 2).
3. $C(i,j)$ for $1 < i \leq n$ and $1 < j \leq m$.
4. Extract the Fréchet distance after processing $C(n,m)$.

To enter the free space of $C(1,1)$, we can argue similarly as for the VMFD algorithm where the distance between π_0 and σ_0 is the minimum ε to get in the free space of $C(1,1)$. The extraction of the Fréchet distance after processing $C(n,m)$ is again similar, i.e. the Fréchet distance is the maximum of $C(n,m).\varepsilon$ and the distance between π_n and σ_m. Processing $C(i,1)$ is in one way similar as processing $C(i,j)$, except that there does not exists a path from the lower cells to $C(i,1)$. We create a row below the grid G where each $C(i,0).\varepsilon = \infty$, such that we can choose to enter the free space from below, but we never will. Hence, the

most complicated step is to process $C(i,j)$ where we can come from below or from the left; the only exception $C(1,1)$ is handled differently.

Assume that we already know the value $C(i+1,j).\varepsilon^L$ such that there exists a bimonotonic increasing path from $(0,0)$ to the free space of $C(i+1,j)$ where we enter $C(i+1,j)$ from the left, i.e. from $C(i,j)$. Furthermore we have a value $C(i+1,j).\alpha^L$ as the minimum height on the passthrough of $L(i+1,j)$ such that we have a bimonotonic increasing path for $C(i+1,j).\varepsilon^L$ and we denote the combination of ε^L and α^L as $C(i+1,j).(\alpha^L,\varepsilon^L)$. We want to minimize ε^L, however for the minimum ε^L we want to minimize α^L as well. When we have a bimonotonic increasing path from $(0,0)$ to the free space of $C(i+1,j)$ where we enter $C(i+1,j)$ from the left, we have that this path must enter row j within some cell — if $j=1$ at $C(1,1)$ cell from $(0,0)$ as a special case. We define the column where we enter row j from below as h, with $h \in \{1,\ldots,i\}$.

If we do not yet know $C(i+1,j).(\alpha^L,\varepsilon^L)$, there are multiple candidates for the value of h, i.e. $h \in \{1,\ldots,i\}$. To find the minimum (α^L,ε^L), we need to have a monotonically increasing path between all cell edges $L(h+1,j)$ to $L(i+1,j)$. It is easy to see that if we have less of these cell edges, finding a monotonically increasing path from $C(h,j)$ to $C(i+1,j)$ is easier and likely to be less restricting. However we also do not want to come from below in $C(h,j)$ if $C(h,j).\varepsilon^B > C(i+1,j).\varepsilon^L$.

A simple property is that $C(i+1,j).\varepsilon^L \geq C(i,j).\varepsilon$. We argue that to calculate $C(i+1,j).(\alpha^L,\varepsilon^L)$, we find the maximum h where $C(h,j).\varepsilon^B \leq C(i+1,j).\varepsilon^L$. From that we can argue that if we enter row j in column h^* while calculating $C(i,j).(\alpha^L,\varepsilon^L)$, for the maximum column h where we enter row j to enter the free space of $C(i+1,j)$ from the left we have $h^* \leq h$. We can then prove that the minimum $C(i+1,j).(\alpha^L,\varepsilon^L)$ lies on an upper envelope constructed of several lines and unimodal functions. The minimum $C(i+1,j).(\alpha^L,\varepsilon^L)$ is the minimum point on this upper envelope. Given that we have a candidate h^* for h, $h^* \leq h$, we prove that $C(i+1,j).(\alpha^L,\varepsilon^L)$ is the minimum of a upper envelope and then $h^* = h$, or $h^* < h$ and we increase h^* to a new candidate.

Let us first prove the lemma that $C(i+1,j).\varepsilon^L$ is not less than the minimum ε to enter the free space of the cell on the left.

Lemma 4. *$C(i+1,j).\varepsilon^L$ is larger than or equal to both $C(i,j).\varepsilon$ and the minimum of $L(i+1,j)(\alpha)$ with $\alpha \in [0,1]$.*

Proof. We prove this by contradiction. Suppose that $C(i+1,j).\varepsilon^L < C(i,j).\varepsilon$, then we have a bimonotonic increasing path that cannot go through the reachable free space of $C(i,j)$. Since any increasing path from $(0,0)$ to enter $C(i+1,j)$ from the left must go over $C(i,j)$, this is impossible. Similarly, to enter $C(i+1,j)$ from the left, we must pass through $L(i+1,j)$ at least at some height which cannot be lower than the minimum of $L(i+1,j)(\alpha)$ for $\alpha \in [0,1]$. □

For $C(i+1,j).(\alpha^L,\varepsilon^L)$, given that we may enter row j in column $h^* \in \{1,\ldots,i\}$, we need to maintain a monotonically increasing path over the cell edges $L(h^*,j)$ to $L(i+1,j)$. If we select a higher value for h^*, we reduce the amount of cell edges to which we need to satisfy the monotonically increasing path.

Lemma 5. *For processing $C(i+1,j).\varepsilon^L$, we enter row j from below in column h where $h = \max\{h' \mid C(h',j).\varepsilon^B \leq C(i+1,j).\varepsilon^L \wedge h' < i+1\}$.*

Proof. We prove this by contradiction. Suppose that we enter row j in column h^*. If $h < h^* < i+1$, we get either $C(h^*,j).\varepsilon^B > C(i+1,j).\varepsilon^L$ and we cannot enter row j at column

h^*, or $C(h^*, j).\varepsilon^B \leq C(i+1, j).\varepsilon^L$ which implies we prefer h^* as a better candidate than h (although not necessarily the best). If $h^* < h$, then by choosing h we can reduce the number of unimodal functions to satisfy on the path from $C(h, j)$ to $C(i+1, j)$ without invalidating $C(i+1, j).\varepsilon^L$, which gives column h to be the best option to enter row j. □

With this lemma we can prove that if we decide to enter row j in column h^* for the bimonotonic increasing path to enter the free space of $C(i, j)$ by the left, then to enter the free space of $C(i+1, j)$ from the left we enter row j in column h where $h^* \leq h$.

Lemma 6. *Given that we enter row j in column h^* where $h^* < i$ for calculating $C(i, j).(\alpha^L, \varepsilon^L)$, then for calculating $C(i+1, j).(\alpha^L, \varepsilon^L)$ we enter row j in column h where $h^* \leq h < i+1$.*

Proof. Since $C(i+1, j).\varepsilon^L \geq C(i, j).\varepsilon$, we derive that $C(i, j).\varepsilon^B \leq C(i+1, j).\varepsilon^L$ or $C(i, j).\varepsilon^L \leq C(i+1, j).\varepsilon^L$. If we assume the first, by Lemma 5 we enter row j in column $h = i$. If we assume the second, then get that $C(h, j).\varepsilon^B \leq C(i, j).\varepsilon^L \leq C(i+1, j).\varepsilon^L$ and hence we do not to enter row j in column for h' where $h' \leq h^* < h$. □

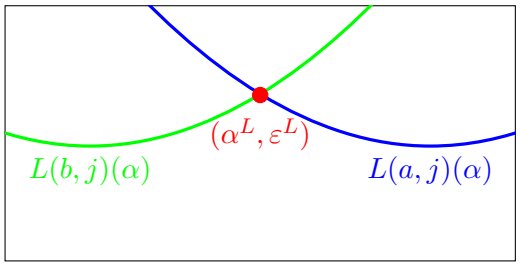
We combine Lemma 4 and Lemma 6.

Lemma 7. *$C(i, j).\varepsilon$ is larger than or equal to $C(k, j).\varepsilon$ for $k \in \{h, \ldots, i\}$ if we enter row j in column h for cell $C(i, j)$.*

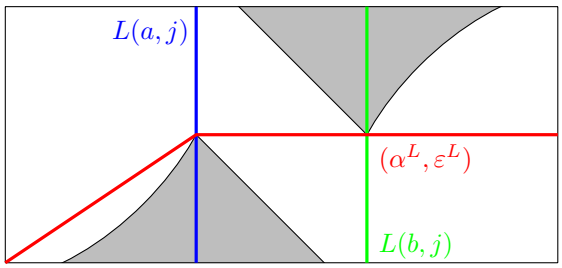
Proof. If we enter row j in column h for calculating $C(i, j)$, then there exists a path from $C(h, j)$ to $C(i, j)$, possibly with $h = i$. Since we do not decrease h by Lemma 6, by recursively use Lemma 4 from i to h the path from $C(h, j)$ to $C(i, j)$ must have a monotonically increasing path over $C(k, j).\varepsilon$ for $k \in \{h, \ldots, i\}$. □

To calculate $C(i+1, j).(\alpha^L, \varepsilon^L)$, an intuitive idea is that since all $L(h+1, j)(\alpha)$ to $L(i+1, j)(\alpha)$ are unimodal functions, we can create an upper envelope of those unimodal functions. The minimum pair $(\alpha^L, \varepsilon^L)$ on this upper envelope would be the best candidate for $C(i+1, j).(\alpha^L, \varepsilon^L)$. In Figure 3.1a we observe this behaviour of two functions $L(a, j)(\alpha)$ and $L(b, j)(\alpha)$, with the red dot on the intersection as minimum pair $(\alpha^L, \varepsilon^L)$ and indeed this is the minimum path in the FSD as shown in Figure 3.1b. However, if we would alternate the order of the two functions $L(a, j)(\alpha)$ and $L(b, j)(\alpha)$, as in Figure 3.1c, the red dot $(\alpha^L, \varepsilon^L)$ does not lie on the intersection and is not even satisfied by the unimodal function $L(a, j)(\alpha)$, yet it is the minimum pair for $(\alpha^L, \varepsilon^L)$ as we observe in the FSD in Figure 3.1d. We see that this is because we get a strictly increasing monotonic path in the FSD. If this is the case, the preceding unimodal functions do not necessarily need to be entirely satisfied, i.e. the minimum $(\alpha^L, \varepsilon^L)$ does not necessarily need to lie above all preceding unimodal functions.

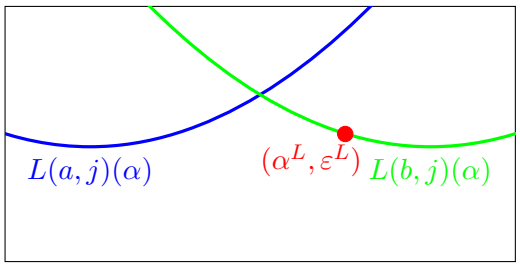
We can prove this differently, that is for all unimodal functions $L(h+1, j)(\alpha)$ to $L(i+1, j)(\alpha)$, we only need the last unimodal function to be entirely satisfied and for all preceding unimodal functions only the decreasing part satisfied, rather than the entire unimodal function. We have an illustration of this behaviour in Figure 3.2. Note that from Lemma 7 we have for each decreasing part of the unimodal function also a lower bound of the minimum of the unimodal function, which we have in Figure 3.2 as dotted lines. In the upper envelope we include the two horizontal lines $C(h, j).\varepsilon^B$ and $C(i, j).\varepsilon$, since both these lines must be satisfied as well by Lemma 4 and Lemma 5. We get the following lemma.



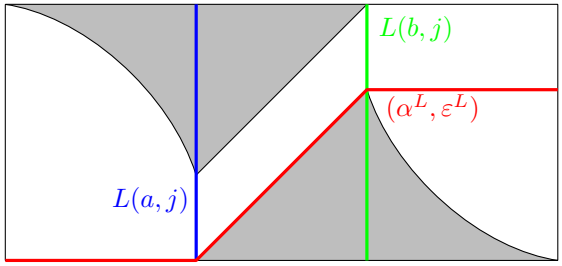
(a) Two unimodal functions of ε against α, where the intersection is the minimum to handle the VVE-event.



(b) The FSD of the Figure a.



(c) Two unimodal functions with of ε against α, where the minimum $(\alpha^L, \varepsilon^L)$ lies below $L(a,j)(\alpha)$.



(d) The FSD of the Figure c.

Figure 3.1: Finding the minimum $(\alpha^L, \varepsilon^L)$ of two unimodal functions where we insert $L(a,j)(\alpha)$ first and later $L(b,j)(\alpha)$. The red minimum $(\alpha^L, \varepsilon^L)$ differs on the insertion order. Note that we draw the unimodal functions for simplicity as parabolas as function of α.

Lemma 8. *The minimum $C(i+1,j).(\alpha^L, \varepsilon^L)$ lies on the upper envelope created by the decreasing part of $L(k,j)(\alpha)$ for all $k \in \{h+1, \ldots, i\}$, the horizontal lines $C(i,j).\varepsilon$ and $C(h,j).\varepsilon^B$ and the entire unimodal function $L(i+1,j)(\alpha)$, where h is the column where we enter row j.*

Proof. Let us prove this by contradiction, then either $(\alpha^L, \varepsilon^L)$ lies below or above the upper envelope.

If $(\alpha^L, \varepsilon^L)$ lies below the upper envelope, then there exists a line that is not satisfied. If this line is the horizontal line $C(i,j).\varepsilon$, then this is a contradiction of Lemma 4. If $(\alpha^L, \varepsilon^L)$ lies below $C(h,j).\varepsilon^B$, then we cannot enter row j at column h which is a contradiction to our assumption that we enter row j in $C(h,j)$. If $(\alpha^L, \varepsilon^L)$ lies below the decreasing part of $L(k,j)(\alpha)$ for some $k \in \{h+1, \ldots, i\}$, we require a passthrough over $L(k,j)$ at most height α^L, which is not satisfied for this $L(k,j)(\alpha)$. If the unimodal function $L(i+1,j)(\alpha)$ is not satisfied, then this is either due to a too low value for ε^L and passing through the unimodal function is not possible, or α^L is too high and we have a decreasing path before entering $C(i+1,j)$.

If $(\alpha^L, \varepsilon^L)$ lies above the upper envelope, then we argue that we can find a better $(\alpha^*, \varepsilon^*)$ such that there still exists a valid path, however for $\varepsilon^* < \varepsilon^L$ or $\alpha^* < \alpha^L$ with $\varepsilon^* = \varepsilon^L$, i.e. a candidate $(\alpha^*, \varepsilon^*)$ that lies on the upper envelope but has a lower ε or α.

If $\alpha^L = 0$ and $(\alpha^L, \varepsilon^L)$ lies above the left upper envelope, then we decrease ε^L to ε^* such that $(0, \varepsilon^*)$ is the point on the upper envelope. Since $(0, \varepsilon^*)$ lies on the upper envelope, for every cell edge $L(k,j)$ with $k \in \{h+1, \ldots, i+1\}$ there is a passthrough at $\alpha = 0$ for ε^*. By

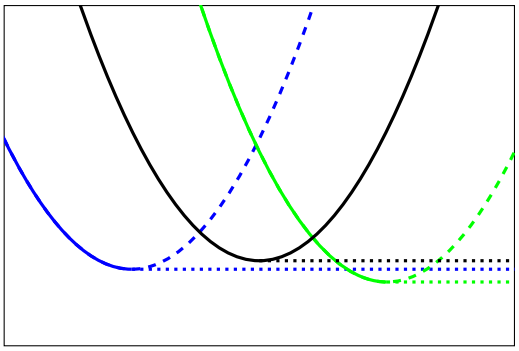


Figure 3.2: An upper envelope of the decreasing part of preceding unimodal functions (blue and green) and one new full unimodal function (black). The dotted lines are an implicit lower bound we get from Lemma 7.

this there exists a bimonotonic increasing path from $(0,0)$ via $C(h,j)$ to $C(i+1,j)$ at $\alpha = 0$ for ε^* which implies that $(\alpha^L, \varepsilon^L)$ was not the minimum.

Assume that $\alpha^L > 0$ and $(\alpha^L, \varepsilon^L)$ lies above the upper envelope and moreover let $(\alpha^*, \varepsilon^L)$ be the point on the upper envelope such that ε^L is equal, but where $\alpha^* < \alpha^L$, i.e. we shift $(\alpha^L, \varepsilon^L)$ to the left until we hit the upper envelope or $\alpha^* = 0$. If $\alpha^* = 0$, then we argue that there exist a point $(0, \varepsilon^*)$ which is a better candidate as shown before. Hence we have that $(\alpha^*, \varepsilon^L)$ with $\alpha^* > 0$ is on the upper envelope and prove that this is a better point than $(\alpha^L, \varepsilon^L)$ since this has a lower α^L. With the property that $C(i,j).\varepsilon = \min(C(i,j).\varepsilon^L, C(i,j).\varepsilon^B)$, by Lemma 7 we have that $C(i,j).\varepsilon$ is equal to or larger than $C(k,j).\varepsilon$ for all $k \in \{h+1, \ldots, i\}$ where h is the cell where we come from below for calculating $C(i+1,j).\varepsilon^L$, i.e. for $C(i+1,j).\varepsilon^L$ there exists a valid path from $(0,0)$ to the free space of all cells $C(k,j)$. If the horizontal lines $C(i,j).\varepsilon$ and $C(h,j).\varepsilon^B$ are satisfied by $(\alpha^L, \varepsilon^L)$, then these are also satisfied by $(\alpha^*, \varepsilon^L)$ and all cells $C(k,j)$ are still accessible. Hence, the point $(\alpha^*, \varepsilon^L)$ lies on the decreasing part of a unimodal function $L(k,j)(\alpha)$ for at least one $k \in \{h+1, \ldots, i\}$ or the entire unimodal function $L(i+1,j)(\alpha)$. In the latter case, if $(\alpha^*, \varepsilon^L)$ lies on $L(i+1,j)(\alpha)$, then it lies on the decreasing part of $L(i+1,j)(\alpha)$ and is similar to the first case. Hence, $(\alpha^*, \varepsilon^L)$ lies on one or more $L(k,j)(\alpha)$ for $k \in \{h+1, \ldots, i+1\}$. Now since $(\alpha^L, \varepsilon^L)$ is already a valid point for $C(i+1,j)$ (but not the minimum) we decrease α^L to α^*. Now let $(\alpha^L, \varepsilon^L)$ be a valid point, but $(\alpha^*, \varepsilon^L)$ invalid. For the first there exists a bimonotonic increasing path, hence for second there must exists a path which must strictly decreasing. By that, there is a unimodal function $L(k,j)(\alpha)$ with $k \in \{h+1, \ldots, i+1\}$ such that this is not satisfied. However, since $(\alpha^*, \varepsilon^L)$ lies on the specified upper envelope it is satisfied by all unimodal functions and this case cannot occur.

As a result, $(\alpha^L, \varepsilon^L)$ cannot lie below or above the upper envelope and hence must be on the upper envelope. □

Lemma 9. *The minimum $(\alpha^L, \varepsilon^L)$ for $C(i+1,j)$ is the leftmost lowest point $(\alpha^L, \varepsilon^L)$ on the upper envelope created by the entire unimodal function $L(i+1,j)(\alpha)$, the horizontal lines $C(i,j).\varepsilon$ and $C(h,j).\varepsilon^B$ and all decreasing parts of the unimodal functions $L(k,j)(\alpha)$ for $k \in \{h+1, \ldots, i\}$, where h is the column in which we enter row j.*

Proof. The minimum $(\alpha^L, \varepsilon^L)$ for $C(i+1,j)$ lies on the upper envelope as described by Lemma 8. Since we want to minimize ε^L and if possible α^L, the minimum $(\alpha^L, \varepsilon^L)$ is the

leftmost lowest point on this upper envelope. □

By Lemma 9 we know how to pick the optimal $C(i+1,j).(\alpha^L,\varepsilon^L)$ if we have the optimal value for h. By Lemma 5 we know which value of h we need to choose if we know $C(i+1,j).(\alpha^L,\varepsilon^L)$. Let us have a candidate column h^* where we enter row j from below where $h^* \leq h$, which by Lemma 6 may be the column where we enter row j for calculating $C(i,j).(\alpha^L,\varepsilon^L)$. We construct the upper envelope similar to Lemma 9, however where we enter row j in column h^*, i.e. we may have more unimodal functions in the upper envelope than necessary. If $h^* = h$, then the minimum point on this upper envelope is $C(i+1,j).(\alpha^L,\varepsilon^L)$. However if $h^* < h$, then we try a better candidate for h^* while maintaining that we do not overestimate $C(i+1,j).\varepsilon^L$ and get a contradiction to Lemma 5.

Lemma 10. *Given that we come from below in column h for calculating $C(i+1,j).(\alpha^L,\varepsilon^L)$, then if we construct the upper envelope UE^* similar to Lemma 9 for unimodal functions $L(h^*+1,j)(\alpha)$ to $L(i+1,j)(\alpha)$ with $h^* \leq h$, we either get $h^* = h$ and the minimum point on the upper envelope for h^* is the minimum $C(i+1,j).(\alpha^L,\varepsilon^L)$, or we have $h \in \{h^*+1,\ldots,i\}$ where $C(h,j).\varepsilon^B$ is less than or equal to the minimum of the upper envelope.*

Proof. If $h^* = h$, then by Lemma 9 we get that $C(i+1,j).(\alpha^L,\varepsilon^L)$ is the minimum of the upper envelope. If $h^* < h$, then by Lemma 5 there exists an $h \in \{h^*+1,\ldots,i\}$ for which $C(h,j).\varepsilon^B \leq C(h^*,j).\varepsilon^B$ or $C(h,j).\varepsilon^B < C(i+1,j).\varepsilon^L$ must hold. Increasing h^* with respect to Lemma 5 will find a better candidate than h^* if $h^* \neq h$. □

Using this lemma we can choose an h^* as a candidate value for h. We enforce that we never over-estimate h^* such that $h^* > h$. A candidate for h^* is by Lemma 6 the column where we come from below while processing $C(i,j).\varepsilon^L$. By Lemma 10 we get that the minimum ε^L is at least as high as the minimum of this upper envelope or the minimum $C(k,j).\varepsilon^B$ for $k \in \{h^*+1,\ldots,i\}$. If both Lemma 9 and Lemma 5 hold for the certain h^*, we have $h^* = h$ and by Lemma 9 the best (α^L,ε^L). Hence, we use Lemma 10 iteratively with an initial candidate h^* and while it is less than h we keep finding a better h^* until $h^* = h$.

A small challenge which we can turn to our advantage is that to calculate $C(i+1,j).(\alpha,\varepsilon)$ we have to get the upper envelope. Since we have processed $C(i,j)$ before processing $C(i+1,j)$, we observe that the upper envelope we used for finding $C(i,j).(\alpha^L,\varepsilon^L)$ is no longer necessary. By Lemma 8 we have that the upper envelope we used to calculate $C(i,j).(\alpha^L,\varepsilon^L)$ consists of the unimodal functions $L(h+1,j)(\alpha)$ to $L(i-1,j)(\alpha)$ and the entire unimodal function $L(i,j)(\alpha)$ where we come from below in column h. Since we assume this h to be h^* for our next cell, we initially start with this upper envelope and can remove $L(k,j)(\alpha)$ if we discover that $k < h$ using Lemma 10.

For simplicity let us assume that we store the entire unimodal function $L(i+1,j)(\alpha)$ in the upper envelope. After calculating $C(i+1,j).(\alpha^L,\varepsilon^L)$, we remove the increasing part of the unimodal function, if possible. Using this method, finding the minimum (α^*,ε^*) of the upper envelope is only a minimum point query on the upper envelope as shown in Lemma 9. By Lemma 6 we do not need any unimodal function $L(h^*,j)(\alpha)$ with $h^* \leq h$ in the upper envelope. Hence we derive that to calculate $C(i+1,j).(\alpha^L,\varepsilon^L)$, we can reuse the upper envelope that we used for calculating $C(i,j).(\alpha^L,\varepsilon^L)$.

We can put the preceeding lemmas into the pseudo-code of Algorithm 2, where we initialize each column and row with an empty upper envelope $\mathcal{UE}$ and an empty list $\mathcal{L}$ of $C(i,j).\varepsilon^B$. Furthermore for every cell the ε must be at least the distance between π_0 and σ_0. We have

$\alpha^L = 0$ for the base cases and the the minimum cost to enter row j as ε initially, which is a lower bound. We again process all cells breadth-first similar to Algorithm 1of the VMFD, since we want all preceding cells processed before processing $C(i,j)$. For each cell we first insert the preceding cell in $\mathcal{L}$ and the new full unimodal function $L(i,j)(\alpha)$ in $\mathcal{UE}$. Note that if these do not exists for e.g. the leftmost column and lowest row, we treat them as if the costs to enter those cells are infinity, i.e. we always prefer a different path. After that, we iteratively use Lemma 10 until we have found the best candidate for $C(i,j).(\alpha^L, \varepsilon^L)$. For this we apply a minimum query on the upper envelope $\mathcal{UE}$ with the two lines $C(i-1,j).\varepsilon$ and $C(h,j).\varepsilon^B = \text{Row}[j].\text{Below}$. By Lemma 6 we may simply remove all cell edges $L(k,j)(\alpha)$ with $k \leq h$ from the upper envelope and $C(i,j).\varepsilon^B$ from the list. This way we process all cells and we extract the Fréchet distance from the last cell using the same approach as for the VMFD.

Algorithm 2 Algorithm for calculating the Fréchet distance.

FrechetDistance(π, σ)

```
1  ▷ π.length = n, σ.length = m
2  ε ← DIST(π(0), σ(0))
3  Column[1, ..., m] ← ⟨(α^L, ε^L) = (0, ε), Below = ε, UE = ∅, L = ∅⟩
4  Row[1, ..., n] ← ⟨(α^B, ε^B) = (0, ε), Below = ε, UE = ∅, L = ∅⟩
5  for (i, j) ← (1, 1) to (n, m) by breadth-first
6       do Row[j].L.ADD(C(i − 1, j))
7          Row[j].UE.ADD(L(i, j)(α))
8          h* ← max_h min{C(h, j).ε^B | C(h, j) ∈ Row[j].L}
9          while C(h*, j).ε^B ≤ Row[j].UE.MINIMUMQUERY(C(i − 1, j).ε, Row[j].Below)
10              do delete all L(x, j)(α) from Row[j].UE with x < h*
11                 delete all C(x, j) from Row[j].L with x ≤ h*
12                 Row[j].Below ← C(h*, j).ε^B
13                 h* ← max_h min{C(h, j).ε^B | C(h, j) ∈ Row[j].L}
14         Row[j].(α^L, ε^L) ← Row[j].UE.MINIMUMQUERY(C(i − 1, j).ε, Row[j].Below)
15         Repeat Line 6 to 14 for Column[i].
16         C(i, j).ε ← min(C(i, j).ε^L, C(i, j).ε^B)
17 return max(C(i, j).ε, DIST(π(n), σ(m)))
```

The asymptotic analysis of this algorithm depends on a set of subproblems. We get at least $\Omega(nm)$ since we have a for-loop over all cells, which we have $\Theta(nm)$. This is similar as we have for the VMFD. Insertion and deletion in $\mathcal{L}$ as in Line 6 and Line 10 take $O(\log m)$ time if this is implemented using a min-heap, but if we use a doubly linked list in which we store only $C(h+1,j)$ to $C(i,j)$ in strictly increasing order, we get amortized $O(1)$ bounds. We can do this according to the following lemma.

Lemma 11. *If we store row$[j].\mathcal{L}$ as a doubly linked list, where we only store all $C(a,j)$ in strictly increasing order, we can find the maximum h with minimum $C(a,j).\varepsilon^B$ in constant time, insert new cells in amortized constant time and delete the first or last element in constant time.*

Proof. We will use the induction hypothesis that the doubly linked list is already sorted, which is true for the empty doubly linked list as well as for containing a single element. We

observe that $C(a,j)$ will not be chosen as the maximum value for h^* if there exists a $C(b,j)$ in the list with $a < b$ and $C(b,j).\varepsilon^B \leq C(a,j).\varepsilon^B$.

If we want to have the highest h^* with $C(h^*,j).\varepsilon^B$ less than or equal to all other values of the list, we have this at the beginning of the doubly linked list if the list is non-empty, since the list is in strictly increasing order.

Let us insert $C(i,j)$ to the doubly linked list. If the last element $C(a,j)$ at the time of insertion in the list has $C(a,j).\varepsilon^B < C(i,j).\varepsilon^B$, we only append $C(i,j)$ and our induction hypothesis holds. If the current last element $C(a,j)$ of the list has $C(a,j).\varepsilon^B \geq C(i,j).\varepsilon^B$, we may delete $C(a,j)$ from the doubly linked list and repeat comparing $C(i,j)$ to the new end of the list. We always terminate this process, although possibly with the doubly linked list containing only the singleton $C(i,j)$.

We only delete cells from the upper envelope if we enter the row in that or a later cell, which we are allowed to do by Lemma 6. As a result, if we have decided to come from below in $C(h^*,j)$, $C(h^*,j).\varepsilon^B$ is less than or equal to $C(a,j)$ with $a < h^*$ and thereby no $C(a,j)$ is stored in the list.

Since we only insert once, we can only delete once. Every deletion is done at the beginning or end of the list, and takes constant time. Insertion is done by appending to the back of the list and deleting only preceding elements, which can only occur once and hence we get an amortized constant running time. The maximum h^* with minimum $C(a,j).\varepsilon^B$ is the first element in the list and can therefore be found in constant time. □

We need to find the distance between two points in $O(\text{DIST}(D,d))$ in Line 2 and 17, where we use D for the complexity of the distance metric and d for the number of dimensions. We furthermore initialize $\Theta(m)$ and $\Theta(n)$ datasets in Line 3 and 4. After that, we apply a for-loop in Line 5 over $\Theta(nm)$ cells.

For each cell, we insert in Line 6 $C(i-1,j)$ into the $\mathcal{L}$ in amortized constant time by Lemma 11 and insert $L(i,j)(\alpha)$ to $\mathcal{UE}$ in Line 7 in $O(\text{INSERT}(D,d,n))$. Note that we insert both elements exactly once. Querying the minimum of $\mathcal{L}$ can be done in constant time, and the while-loop of Line 9 is executed an amortized constant number of times, i.e. once per for-loop and once per deletion of a cell edge of the upper envelope, which is amortized constant. Finding the minimum of the upper envelope in Line 9 needs to be done in $O(\text{MINIMUMQUERY}(D,d,n))$, which we do an amortized constant number of times in the while-loop and once in Line 14. In the while-loop we delete both from the $\mathcal{L}$ and the $\mathcal{UE}$, which can either be done in $O(1)$ or $O(\text{DELETE}(D,d,m))$ respectively. We insert at most once and get an amortized constant number of deletions in the for-loop as well. For each cell, we need to do this columnwise as well as in Line 15. We can put this into the following theorem.

Theorem 2. *The running time of Algorithm 2 is* $O(T(\text{DIST})+nm\cdot(T(\text{INSERT})+T(\text{DELETE})+T(\text{MINIMUMQUERY})))$*, where the* $T()$ *may depend on* n *or* m*, number of dimensions* d *and the distance metric* D*.*

In practice $T(\text{DIST})$ is independent of n or m and hence near-constant. Insertion, deletion and minimum queries in the upper envelope are more interesting. We prove in the following chapters how we can find a suitable data structure for these operations, depending on our distance metric D.

A significant disadvantage to our novel algorithm is that we need the upper envelope $\mathcal{UE}$ and list $\mathcal{L}$ which both may store worst case all its preceding elements. Since we process all cells breadth-first, if we are at the diagonal from $C(1,n)$ to $C(n,1)$ for each cell, we may need

to store $O(n)$ elements per cell. This implies a worst case $\Omega(nm)$ storage and may even be worse for less efficient implementations of the upper envelopes we require.

4 Algorithm for the Euclidean distance

In the previous chapter we have discussed a new algorithm for calculating the Fréchet distance over two polygonal curves. In the algorithm we required an upper envelope of the decreasing part of unimodal functions. What such unimodal function actually looks like depends on the distance metric that is used. In this chapter, we assume that we use the Euclidean distance between two points and explain how our algorithm works. For that, we shall first discuss properties the unimodal functions for the Euclidean case and after that we will show how we can calculate an upper envelope of such unimodal functions.

The Euclidean distance L_2 between two points a and b can be represented as the length of a straight line between these points. For our algorithm we will use the squared Euclidean distance between two points a and b, rather than the Euclidean distance, which does not influence the result, however we can argue about parabolas rather than the squared root of parabolas.

Lemma 12. *Every edge of a cell in the grid, e.g. $L(i,j)$, can be represented as a parabola in α if we use the squared Euclidean distance in $\mathbb{R}^d$.*

Proof. We are interested in how $L(i,j)(\alpha)$ could be represented. Let us have the point π_i and line segment $l = \sigma_{j-1}\sigma_j$. The squared Euclidean distance $L(i,j)(\alpha)$ for $\alpha \in \mathbb{R}$ can then be represented as $L(i,j)(\alpha) = ||\pi(i) - (\sigma(j-1) + \alpha \cdot (\sigma_j - \sigma_{j-1})||^2$. We rewrite this and get

$$L(i,j)(\alpha) = ||l||\alpha^2 + (2 \cdot ||\pi(i) - \sigma(j-1)|| \cdot ||l||) \cdot \alpha + ||\pi(i) - \sigma(j-1)||^2.$$

This is a quadratic formula in α and therefore a parabola. □

We can calculate this parabola $L(i,j)(\alpha)$ in $O(d)$ time. From the proof of Lemma 12 we observe that the α^2-coefficient is constant for a given j, i.e. the length of the line segment $\sigma_{j-1}\sigma_j$. We can therefore treat all parabolas $L(a,j)(\alpha)$ and $L(b,j)(\alpha)$ as pseudolines if we compare these to each other.

Observation 1. *If we compare two parabolas $L(a,j)(\alpha)$ and $L(b,j)(\alpha)$ for the squared Euclidean distance, these parabolas have the same α^2-coefficient and intersect at most once.*

Dynamically maintaining the upper envelope of parabolas is not a simple task, however by Observation 1 we subtract the α^2-coefficient and get lines. For an upper envelope of lines, we can use e.g. the approach of Overmars and Van Leeuwen [1981] or Brodal and Jacob [2002]

to maintain the upper envelope. However we have three difficulties with this approach. The upper envelope used to calculate $C(i+1,j).(\alpha^L,\varepsilon^L)$ using Algorithm 2 consists of decreasing part of the parabolas $L(h^*+1,j)(\alpha)$ to $L(i,j)(\alpha)$, one entire parabola $L(i+1,j)(\alpha)$ and two horizontal lines $C(i,j).\varepsilon$ and $C(h^*,j).\varepsilon^B$. By subtracting the α^2-coefficient we can treat the parabolas as lines, however this makes the two horizontal lines now parabolas, which is the first challenge. The second challenge is that finding the minimum is no longer a straightforward query. The third challenge is that although we can treat all parabolas as lines, we only want the decreasing part. In this chapter we will show how to overcome these challenges.

Due to the complexity of the dynamic convex hulls proposed by Chan [2001] and Brodal and Jacob [2002], we will use the approach of Overmars and Van Leeuwen [1981] for the upper envelope, this data structure is more suitable to adapt for solving our three challenges. The main advantage is that the upper envelope is stored in a balanced binary search tree with height $O(\log n)$.

Let us assume that we only have the decreasing parts of the parabolas $L(k,j)(\alpha)$ for $k \in \{h^*+1,\ldots,i\}$ stored in the upper envelope $\mathcal{UE}$ and $L(i+1,j)(\alpha)$ entirely. Then either the minimum (α^L,ε^L) is the minimum point of this upper envelope, or the minimum (α^L,ε^L) lies on either $C(i,j).\varepsilon$ or $C(h^*,j).\varepsilon^B$. We can split this decision into two parts. First we find the minimum (α^*,ε^*) of this upper envelope without the horizontal lines and if and only if this minimum lies on or above both $C(i,j).\varepsilon$ and $C(h^*,j).\varepsilon^B$ we have $(\alpha^*,\varepsilon^*) = (\alpha^L,\varepsilon^L)$. Otherwise, (α^L,ε^L) lies on the maximum of $C(i,j).\varepsilon$ and $C(h^*,j).\varepsilon^B$ and this line's intersection to the upper envelope (with possibly $\alpha^L = 0$).

With this information we can ignore the horizontal lines $C(i,j).\varepsilon^L$ and $C(h^*,j).\varepsilon^B$ initially. However, we cannot simply ignore the increasing part of the parabolas. We subtract the α^2 coefficient and insert all the resulting lines in the upper envelope. If we search for the minimum (α^*,ε^*), we either find that this is the point we are looking for, or we can ignore one of the two parabolas on which this intersection lies as we will prove in Lemma 13. We define the set S as the set of full parabolas which are actually stored in the upper envelope. This is by Lemma 10 a subset of $\{h^*+1,\ldots,i+1\}$. We prove that we can reduce the size of S as long as we have not found the minimum (α^*,ε^*), however while maintaining that we do not decrease S too much, i.e. we only delete parabolas which will never be on the upper envelope again for cells further to the right, to maintain that we can reuse the upper envelope.

Since we only have the decreasing parts of the parabolas $L(h^*,j)$ to $L(i,j)$ we might have that the upper envelope is not continuous, however we are allowed to ignore this. This can be explained using Figure 3.2. Although we are only interested in the decreasing part, by Lemma 7 we have that the minimum of every parabola $L(k,j)(\alpha)$ for $k \in \{h^*+1,\ldots,i\}$ must be satisfied, as well as the minimum of $L(i+1,j)(\alpha)$ by Lemma 4. Hence, the minimum of the upper envelope including the two horizontal lines is always at least the maximum of all minima of $L(k,j)(\alpha)$ for $k \in \{h^*+1,\ldots,i+1\}$. Hence, this discontinuous part lies on or below the horizontal line $C(i,j).\varepsilon$. Therefore the discontinuous part is irrelevant to us for finding the minimum of the total upper envelope if we include the horizontal line $C(i,j).\varepsilon$.

Lemma 13. *For the minimum (α^*,ε^*) of the upper envelope of full parabolas $L(k,j)(\alpha)$ for $k \in S$ and $S \subseteq \{h^*+1,\ldots,i+1\}$ is:*

1. *the minimum (α^*,ε^*) of the decreasing part of the parabolas $L(k,j)(\alpha)$ for $k \in \{h^*+1,\ldots,i\}$ and full parabola $L(i+1,j)(\alpha)$, or*
2. *(α^*,ε^*) lies on two parabolas $L(a,j)(\alpha)$ and $L(b,j)(\alpha)$, of which we can remove at least*

$L(a,j)(\alpha)$ *or* $L(b,j)(\alpha)$ *from* S*, or*

3. $\varepsilon^* \leq C(i,j).\varepsilon^L$,

Proof. We assume that for a given set S we have the minimum $(\alpha^*, \varepsilon^*)$ of this set of full parabolas. If this point lies on $L(i,j)(\alpha)$, then it must lie on $L(i,j)(\alpha)$'s decreasing part, minimum point or increasing part. If it is the minimum, then this minimum of $L(i,j)(\alpha)$ is indeed the minimum $(\alpha^*, \varepsilon^*)$ of the upper envelope. Similarly, if $(\alpha^*, \varepsilon^*)$ is the minimum of another parabola $L(k,j)(\alpha)$, then this minimum is still the minimum of the upper envelope.

If the minimum $(\alpha^*, \varepsilon^*)$ is not the minimum of any $L(k,j)$ with $k \in S$, then $(\alpha^*, \varepsilon^*)$ must lie on the intersection between two (or more) parabolas. Let this be intersection be between $L(a,j)(\alpha)$ and $L(b,j)(\alpha)$ with $a, b \in S$ and $a < b$. If more than two lines intersect on $(\alpha^*, \varepsilon^*)$, we pick those lines with the largest increase and/or decrease. We have the following case distinction.

1. Increasing on $L(a,j)(\alpha)$ and $b = i+1$: We remove a from S, since $L(a,j)(\alpha)$ lies entirely below $L(i+1,j)$.

2. Decreasing on $L(a,j)(\alpha)$ and $b = i+1$: The intersection is $(\alpha^*, \varepsilon^*)$, which corresponds to a VVE-event between $L(a,j)(\alpha)$ and $L(i+1,j)(\alpha)$ in the RFSD.

3. Decreasing on $L(a,j)(\alpha)$ and decreasing on $L(b,j)(\alpha)$: An increase of α will cause the decrease of ε, hence $(\alpha^*, \varepsilon^*)$ cannot be the minimum.

4. Increasing on $L(a,j)(\alpha)$ and increasing on $L(b,j)(\alpha)$: A decrease of α will cause the decrease of ε, hence $(\alpha^*, \varepsilon^*)$ cannot be the minimum.

5. Increasing on $L(a,j)(\alpha)$ and decreasing on $L(b,j)(\alpha)$: By Observation 1 we have that the decreasing part of $L(a,j)(\alpha)$ lies entirely below $L(b,j)(\alpha)$ and hence $L(a,j+1)(\alpha)$ may be deleted from S.

6. Decreasing on $L(a,j)(\alpha)$ and increasing on $L(b,j)(\alpha)$: This point was required to be satisfied for $C(b,j).\varepsilon^L$. Since $a \in S$, $b \in S$ and $a < b$, we get $\varepsilon^* = C(b,j).\varepsilon^L \leq C(i,j).\varepsilon^L$ by Lemma 7. Hence, this point lies either on or below $C(i,j).\varepsilon^L$. Returning this point is valid.

As a result, we always reduce S or give a point which is either the minimum of the envelope or the minimum of the upper envelope is less than or equal to $C(i,j).\varepsilon^L$. □

Although Lemma 13 is now proven for the Euclidean distance function, this lemma can be used for other distance functions as well which behave as pseudo lines. We will however not go into detail on different distance metrics which may use this same approach.

With Lemma 13 we can start with an initially empty set S of full parabolas in the upper envelope. With Algorithm 2 we insert every parabola exactly once. Every time we query the minimum of the upper envelope (without horizontal lines) we may optimize the set of parabolas in S and hence in the upper envelope, however while maintaining that the $\mathcal{UE}$ we used for calculating $C(i,j)$ can be given to $C(i+1,j)$ without invalidating the set of parabolas according to Lemma 13. More precisely, if after processing $C(i,j)$ we have set S', then the initial set S for processing $C(i+1,j)$ is $S' \cup \{L(i+1,j)(\alpha)\} = S$. Intuitively this implies that if we delete parabola $L(a,j)(\alpha)$ from S for $C(i+1,j).\varepsilon$, for cells further right $L(a,j)(\alpha)$ will not

be on the upper envelope either. Since we only remove a parabola from S in case 1 and case 5 of the proof of Lemma 13, there exists an $L(b,j)(\alpha)$ which is entirely above $L(a,j)(\alpha)$ for the relevant part of $L(a,j)(\alpha)$. If we delete $L(b,j)(\alpha)$, this is due to entering row j in column $h^* \geq b$ and $L(a,j)(\alpha)$ is not in the set of decreasing part of unimodal functions anymore either, or $L(b,j)(\alpha)$ is strictly below $L(c,j)(\alpha)$ for $b < c$ and hence $L(a,j)(\alpha)$ is strictly below $L(c,j)(\alpha)$ as well by transitivity. Therefore the resulting set S of $C(i,j)$ of full parabolas in the upper envelope $\mathcal{UE}$ can still be given as base case for Lemma 13 for $C(i+1,j)$.

By Lemma 13 we have found $(\alpha^*, \varepsilon^*)$, which is not necessarily the minimum of the upper envelope that includes the two horizontal lines. Let us take the maximum of these lines. If $(\alpha^*, \varepsilon^*)$ is below this line, then the (left) intersection between the maximum horizontal line and $\mathcal{UE}$ on S is the result of the minimum query. If $(\alpha^*, \varepsilon^*)$ is above both horizontal lines, then indeed $(\alpha^*, \varepsilon^*)$ is the minimum of $\mathcal{UE}$ including the two horizontal lines by Lemma 13.

Finding the minimum $(\alpha^*, \varepsilon^*)$ on the upper envelope of full parabolas can be done by subtracting the α^2-coefficient and store the resulting lines in the upper envelope as proposed by Overmars and Van Leeuwen [1981]. Since we know that the minimum $(\alpha^*, \varepsilon^*)$ lies on the upper envelope of full parabolas, we can do a binary search on the upper envelope and either find the left and right parabola which intersect at $(\alpha^*, \varepsilon^*)$, or the single parabola which contains the minimum $(\alpha^*, \varepsilon^*)$. This takes $O(\log n)$, the height of the binary search tree containing the upper envelope.

Checking whether $(\alpha^*, \varepsilon^*)$ lies below the horizontal lines $C(i,j).\varepsilon$ and $C(h^*,j).\varepsilon$ can be done in constant time. If it lies below these lines, we can find with a binary search the intersection of the horizontal line with the upper envelope, which again takes $O(\log n)$ time on the binary search tree.

As a result, with Lemma 13 we can find in amortized $O(\log n)$ time the minimum point on the upper envelope $\mathcal{UE}$ with amortized $O(1)$ deletions. For insertions in the upper envelope we first need to calculate the parabola which takes $O(d)$ time. After that, insertion and deletion in this upper envelope can be done in $O(\log^2 n)$ time. Since we insert only once in the upper envelope, we charge the deletions we do in Lemma 13 to the insertion time, and hence get a query time for the minimum point in amortized $O(\log n)$ time and an insertion and deletion in $O(\log^2 n)$ time. We insert this in Theorem 2 and get the following running time.

Theorem 3. *Algorithm 2 calculates the Fréchet distance under the Euclidean distance in $\mathbb{R}^d$ in $O(nm(d + \log^2(nm)))$ time.*

Note that this is a log-factor slower than the algorithm by Alt and Godau [1995], which has a running time of $O(nm \log(nm))$ for fixed dimensions d. It is even slower compared to the algorithm of Buchin et al. [2012a] with the with the running time of $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ for $d = 2$ and $n = m$. Another disadvantage is that the dynamic convex hull of Overmars and Van Leeuwen [1981] may take $O(n \log n)$ space and hence we may require up to $O(nm \log(nm))$ space. However, the implementation of our novel algorithm is more straightforward. Both Algorithm 2 and maintaining the upper enelope using Overmars and Van Leeuwen are not very complex. Although our query on the upper envelope is not a default query for convex hulls, it can be done with a binary search. Furthermore, the hidden constants of the big-O-notation are relatively low compared to the approach of Buchin et al. or the parametric search of Megiddo [1983] as proposed by Alt and Godau.

Note that using the dynamic upper envelope with the dynamic convex hull of Overmars and Van Leeuwen, we may restrict ourselves more than necessary. We do not need every query, only a binary search on the resulting upper envelope is sufficient. Furthermore, although we

may delete in arbitrary order, we insert all lines in the upper envelope in order of occurance in the grid and hence insert all points in the dual plane in the convex hull in sorted order. Although sorting takes $O(n \log n)$ time for each row, using the zone theorem as discussed by De Berg et al. [2008, Theorem 8.6], we can insert all lines in an arrangement of lines in $O(n^2)$. If we traverse over each line, we get the ordering in which we insert all lines in the upper envelope in $O(n^2 + m^2)$ time. Whether this pre-processing leads to better running times is worth pursuing.

Although we have presented Algorithm 2 for the Euclidean case with a running time of $O(nm(d + \log^2(nm)))$ by Theorem 3, we can use other distance metrics as well. In the next chapter we discuss polygonal distance functions for calculating the Fréchet distance, in which we can avoid the $\log^2$ factor.

5 Algorithm for polyhedral distance functions

In Chapter 4 we discussed how our novel algorithm for calculating the Fréchet distance works for two polygonal curves using the Euclidean distance metric. For that we explained an algorithm using upper envelopes in Chapter 3. Although the Euclidean distance is an intuitive measure for the distance between two points, we can use other distance metrics as well. In this chapter we discuss a number of other distance metrics for which the algorithm perform faster. We start with the algorithm for the L_1 distance and after that we explain the algorithm for the L_∞ distance. These can be calculated in a similar way. The L_1 and L_∞ distance are examples of polyhedral distance functions and our algorithm generally works faster for polyhedral distance functions than for the Euclidean distance. Therefore, we also explain how to use polyhedral distance functions which can be used to obtain an approximation algorithm for the Fréchet distance under the Euclidean distance. Finally, we include how we can use a combination of the decision problem in combination with the approximation algorithm we propose and get an even better running time.

5.1 Algorithm for L_1

The distance function L_1 is also known as the Manhattan distance function. This name is based on that the distance between two points is only achieved if you can go over axis-aligned paths — like the streets in Manhattan — rather than the straight line as in the Euclidean distance. Mathematically this distance function between two points a and b is defined as $L_1(a, b) = |a.x - b.x| + |a.y - b.y|$. It gives a diamond shape as unit sphere in 2-dimensions as shown in Figure 2.2b in Chapter 2. In higher dimensions, this becomes a cross-polytope. Note that this distance function is convex, hence the free space of all cells in the FSD are convex and thereby functions representing all cell edges $L(i, j)$ and $B(i, j)$ are unimodal functions.

To calculate the Fréchet distance for L_1 between two polygonal curves π and σ we can use Algorithm 2. We iterate over all cells and maintain the minimum ε to enter the free space of each cell. The main difference between the Euclidean distance discussed in Chapter 4 is that the unimodal functions $L(i, j)(\alpha)$ and $B(i, j)(\alpha)$ have different properties.

Lemma 14. *Every edge of a cell in the grid, e.g. $L(i, j)(\alpha)$, can be represented as an upper envelope of at most $d + 1$ lines if we use the L_1 distance in $\mathbb{R}^d$.*

Proof. We want to derive the behaviour of $L(i,j)(\alpha)$ in d dimensions. We consider the point $p = \pi_i$ and the line segment $l = \sigma_{j-1}\sigma_j$. We apply an affine transformation of the line segment to a line, i.e. we have $l\colon [0,1] \to \mathbb{R}^d$ and extend this to $l'\colon \mathbb{R} \to \mathbb{R}^d$.

We take the mapping $f\colon \mathbb{R} \to \mathbb{R}$ defined by $f(\alpha) = \sum_{k=1}^{d} |p.x_k - l'(\alpha).x_k|$, i.e. $f(\alpha)$ is the L_1 distance between $l'(\alpha)$ and p. We have that for each dimension k there exists an α such that $|p.x_k - l'(\alpha).x_k| = 0$ and we denote this value as α_k, or $l'(\alpha).x_k = 0$. If α_k exists, then for $\alpha < \alpha_k$ the distance in this dimension k is decreasing with $p.x_k - l'(\alpha).x_k$, and for $\alpha > \alpha_k$ it is increasing with $p.x_k - l'(\alpha).x_k$, i.e. in both cases linear. If $l'(\alpha).x_k = 0$, the slope for dimension k is constant.

Let us sort all these α_k, then between any two consecutive values all distances in the different dimensions remain either in increasing or decreasing order and at α_k the function is continuous. Since $f(\alpha)$ is the sum of all these linear functions, $f(\alpha)$ consists of at most $d+1$ different linear parts. If we take $\alpha \in [0,1]$, we have that we $f(\alpha)$ still has at most $d+1$ different parts, which is the unimodal function $L(i,j)(\alpha)$. □

As an upper envelope of lines we have that $L(i,j)(\alpha)$ (and $B(i,j)(\alpha)$) are unimodal functions. By Lemma 14 we need to insert $O(d)$ lines in the upper envelope of Algorithm 2. The slope of each line can be calculated in $O(d)$. However, we may have $O(nmd)$ lines in the upper envelope if, basically, we insert every $L(i,j)$, which would cause an insertion and deletion time of $O(d(\log^2(nmd) + d))$ using the dynamic convex hull of Overmars and Van Leeuwen [1981]. Where for the Euclidean case we could not use the dynamic convex hull of Brodal and Jacob [2002] since the upper envelope was on a combination of parabolas and lines, we now can use their dynamic convex hull since we only have lines. We simply insert all lines of Lemma 14 — both increasing and decreasing — and the two horizontal lines $C(i,j).\varepsilon$ and $C(h,j).\varepsilon^B$. After extracting minimum $(\alpha^L, \varepsilon^L)$, we delete the strictly increasing lines. For this, we insert $O(d)$ lines and delete $O(d)$ lines after calculating $C(i+1,j).(\alpha^L, \varepsilon^L)$. Finding the minimum is an extreme-point query which takes $O(\log(nmd))$ time. As a result, we get an $O(nmd(\log(nmd) + d))$ running time using Brodal and Jacob's dynamic convex hull. Although this is indeed faster than the Euclidean case if $d = o(\log(nm))$, we can get a better performance if d is not arbitrarily large. We prove that although we have at most $d+1$ lines for each $L(i,j)$, these $d+1$ are parallel to a (sub)set of 2^d lines.

Lemma 15. *Given the upper envelope representing $L(i,j)$ in L_1, all $d+1$ lines are parallel to a subset of 2^d lines for fixed j.*

Proof. Given any point p, we get that the slope is constant in all directions, i.e. $|p.x_k - l'(\alpha).x_k|$. Since p has a fixed x_k, the slope only depends on $l'(\alpha).x_k$.

Given an unknown p, there exists $d!$ different sorted orderings for α_k. However we are only interested for a certain α in which of the k dimensions the distance $|p.x_k - l'(\alpha).x_k|$ is increasing, the ordering of these is irrelevant. As a result, given any value of α, the distance in all d dimensions is either increasing or decreasing and hence the slope is constant. We get that there exists 2^d different orderings of α_k. Let us for each 2^d orderings have a line with this exact slope. Then, for a fixed line segment, e.g. $L(i,j)$ with fixed j, all lines of the upper envelope must be parallel to any of the 2^d slopes. □

Simply inserting all line segments of $L(i,j)$ in the upper envelope would give an $O(\log(nmd))$ insertion and deletion time to the upper envelope. By Lemma 15 however we have that there exists only 2^d different lines and all other lines are parallel. Since we want an upper envelope

of all these lines, we only need to store the highest line. Let us for each of the 2^d possibly available lines create a doubly linked list named maxlist $\mathcal{M}$, which we can use similar as the data structure for Row$[j].\mathcal{L}$ and maintain it in amortized $O(1)$ time using Lemma 11. Where in Row$[j].\mathcal{L}$ we are interested in finding the minimum point on the list and store everything in strictly increasing order, let us for each $\mathcal{M}$ store everything in strictly decreasing order, which can be done analogously. Although we may have up to 2^d times a list $\mathcal{M}$, the upper envelope of a row consists of $O(2^d)$ lines.

For each new $L(i,j)(\alpha)$ we add if we process $C(i,j)$ we insert all $O(d)$ lines of $L(i,j)(\alpha)$ into their corresponding $\mathcal{M}$. Calculating to which $\mathcal{M}$ every line which must be added can be done in $O(d)$ time per line segment of $L(i,j)(\alpha)$, i.e. $O(d^2)$ for every cell edge. If an insertion or deletion in a $\mathcal{M}$ changes the maximum line, we replace that line in the upper envelope, which occurs amortized $O(1)$ times insertion for each direction d and hence amortized $O(d)$ times for each cell. As a result, we get an amortized $O(d^2 + d\log 2^d) = O(d^2)$ insertion and deletion time for maintaining the upper envelope using Brodal and Jacob [2002]. The disadvantage to this approach is however that we need to initialize $(n+m)2^d$ sets of $\mathcal{M}$. For each $\mathcal{M}$, we need to calculate its slope, which takes an additional $O(d)$ time for initialization.

As a result, we can choose either to use the same approach as the Euclidean distance, insert all $O(d)$ lines forming the upper envelope as in Lemma 14 and get an $O(d\log(nd))$ insertion and deletion time of $L(i,j)$ and a query time of $O(\log(nd))$, or we can use the approach where for each possible candidate for a slope we initialize maintain a $\mathcal{M}$ which gives us an amortized $O(d^2)$ insertion and deletion time, a query time of $O(d)$and an initialization time of $O((n+m)d2^d)$. For a fixed number of dimensions, the latter is arguably better. If we insert these running times in Theorem 2, we can get the following theorem.

Theorem 4. *We can calculate the Fréchet distance under the L_1 distance metric in $\mathbb{R}^d$ in $O(\min(nmd^2 + (n+m)d2^d, nmd(\log(nmd) + d)))$.*

5.2 Algorithm for L_∞

The distance function L_∞ is known as the maximum norm. The distance between two points can be defined as the maximum of the distances between the two points in any axis-aligned direction. Mathematically this can be defined as $L_\infty(a,b) = \max\{|a.x - b.x|, |a.y - b.y|\}$ and gives a square as a unit sphere in $\mathbb{R}^2$ as shown in Figure 2.2c in Chapter 2 and a hypercube in higher dimensions. The L_∞ distance function is a convex distance function, hence the free space of all cells in the FSD are convex and thereby functions representing all cell edges $L(i,j)(\alpha)$ and $B(i,j)(\alpha)$ are unimodal functions.

The Fréchet distance using L_∞ as distance metric between two polygonal curves π and σ can be calculated using Algorithm 2. Intuitively this is very similar to the L_1 distance as in Section 5.1, hence we shall use the same approach. Let us first prove that every unimodal function $L(i,j)(\alpha)$ and $B(i,j(\alpha))$ can again be defined as the upper envelope, although different than the L_1 distance.

Lemma 16. *Every edge of a cell in the grid, e.g. $L(i,j)$, can be represented as the upper envelope of at most $2d$ lines if we use the L_∞ distance in $\mathbb{R}^d$.*

Proof. We want to derive the behaviour of $L(i,j)(\alpha)$ in d dimensions. We consider the point $p = \pi_i$ and the line segment $l = \sigma_{j-1}\sigma_j$. We apply an affine transformation over the line segment to a line, i.e. we have $l\colon [0,1] \to \mathbb{R}^d$ and extend this to $l'\colon \mathbb{R} \to \mathbb{R}^d$.

We take the mapping $f\colon \mathbb{R} \to \mathbb{R}$ defined by $f(\alpha) = \max_{k\in\{1,\dots,d\}} |p.x_k - l'(\alpha).x_k|$, i.e. $f(\alpha)$ is the L_∞ distance between $l'(\alpha)$ and p. For each dimension k there exists either an α where $|p.x_k - l'(\alpha).x_k| = 0$ or $l'(\alpha).x_k$ is axis-aligned. Let us denote this as α_k if it exists. For $\alpha < \alpha_k$ this function is decreasing in with $p.x_k - l'(\alpha).x_k$, and for $\alpha > \alpha_k$ it is increasing with $p.x_k - l'(\alpha).x_k$, i.e. in both cases linear. If α_k does not exists, $|p.x_k - l'(\alpha).x_k|$ is a constant and hence also linear.

Since we are only interested in the maximum distance for $k \in \{1, \dots, d\}$, we take both the increasing and decreasing lines of k. As a result, $f(\alpha)$ consists of at most $2d$ lines for $\alpha \in \mathbb{R}$. If we take $\alpha \in [0, 1]$, we still have that $f(\alpha)$ has at most $2d$, different parts. □

Where for L_1 we could prove that the unimodal functions $L(i, j)(\alpha)$ could be represented as an upper envelope of at most $d + 1$ lines, we now have that it can be represented as an upper envelope of at most $2d$ lines. However, we argue that for a fixed row j, all possible $O(d)$ lines constructing the upper envelope of $L(i, j)(\alpha)$ are parallel.

Lemma 17. *Given the upper envelope representing $L(i, j)$ in L_∞, all possible $2d$ lines of the upper envelope are parallel for fixed j.*

Proof. Given any point p, we get that the slope is constant in all directions, i.e. $|p.x_k - l'(\alpha).x_k|$. Since p has a fixed x_k, the slope only depends on $l'(\alpha).x_k$. For each dimension k, the slope is constant. □

Again we can use the approach similar to the Euclidean distance where we simply insert all $O(d)$ lines, which would give an insertion and deletion time of $O(d \log(nmd))$, or the approach similar to L_1 with a set of doubly linked lists $\mathcal{M}$. Calculating the slope of each direction is easier for the L_∞ case and can be done in $O(d)$ time and initializing the set of $\mathcal{M}$ is also easier and can be done in $O((n + m)d)$ time. Since the upper envelope consists of at most $O(d)$ line segments, maintenance of the $\mathcal{M}$s can be done in amortized $O(d)$ time and maintenance of the upper envelope in $O(d \log d)$ time using the upper envelope of Brodal and Jacob [2002]. Querying can be done in the similar way as for the L_1 distance, i.e. it takes $O(\log d)$ time. The total running time of our algorithm by using this approach would be $O(nmd \log d + (n + m)d) = O(nmd \log d)$.

We can actually achieve better bounds by avoiding the dynamic upper envelope completely. If the set of $\mathcal{M}$s is sorted, we can compute in linear time the upper envelope containing the maximum of lines, by the point-line duality as explained in Section 2.1 and Graham scan. Pre-sorting every possible slope of $\mathcal{M}$ for each row and column takes $O((n+m)d \log d)$ time. The slope of each part of the unimodal function is the maximum difference in any dimension of which implies we have $2d$ lines as proven in Lemma 16. Hence, we can insert and delete in all the $\mathcal{M}$s in $O(d)$ time and find the minimum of the upper envelope in $O(d)$ time. If we insert the time for these operations on the upper envelope into Theorem 2, we get the following theorem for the L_∞ distance metric.

Theorem 5. *We can calculate the Fréchet distance under the L_∞ distance metric in $\mathbb{R}^d$ in $O(nmd + (n + m)d \log d)$ time.*

If we would calculate the L_∞ distance using the dynamic upper envelope of Brodal and Jacob [2002] we would get an $O(nmd \log d)$ time algorithm, which is asymptotically slower. Note that if we simply would insert all lines, similar to how we can solve it for the Euclidean

case, we get an $O(nmd \log(nmd))$ running time, which is again asymptotically slower. Furthermore we see that the L_∞ distance has a better performance than for the L_1 distance, which is caused by the smaller number of facets the L_∞ distance function has in $\mathbb{R}^d$. For the hypercube of L_∞ this is $2d$, for the cross-polytope of L_1 this is 2^d.

5.3 Algorithm for k-regular polygon in $\mathbb{R}^2$

In the previous sections we discussed two distance metrics L_1 and L_∞. Although these are fairly easy to calculate, especially in fixed dimensions, these are not very similar to the Euclidean distance. Next we consider distance metrics with other convex polygons as unit circles, specifically a k-regular polygon that has k sides of equal length, with $k \geq 3$. An example of an 8-regular polygon is shown in Figure 2.2b in Chapter 2. A 4-regular polygon may have the same unit sphere as the L_1 or L_∞ unit sphere (depending on the rotation) and for $k = \infty$ we get a circle as unit sphere, similar to the Euclidean distance. It can be observed that this k-regular polygon is convex, hence we may use our novel algorithm to calculate the Fréchet distance using the k-regular polygon. We shall use the notion L_{kp} for this distance function.

A k-regular polygon is constructed of k line segments which we denote by s_1 to s_k. To use this k-regular polygon as a distance function between two points a and b we need to increase the unit sphere around a such that it 'hits' b. The distance can be obtained by projecting b on to the unit sphere of a and query to which k line segment it is projected. Let this be s_i. We find the line s' such that $b \in s'$ and $s' || s_i$, i.e. s' and s_i are parallel. The minimum Euclidean distance between s' and a is the L_{kp} distance from a to b. Mathematically, we can define this as

$$L_{kp}(a, b) = \min_{i \in \{1,\ldots,k\}} \{\min L_2(a, s') \mid b \in s' \wedge s' || s_i\}.$$

Note that if we choose an arbitrary value for k, for example $k = 3$, we may get that $L_{kp}(a, b) \neq L_{kp}(b, a)$, which would be counter-intuitive. An example of this is in Figure 5.1. This is due to the fact that a k-regular polygon is not centrally symmetric. Let us project a to segment s_i of the unit sphere of b and project b to segment $t(j)$ of the unit sphere of a. If s_i and t_j are parallel, then the distance from b to a is equal to the distance from a to b. If we choose k to be a power of 2, then for each segment s_i there exists a parallel segment s_j on the exact opposite of the k-regular polygon. We therefore implicitly require that k is a power of 2, otherwise Lemma 2 does not apply.

Let us prove that for $\mathbb{R}^2$ the functions representing the cell edges $L(i, j)(\alpha)$ and $B(i, j)(\alpha)$ are an upper envelope of $O(k)$ line segments.

Lemma 18. *Every edge of a cell in the grid, e.g. $L(i, j)(\alpha)$, can be represented as the upper envelope in $\mathbb{R}^2$ of at most k lines if we used a k-regular polyhedral distance function as distance metric in $\mathbb{R}^2$.*

Proof. We want to derive the behaviour of $L(i, j)(\alpha)$ in using L_{kp} as distance function. We consider the point $p = \pi_i$ and the line segment $l = \sigma_{j-1}\sigma_j$. We apply an affine transformation over the line segment to a line, i.e. we have $l \colon [0, 1] \to \mathbb{R}^2$ and extend this to $l' \colon \mathbb{R} \to \mathbb{R}^2$. We furthermore assume that p is at the origin, otherwise we translate p to the origin and l' correspondingly, which does not affect the distance. We fix point p and make k rays starting

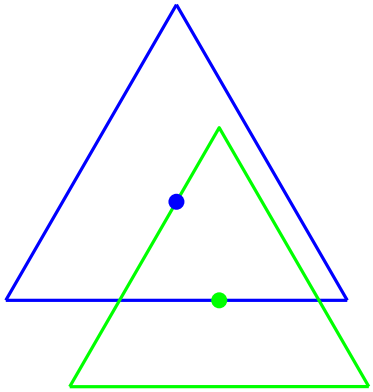


Figure 5.1: A counter-example that for $k = 3$ we have that $L_{kp}(a, b) \neq L_{kp}(b, a)$. The distance from the blue point to the green point is larger than the other way round.

at p which pass through the intersection of the line segments of the k-regular polyhedral unit sphere. We define these rays as r_a with $a \in \{1, \ldots, k\}$ and the intersection between r_a lies on s_a and s_{a+1} for $a \in [1, k-1]$ and r_a lies on s_k and s_1. We get that where s_a and s_{a+1} intersect, we have a passthrough of ray r_a.

We take the mapping $f\colon \mathbb{R} \to \mathbb{R}$ defined by $f(\alpha) = \min_{a\in\{1,\ldots,k\}}\{\min L_2(p, l') \mid l(\alpha) \in l' \wedge l' || s_a\}$, i.e. $f(\alpha)$ is the L_{kp} distance between $l'(\alpha)$ and p. Now let some point $l'(\alpha)$ be between rays r_a and r_{a+1}, projecting $l'(\alpha)$ to the unit sphere implies we project it to s_{a+1}.

For a line l', this line may intersect every ray r_a at most once (with $p \in l'$ as a special case). As a result, we may split l' into at most k segments which we define as l'_a with $a \in \{1, \ldots, k\}$. The distance from p to $l'_a(\alpha)$ can be represented as the minimum distance from p to $s_a(\alpha)$ such that $l'_a(\alpha) = s_a(\alpha)$. We rotate s_a around p such that s_a is parallel to the y-axis and right of p and rotate $l'_a(\alpha)$ correspondingly. This rotation does not change the L_{kp} distance, however the distance from p to $l'_a(\alpha)$ is now the difference in the x-direction. Since $l'_a(\alpha)$ is a straight line, the difference is a straight line as well. Let us denote this difference of distance as $d_a(\alpha)$.

We define m as the minimum distance between p to l', i.e. $m = \min_\alpha f(\alpha)$. Since a k-regular polygon is convex, by Lemma 3 such minimum exists. We have that this minimum distance is achieved by projecting a point to at most two different faces of the k-regular polygon or l' is (partially) parallel to a face of the k-regular polygon. We define α_m which is the α for which this minimum distance is achieved. For $\alpha' < \alpha_m$, we have that $l'(\alpha')$ is projected to face s_a for some $a \in \{1, \ldots, k\}$. For this, we have a strictly decreasing difference from p to $l'(\alpha')$, which is $d_a(\alpha')$ and therefore a line. Similarly we have this for $\alpha' > \alpha_m$ where the lines are increasing.

Let us take the upper envelope of all lines $d_a(\alpha)$ for $a \in \{1, \ldots, k\}$ if for some $\alpha \in \mathbb{R}$ $l'(\alpha)$ is projected to segment s_a. We show that $d_a(\alpha)$ is part of the upper envelope for α if and only if for that α l'_a is projected to s_a. Let us assume that this is not the case and there exists some b for which $d_b(\alpha)$ is higher on the upper envelope for the given α, i.e. $d_b(\alpha) > d_a(\alpha)$. We extend the unit sphere around p to $d_b(\alpha)$ and hence it intersects at $l'(\alpha)$. This implies $l'(\alpha)$ is projected to s_b, which is a contradiction to our assumption that $l'(\alpha)$ is projected to s_a.

Although we this now proven for $f(\alpha)$ with $\alpha \in \mathbb{R}$, if we take $\alpha \in [0, 1]$ we have a similar upper bound of at most k lines. □

Since we have a k-regular polygon, by Lemma 18 we have that every $L(i,j)(\alpha)$ can be represented as an upper envelope of k lines. Similar as to L_1 and L_∞, we prove that for a fixed row j all k lines of upper envelopes on that row are parallel.

Lemma 19. *Given the upper envelope representing $L(i,j)$ in L_{kp}, all possible k lines of the upper envelope are parallel for fixed j.*

Proof. Given any point p, we get that the slope of every line $d_i(\alpha)$ depends on the rotation of s_i with $i \in \{1,\ldots,k\}$ around p and the line segment $l'(\alpha)$ as proven in Lemma 18. Since p is constant, the distance only depends on $l'(\alpha)$ and hence on $l(\alpha)$. Since $l(\alpha)$ is a single line for fixed j, all possible k lines representing $L(i,j)$ have a constant slope. □

Given Lemma 18 and Lemma 19, we can construct the upper envelope for a row in the similar way as for L_∞. Rather than having d dimensions and hence $O(d)$ possible candidates for $\mathcal{M}$, we have k possible sets of $\mathcal{M}$ and the upper envelope of lines consists of at most $O(k)$ lines. Where for L_∞ we required that the sets of $\mathcal{M}$ are sorted, we can drop this requirement; given the k-regular polygon we have the faces consecutively and hence the lists $\mathcal{M}$ we assign to each face are consecutively stored. The initialization can be done in $O((n+m)k)$. Finding to which $\mathcal{M}$ the point $L(i,j)(0)$ needs to be stored can be found $O(k)$ time. All the following faces to which $L(i,j)$ can be projected are consecutive and therefore the insertion takes amortized $O(k)$ time in total. We insert the performance of these operations into Theorem 2 and derive the following running time, where we observe that the initialization can be computed in less than $o(nmk)$ time.

Theorem 6. *We can calculate the Fréchet distance using a k-regular polygon in $\mathbb{R}^2$ in $O(nmk)$ time for k a power of 2.*

5.4 Approximating the Euclidean distance function

In the previous section we have discussed the distance function for L_{kp}, which is a k-regular polygon in $\mathbb{R}^2$. We can prove that we can calculate a k-regular polygon distance function which approximates the Euclidean distance function by a certain error rate. We want to calculate the $(1+\varepsilon)$-approximation to the Fréchet distance between the two curves π and σ. Note that this ε is the approximation error used in approximation algorithms, rather than the Fréchet distance to enter the free space of a cell or a candidate for the Fréchet distance as used by Alt and Godau.

If we take $k = \infty$ we approximate the Fréchet distance to $\varepsilon = 0$, however this would increase the calculation time drastically using Theorem 6. Let us show how we can choose k based on ε. For simplicity we now assume that we calculate Fréchet distance in $\mathbb{R}^2$. We shall later extend this to higher dimensions. We first prove the approximation error for L_1 and L_∞ in $\mathbb{R}^2$.

Lemma 20. *The Fréchet distance calculated by the L_1 distance metric approximates the Fréchet distance of the Euclidean distance by $\varepsilon = \sqrt{2}-1$ in $\mathbb{R}^2$.*

Proof. The difference between the unit spheres of radius 1 of L_1 and L_2 is maximized at the four points where $|x| = |y|$. For L_2 the Euclidean distance to these points is 1 at $|x| = |y| = 1/\sqrt{2}$. If we take the L_1 distance on these points, i.e. $L_2(|x|,|y|) = |x| + |y| = \sqrt{2}$,

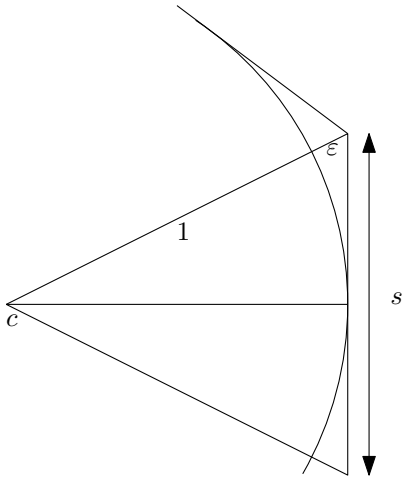


Figure 5.2: Given a fixed ε, we can calculate the size of s which is the length of a side of the k-regular polygon we can use for approximating the Euclidean distance.

we observe that we overestimate the Fréchet distance of the Euclidean distance with at most $\varepsilon = \sqrt{2} - 1$. □

Lemma 21. *The Fréchet distance calculated by the L_∞ distance metric approximates the Fréchet distance of the Euclidean distance by $\varepsilon = \sqrt{2} - 1$ in $\mathbb{R}^2$.*

Proof. The difference between the unit spheres of radius 1 of L_∞ and L_2 is maximized at the four points where $|x| = |y|$. For L_2 the Euclidean distance to these points is 1 at $|x| = |y| = 1/\sqrt{2}$. If we take the L_∞ distance on these points, i.e. $L_\infty(|x|, |y|) = \max\{|x|, |y|\} = 1/\sqrt{2}$, we may underestimate the Fréchet distance under the Euclidean distance by this factor. If we would divide the outcome of this metric by $\sqrt{2}$ we may overestimate the Fréchet distance by $\varepsilon = \sqrt{2} - 1$. □

If we have $\varepsilon \geq \sqrt{2}-1$, the L_1 and L_∞ distances would be a valid metric for approximating the Euclidean distance in $\mathbb{R}^2$. If we decrease our approximation error ε to less than $\sqrt{2} - 1$, L_1 and L_∞ are no longer useful. However we can construct an L_{kp} distance metric with k chosen such that the unit sphere of L_{kp} approximates the L_2 distance by ε.

We assume $0 < \varepsilon < \sqrt{2} - 1$. We calculate the value for k using Figure 5.2. Given ε as the approximation error, we have the length of a line segment of the k-regular polygon as $s = 2\sqrt{2\varepsilon + \varepsilon^2}$. Since we have an upper bound of ε, we reduce s to $2\sqrt{2\varepsilon} < s < 2$. For arbitrarily small ε we overestimate k by placing all segments s on the square of radius 1 rather than tight around the circle. We get the necessary k for the approximation k-regular graph as $k \leq 4/s < 4/(2\sqrt{2\varepsilon})$, hence $k = O(\sqrt{\varepsilon})$. For a given ε we can simply insert in our value for k in the k-regular polygon algorithm of Theorem 6, where we may need to increase k to the nearest power of 2, which only gives a constant overhead.

Theorem 7. *We can calculate a $(1 + \varepsilon)$-approximation of the Fréchet distance under the Euclidean distance using a k-regular polygon in $\mathbb{R}^2$ in $O(nm/\sqrt{\varepsilon})$ time.*

For the k-regular polygon we implicitly limit ourselves to $\mathbb{R}^2$. This is basically since for higher dimensions there do not exists arbitrarily many regular polytopes, e.g. for $\mathbb{R}^3$ there

exists only five regular polyhedrons, hence given an ε we do not necessarily have a k-regular polytope that allows us to approximates the Euclidean distance sufficiently. Luckily we do not need to find a polyhedron. We defined the unimodal function $L(i,j)(\alpha)$ as the distance between a line segment $l = \sigma_{j-1}\sigma_j$ and a point $p = \pi_i$. If we have that p lies on l or the extension of l, there exists infinitely many hyperplaces containing both p and l, which we select one. If this is not the case, then we have that there exists only one hyperplane containing both p and l. Given a hyperplane containin both p and l, we can use the k-regular polyhedron as distance function between p and l. Note that if we calculate the upper envelope of row j, which contains the unimodal function $L(h,j)(\alpha)$ with $h \neq i$, then we might have a different hyperplane containing $p' = \pi_h$ and l than for $p = \pi_i$ and l. However this does not influence the algorithm in any way, since we do not need to compare p and p' directly, only with respect to l.

Finding the hyperplane could be difficult, however let us calculate the (Euclidean) distance from π_i to both σ_{j-1} and σ_j, i.e. the endpoints of l, which can be done in $O(d)$ time. We draw l in $\mathbb{R}^2$ and given both distances from the endpoints we can find the location of p in $\mathbb{R}^2$ in constant time. As a result, for each cell edge we have an $O(d)$ additional computation time to calculate distance to the k-regular polygon. If we plug this into Theorem 7 we can get the following Theorem.

Theorem 8. *We can calculate a $(1+\varepsilon)$-approximation of the Fréchet distance under the Euclidean distance using a k-regular polygon in $\mathbb{R}^d$ in $O(nmd/\sqrt{\varepsilon})$.*

To some extent, our work is about avoiding to use the decision version of the Fréchet distance problem. But let us now conclude with a result that combines our algorithm with the decision version to improve upon Theorem 8. Consider an algorithm that – given two polygonal curves – first computes a $\sqrt{2}$-approximation of the Fréchet distance using Theorem 8. This computes a value δ^* such that the Fréchet distance δ_2 under the L_2 distance lies in the interval $[\delta^*/\sqrt{2}, \delta^*]$. On this interval we now perform a binary search (invoking the decision version in every step) with $\lceil \log \frac{\sqrt{2}-1}{\varepsilon} \rceil$ steps to find an interval $[\delta, \delta']$ of length at most $\delta^*\varepsilon$ that contains δ_2. Now $\delta_2 \leq \delta' \leq (1+\varepsilon)\delta \leq (1+\varepsilon)\delta_2$. This gives the following results.

Theorem 9. *We can calculate a $(1+\varepsilon)$-approximation of the Fréchet distance under the Euclidean distance in $O(nmd + T(m,n,d)\log(1/\varepsilon))$ time, where $T(m,n,d)$ is the time needed to solve the decision problem for the Fréchet distance for the Euclidean distance in $\mathbb{R}^d$.*

Solving the decision version takes $O(nmd)$ time [Alt and Godau, 1995], or for $d = 2$ and $n = m$ in $O(n^2(\log\log n)^{3/2}/\sqrt{\log n})$ time [Buchin et al., 2012a].

6 Conclusion

In this thesis propose a new algorithm for calculating the Fréchet distance. The well-known algorithm for calculating the Fréchet distance of Alt and Godau [1995] has a running time of $O(n^2 \log n)$ and only recently Buchin et al. [2012a] have been able to improve this to $O(n^2 \sqrt{\log n}(\log \log n)^{3/2})$. These algorithms, and all other known algorithms for the Fréchet distance, are based on solving the decision variant of the Fréchet distance efficiently. In our algorithm we approached the problem differently, which gives a new perspective on the problem of computing the Fréchet distance. Rather than using the Free Space Diagram (FSD) as a decision model, we use its general idea and traverse over a similar grid and by slowly increasing the size of this we calculate the Fréchet distance iteratively.

Our algorithm is defined independently of the distance metric we use. We give a solution for the Euclidean distance that calculates the Fréchet distance in $O(n^2 \log^2 n)$, which is slower than the approach of Alt and Godau. However the algorithm we propose has advantages over the parametric search approach; both the hidden constants in the big-O-notation will not be very large and the implementation will be easier. Another advantage is that we can use different distance metrics, e.g. a k-regular polygon. For a given k we can calculate the Fréchet distance in $O(n^2 k)$ in 2 dimensions. If we use a k-regular polygon which approximates the Euclidean distance by a certain error ε in 2 dimensions, we can approximate the Euclidean distance. We extend this to d dimensions and get a $(1+\varepsilon)$-approximation algorithm which runs in $O(n^2 d/\sqrt{\varepsilon})$ time. While the improvement in running time compared to the exact algorithm is rather modest, this is as far as we know the first approximation algorithm of the Fréchet distance that runs in quadratic time without making assumptions on the curves. Another strong property is that it has only a low dependency on d. If we combine our approach with the decision problem, we can approximate the Fréchet distance in $O(n^2 d \log(1/\varepsilon))$ time or even $O(n^2 + n^2(\log \log n)^{3/2} \log(1/\varepsilon)/\sqrt{\log n})$ time for $d = 2$.

We propose an algorithm that can calculate the Fréchet distance which does not have the complexity of handling the VVE-events as by Alt and Godau. We can come close to an $O(n^2)$ computation time for fixed dimensions and polygonal distance metrics. It is worth pursuing whether we can avoid the complexity of the dynamic upper envelope we use for the Euclidean case, as we achieve in the polyhedral case. Currently, Algorithm 2 uses a breadth-first computation over all $\Theta(nm)$ cells. If we know that large areas are reachable or not, i.e. the FSD has clusters of free or non-free cells, we would like to avoid these. Furthermore, it would be interesting to see how our novel algorithm performs in practice.

Bibliography

H. Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, pages 235–248. Springer, 2009.

H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1–2):75–91, 1995.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, third edition, 2008.

S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st International Conference on Very Large Data Bases*, pages 853–864. VLDB Endowment, 2005.

G. Brodal and R. Jacob. Dynamic planar convex hull. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002*, pages 617–626. IEEE, 2002.

K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog. In *Proceedings of the 23rd European Workshop on Computational Geometry*, pages 170–173. Citeseer, 2007.

K. Buchin, M. Buchin, and J. Gudmundsson. Detecting single file movement. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM GIS)*, pages 288–297, 2008.

K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry and Applications, special issue on 19th International Symposium on Algorithms and Computation (ISAAC)*, 21(3):253–282, 2011.

K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog—with an application to Alt's conjecture. *arXiv/1209.4403*, 2012a. URL `http://arxiv.org/abs/1209.4403`.

M. Buchin, S. Dodge, and B. Speckmann. Context-aware similarity of trajectories. In *Proceedings of the 6th International Conference on Geographic Information Science (GIScience)*. Springer, 2012b.

T. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16(4):361–368, 1996.

T. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *Journal of the ACM (JACM)*, 48(1):1–12, 2001.

A. Cook and C. Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms (TALG)*, 7(1):9, 2010.

A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. In *Proceedings of the 27th annual ACM symposium on Computational Geometry*, pages 448–457. ACM, 2011.

M. Henzinger, P. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *Journal of Computer and System Sciences*, 55(1):3–23, 1997.

M. Jiang, Y. Xu, and B. Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 6(1):51–64, 2008.

D. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.

N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM (JACM)*, 30(4):852–865, 1983.

M. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, volume 1, pages 108–115. IEEE, 1999.

R. van Oostrum and R. Veltkamp. Parametric search made practical. In *Proceedings of the 18th Annual Symposium on Computational Geometry*, pages 1–9. ACM, 2002.

M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166–204, 1981.

R. Sriraghavendra, K. Karthik, and C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proceedings of the 9th International Conference on Document Analysis and Recognition, 2007*, volume 1, pages 461–465. IEEE, 2007.