Realistic Analysis for Algorithmic Problems on Geographical Data

Werkelijkheidsgetrouwe Analyse voor Algoritmische Problemen met Geografische Data
(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof.dr. G.J. van der Zwaan,
ingevolge het besluit van het college voor promoties

in het openbaar te verdedigen op
maandag 16 september 2013 des middags te 4.15 uur

door

Anne Driemel

geboren op 19 mei 1983 te Rüdersdorf bei Berlin, Duitsland

Promotoren:
Prof.dr. M. J. van Kreveld
Prof.dr. M. de Berg

This work was supported by the Netherlands Organisation for Scientific Research (NWO) under project no. 612.065.823 (*Realistic Input Models for Geographic Applications*).

Contents

1 Introduction **1**

1.1 Motivation 1

1.1.1 Trajectory analysis 3

1.1.2 Shape matching 4

1.1.3 Terrain analysis 5

1.2 Realistic analysis 6

1.2.1 Imprecise input models 7

1.2.2 Probabilistic analysis 7

1.2.3 Approximation algorithms 8

1.2.4 Realistic input models 8

1.3 Contributions of this thesis 10

1.3.1 Approximating the Fréchet distance 10

1.3.2 Data structures for Fréchet-distance queries 11

1.3.3 The Fréchet distance with shortcuts 12

1.3.4 Flow computations on imprecise terrains 13

1.3.5 The complexity of Voronoi diagrams on terrains 14

2 Introduction to the Fréchet distance **15**

2.1 Basic definitions 15

2.2 State of the art 16

2.3 Basic concepts 17

2.4 Alt and Godau's algorithm 19

2.5 Notation 20

3 Approximating the Fréchet distance **21**

3.1 On c-packed curves 21

3.1.1 Definition and basic properties 21

3.1.2 Curve simplification 22

3.1.3 Bounding the free-space complexity 23

3.2 The algorithm 24

3.2.1 Computing the reachable free space 24
3.2.2 The approximate decision procedure 26
3.2.3 Searching for the Fréchet distance 27
3.2.4 The resulting algorithm 31
3.3 Analysis of the algorithm 31
3.3.1 Correctness 31
3.3.2 Running time 33
3.3.3 The result 34
3.4 Extension to low-density curves 35
3.4.1 Low density curves can be long only if they pay for it 35
3.4.2 Accounting for many reachable free-space cells 37
3.5 Extension to bounded curves 39
3.6 Extension to closed packed curves 41
3.7 Fatness implies packedness 43
3.8 Concluding remarks 44

4 Data structures for Fréchet-distance queries 47
4.1 Useful lemmas for curves and segments 47
4.2 Data structure for queries with single segments 49
4.2.1 Stage 1: Achieving a constant-factor approximation 49
4.2.2 Stage 2: A segment query to the entire curve 51
4.2.3 Stage 3: A segment query to a subcurve 53
4.3 Data structure for queries with polygonal curves 56
4.3.1 Universal vertex permutation 56
4.3.2 Extending the data structure 60
4.4 Concluding remarks 62

5 The Fréchet distance with shortcuts 65
5.1 Introduction 65
5.2 Preliminaries 68
5.2.1 The k-shortcut Fréchet distance 68
5.2.2 Tunnels in the free space diagram 69
5.3 The vertex-restricted shortcut Fréchet distance 72
5.3.1 Canonical tunnels and gates 72
5.3.2 A polynomial-time exact algorithm 73
5.3.3 Tunnel events 76
5.3.4 Analysis of the exact algorithm 78
5.3.5 A near-linear time approximation algorithm 80
5.3.6 Analysis of the approximation algorithm 86
5.3.7 Extension to the k-shortcut Fréchet distance 90
5.4 The continuous shortcut Fréchet distance 95
5.4.1 NP-hardness reduction 95
5.4.2 Correctness of the reduction 102
5.4.3 Algorithmic results 110
5.5 Concluding remarks 117

6 Flow computations on imprecise terrains 121
6.1 Introduction . . . 121
6.1.1 Basic definitions and notation . . . 123
6.2 NP-hardness in the surface model . . . 123
6.2.1 Flow model . . . 124
6.2.2 Overview of the construction . . . 124
6.2.3 Details of the construction . . . 127
6.2.4 Analysis of flow through a gadget . . . 129
6.2.5 Correctness of the NP-hardness reduction . . . 130
6.3 Watersheds in the network model . . . 131
6.3.1 Flow model . . . 131
6.3.2 Flow paths are stable . . . 132
6.3.3 Potential watersheds . . . 133
6.3.4 Computing potential watersheds in linear time . . . 138
6.3.5 Potential downstream areas . . . 140
6.3.6 Persistent watersheds . . . 142
6.4 Regular terrains . . . 145
6.4.1 Characterization of regular terrains . . . 145
6.4.2 Computing proxies and regular terrains . . . 147
6.4.3 Nesting properties of imprecise watersheds . . . 149
6.4.4 Fuzzy watershed boundaries . . . 151
6.4.5 The fuzzy watershed decomposition . . . 154
6.4.6 Disconnected persistent watersheds . . . 157
6.5 Concluding remarks . . . 157

7 Complexity of Voronoi diagrams on terrains 161
7.1 Introduction . . . 161
7.2 Preliminaries . . . 163
7.2.1 Voronoi diagrams on terrains . . . 163
7.2.2 The input model . . . 163
7.2.3 The complexity of the Voronoi diagram . . . 165
7.3 Upper bound . . . 167
7.4 Lower bound . . . 170
7.4.1 A simple example . . . 170
7.4.2 Farming – an $\Omega(n\sqrt{m})$ example . . . 171
7.4.3 Industrial farming – an $\Omega\big(nm^{2/3}\big)$ example . . . 173
7.5 Concluding remarks . . . 178

8 Conclusions 179
8.1 Summary . . . 179
8.2 Outlook . . . 180
8.2.1 Shape-matching protein structures . . . 180
8.2.2 Towards robust hydrological computations . . . 181

References 187

Publications 199

Samenvatting **201**

Zusammenfassung **205**

Summary **209**

Acknowledgements **213**

Curriculum Vitae **215**

Colophon **217**

CHAPTER 1

Introduction

Geographical information science is no more about geographical information systems than astronomy is about telescopes.[1]

1.1 Motivation

Many of our every-day problems are spatial, for example the question where the next coffee place is or how to get back to the hotel. We increasingly solve them with the help of the computer. While it used to be prevalent to ask people for directions on the street, today modern gadgets such as smart phones and navigation systems can provide an answer just as conveniently.

Underlying this technology are so-called *geographic information systems (GIS)*. The advancement of these systems is closely linked to automated cartography. In the most general terms, a geographic information system processes spatially referenced data and provides a visualization or summary (e.g., a map) of certain aspects of the data. A specific type of geographical data are trajectories of moving objects. The temporal component which is present in this tracking data is posing new challenges to GIS technology and is receiving increasing attention [47, 48]. Figure 1.1 demonstrates the need to address this challenge. The figure shows an automatically generated map of vessels driving in the Aegean sea over the period of one day. Since the temporal information was omitted from the data, the individual trajectories are not recognizable and the picture is cluttered.

Traditionally, geographic information systems are designed for experts to support decision making and for research purposes [93]. They are aimed at investigating

[1]In its original form, the statement "Computer science is no more about computers than astronomy is about telescopes" is sometimes attributed to E.W. Dijkstra.

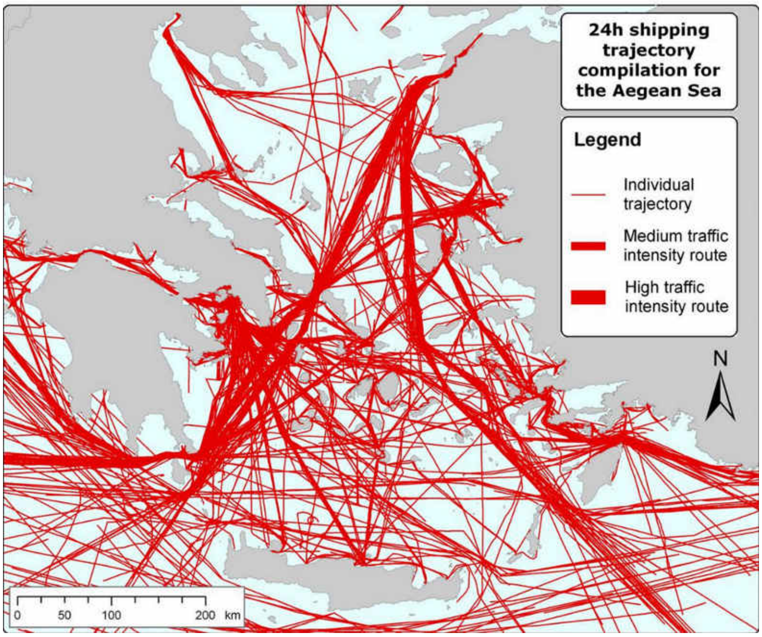

24h shipping trajectory compilation for the Aegean Sea
Legend
Individual trajectory
Medium traffic intensity route
High traffic intensity route
N
0 50 100 200 km

Figure 1.1: Example of an automatically generated map showing vessel trajectories. (Source: Archipelagos Institute of Marine Conservation, http://www.archipelago.gr/)

complex questions, such as: Where should one build houses such that they will not get flooded in the next twenty to fifty years? What are the prime locations for new businesses which maximize the number of walk-in customers? How should radio towers be distributed to provide maximal coverage of the served area and at the same time minimize the interference?

Computer science is a research area that is nourished by problems arising in application areas. Typically, a problem is first modelled in the form of a mathematical question with the goal to solve the problem on this abstract level. For example, instead of studying the problem of predicting surface flow for a particular geographical region, we assume the terrain is given as a triangulated surface of points in three-dimensional space and we assume any water stream follows the path of steepest descent. We can define a watershed map that subdivides the surface into regions, where each region consists of the surface points that send water to the same local minimum. This helps us to predict the volumes of water that arrive at a certain point in the river. During this process, the abstract problem may become disconnected from the original application and may develop a life of its own, given that it is algorithmically interesting enough. Much later and without anticipation, the same abstract problem might appear in a different context again. Watershed maps, for example, are also used in image segmentation. In this way, solutions become universally applicable and are continuously extended. This is where computer science distinguishes itself as a science from basic engineering. The statement at the beginning of this chapter alludes to this fundamental difference. It could have been by E.W. Dijkstra, who was a vivid promotor of the mathematical view onto problems which arise in the development of computer programs [63, 66, 64, 65] and who received the Turing award for his eloquent insistence. In a similar spirit, but less provocatively, Michael Goodchild distinguishes:

> "*geographic information science (GIScience)*, which is the research field that studies the general principles underlying the acquisition, management, processing, analysis, visualization, and storage of geographic data; and *geographic information systems (GIS)*, which are computer software packages designed to carry out these activities" [80].

The analysis of geographical data stands in the long tradition of the old mathematical field of geometry. The origin of the word derives from ancient greek: *geo-* "earth" and *-metron* "measurement". Today, *computational geometry* is the branch of theoretical computer science which has a focus on geometrical algorithms and data structures. Thus, the algorithmic problems that arise in the development of GIS technology, besides being of interest to GIScience, also fall into the area of computational geometry.

In theoretical computer science, the standard way to measure the efficiency of an algorithm is to analyze its running time and space requirements in the worst case. This fundamental technique helps us to distinguish hard problems from tractable ones and thus to reason about the computational complexity of problems. However, the worst-case analysis does not always predict the actual behaviour of an algorithm and thus it is sometimes less suitable to measure the efficiency in practice. This is a limitation of the abstraction process described earlier.

In this thesis, we want to bridge the gap between theory and practice by using *realistic analysis*. It spans a collection of techniques that allow for a theoretical analysis with provable bounds which are meaningful in practice. We will discuss these techniques in more detail in Section 1.2. Next, we outline the general topics of this thesis. The specific contributions in these subject areas are then discussed in Section 1.3.

1.1.1 Trajectory analysis

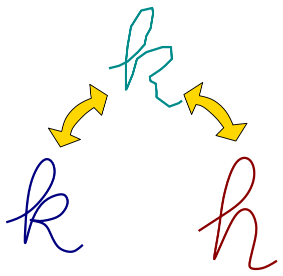
A trajectory is a curve parametrized by time. Its representation usually follows the way it is recorded, that is, as a series of time-stamped positions. The research area of trajectory analysis is relatively young. With the advent and wide availability of digital tracking devices, such as GPS-sensors and mobile phones, an increasing amount of tracking data is being collected [98, 106]. The research objectives are diverse. Researchers might be interested, for example in the movement behaviour of animals, which they have tagged with location sensors [39, 85]. Similarly, traffic analysts may be interested in the navigational strategies used by taxi drivers or bicyclists in large cities [160, 22]. Another source for trajectory data lies in meteorology. The ability to trace and predict the trajectory of a hurricane can be crucial in disaster management and evacuation planning [22, 152]. Figure 1.1 demonstrates the challenges in visualizing and structuring the amount of data being collected.

With all of this data available, there is a growing demand for efficient algorithms and data structures to manipulate, store and analyze this data. One of the problems that are relevant here is the *shape matching problem*, which we discuss in detail further below. A specific variant of this problem is finding the path of a moving object in a street map. Assume we are given the trajectory of the object and a geometric graph (the street map). To solve the problem, one can find the path in the graph which has the closest similarity to the trajectory over all possible paths in the graph. We study shape matching techniques for curves and data structures to support similarity queries in Chapters 2-5.

Another problem is the *trajectory segmentation problem*. The goal is to partition the trajectory into subsequences that are homogeneous according to some particular measure. As a high-level example, consider the trajectory of a person commuting between home and work. A meaningful segmentation would distinguish the time periods where the person is either (i) walking, (ii) using the train, (ii) using the car, (iii) using the bicycle, or (iv) staying in one place.[2]

1.1.2 Shape matching

The *shape matching problem* is to compute whether two given shapes are similar and how similar they are. It is an active research topic within computational geometry [9], as well as in many other research areas.



Input shapes may be handwritten characters [121, 138] or outlines of objects depicted in an image [134], protein chains in structural biology [94] or trajectories of moving objects [157, 32].

We can identify two fundamental problems:
(A) how to define similarity and
(B) how to compute this similarity efficiently.

For the first problem, one can define a similarity measure, which is a function that returns a high score if the input curves are very similar. More commonly, a *distance measure* is defined, where the score is inversely proportional to the similarity. For computational reasons it can be desirable if the triangle inequality is satisfied. Ideally, this function closely models human perception and judgement.

Depending on the application setting, the appropriate definition of similarity might be transformation invariant. For example, when comparing molecular structures, their actual placement and orientation in three-dimensional space must be ignored (invariance under rigid motions). In imaging, the same object might be depicted from slightly different perspectives (invariance under affine transformations). The semantics of the geographical positions visited by a trajectory are global and therefore transformation invariance is usually not desirable in this case. The distance measures which we discuss in this thesis are not invariant under transformations.

A very general and well-known distance measure in mathematics is the so-called *Hausdorff distance*. Given two point sets A and B, consider the largest distance that any point in A has to a closest point in B and the largest distance that any point in B has to a closest point in A. The maximum over both directions is the Hausdorff distance. Another distance measure is the so-called *Fréchet distance*, which we formally define in Chapter 2. The Fréchet distance is very similar to the Hausdorff distance, however it is better suitable for curves, since it takes the continuity of the

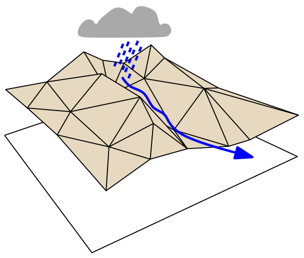
[2]The author of this thesis has published on this topic, see [38] and [18], however this material was omitted from the thesis.

curves into account. Intuitively, it corresponds to the maximum distance a point on the first curve has to travel as this curve is being continuously deformed into the second curve. Hence, the Fréchet distance is dominated by the maximum distance (the so-called *bottleneck*) that two points on the curves have under the Fréchet matching.

Compared to other distance measures which use an averaging mechanism, the bottleneck makes it quite sensitive to noise and therefore less applicable to real-world data. In Chapter 5 we introduce a new variant of the Fréchet distance which is more robust against input noise and study the computational problems that arise from this definition. Furthermore, in Chapter 3 we study the problem of computing the standard Fréchet distance and in Chapter 4 we study data structures that support queries for the Fréchet distance between partial curves. In Chapter 8 we come back to these definitions in the context of protein structure comparison.

1.1.3 Terrain analysis

Naturally, many geographical questions involve the surface of the earth. *Hydrological modelling* is an example of a research area within GIScience where the interplay of elevations on the surface is of huge importance. The main question is where water resulting from precipitation ends up once it hits the surface and how river networks are formed resulting from this flow of water [26].



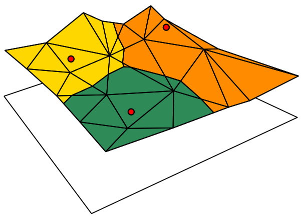
We can also ask the question the other way around: Given a point on the terrain, which other places can it possibly receive water from and how much?

Watershed maps can be computed to answer this question [143, 49]. However, there are still many open questions and challenges that remain. The computations have to be done on a digital model of the actual geographic terrain, which inevitably contains measurement errors. However, hydrological computations are known to be extremely sensitive to elevation error [154, 26].

A mathematical model of a terrain is a height function on a two-dimensional domain. We can distinguish two basic types of models. The model most commonly used in computational geometry is the polyhedral terrain. In this model, a terrain is defined by a triangulated set of points in the domain, where every point has a certain height. The surface of the terrain is defined by the triangulation lifted according to these heights. The second model which is very popular in GIS, is a grid terrain. In this model, the domain is formed by a grid and here every grid point has a certain height. One can picture the surface of this type of terrain as a bed of needles with different lengths.

The main difference between the two models is that the polyhedral terrain provides a continuous model of the surface, whereas the grid terrain serves as a discrete model of the same. The type of model (polyhedral vs. grid) used in hydrological computations can have a large influence on the outcome and difficulty of the computations. We study these relations in more detail in Chapter 6. In Chapter 8 we give an extensive interdisciplinary literature review on flow computations on digital terrains and evaluate our results in this context.

The second topic which lies in the area of terrain analysis and which we study in this thesis involves the shortest paths along the terrain surface. Such a shortest path is called a *geodesic* and the length of the path is called the geodesic distance. Mathematically, the terrain surface induces a distance measure on the set of points which forms its surface.



The *Voronoi diagram* is defined as a subdivision of a space into cells according to a set of sites, which we here define as a set of points. The cell that corresponds to a site consists of all the points for which the site is the nearest neighbor out of the set of sites.

One can define the Voronoi diagram on the terrain surface with respect to the geodesic distances. Such geodesic Voronoi diagrams are an important data structure used in wireless networks [107, 139], surface reconstruction and analysis [104, 117, 140], and many other problems. An interesting question is thus, what the complexity of such a geodesic Voronoi diagram is in practice. We give answers to this question in Chapter 7.

1.2 Realistic analysis

The worst-case analysis of the running time and space complexities as a function of the input size is a fundamental method in algorithm design. However, it fails to describe the actual behaviour when the worst case is a contrived geometric configuration which would never occur in practice.

For instance in practice, the complexity of Voronoi diagrams on terrains can be observed to be linear in the size of the terrain. In the example shown on page 7, however, a construction of a terrain is described, such that the Voronoi diagram of two particular points on the surface has quadratic complexity. Note that the terrain is very specific and contrived in its shape and representation. The example illustrates the discrepancy between the analysis and the reality which we are facing when working with geographical input.

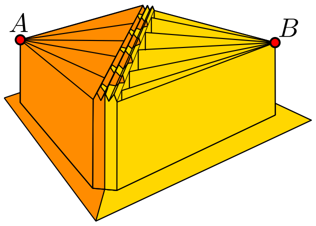
There are different approaches to reasoning about algorithms and data structures for real data that allow a theoretical analysis with provable bounds. Known techniques that have been used in computational geometry are approximation algorithms [86], probabilistic analyses such as expected [128] and smoothed analysis [137], imprecise input models [112], and realistic input models [56].

Thus, we have a collection of sophisticated tools at our disposal, which we make use of in this thesis. In Chapter 3 and Chapter 5, we give several approximation algorithms and use realistic input models in the analysis. In Chapter 6 we use imprecise input models. In Chapter 7 we use a combination of realistic input models and a probabilistic analysis.

1.2.1 Imprecise input models

Inarguably, real-world data is imprecise and contains measurement errors. Similarly, the limitations of floating point arithmetic in modern processors can lead to the output being incorrect. On the other hand, the worst-case configurations which we deal with

Example 1.2 A high complexity Voronoi diagram on a terrain.



The *bisector* between two points A and B on a polyhedral terrain is formed by the set of points that have equal geodesic distance to A and to B. Subdividing the surface into two cells, it represents the Voronoi diagram of A and B on the terrain. In the worst case, the bisector can have a quadratic complexity. A construction that shows this is depicted in the figure to the right. We place A and B at the centers of two chinese fans, each of linear complexity. The arms of the fans connect to a linear number of ridges on two opposite sides and are shifted against each other. The construction is such that the shortest path from any point on the bisector to either A or B will follow one of the lines on top of the corresponding fan. This has the effect that the bisector crosses the ridge triangles in a zig-zag manner leading to a quadratic number of intersections with the triangles that form the terrain surface. This example was previously mentioned in [119].

in the theoretical analysis are often very specific. In some cases, a perturbation in the right place might break the mechanics of the construction. This problem is not specific to geometrical data. To bound numerical errors in mathematical computations, the theory of interval arithmetic has been developed long ago. Instead of doing the computations with seemingly exact values, we can use intervals which contain the true exact value to bound the computational error in every step of the computations. In this way, the error in the output of the computations can be controlled.

In computational geometry, the input data is often described using higher dimensional coordinates. Here, the equivalent to intervals are disks or balls, which contain the true coordinates. In some cases, the increased correctness comes at the cost of making the computational problems harder. In other cases, it is possible to prove a lower complexity that is closer to the observed complexities, for example in [55].

1.2.2 Probabilistic analysis

Randomized algorithms, such as quicksort, are used in many state of the art software libraries. The efficiency observed in practice can also be proven in theory if we use a probabilistic analysis for the running time. The basic idea is the following. Instead of bounding the complexity in the worst case, we assume a probability distribution over all possible input instances and bound the complexity in the expected case. Alternatively, if the algorithm is randomized, one can take the probability over the internal randomness of the algorithm. We can also combine the idea of imprecise input models, which we discussed above, with a probabilistic analysis. Indeed, the idea of the so-called *smoothed analysis* is to analyze the expected complexity under small perturbations of the input. Analysing the complexity of an algorithm often reduces to analysing the complexity of a combinatorial or geometric structure. The same holds true for analysing the costs and performance of a data structure. In both cases a probabilistic analysis can give new insights.

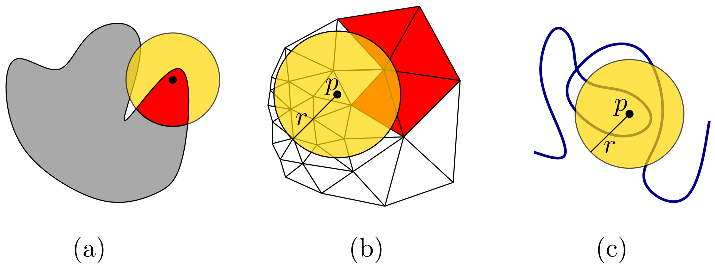


Figure 1.3: Illustration to the definitions of (a) fatness, (b) low-density of a set of triangles and (c) packedness of a curve.

1.2.3 Approximation algorithms

If we acknowledge that the input is imprecise, then we might also be satisfied with an approximate answer to the studied questions. For this, the theory of approximation algorithms has been developed. An approximation algorithm is an algorithm which does not give the exact answer. However, the output can be proven to be correct up to a certain factor. This factor is usually independent of the input size. In some cases, the algorithm can even output a result which is arbitrarily close to the exact result. This can be modeled by proving an approximation factor of $(1 + \varepsilon)$, where $\varepsilon > 0$ is an input parameter to the algorithm. In this case, the running time and space complexities also depend on the value of ε and this parameter has to be taken into account in the analysis.

1.2.4 Realistic input models

The idea behind so-called *realistic input models* is to carry out the worst-case analysis with respect to parameters that describe how “unrealistic” the input is. Typically, these parameters constrain shape or distribution in space. Using this technique, one can sometimes explain an observed discrepancy between actual and theoretical running times and reveal inherent properties of realistic input that make the problem computationally easy or hard. More importantly, this gives rise to new algorithms that are more efficient by exploiting the observed properties. The most commonly used realistic input models are *fatness* and *low density*. In this thesis we introduce a third model, called *packedness* and analyze its relationship to previous models. We give a brief overview over these models. For a more extensive discussion we refer to the paper by de Berg *et al.* [56].

Fatness. Arguably, fatness is one of the earliest realistic input models that have been used in computational geometry, albeit without this particular label. Alt *et al.* [11] show among other results that the union complexity of n wedges is in $O(n)$ if the opening angles of the wedges are bounded from below. Note that without this restriction, the complexity is $\Theta(n^2)$, since skinny wedges can be arranged to form a grid-like structure with many holes. Later, this concept was used also by Matoušek *et al.* in [115] when they introduced so-called *fat triangles*, the definition being again that the inside angles are bounded from below. Over time, the concept of fatness has

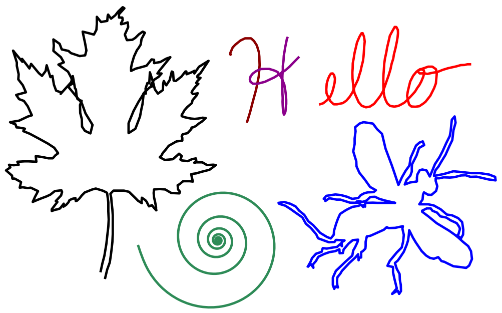


Figure 1.4: Examples of curves that are c-packed for a small constant c.

been generalized further beyond geometric primitives. One of the most recent results by de Berg [50] deals with fat objects which are not necessarily convex or polygonal. According to this definition, an object is *(locally)* γ*-fat*, if for any ball centered inside the object, and which does not fully contain the object, the connected component of the object inside the ball, which contains the center, takes up at least a ratio of γ of the volume of the ball. See Figure 1.3 (a) for an illustration.

The class of shapes defined by this definition is more general than a definition of fatness previously suggested by Efrat, which is the class of (α, β)-covered shapes [74].

Low density. The idea of parametrizing the density of a set of geometric objects was introduced initially for the robot motion planning problem [148, 147]. It also proved to be a useful input assumption for shortest-path computations on polyhedra and Voronoi diagrams on terrains [130, 17, 69]. A set of objects is γ*-low-density*, if for any ball of any radius, the number of objects intersecting the ball that are larger than the ball is less than γ. See Figure 1.3 (b) for an illustration. This definition allows for a very general class of realistic input. A large ball can contain many triangles as long as they are small. Furthermore, like the definition of fatness, it is scale independent. There exists a strict hierarchy of models, as shown in [56]. According to this hierachy, low-density is strictly more general than fatness, since a set of non-intersecting fat objects is always low-density, but a set of low-density objects does not need to consist of fat objects. We can also apply the low-density assumption to the set of edges of a polygonal curve and obtain a very general class of curves.

Straightness and boundedness. Alt *et al.* study the problem of computing certain similarity measures for restricted classes of curves in [14]. Following their definitions, a curve is called γ*-straight*, if the euclidian distance between two points on the curve is only a factor γ smaller than their distance along the curve. Secondly, a curve is called γ*-bounded*, if for any two points on the curve, the portion connecting them stays within a radius of γ to either point. These models bound the dilation or detour of the curve, which is a natural way to restrict such a curve. However, note that the resulting classes of curves are not very general, since the restriction on the detour disallows the curve to revisit places.

Packedness. In this thesis, we introduce a new class of curves, called c-packed curves. A curve X is ***c*-packed** if the total length of X inside any ball is bounded by c times the radius of the ball. See Figure 1.3 (c) for an illustration. A c-bounded curve might have arbitrary length while maintaining a finite diameter, and as such may not be c-packed. However, in some sense c-packed curves are considerably more general and a more natural family of curves. For example, a c-packed curve might self cross and revisit the same location several times, and concatenating two c-packed curves always results in a $(2c)$-packed curve, none of which can be claimed for c-bounded curves. Figure 1.4 shows examples of curves which are c-packed for a small constant c. Like fatness and low-density, the definition is scale independent. Intuitively, c-packed curves behave reasonably in any resolution. We will see that the class of c-packed curves is strictly more general than the class of low-density curves. In section Section 3.1 we study basic properties of c-packed curves.

1.3 Contributions of this thesis

1.3.1 Approximating the Fréchet distance

The Fréchet distance is used in shape matching to define similarity between curves. An intuitive definition can be given as follows. Imagine walking forwards along the two curves simultaneously. At any point in time, the two positions have to stay within distance δ from each other. The minimal δ for which such a traversal is possible is the Fréchet distance. In Chapter 2 we give an extensive introduction to this distance measure and explain standard concepts used in the literature.

To date, the best known algorithm to compute the Fréchet distance between two polygonal curves takes time roughly quadratic in the number of vertices. The algorithm consists of a decision procedure, which is used in a search over candidate values for the Fréchet distance. In Chapter 3 we present a simple and practical $(1+\varepsilon)$-approximation algorithm for the Fréchet distance between two polygonal curves in $\mathbb{R}^d$. We show that our algorithm has near-linear running time for c-packed polygonal curves and similar results for other input models. Underlying the algorithm are several new insights. We simplify the input curves during the decision procedure to the appropriate resolution to reduce the complexity of the algorithm. The choice of the resolution has to be balanced to preserve enough information to reason about the original curves. Secondly, we show how to approximate the candidate values, such that the search for the Fréchet distance can be done efficiently without using parametric search or random sampling.

We extend the analysis and show that also for low-density curves in the plane the algorithm runs in near-linear time and is still subquadratic in higher dimensions. This relies on a new packing lemma showing that, if the simplification of a low-density curve is long inside a relatively small area, then the original curve must contain many vertices in the vicinity of this region. We also improve upon the result by Alt *et al.* [14] for κ-bounded curves and we show how to adapt our algorithm to handle closed curves. These bounds imply that the presented algorithm provides fast approximation for the Fréchet distance for all these types of curves. The new results are summarized in Table 1.5. The material in Chapter 3 has been published in [70] and in [71].

Curves type	Running time	See
c-packed	$O(cn/\varepsilon + cn \log n)$	Theorem 3.3.5
κ-straight	Same as 2κ-packed	Lemma 3.5.2
κ-bounded	$O\big((\kappa/\varepsilon)^d n \log n\big)$	Theorem 3.5.6
$O(1)$-low-density	$O\left(\frac{n^{2(d-1)/d}}{\varepsilon^2} \log n\right)$	Theorem 3.4.7
c-packed & closed	$O\big(c^2 n/\varepsilon^2 + c^2 n \log n\big)$	Theorem 3.6.4

Table 1.5: Summary of new results for computing a $(1+\varepsilon)$-approximation to the Fréchet distance between two polygonal curves with n vertices in $\mathbb{R}^d$. For κ-bounded and low-density curves, the running time can be slightly improved, see [71].

1.3.2 Data structures for Fréchet-distance queries

In Chapter 4 we present a data structure that preprocesses a given polygonal curve Z, such that given a query segment h, and two points p, p' on Z (and the edges containing them), it $(1+\varepsilon)$-approximates the Fréchet distance between h and the subcurve of Z between p and p'. This data structure is used by the algorithms described in the proceeding Chapter 5. The data structure works for any polygonal curve, requires $O(n \log^2 n)$ time and $O(n)$ space for preprocessing, regardless of the packedness or density of the input, and can answer such queries in $O(\log n \log\log n)$ time (ignoring the dependency on ε).

We extend the data structure as follows. First, we show how to preprocess a polygonal curve in near-linear time and space, such that, given a number $k \in \mathbb{N}$, one can compute a simplification in $O(k)$ time which has $K = 2k-1$ vertices of the original curve and has approximately optimal Fréchet distance to the original curve, compared to any curve which uses k vertices. Surprisingly, this can be done by computing a permutation of the vertices of the input curve, such that this simplification is the subcurve defined by the first K vertices in this permutation. Namely, we compute an ordering of the vertices of the curves by their "Fréchet error".

Secondly, we use this vertex permutation to extend the initial data structure to support queries with polygonal curves of multiple segments (as opposed to single segments). The resulting data structure can answer queries with a constant approximation factor in $O(k^2 \log n \log(k \log n))$ time, where k is the number of vertices of the query curve. The preprocessing takes $O(n \log^3 n)$ time and uses $O(n \log n)$ space.

To our knowledge these are the first data structures to support these types of queries, apart from recent results by de Berg *et al.* [53] and Gudmundsson and Smid [83]. De Berg *et al.* provide a data structure to count the number of subcurves that are within a certain Fréchet distance from a query segment up to a constant approximation factor. Gudmundsson and Smid study the problem under the packedness assumption. In the concluding remarks of the chapter we outline how our data structures could be extended to the case where the subcurve of Z is not fixed. The material in Chapter 4 has been published in [67] and will be published in [68].

1.3.3 The Fréchet distance with shortcuts

The standard Fréchet distance is a bottleneck distance measure and hence quite sensitive to outliers. In Chapter 5 we introduce the notion of a more robust Fréchet distance. Again, imagine walking along the two curves simultaneously while maintaining a maximal distance between the two positions. However, we are now allowed to "shortcut" parts of the input curves. This is a natural approach for handling noise, in particular batched outliers. Consider the example of a bird navigating along a coastline. The bird ignores "detours" like harbors or river mouths and instead follows a shortcut across, see the figure to the right. Using the new shortcut Fréchet distance, we can detect the similarity of the trajectory to the coastline. The distance measure automatically cuts across outliers and ignores data specific "detours". Hence it can produce more meaningful results when dealing with real-world data than the standard Fréchet distance. It can also be interpreted as a partial distance measure which is parameter-free. We study the complexity of computing the shortcut Fréchet distance in the case where shortcuts can be taken on one of the two curves. The results can be summarized as follows.

We first study the shortcut Fréchet distance in the case where shortcuts have to start and end at input vertices of one of the two curves. For a prescribed parameter $\varepsilon > 0$, we present an algorithm for computing a $(3+\varepsilon)$-approximation to the shortcut Fréchet distance between two given c-packed polygonal curves of total complexity n. The running time of the new algorithm is near-linear if the number of shortcuts is small or unbounded. More precisely, if we allow an unbounded number of shortcuts the running time is $O\left(c^2 n \log^3 n\right)$. For the case that only k shortcuts are allowed, we present a variant of this algorithm with running time $O\left(c^2 kn \log^3 n\right)$. In the analysis we use techniques developed in Chapter 3 and follow the general approach of devising a decision procedure which is used to search over the critical values, the candidate values for the shortcut Fréchet distance. The shortcuts introduce a new type of candidate value. We characterize these values and show how to compute them. Furthermore, we show that these values have a certain monotonicity property, which makes it possible to search over them efficiently. We also present a polynomial-time exact algorithm to compute the shortcut Fréchet distance in the vertex-restricted case.

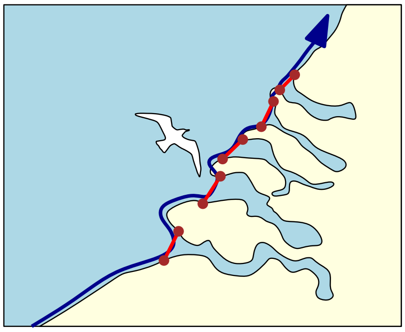


Figure 1.6: A trajectory (blue) of a seagull on the way to its nest, navigating along the coastline of Zeeland (the Netherlands) while taking shortcuts (red).

In the second part of the chapter, we show that, surprisingly, in the general case, where shortcuts can be taken at any point along a curve, the problem of computing the shortcut Fréchet distance exactly is NP-hard. This constitutes the first hardness result for a variant of the Fréchet distance between two polygonal curves. Specifically, we describe a polynomial-time reduction from SUBSET-SUM to the decision problem. An important observation is that the reachable free-space of the matchings may fragment into an exponential number of components. We use this fact in our reduction together with a mechanism that controls the sequence of free-space components that may be visited.

Furthermore, we show how to combine the algorithmic layout used in the first part of the chapter with a line-stabbing algorithm of Guibas *et al.* [84] to obtain a 3-approximation algorithm for the decision problem which runs in $O(n^3 \log n)$ time. The approximation scheme used by this algorithm naturally prevents the reachable free space from being fragmented. While traversing the free space diagram, the line-stabbing algorithm enables us to compute the possible endpoints of shortcuts in the interior of edges.

In the concluding remarks of the chapter we give an extensive discussion of open problems. In particular we discuss some challenges in extending the decision algorithm to the computation problem. Furthermore, we discuss the case where shortcuts can be taken on both curves and conjecture that many of our results carry over.

The material in Chapter 5 has been partially published in [67] and in [37] and will be partially published in [68].

1.3.4 Flow computations on imprecise terrains

In Chapter 6 we study water flow computation on imprecise terrains. The problem has been extensively studied under the assumption that the given elevation data is exact. The watershed of a point p is commonly defined as the set of points that send water to p via a flow path. On imprecise terrains, we introduce the notion of a *potential watershed*, which consists of the points that could potentially send water to p via a flow path, and the *persistent watershed*, which consists of the points that persistently send water to p in some way or another.

We consider two approaches to modeling flow on an imprecise terrain: one where water flows across the surface of a polyhedral terrain in the direction of steepest descent, and one where water only flows along the edges of a predefined graph, for example a grid or a triangulation. The second model is widely adopted in GIS applications, for example in the form of the D-8 grid model. In both cases we allow each vertex an imprecise elevation, given by an interval of possible values, while its (x, y)-coordinates are fixed. For the first model, we show that the problem of deciding whether one vertex may be contained in the watershed of another is NP-hard.

In contrast, for the second model we give a simple $O(n \log n)$-time algorithm to compute the potential and persistent watershed of a vertex, or a set of vertices, where n is the number of edges of the graph. On a grid model, we can compute the same in $O(n)$ time. We extend these techniques and achieve the same running times for computing the *potential downstream area* of a point.

In order to still extend these results, we define a certain class of imprecise terrains which we call *regular*. We prove that persistent watersheds satisfy certain nesting conditions on regular terrains and give an algorithm that turns a non-regular terrain into a regular one. This leads to efficient computations of potential watershed boundaries, and to a natural definition of a *potential ridge*, which delineates the persistent watersheds of the "main" minima of a regular terrain. On regular terrains, the potential ridge is equal to the union of the areas where the potential watersheds of these minima overlap and we can compute it in $O(n \log n)$ time.

The material in Chapter 6 has been previously published in [72] and will be published in [73].

1.3.5 The complexity of Voronoi diagrams on terrains

In Chapter 7 we investigate the combinatorial complexity of geodesic Voronoi diagrams on polyhedral terrains using a probabilistic analysis. Aronov *et al.* [17] prove that, if one makes certain realistic input assumptions on the terrain, this complexity is $\Theta(n+m\sqrt{n})$ in the worst case, where n denotes the number of triangles that define the terrain and m denotes the number of Voronoi sites.

We prove that under a relaxed set of assumptions the Voronoi diagram has expected complexity $O(n + m)$, given that the sites have a uniform distribution on the domain of the terrain (or the surface of the terrain). Furthermore, we present a worst-case construction of a terrain which implies a lower bound of $\Omega(nm^{2/3})$ on the expected worst-case complexity if these assumptions on the terrain are dropped. This lower bound may serve as a justification for the input assumptions made earlier, since it implies that the randomness assumption by itself is not sufficient to prove a low complexity in our analysis. The construction that leads to this lower bound is intricate and requires a careful balancing of the variance of the distances of the sampled sites, and how closely they can be packed together.

The material in Chapter 7 has been previously published in [69].

CHAPTER 2

Introduction to the Fréchet distance

2.1 Basic definitions

Basic notation. Let X be a curve in $\mathbb{R}^d$; that is, a continuous mapping from $[0, 1]$ to $\mathbb{R}^d$. We will identify X with its image $X([0, 1]) \subseteq \mathbb{R}^d$ if it is clear from the context. The curve X is ***closed*** if $X(0) = X(1)$. We use $\|\cdot\|$ to denote the Euclidean distance as well as the length of a curve. Given two curves X and Y that share an endpoint, let $X \oplus Y$ denote the ***concatenated*** curve. We denote with $X[x, x']$ the subcurve of X from $X(x)$ to $X(x')$ and with $X\langle \mathsf{p}, \mathsf{p}'\rangle$ the subcurve of X between the two points $\mathsf{p}, \mathsf{p}' \in X$. In Section 2.5 we give an overview of the general notation used in Chapters 3-5.

The Hausdorff distance. Let $X : [0, 1] \to \mathbb{R}^d$ and $Y : [0, 1] \to \mathbb{R}^d$ be two polygonal curves. The ***directed Hausdorff distance*** is defined as

$$d_{\mathcal{H}}(X, Y) = \max_{x\in[0,1]} \min_{y\in[0,1]} \|X(x) - Y(y)\| .$$

This corresponds to the maximum distance a point on X has to its closest point on Y. The undirected Hausdorff distance is defined as the maximum over both directions.

The Fréchet distance. A ***reparameterization*** is a one-to-one and continuous function $f : [0, 1] \to [0, 1]$. It is ***orientation-preserving*** if it maps $f(0) = 0$ and $f(1) = 1$. The Fréchet distance is defined only for oriented curves, as we need to match the start and end points of the curves. The orientation of the curves we use will be understood from the context. We define the ***width*** of an orientation-preserving reparameterization f of X with respect to Y as

$$\mathrm{width}_{X,Y}(f) = \max_{\alpha\in[0,1]} \|X(f(\alpha)) - Y(\alpha)\| .$$

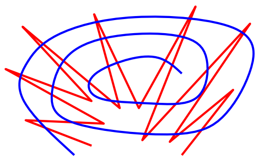
Now, the ***Fréchet distance*** between the two curves is

$$d_{\mathcal{F}}(X,Y) = \inf_{f:[0,1]\to[0,1]} \mathrm{width}_{X,Y}(f)\,.$$

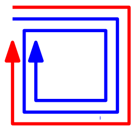
The Fréchet distance can be interpreted as the maximum length of a leash one needs to walk a dog, where the dog walks monotonically along X according to f, while the handler walks monotonically along Y according to its original parametrization. In this analogy, the Fréchet distance is the shortest possible leash admitting such a traversal of the curves. For the ***discrete Fréchet distance***, the handler and the dog turn into frogs and the vertices are stepping stones between which the frogs are allowed to jump during a traversal. The Fréchet distance complies with the triangle inequality; that is, for any three curves X, Y and Z we have that $d_{\mathcal{F}}(X,Z) \leq d_{\mathcal{F}}(X,Y) + d_{\mathcal{F}}(Y,Z)$.

2.2 State of the art

Comparing geometric shapes is a task that arises in a wide arena of applications. The Fréchet distance and its variants (e.g., dynamic time-warping [100]) have been used as similarity measures in applications such as matching of time series in databases [102], comparing melodies in music information retrieval [133], speech recognition [105], signature and handwriting recognition [121, 138], matching coastlines over time [114], as well as in map-matching of vehicle tracking data [30, 157], and moving objects analysis [31, 32].



Informally, the Fréchet distance between two curves is the maximum distance a point on the first curve has to travel as this curve is being continuously deformed into the second curve. Unlike the Hausdorff distance, which is solely based on nearest neighbor distances between points on the curves, the Fréchet distance requires a continuous and order-preserving assignment of points to measure these distances. The figure to the right shows an example of two dissimilar curves that have a small Hausdorff distance, but a large Fréchet distance.



Moreover, the Fréchet distance between two curves might be arbitrarily larger than their Hausdorff distance, as demonstrated by the figure on the left. In many applications, the course of the curve is important, for example when comparing trajectories. This makes the Fréchet distance appear like a better measure of similarity in these cases and shows that it is better suited for comparing curves with respect to their intrinsic structure.

For two polygonal curves in the plane of total complexity n, computing their Hausdorff distance can be done in $O(n \log n)$ time [9]. However, computing their Fréchet distance takes roughly quadratic time according to the best known algorithms to date. After publication in the seminal paper by Alt and Godau [12], their $O(n^2 \log n)$-time algorithm remained the state of the art for more than a decade. This led Alt to conjecture that the problem of deciding whether the Fréchet distance between two curves is smaller or equal a given value is 3SUM-hard. It has been an open problem to find a subquadratic algorithm for computing the Fréchet distance for two curves. However, recently, there has been some progress in improving upon the quadratic running time of the decision algorithm. First, Agarwal *et al.* presented a subquadratic time algorithm for the discrete Fréchet distance, which only considers distances between vertices of the curves [3]. Buchin *et al.* build upon their work and give an algorithm for the original Fréchet distance [34]. Their algorithm is randomized and takes $o(n \log n)$ expected time overall to compute the Fréchet distance. The decision algorithm they present is deterministic and takes subquadratic time. The only lower bound known for the decision problem is $\Omega(n \log n)$ and was given by Buchin *et al.* [33]. A randomized algorithm simpler than the one by Alt and Godau, which has the same running time, but avoids parametric search (see Section 2.4), was recently presented by Har-Peled and Raichel [88].

The only subquadratic algorithms known are for quite restricted classes of curves such as for closed convex curves and for κ-bounded curves [14]. For closed convex curves the Fréchet distance equals the Hausdorff distance and for κ-bounded curves the Fréchet distance is at most $(1+\kappa)$ times the Hausdorff distance, and hence the $O(n \log n)$ algorithm for the Hausdorff distance can be applied to compute a $(1+\kappa)$-approximation. Aronov *et al.* [20] provided a near-linear time $(1+\varepsilon)$-approximation algorithm for the discrete Fréchet distance for a restricted class of curves called *backbone curves.*

2.3 Basic concepts

Observation 2.3.1 We state a simple observation which is used throughout the chapters on the Fréchet distance. Given two directed segments pq and uv, it holds $d_{\mathcal{F}}(\mathsf{pq}, \mathsf{uv}) = \max(\|\mathsf{u} - \mathsf{p}\|, \|\mathsf{v} - \mathsf{q}\|)$. To see this, consider the uniform parameterization $\gamma(t) = t\mathsf{p} + (1-t)\mathsf{q}$ and $\pi(t) = t\mathsf{u} + (1-t)\mathsf{v}$, for $t \in [0,1]$. It is easy to verify that $f(t) = \|\gamma(t) - \pi(t)\|$ is convex, and as such $f(t) \le \max(f(0), f(1))$, for any $t \in [0,1]$.

Free-space diagram. For two polygonal curves $X : [0,1] \to \mathbb{R}^d$ and $Y : [0,1] \to \mathbb{R}^d$ the square $[0,1]^2$ represents their ***parametric space***. We assume that the parametrizations of the curves are uniform on an edge. We write V_X and V_Y to denote the set of vertices of the curves X and Y, respectively. For a point $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p}) \in [0,1]^2$, we define its ***elevation*** to be $\mathrm{dist}(\mathsf{p}) = \|X(x_\mathsf{p}) - Y(y_\mathsf{p})\|$. For a given parameter $\delta > 0$, the *δ-**free space*** of X and Y is defined as

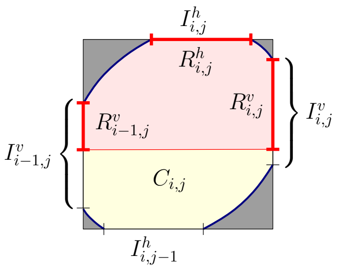
$$\mathcal{D}_{\le\delta}(X,Y) = \left\{ \mathsf{p} \in [0,1]^2 \;\middle|\; \mathrm{dist}(\mathsf{p}) \le \delta \right\}.$$

This is a level set of the height function defined by the elevations of points in the parametric space.

The parametric space can be broken into a (not necessarily uniform) grid called the ***free-space diagram***, where a vertical line corresponds to a vertex of X and a horizontal line corresponds to a vertex of Y.

Every two segments of X and Y define a ***free-space cell*** in this grid. In particular, let $C_{i,j} = C_{i,j}(X,Y)$ denote the (closed) free-space cell that corresponds to the ith edge of X and the jth edge of Y. The cell $C_{i,j}$ is located in the ith column and jth row of the grid.

It is known that the free space, for a fixed δ, inside such a cell $C_{i,j}$ (i.e., $\mathcal{D}_{\le\delta}(X,Y) \cap C_{i,j}$) is the clipping of an affine transformation of a disk to the cell [12], see the figure on the right; as such, it is convex and of constant complexity. We call the intersection of the free space with an edge of the grid a ***free space interval***. Let $I^h_{i,j}$ denote the horizontal free space interval at the top boundary of $C_{i,j}$, and $I^v_{i,j}$ denote the vertical free-space interval at the right boundary.



We define the ***complexity*** of the free-space diagram for distance δ, denoted by $\mathcal{N}_{\leq\delta}(X,Y)$, as the total number of grid cells that have a non-empty intersection with $\mathcal{D}_{\leq\delta}(X,Y)$.

Reachable free space. The Fréchet distance between X and Y is at most δ if and only if there exists a monotone path in the free space diagram between $(0,0)$ and $(1,1)$ that is fully contained in $\mathcal{D}_{\leq\delta}(X,Y)$. The ***reachable free space*** of two curves X and Y, denoted by $\mathcal{R}_{\leq\delta}(X,Y)$, is the set of points in $\mathcal{D}_{\leq\delta}(X,Y)$ that are reachable from $(0,0)$ by a monotone path. Let the ***reachability intervals*** $R^h_{i,j} \subseteq I^h_{i,j}$ and $R^v_{i,j} \subseteq I^v_{i,j}$ consist of the points $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p})$ on the boundary of $C_{i,j}$ that are reachable by a monotone path from $(0,0)$ to p.

Fréchet matching. A reparameterization that realizes the Fréchet distance (i.e., a reparametrization of width $d_\mathcal{F}(X,Y)$) does not always exist. Figure 2.1 shows such a case. The depicted path is not strictly monotone and thus does not correspond to a reparameterization. However, it can be perturbed to obtain a reparameterization which has a width that is arbitrarily close to $d_\mathcal{F}(X,Y)$. We call a monotone path from $(0,0)$ to $(1,1)$ in the free-space diagram, which is not necessarily strictly monotone, a ***matching*** between the two curves. Lifting the path onto the height function that represents the elevations over the parametric space, we call the maximal elevation reached by the path the ***width*** of the matching. We write $X \Leftrightarrow_\delta Y$ to denote a matching of width δ and we omit δ from the notation if it is clear from the context.

Free-space events. We denote with $\mathbf{b}(\mathsf{p},\delta)$ the ball of radius δ centered at a point p. To compute the Fréchet distance consider increasing δ from 0 to ∞. As δ increases, structural changes happen to the free-space diagram. We are interested in the radii

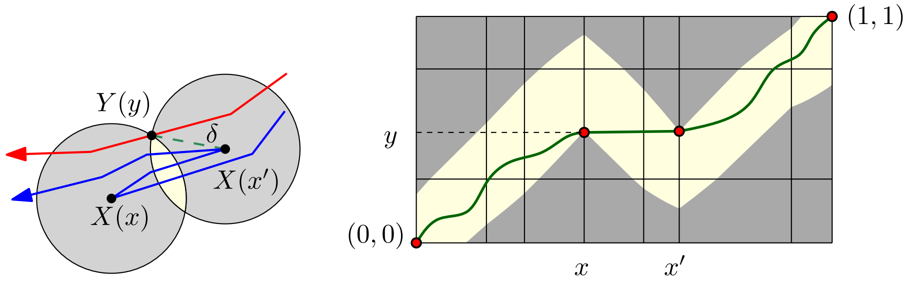
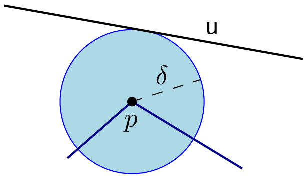


Figure 2.1: Two curves X and Y and their free-space diagram with a matching indicated, i.e., a monotone path. In this example $d_\mathcal{F}(X,Y)$ coincides with a vertex-vertex-edge event.

(i.e., the value of δ) of these events. These values are called ***critical values***. For a set of numbers U, an ***atomic interval*** is a maximum interval of $\mathbb{R}$ that does not contain any point of U in its interior. The Fréchet distance can be computed using a binary search on the atomic intervals of the critical values.



Consider a segment $\mathsf{u} \in X$ and a vertex $\mathsf{p} \in Y$, a ***vertex-edge event*** corresponds to the minimum value δ such that u is tangent to $\mathbf{b}(\mathsf{p}, \delta)$. In the free-space diagram, this corresponds to the event that a free-space interval consists of one point only. The line supporting this boundary edge corresponds to p, and the other dimension corresponds to u. Naturally, the event could happen at a vertex of u. The second type of event, a ***vertex-vertex-edge event*** (also called monotonicity event), corresponds to the common distance of two vertices on one curve to the intersection point of their bisector with a segment on the other curve. See Figure 2.1 for an example. The third type of event happens when two corresponding endpoints of the curves are at distance δ to each other. More generally, we call the the distances between any pair of points of $V_X \cup V_Y$ ***vertex-vertex event*** values. Analogously, we refer to the distance between two edges as ***edge-edge event*** value. This type of event happens if two edges intersect in the plane or in higher dimension pass close to each other.[1] Note that each one of the described critical values can be computed in $O(1)$ time.

2.4 Alt and Godau's algorithm

It is instructive to give a short summary of the algorithm by Alt and Godau which computes the Fréchet distance. The basic layout has been adopted by all subsequent algorithms to date, and the concepts introduced in their paper, which we discussed in the previous section, are now standard in the literature.

So, let $X : [0, 1] \to \mathbb{R}^d$ be a polygonal curve with n vertices and let $Y : [0, 1] \to \mathbb{R}^d$ be a polygonal curve with m vertices. One can compute the Fréchet distance between X and Y in $O(nm \log nm)$ time using the algorithm sketched below [12].

Algorithm. The algorithm consists of two parts:

(A) a decision procedure **Decider**(X, Y, δ) to answer the question whether the Fréchet distance between X and Y is smaller or equal than a given value δ and

(B) an algorithm that invokes **Decider**(X, Y, δ) in a search for the minimum value of δ for which the decision procedure returns "yes".

The decision procedure **Decider**(X, Y, δ) computes the δ-free space diagram and searches for a monotone path from $(0, 0)$ to $(1, 1)$ which stays inside the free space. Such a path can be computed, if it exists, in $O(nm)$ time by dynamic programming. The path also encodes the matching of width δ. To facilitate the search in (B), Alt and Godau identify critical values (see Section 2.3) which are candidate values for the Fréchet distance. These are the values of δ for which a free-space event happens. The number of vertex-edge and vertex-vertex events is bounded by $O(nm)$ and the number of vertex-vertex-edge events is bounded by $O(n^2m + m^2n)$. Each of these

[1]Edge-edge events only become relevant if one allows shortcuts on the input curves (Chapter 5).

values can be computed in $O(1)$ time. Alt and Godau showed that not all of these critical event values have to be computed. Instead of doing a binary search on the atomic intervals of the critical values, one can use a variant of the parametric search technique based on sorting. The authors note that this technique, even though it leads to a low asymptotic running time, is not really applicable in practice because of the enormous constants involved. To avoid the parametric search, one can use a more recent algorithm by Har-Peled and Raichel [88], which has the same running time.

2.5 Notation

X	curve parametrized by $x \in [0,1]$
Y	curve parametrized by $y \in [0,1]$
x_p	x-coordinate of point p
y_p	y-coordinate of point p
$\mathrm{dist}(\mathsf{p})$	elevation of $\mathsf{p} \in [0,1]^2$
$X \oplus Y$	concatenation of X and Y
$X[x, x']$	subcurve of X from $X(x)$ to $X(x')$
$X\langle \mathsf{p}, \mathsf{p}' \rangle$	subcurve of X between the two points $\mathsf{p}, \mathsf{p}' \in X$
$\mathsf{b}(\mathsf{p}, r)$	ball centered at p with radius r
$\mathbb{S}(\mathsf{p}, r)$	sphere centered at p with radius r
$\lVert \mathsf{p} - \mathsf{q} \rVert$	Euclidean distance between points p and q
$\lVert X \rVert$	length of curve X
$d_\mathcal{F}(X, Y)$	Fréchet distance between X and Y
$d_\mathbb{S}(k, X, Y)$	k-shortcut Fréchet distance between X and Y
$d_\mathbb{S}(X, Y)$	shortcut Fréchet distance between X and Y
$X \Leftrightarrow_\delta Y$	matching of width δ between X and Y
$\mathcal{D}_{\leq\delta}$	δ-free space
$\mathcal{R}_{\leq\delta}$	reachable free space
$\mathcal{N}_{\leq\delta}$	complexity of the δ-free space diagram
$C_{i,j}$	free space cell in the ith column and the jth row
$I^v_{i,j}$	vertical free space interval at the right side of $C_{i,j}$
$I^h_{i,j}$	horizontal free space interval at the top side of of $C_{i,j}$
$R^v_{i,j}$	vertical reachability interval at the right side of $C_{i,j}$
$R^h_{i,j}$	horizontal reachability interval at the top side of $C_{i,j}$
V_X	vertex set of polygonal curve X
$\mathcal{D}(\mathsf{P})$	pairwise distances of elements in point set P

CHAPTER 3

Approximating the Fréchet distance

In this chapter we study the problem of approximating the Fréchet distance between two given polygonal curves X and Y in $\mathbb{R}^d$. In Section 3.1, we introduce the notion of c-packed curves, and study their behavior under simplification. In Section 3.2, we describe the basic elements of the approximation algorithm. We present an approximate decider procedure which uses curve simplification to reduce the complexity of the input curves. In Section 3.3 we analyze the correctness and running time of the resulting algorithm for c-packed curves. In Sections 3.4 and 3.5 we extend the analysis to other families of curves, in particular to low-density curves and κ-bounded curves. In Section 3.6, we extend the algorithm to closed curves. As an additional result, we show in Section 3.7 that the outlines of fat objects are c-packed. We conclude with a discussion in Section 3.8.

3.1 On c-packed curves

We introduce a new family of curves, called c-packed curves, for which we can approximate the Fréchet distance quickly, given that the constant c is a small. We expect this to be the case for most curves arising in practical applications. We prove that the free-space complexity is linear for any two c-packed curves X and Y, if they are simplified to the appropriate resolution. This will imply that our algorithm works in near-linear time for c-packed curves, which is one of our main results.

3.1.1 Definition and basic properties

Definition 3.1.1 A curve X in $\mathbb{R}^d$ is ***c-packed*** if for any point p in $\mathbb{R}^d$ and any radius $r > 0$, the total length of X inside the ball $\mathbf{b}(\mathsf{p}, r)$ is at most cr.

Comparison to other models. The boundary of convex polygons, algebraic curves of bounded maximum degree, the boundary of (α, β)-covered shapes [74], and the boundary of γ-fat shapes [50] are all c-packed. Indeed, the boundaries of (α, β)-covered shapes and γ-fat shapes are assumed to be formed by a bounded number of algebraic curves of bounded maximum degree. If one removes the requirement that a γ-fat curve be of bounded descriptive complexity, then also fractal curves, like the Koch's snowflake, which can have infinite length within a bounded area, can be fat [29]. Naturally, these curves cannot be c-packed. Interestingly, one can show that (α, β)-covered polygons are c-packed even if they have unbounded complexity, see Section 3.7 and also the result of Bose *et al.* [29]. It is easy to verify that c-packed curves are also low-density [56], but a low-density curve might not be c-packed, for any constant c, see Section 3.4.

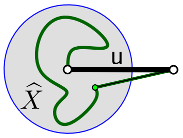
3.1.2 Curve simplification

During the course of the algorithm, we will simplify the input curves. A simplification of a polygonal curve X is a curve which is similar to X, but has fewer vertices. We suggest a straightforward greedy algorithm for curve simplification, which is sufficient for our purposes. We will see that the simplification algorithm preserves the c-packedness up to a constant factor, see Lemma 3.1.5. We comment that Agarwal *et al.* [4] suggested a more aggressive (but slightly slower and more complicated) simplification algorithm that could be used instead.

Algorithm 3.1.2 Given a polygonal curve $X = \mathsf{p}_1\mathsf{p}_2\mathsf{p}_3 \ldots \mathsf{p}_k$ and a parameter $\mu > 0$, consider the following simplification algorithm: First mark the initial vertex p_1 and set it as the current vertex. Now scan the polygonal curve from the current vertex until it reaches the first vertex p_i that is within distance at least μ from the current vertex. Mark p_i and set it as the current vertex. Repeat this until reaching the final vertex of the curve, and also mark this final vertex. Consider the curve that connects only the marked vertices, in their order along X. We refer to the resulting curve as the ***μ-simplification*** of X and we denote it with $X' = \mathbf{simpl}(X, \mu)$.

The simplified curve has the useful property that all its edges are of length at least μ, except for the last edge, which might be shorter. For simplicity of exposition, we assume that the last segment in the simplified curve also has length at least μ. Our arguments can be easily modified to handle the case that the last edge has length less than μ. The next lemma does not require the input curve to be c-packed.

Lemma 3.1.3 *Any polygonal curve X in $\mathbb{R}^d$ is within Fréchet distance μ to its μ-simplified curve; that is, $d_{\mathcal{F}}\Big(X, \mathbf{simpl}(X, \mu)\Big) \le \mu$, for any $\mu \ge 0$.*

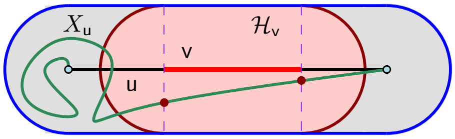


Proof: Consider a segment u of $\mathbf{simpl}(X, \mu)$ and the portion $\widehat{X}$ of X that corresponds to it. Clearly, all the vertices of $\widehat{X}$ are contained inside a ball of radius μ centered at the first endpoint of u visited by X, except the last vertex of $\widehat{X}$. As such, one can parameterize u and $\widehat{X}$, such that initially the point stays on the vertex of u while visiting all vertices of $\widehat{X}$ (except the last one), and then simultaneously move on u and the last segment of $\widehat{X}$, in such a way that the distance is always at most μ. $\square$

We now study the class of c-packed curves under the curve simplification described above. The following two lemmas testify that when curve simplification is applied to a c-packed curve, the packedness constant increases by at most a constant factor independent of the simplification parameter.

Lemma 3.1.4 *Let X be a curve in $\mathbb{R}^d$, let $\mu > 0$ be a parameter, and let $X' =$ $\mathbf{simpl}(X, \mu)$ be the simplified curve. Then $\|X \cap \mathbf{b}(\mathsf{p}, r + \mu)\| \geq \|X' \cap \mathbf{b}(\mathsf{p}, r)\|$ for any ball $\mathbf{b}(\mathsf{p}, r)$.*

Proof: Let u be a segment of X' that intersects $\mathbf{b}(\mathsf{p}, r)$ and let $\mathsf{v} = \mathsf{u} \cap \mathbf{b}(\mathsf{p}, r)$ be this intersection. Let X_u be the portion of X that got simplified into u. Observe that X_u is a polygonal curve that lies inside a hippodrome of radius μ around u; that is, $X_\mathsf{u} \subseteq \mathcal{H}_\mathsf{u} = \mathsf{u} \oplus \mathbf{b}(0, \mu)$, where $\oplus$ denotes the Minkowski sum of the two sets, see the figure on the right.



Erect two hyperplanes passing through the endpoints of v that are orthogonal to v, and observe that X_u must intersect both hyperplanes. Hence, we conclude that the portions of X_u in the hippodrome $\mathcal{H}_\mathsf{v} = \mathsf{v} \oplus \mathbf{b}(0, \mu)$ are of length at least $\|\mathsf{v}\|$. Clearly, $\mathsf{v} \subseteq \mathbf{b}(\mathsf{p}, r)$ implies that $\mathcal{H}_\mathsf{v} \subseteq \mathbf{b}(\mathsf{p}, r + \mu)$, which in turn implies that $X_\mathsf{u} \cap \mathcal{H}_\mathsf{v} \subseteq \mathbf{b}(\mathsf{p}, r + \mu)$ and thus $\|X_\mathsf{u} \cap \mathbf{b}(\mathsf{p}, r + \mu)\| \geq \|\mathsf{v}\|$.

Summing over all segments v in $X' \cap \mathbf{b}(\mathsf{p}, r)$ implies the claim. □

Lemma 3.1.5 *Let X be a c-packed curve in $\mathbb{R}^d$, let $\mu > 0$ be a parameter, and let $X' = \mathbf{simpl}(X, \mu)$ be the simplified curve. Then X' is a $6c$-packed curve.*

Proof: Let $\mu = d_\mathcal{F}(X, X')$. Assume, for the sake of contradiction, that $\|X' \cap \mathbf{b}(\mathsf{p}, r)\| > 6cr$ for some $\mathbf{b}(\mathsf{p}, r)$ in $\mathbb{R}^d$. If $r \geq \mu$, then set $r' = 2r$ and Lemma 3.1.4 implies that

$$\|X \cap \mathbf{b}(\mathsf{p}, r')\| \geq \|X \cap \mathbf{b}(\mathsf{p}, r + \mu)\| \geq \|X' \cap \mathbf{b}(\mathsf{p}, r)\| > 6cr = 3cr',$$

which contradicts the fact that X is c-packed.

If $r < \mu$ then let U denote the segments of X' intersecting $\mathbf{b}(\mathsf{p}, r)$ and let $k = |U|$. Observe that $k > 6cr/2r = 3c$, as any segment can contribute at most $2r$ to the length of X' inside $\mathbf{b}(\mathsf{p}, r)$. Therefore we have that

$$\|X' \cap \mathbf{b}(\mathsf{p}, 2\mu)\| \geq \|X' \cap \mathbf{b}(\mathsf{p}, r + \mu)\| \geq \|U \cap \mathbf{b}(\mathsf{p}, r + \mu)\| \geq k\mu,$$

since every segment of the simplified curve X' has a minimal length of μ. By Lemma 3.1.4, this implies that $\|X \cap \mathbf{b}(\mathsf{p}, 3\mu)\| \geq \|X' \cap \mathbf{b}(\mathsf{p}, 2\mu)\| \geq k\mu > 3c\mu$, which is a contradiction to the c-packedness of X.

□

3.1.3 Bounding the free-space complexity

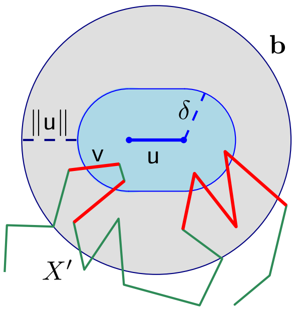
In the following, we are interested in the maximum complexity of the reachable free space of two c-packed curves when considering any radius δ and simplifying the curves with radius $\varepsilon\delta$. The reasons will become apparent only shortly after, in Lemma 3.2.3

and Lemma 3.2.4, where we show that the simplification radius chosen this way enables us to either (i) compute a $(1+\varepsilon)$-approximation of the Fréchet distance, or (ii) solve the decision problem exactly using the simplified curves (see Section 3.2.3.6).

Recall from Section 2.3 that the free-space complexity $\mathcal{N}_{\leq\delta}$ is the number of free-space cells that have non-empty intersection with the δ-free space (see Section 2.3).

Lemma 3.1.6 *Given two c-packed curves X and Y with a total number of n vertices in $\mathbb{R}^d$, and parameters $\varepsilon > 0$ and $\delta > 0$. For $\mu = \varepsilon\delta$, let $X' = \mathbf{simpl}(X,\mu)$ and $Y' = \mathbf{simpl}(Y,\mu)$ denote their simplifications. It holds that $\mathcal{N}_{\leq\delta}(X',Y') \leq n(9c + 6c/\varepsilon)$.*

Proof: A free space cell of $\mathcal{D}_{\leq\delta}(X',Y')$ corresponds to two segments $\mathsf{u} \in X'$ and $\mathsf{v} \in Y'$.



The free space in this cell is non-empty if and only if there are two points $\mathsf{p} \in \mathsf{u}$ and $\mathsf{q} \in \mathsf{v}$ such that $\|\mathsf{p}-\mathsf{q}\| \leq \delta$. We charge this pair of points to the shorter of the two segments. We claim that a segment cannot be charged too many times.

Indeed, consider a segment $\mathsf{u} \in X'$, and consider the ball $\mathbf{b}$ of radius $r = (3/2)\,\|\mathsf{u}\| + \delta$ centered at the midpoint of u, see the figure on the right. Every segment $\mathsf{v} \in Y'$ that participates in a close pair as above and charges u for it, is of length at least $\|\mathsf{u}\|$, and the length of $\mathsf{v} \cap \mathbf{b}$ is at least $\|\mathsf{u}\|$. Since Y' is $6c$-packed, by Lemma 3.1.5, we have that the number of such charges is at most

$$c' = \frac{\|Y' \cap \mathbf{b}\|}{\|\mathsf{u}\|} \leq \frac{6cr}{\|\mathsf{u}\|} = \frac{6c((3/2)\|\mathsf{u}\| + \delta)}{\|\mathsf{u}\|} \leq 9c + \frac{6c\delta}{\mu} = 9c + \frac{6c}{\varepsilon},$$

since $\|\mathsf{u}\| \geq \mu$ and $\delta = \mu/\varepsilon$. We conclude that there are at most $c'n$ free-space cells that contain a point of $\mathcal{D}_{\leq\delta}$. $\square$

3.2 The algorithm

In this section we devise the basic approximation algorithm for the Fréchet distance between two given polygonal curves. In the general layout we follow Alt and Godau (see Section 2.4) by first defining a decision procedure and then a search strategy which uses the decision procedure. In the section after (Section 3.3) we then analyze the correctness and running time of the resulting algorithm assuming that the input curves are c-packed and show how to use the algorithm to get a faster approximation algorithm.

3.2.1 Computing the reachable free space

The reachable free space, denoted by $\mathcal{R}_{\leq\delta}$ (see Section 2.3), has finite complexity inside each grid cell, and we need to describe it only for the grid cells that have non-empty intersection with $\mathcal{R}_{\leq\delta}$. Clearly, generating only those grid cells is sufficient to decide if there is a monotone path between $(0,0)$ and $(1,1)$, which is equivalent to deciding if the Fréchet distance between X and Y is smaller or equal to δ. In particular, to fully describe $\mathcal{R}_{\leq\delta}$, we will specify the reachability intervals $R^h_{i,j}$ and $R^v_{i,j}$ for each cell $C_{i,j}$, which describe the intersection of $\mathcal{R}_{\leq\delta}$ with the top and right boundary of $C_{i,j}$. These intervals contain all the needed information, since $\mathcal{R}_{\leq\delta} \cap C_{i,j}$ is convex.

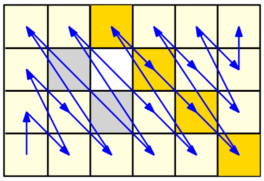
The complexity of the reachable free space, for distance δ, denoted by $\mathcal{N}_{\leq\delta}(X, Y)$, is the total number of grid cells which have non-empty intersection with $\mathcal{R}_{\leq\delta}$. One can compute this set of cells and extract an existing monotone path in $O(\mathcal{N}_{\leq\delta}(X, Y))$ time, by performing a BFS of the grid cells that visits only the reachable cells. This yields the following relatively easy result. We include the details both for the sake of completeness and because the algorithm we suggest is engagingly simple.

Lemma 3.2.1 *Given two polygonal curves X and Y in $\mathbb{R}^d$, and a parameter $\delta \geq 0$, one can compute a representation of $\mathcal{R}_{\leq\delta}(X, Y)$ in $O(\mathcal{N}_{\leq\delta}(X, Y))$ time. Furthermore, one can decide if $d_{\mathcal{F}}(X, Y) \leq \delta$, and if this is the case also extract a matching of width at most δ in $O(\mathcal{N}_{\leq\delta}(X, Y))$ time. We denote the algorithm which computes this information with* **exactDecider**(X, Y, δ).

Proof: We create a directed graph G that has a node $v(i, j)$ for every reachable free-space cell $C_{i,j}$. With each node $v(i, j)$ we store the free-space intervals $I^h_{i,j}$ and $I^v_{i,j}$ as well as the reachability intervals $R^h_{i,j} \subseteq I^h_{i,j}$ and $R^v_{i,j} \subseteq I^v_{i,j}$.

Each node $v(i, j)$ can have an outgoing edge to its right and top neighbor; an edge between these vertices exists if and only if the corresponding reachability interval between them is nonempty. In particular, a monotone path from $(0, 0)$ to a point $(x, y) \in C_{i,j}$ in $\mathcal{R}_{\leq\delta}$ corresponds to a monotone path in the graph G from $v(1, 1)$ to $v(i, j)$. Furthermore, any such monotone path has exactly $k = i + j - 2$ edges.

We compute the graph G on the fly by performing a BFS on it, starting from $v(1, 1)$, and ensuring that when the BFS visits a node $v(i, j)$ it enqueues the vertices $v(i, j + 1)$ and $v(i + 1, j)$, in this order, to the BFS queue (if they are connected to $v(i, j)$, naturally).



This implies that at any point in time, and for any k, the BFS queue contains the reachable nodes on the kth diagonal (i.e., all nodes $v(i, j)$ such that $i + j = k - 1$) of the diagram sorted from left to right. However, the same node might appear twice (consecutively) in this queue.

In every iteration, the BFS dequeues the one or two copies of the same node $v(i, j)$ and merges the two copies of the same vertex into one if necessary. Now, the one or two vertices (i.e., $v(i - 1, j)$ and $v(i, j - 1)$) that have incoming edges to $v(i, j)$ are known, as are their reachability intervals. Therefore one can compute the reachability intervals for $v(i, j)$ in constant time. Now, $v(i, j + 1)$ is enqueued if and only if the top side of the cell $C_{i,j}$ is reachable by a monotone path (i.e., $R^h_{i,j} \neq \emptyset$), and $v(i + 1, j)$ is enqueued if and only if the right side of the cell $C_{i,j}$ is reachable by a monotone path (i.e., $R^v_{i,j} \neq \emptyset$). Since $\mathcal{R}_{\leq\delta}(X, Y) \cap C_{i,j}$ is convex and of constant complexity, this can be done in constant time.

Clearly, the BFS takes time linear in the size of G and it computes the reachability information for all reachable free-space cells of $\mathcal{R}_{\leq\delta}(X, Y)$. Now, one can check if $(1, 1)$ is reachable by inspecting the last cell handled. If this cell is not the cell in the top

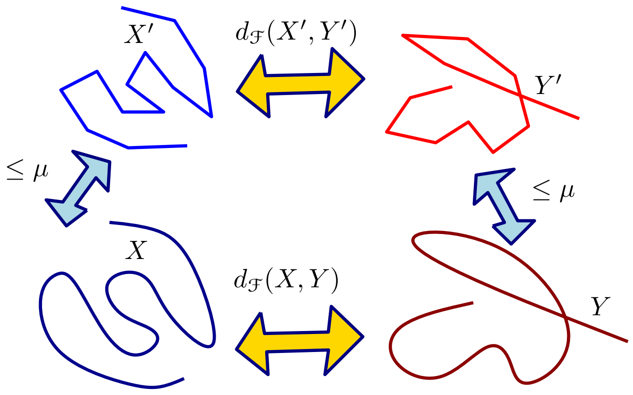


Figure 3.1: The idea of the approximate decision procedure using simplification.

right corner of the free-space diagram, then we conclude that $(1, 1)$ is not reachable. Otherwise, we check if the top right corner of this cell is monotonically reachable from the origin by inspecting the computed reachability intervals. The monotone path realizing this can be extracted in linear time, by introducing backward edges in the graph and tracing a path back to the origin. □

Observation 3.2.2 One can compute all vertex-edge events with radius at most δ in $O(\mathcal{N}_{\le\delta}(X, Y))$ time as follows. We compute the graph representation of $\mathcal{R}_{\le\delta}(X, Y)$ using **exactDecider**(X, Y, δ) (Lemma 3.2.1). Next, for each reachable cell consider the vertex-edge events at its top and right boundaries and compute their event radii. Recall that a cell boundary corresponds to an edge from the one curve and a vertex from the other curve. Clearly, any cell boundary can be used by the matching of width at most δ, if and only if the corresponding event radius is smaller or equal δ.

3.2.2 The approximate decision procedure

The idea underlying this approximate decision procedure is illustrated in Figure 3.1. We simplify the two input curves to a resolution that is (roughly) an ε-fraction of the radius we care about (i.e., δ), and we then use the exact decision procedure on these two simplified curves. Since the Fréchet distance complies with the triangle inequality and by Lemma 3.1.3, we can approximately infer the original distance from this information.

Lemma 3.2.3 *Let X and Y be c-packed polygonal curves in $\mathbb{R}^d$, and let $0 < \varepsilon \le 1$ and $\delta > 0$ be two parameters. Then there is an algorithm* **decider**$(X, Y, \varepsilon, \delta)$ *that outputs in $O(nc/\varepsilon)$ time one of the following:*

(A) "$d_{\mathcal{F}}(X, Y) \le (1 + \varepsilon)\delta$", and a matching $X \Leftrightarrow Y$ of width at most $(1 + \varepsilon)\delta$, and this happens if $d_{\mathcal{F}}(X, Y) \le \delta$.

(B) "$d_{\mathcal{F}}(X, Y) > \delta$" if $d_{\mathcal{F}}(X, Y) > (1 + \varepsilon)\delta$.

(C) If $d_{\mathcal{F}}(X, Y) \in (\delta, (1 + \varepsilon)\delta]$ then the algorithm outputs either of the above outcomes.

Proof: Set $\mu = (\varepsilon/4)\delta$. Compute in linear time the curves $X' = \textbf{simpl}(X, \mu)$ and $Y' = \textbf{simpl}(Y, \mu)$ using Algorithm 3.1.2. Let $\delta' = \delta + 2\mu$ and let $\varepsilon' = \mu/\delta' = \varepsilon/(4 + 2\varepsilon)$. Using $\textbf{exactDecider}(X', Y', \delta')$ (Lemma 3.2.1) we can decide whether $d_{\mathcal{F}}(X', Y') \leq \delta'$ in

$$O(\mathcal{N}_{\leq \delta'}(X', Y')) = O(nc/\varepsilon') = O(nc/\varepsilon)$$

time, by Lemma 3.1.6 (with δ' and ε') and since $\varepsilon/(4 + 2\varepsilon) \geq \varepsilon/6$ for $\varepsilon \leq 1$.

If so, we output the matching as a proof that

$$d_{\mathcal{F}}(X, Y) \leq d_{\mathcal{F}}(X, X') + d_{\mathcal{F}}(X', Y') + d_{\mathcal{F}}(Y', Y) \leq \delta' + 2\mu = \delta + 4(\varepsilon/4)\delta = (1 + \varepsilon)\delta.$$

On the other hand, if $d_{\mathcal{F}}(X', Y') > \delta'$, then this implies, by the triangle inequality, that

$$d_{\mathcal{F}}(X, Y) \geq d_{\mathcal{F}}(X', Y') - d_{\mathcal{F}}(X, X') - d_{\mathcal{F}}(Y', Y) > \delta' - 2\mu = \delta.$$

Therefore, the algorithm outputs "$d_{\mathcal{F}}(X, Y) > \delta$" in this case. □

3.2.2.1 How to use the approximate decider in a binary search

In order to use Lemma 3.2.3 to perform a binary search for the Fréchet distance, we can turn the approximate decision procedure into a precise one as follows.

Lemma 3.2.4 *Let X and Y be two c-packed polygonal curves in $\mathbb{R}^d$, and let $0 < \varepsilon \leq 1$ and $\delta \geq 0$ be two parameters. Then, there is an algorithm* $\textbf{Decider}(X, Y, \delta, \varepsilon)$ *that, in $O(cn/\varepsilon)$ time, returns one of the following outputs: (i) a $(1 + \varepsilon)$-approximation to $d_{\mathcal{F}}(X, Y)$, (ii) $d_{\mathcal{F}}(X, Y) < \delta$, or (iii) $d_{\mathcal{F}}(X, Y) > \delta$. The answer returned is correct.*

Proof: If $\delta = 0$, then we can determine in $O(n)$ time if the two curves are identical and return either $d_{\mathcal{F}}(X, Y) = \delta$ or $d_{\mathcal{F}}(X, Y) > \delta$ correspondingly. So assume that $\delta > 0$. Let $\delta' = \delta/(1 + \varepsilon')$, for $\varepsilon' = \varepsilon/3$. We call $\textbf{decider}(X, Y, \varepsilon', \delta)$ (see Lemma 3.2.3). If the call returns "$d_{\mathcal{F}}(X, Y) > \delta$", then we return this result.

Otherwise, we call $\textbf{decider}(X, Y, \varepsilon', \delta')$. If it returns that "$d_{\mathcal{F}}(X, Y) \leq (1 + \varepsilon')\delta'$" then $d_{\mathcal{F}}(X, Y) \leq (1 + \varepsilon')\delta' = \delta$, and we return this result.

The only remaining possibility is that the two calls returned "$d_{\mathcal{F}}(X, Y) \leq (1+\varepsilon')\delta$" and "$d_{\mathcal{F}}(X, Y) > \delta'$". But then we have found the required approximation, since $\frac{(1+\varepsilon')\delta}{\delta'} = (1 + \varepsilon')^2 < (1 + \varepsilon)$ for $0 < \varepsilon \leq 1$.

□

3.2.3 Searching for the Fréchet distance

In the previous sections we studied the decision problem. In this section, we will devise a search strategy to find the minimal value of δ for which the decision procedure returns "yes". The resulting algorithm is described in the following section (Section 3.2.4).

3.2.3.1 Searching in a fixed interval

It is straightforward to perform a binary search on an interval $[\alpha, \beta]$ to approximate the value of the Fréchet distance, if it falls inside this interval. Indeed, partition this interval into subintervals of length $\varepsilon\alpha$ and perform a binary search to find the interval that contains the Fréchet distance. There are $O(\beta/\varepsilon\alpha)$ intervals, and this would require $O(\log(\beta/\varepsilon\alpha))$ calls to **Decider**. By using exponential subintervals, one can do slightly better, as testified by the following lemma.

Lemma 3.2.5 *Given two curves X and Y in $\mathbb{R}^d$, a parameter $0 < \varepsilon \leq 1$, and an interval $[\alpha, \beta]$ with $\alpha, \beta > 0$, one can perform a binary search in $[\alpha, \beta]$ and obtain a $(1+\varepsilon)$-approximation to $d_{\mathcal{F}}(X, Y)$ if $d_{\mathcal{F}}(X, Y) \in [\alpha, \beta]$, or report that $d_{\mathcal{F}}(X, Y) \notin [\alpha, \beta]$. The algorithm, denoted by* **searchInterval**$(X, Y, [\alpha, \beta], \varepsilon)$*, takes $O\left(\log \dfrac{\log(\beta/\alpha)}{\varepsilon}\right)$ calls to* **Decider**.

Proof: Let $\alpha_i = \alpha(1+\varepsilon)^i$ for $i = 0, \ldots, M = \lfloor \log_{1+\varepsilon}(\beta/\alpha) \rfloor$ and $\alpha_{M+1} = \beta$. Perform a binary search, using **Decider**$(X, Y, \delta, \varepsilon)$ to find the two values α_i and α_{i+1} such that $\alpha_i \leq \delta = d_{\mathcal{F}}(X, Y) \leq \alpha_{i+1}$. Since $\alpha_{i+1} = (1+\varepsilon)\alpha_i$, we conclude that we found the required approximation.

It might be that during this procedure one of the calls to **Decider**$(X, Y, \delta, \varepsilon)$ found the required approximation, and in this case we abort the binary search and just return this approximation.

This process requires $O(\log M) = O\big(\log \log_{1+\varepsilon}(\beta/\alpha)\big)$ calls to **Decider**. Observe that

$$M = \log_{1+\varepsilon} \frac{\beta}{\alpha} = \frac{\ln(\beta/\alpha)}{\ln(1+\varepsilon)} = O\left(\frac{1}{\varepsilon} \log \frac{\beta}{\alpha}\right).$$

Indeed, $e^{x/2} \leq 1 + x \leq e^x$ for $x \in [0, 1]$, and this implies that $x/2 \leq \ln(1+x) \leq x$, which is the inequality used above. □

3.2.3.2 Searching over events

Clearly, the procedure **searchInterval**$(X, Y, [\alpha, \beta], \varepsilon)$ alone does not suffice to solve our main problem, since the interval of distances we are searching over might have arbitrarily large "spread" (i.e., $\log \beta/\alpha$ might be arbitrarily large). However, the Fréchet distance must be sufficiently close to a free-space event in one of the "approximate" diagrams, i.e., a free-space diagram of the two simplified curves. Thus, we can identify two kinds of critical values to search over, which are candidate values for the approximate Fréchet distance. These are the events where (i) the simplification of an input curve changes, or (ii) the reachability within the approximate free-space diagram changes (i.e., a free-space event; see Section 2.1).

The traditional solution to overcome this problem is to use parametric search. However, in our case, since we are only interested in approximation, we can use a simpler, "approximate", search. It is sufficient to search over a set of values which approximate the event values by a constant factor, since we will use Lemma 3.2.5 to refine the resulting search interval in the main algorithm. In effect, we will use this lemma to turn a constant factor approximation of the Fréchet distance into a $(1+\varepsilon)$-approximation.

Algorithm 3.2.6 Let **searchEvents**(X, Y, Z, ε) denote the algorithm that performs a binary search over the values of Z, to compute the atomic interval of Z that contains the Fréchet distance between X and Y. This procedure uses **Decider** (Lemma 3.2.4) to perform the decisions during the search. The decision procedure may also return a $(1+\varepsilon)$-approximation to $d_{\mathcal{F}}(X, Y)$ if it fails to make an exact decision.

3.2.3.3 Simplification events

Consider the events when the simplified curves change, see Algorithm 3.1.2. Consider the set of all pairwise distances between vertices of X and Y (i.e., $V_X \cup V_Y$). Observe that it breaks the real line into $\binom{n}{2} + 1$ atomic intervals, such that in each such interval the simplification does not change. Thus **simpl**(X, μ) (resp. **simpl**(Y, μ)) might result in $O(n^2)$ different curves depending on the value of μ, where n is the total number of vertices of X and Y. As a first step we would therefore like to use **searchEvents** (Algorithm 3.2.6) to perform a binary search over those distances to find the atomic interval that contains the required Fréchet distance. Naively, this would require us to perform distance selection. However, it is believed that exact distance selection requires $\Omega(n^{4/3})$ time in the worst case [76]. To overcome this we will perform an approximate distance selection, as suggested by Aronov *et al.* [20].

Lemma 3.2.7 *Given a set* P *of* n *points in* $\mathbb{R}^d$*, let* $\mathcal{D}(\mathsf{P})$ *be the set of all pairwise distances of points in* P*. Then, one can compute in* $O(n \log n)$ *time a set* Z *of* $O(n)$ *numbers, such that for any* $y \in \mathcal{D}(\mathsf{P})$*, there exist numbers* $x, x' \in Z$ *such that* $x \le y \le x' \le 2x$*. Let* **approxDistances**(P) *denote this algorithm.*

Proof: Compute an 8-well-separated pairs decomposition of P. Using the algorithm of Callahan and Kosaraju [41] this can be done in $O(n \log n)$ time, and it results in a set of pairs of subsets $\{(A_1, B_1), \ldots, (A_m, B_m)\}$, where $m = O(n)$, such that for any two points $\mathsf{p}, \mathsf{q} \in \mathsf{P}$ there exists a pair (A_i, B_i) in the above decomposition, such that: (i) $\mathsf{p} \in A_i$ and $\mathsf{q} \in B_i$ (or vice versa), and (ii) $\max(\operatorname{diam}(A_i), \operatorname{diam}(B_i)) \le \min_{\mathsf{p}_i \in A_i, \mathsf{q}_i \in B_i} \|\mathsf{p}_i - \mathsf{q}_i\| / 8$.

This implies that the distance of any pair of points in A_i and B_i, respectively, are the same up to a small constant. As such, for every pair (A_i, B_i), for $i = 1, \ldots, m$, we pick representative points $\mathsf{p}_i \in A_i$ and $\mathsf{q}_i \in B_i$, and set $\ell_i = (3/4) \|\mathsf{p}_i - \mathsf{q}_i\|$. Let $Z = \{\ell_1, \ldots, \ell_m, 2\ell_1, \ldots, 2\ell_m\}$ be the computed set of values.

Consider any pair of points $\mathsf{p}, \mathsf{q} \in \mathsf{P}$. For the specific pair (A_i, B_i) that contains the pair of points p and q that we are interested in, we have that

$$\ell_i = (3/4) \|\mathsf{p}_i - \mathsf{q}_i\| \le \|\mathsf{p}_i - \mathsf{q}_i\| - \operatorname{diam}(A_i) - \operatorname{diam}(B_i) \le \|\mathsf{p} - \mathsf{q}\|.$$

At the same time, we have that

$$\|\mathsf{p} - \mathsf{q}\| \le \|\mathsf{p}_i - \mathsf{q}_i\| + \operatorname{diam}(A_i) + \operatorname{diam}(B_i) \le (5/4) \|\mathsf{p}_i - \mathsf{q}_i\| \le 2\ell_i,$$

thus establishing the claim. □

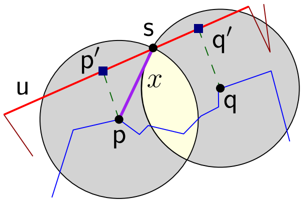
3.2.3.4 Monotonicity events

The following lemma testifies that the radius of a vertex-vertex-edge event must be "close" to either a vertex-edge event or to the distance between two vertices. Since we will approximate the vertex-vertex distances (i.e., the simplification events) and perform a binary search over them, this implies that we further only need to consider vertex-edge events. Furthermore, by Observation 3.2.2, the number of those vertex-edge events which remain in the resulting search range can be bounded by the complexity of the reachable free space.

Lemma 3.2.8 *Let x be the radius of a vertex-vertex-edge event involving vertices $\mathsf{p}, \mathsf{q} \in \mathbb{R}^d$ and a segment $\mathsf{u} \in \mathbb{R}^d$. Let $\mathcal{W}$ be the set of pairwise distances of the vertex set $V_X \cup V_Y$. Then there exists a number y such that $y/2 \leq x \leq 3y$, and y is either in $\mathcal{W}$ or y is the radius of a vertex-edge event.*

Proof: Let s be the intersection point of $\mathbb{S}(\mathsf{p}, x) \cap \mathbb{S}(\mathsf{q}, x)$ which lies on u. Let p' (resp., q') be the closest point on u to p (resp., q).

Clearly $\|\mathsf{p}' - \mathsf{q}'\| \leq \|\mathsf{p} - \mathsf{q}\|$. Indeed, let ℓ be the line supporting u. Project the line segment pq perpendicular to ℓ onto ℓ. The resulting line segment is shorter that $\|\mathsf{p} - \mathsf{q}\|$ and contains both p' and q', unless $\mathsf{p}' = \mathsf{q}'$ coincide with an endpoint of u. Furthermore, since the ε-environment around s on u is covered by $\mathbf{b}(\mathsf{p}, x) \cup \mathbf{b}(\mathsf{q}, x)$, the point s lies on the segment $\mathsf{p}'\mathsf{q}'$.



By the triangle inequality, this implies that

$$x = \|\mathsf{p} - \mathsf{s}\| \leq \|\mathsf{p} - \mathsf{p}'\| + \|\mathsf{p}' - \mathsf{s}\| \leq \|\mathsf{p} - \mathsf{p}'\| + \|\mathsf{p}' - \mathsf{q}'\| \leq \|\mathsf{p} - \mathsf{p}'\| + \|\mathsf{p} - \mathsf{q}\|.$$

A similar argument implies that

$$x = \|\mathsf{p} - \mathsf{s}\| \geq \|\mathsf{p} - \mathsf{p}'\| - \|\mathsf{p}' - \mathsf{s}\| \geq \|\mathsf{p} - \mathsf{p}'\| - \|\mathsf{p}' - \mathsf{q}'\| \geq \|\mathsf{p} - \mathsf{p}'\| - \|\mathsf{p} - \mathsf{q}\|.$$

If $\|\mathsf{p} - \mathsf{p}'\| \geq 2\|\mathsf{p} - \mathsf{q}\|$ then the above implies that $x \in [1/2, 3/2]\|\mathsf{p} - \mathsf{p}'\|$. If p' is an endpoint of u then $\|\mathsf{p} - \mathsf{p}'\|$ is in $\mathcal{W}$. Otherwise, $\|\mathsf{p} - \mathsf{p}'\|$ is the radius of the vertex-edge event between p and u. In either case, this implies the claim.

If $\|\mathsf{p} - \mathsf{p}'\| \leq 2\|\mathsf{p} - \mathsf{q}\|$ then

$$x = \|\mathsf{p} - \mathsf{s}\| \leq \|\mathsf{p} - \mathsf{p}'\| + \|\mathsf{p} - \mathsf{q}\| \leq 2\|\mathsf{p} - \mathsf{q}\| + \|\mathsf{p} - \mathsf{q}\| = 3\|\mathsf{p} - \mathsf{q}\|,$$

and of course $\|\mathsf{p} - \mathsf{q}\| \in \mathcal{W}$. Now, the two balls of radius x centered at p and q, respectively, cover the segment pq, and we have that $\|\mathsf{p} - \mathsf{q}\|/2 \leq x$, which implies the claim. □

3.2.3.5 Edge-edge events

It might happen that two long edges intersect in their middle (or in higher dimension pass close to each other), and thus contribute an isolated connected component to $\mathcal{D}_{\leq\delta}(X, Y)$. Such a connected component is a convex set lying completely in the interior of the grid cell of the two segments. In Section 2.3, we referred to the event that such a component is created in the free space as edge-edge event. Since it is not reachable by a monotone path in the diagram from $(0, 0)$, we can just ignore it. The corresponding component will grow as δ increases until it hits the boundary of the grid cell. At this point, a vertex-edge event will happen.

3.2.3.6 Searching with a fixed simplification

We will use the decision procedure in a binary search over the simplfication events for the atomic interval that contains the desired $(1+\varepsilon)$-approximation. Intuitively, when further refining the search in this interval, the simplification of the input curves carried out by **Decider**, will always yield the same simplified curves, since there are no simplification events in this interval. Thus, we have found simplifications X' and Y', such that $d_{\mathcal{F}}(X', Y')$ yields the desired $(1+\varepsilon)$-approximation. Clearly, an approximation of $d_{\mathcal{F}}(X', Y')$ suffices for our result.

Let **searchIntervalExact**$(X, Y, [\alpha, \beta], \varepsilon)$ be the variant of **searchInterval** from Lemma 3.2.5 that uses **exactDecider** (Lemma 3.2.1) directly instead of calling **Decider**. This version searches for the Fréchet distance in the given interval, but does not perform simplification before calling the decision procedure. It returns a $(1+\varepsilon)$-approximation of the Fréchet distance, given that it is contained in this interval. Note that the correctness of Lemma 3.2.5 is not affected by this modification. Similarly, let **searchEventsExact** be the variant of **searchEvents** (Algorithm 3.2.6) that uses **exactDecider** directly instead of calling **Decider**.

3.2.4 The resulting algorithm

The resulting approximation algorithm is layed out in Algorithm 3.2.9. It will be used by the final approximation algorithm as a subroutine. We first analyze this basic algorithm. We will then show how to use it, in Lemma 3.3.4 below, to get a faster approximation algorithm. The algorithm in Algorithm 3.2.9 performs numerous calls to **Decider**, with approximation parameter $\varepsilon > 0$. If any of these calls discover the approximate distance, then the algorithm immediately stops and returns the approximation. Therefore, at any point in the execution of the algorithm, the assumption is that all previous calls to **Decider** returned a direction where the optimal distance must lie. In particular, a call to **searchInterval**$(X, Y, \mathcal{I}, \varepsilon)$, would either find the approximate distance in the interval $\mathcal{I}$ and return immediately, or the desired value is outside this interval.

3.3 Analysis of the algorithm

3.3.1 Correctness

Lemma 3.3.1 *For any $x, y \in (2\alpha, \beta/2)$, for α, β computed in line 3, it holds that* **simpl**$(X, x) =$ **simpl**(X, y) *and* **simpl**$(Y, x) =$ **simpl**(Y, y).

Proof: The interval (α, β) does not contain any value of Z. Hence, by Lemma 3.2.7, $(2\alpha, \beta/2)$ does not contain any value of the pairwise distances between vertices of the vertex set of X and Y which implies that the simplification is the same for any value inside this interval. □

Lemma 3.3.2 *Given two polygonal curves X and Y, and a parameter $0 < \varepsilon \leq 1$, the algorithm* **approxFréchet**(X, Y, ε) *computes a $(1+\varepsilon)$-approximation to $d_{\mathcal{F}}(X, Y)$.*

Algorithm 3.2.9 approxFréchet(X, Y, ε)

```
Input: polygonal curves $X$ and $Y$; approximation error $\varepsilon \in (0, 1]$
 1: Assert that $d_{\mathcal{F}}(X, Y) > 0$ by checking if $X$ is identical to $Y$
 2: $Z \leftarrow$ approxDistances$(\mathsf{P})$, where $\mathsf{P} = V_X \cup V_Y$ (Lemma 3.2.7)
 3: $[\alpha, \beta] \leftarrow$ searchEvents$(X, Y, Z, \varepsilon)$ (Algorithm 3.2.6)
 4: If $(\alpha > 0)$ call searchInterval$(X, Y, [\alpha, 4\alpha'], \varepsilon)$, where $\alpha' = (30/\varepsilon)\alpha$
    (Lemma 3.2.5)
 5: If $(\beta > 0)$ call searchInterval$(X, Y, [\beta'/4, \beta], \varepsilon)$, where $\beta' = \beta/3$
 6: Let $X' =$ simpl$(X, \mu)$ and $Y' =$ simpl$(Y, \mu)$, for $\mu = 3\alpha$
    (Algorithm 3.1.2)
 7: Let $\mathcal{V}'$ be the set of all vertex-edge distances of $X'$ and $Y'$
 8: Compute $Z' = \{v \in \mathcal{V}' \mid \alpha' \le v \le \beta'\}$ using exactDecider$(X', Y', \beta')$
    (see Observation 3.2.2)
 9: $[\alpha'', \beta''] \leftarrow$ searchEventsExact$(X', Y', Z', \varepsilon)$ (Algorithm 3.2.6)
10: If $(\alpha'' > 0)$ call searchIntervalExact$(X', Y', [\alpha'', 4\alpha''], \varepsilon/4)$
11: If $(\beta'' > 0)$ call searchIntervalExact$(X', Y', [\beta''/4, \beta''], \varepsilon/4)$
12: The algorithm terminated in one of the lines above. Let $f$ be the matching
    $X' \Leftrightarrow Y'$, let $g$ be the matching $X \Leftrightarrow X'$ and let $h$ be the matching $Y \Leftrightarrow Y'$
    resulting from the last executed search.
13: Return the matching $X \Leftrightarrow Y$ resulting from chaining $f, g$ and $h$ and the
    resulting width $\Delta$ as an approximation to $d_{\mathcal{F}}(X, Y)$
```

Proof: If the algorithm found the approximation before line 6, then clearly it is the desired approximation, and we are done. In particular, this is the case if (i) the assertion in line 1 fails or (ii) if $4\alpha' > \beta'/4$. Otherwise, by Lemma 3.2.4 we know that $d_{\mathcal{F}}(X, Y) \in [\alpha, \beta]$ computed in line 3. By line 4 and 5 it must be that $d_{\mathcal{F}}(X, Y) \in [4\alpha', \beta'/4]$. Since $\mu = 3\alpha = (\varepsilon/10)\alpha' \le 4\alpha' \le \beta'/4$, it follows, by the triangle inequality, that

$$d_{\mathcal{F}}(X', Y') \le d_{\mathcal{F}}(X', X) + d_{\mathcal{F}}(X, Y) + d_{\mathcal{F}}(Y, Y') \le 2\mu + \beta'/4 < \beta'.$$

A similar argument shows that $d_{\mathcal{F}}(X', Y') > \alpha'$. We conclude that $d_{\mathcal{F}}(X', Y') \in [\alpha', \beta']$ and the algorithm continues the search with the simplified curves X' and Y'.

The binary search in line 9 over the vertex-edge distances of X' and Y' returns an atomic interval $[\alpha'', \beta'']$ of this set that contains $d_{\mathcal{F}}(X', Y')$. By Lemma 3.2.5, one of the two searches in line 10 and line 11 will return a $(1 + \varepsilon)$-approximation to $d_{\mathcal{F}}(X', Y')$, given that this Fréchet distance lies in one of the respective intervals that we perform the search on. We claim that this is the case.

Indeed, the interior of $[\alpha'', \beta'']$ does not contain any of

(i) $\mathcal{W}$: the pairwise distances of $V_{X'}$ and $V_{Y'}$, or
(ii) $\mathcal{V}'$: the vertex-edge distances of X' and Y'.

Therefore, the interval $[\alpha'', \beta'']$ might contain only vertex-vertex-edge events of X' and Y'. By Lemma 3.2.8, for a vertex-vertex-edge event with radius r there exists a value $y \in \mathcal{V}' \cup \mathcal{W}$ that 3-approximates r; that is, such that $y/2 \le r \le 3y$. But since there is no value of $\mathcal{V}' \cup \mathcal{W}$ in the interior of $[\alpha'', \beta'']$, and therefore, for any such $r \in [4\alpha'', \beta''/4]$, we have that $r \notin [y/3, 3y]$ for any $y \in \mathcal{V}' \cup \mathcal{W}$

We conclude that no vertex-vertex-edge event, vertex-edge event, or vertex-vertex event of X' and Y' lies in the interval $[4\alpha'', \beta''/4]$. Since the Fréchet distance must be equal to one such value, it follows that $d_{\mathcal{F}}(X', Y') \notin (4\alpha'', \beta''/4)$, but this implies that either $d_{\mathcal{F}}(X', Y') \in [\alpha'', 4\alpha'']$ or $d_{\mathcal{F}}(X', Y') \in [\beta''/4, \beta'']$. In either case, the above algorithm would have found the approximate distance (either in line 10 or 11). Thus, we found a value δ, such that $\delta \in [d_{\mathcal{F}}(X', Y'), (1+\varepsilon/4)d_{\mathcal{F}}(X', Y')]$.

By the triangle inequality we conclude that the returned value Δ satisfies

$$\begin{aligned}\Delta &\le d_{\mathcal{F}}(X, X') + \delta + d_{\mathcal{F}}(Y, Y') \le d_{\mathcal{F}}(X, X') + (1+\varepsilon/4)d_{\mathcal{F}}(X', Y') + d_{\mathcal{F}}(Y', Y)\\ &\le (1+\varepsilon/4)(2\mu + d_{\mathcal{F}}(X, Y) + 2\mu) \le 5\mu + (1+\varepsilon/4)d_{\mathcal{F}}(X, Y) \le (1+\varepsilon)d_{\mathcal{F}}(X, Y),\end{aligned}$$

since $5\mu = 15\alpha = (\varepsilon/2)(30/\varepsilon)\alpha = (\varepsilon/2)\alpha' \le (\varepsilon/2)d_{\mathcal{F}}(X, Y)$.

Note that $\Delta \ge d_{\mathcal{F}}(X, Y)$ since it is the width of a specific matching between the two curves. $\square$

3.3.2 Running time

Lemma 3.3.3 *Given two c-packed polygonal curves X and Y with a total number of n vertices in $\mathbb{R}^d$, and a parameter $0 < \varepsilon \le 1$, the running time of* **approxFréchet**(X, Y, ε) *is $O((cn/\varepsilon)\log n)$.*

Proof: Checking if the two curves are identical takes $O(n)$ time in line 1. Computing Z (and sorting it) takes $O(n \log n)$ time by Lemma 3.2.7. The calls in line 3, 4 and 5 perform $O(\log n + \log(1/\varepsilon)) = O(\log n)$ calls to **Decider**, by Lemma 3.2.5. (Here, we assume that $\varepsilon > 1/n$, otherwise we can just use the algorithm of Alt and Godau [12] since its running time is faster than our approximation algorithm in this case.) Each call to **Decider** takes $O(nc/\varepsilon)$ time, so overall this takes $O(cn/\varepsilon \log n)$ time. Computing the simplifications in line 6 with Algorithm 3.1.2 takes $O(n)$ time.

Computing and sorting the set of vertex-edge events in line 8 takes $O(N \log N)$ time, where $N = \mathcal{N}_{\le\beta'}(X', Y')$, since $|Z'| \le N$ by Observation 3.2.2. The binary search in line 9 requires $O(\log N)$ calls to the algorithm **exactDecider** (Lemma 3.2.1). The two calls to **searchIntervalExact** require $O(\log(1/\varepsilon))$ calls to **exactDecider**. Now, observe that all these calls to **exactDecider** are done with values of $\delta \in [\alpha', \beta']$. Since the complexity of the reachable free space is monotone in δ, we can bound the running time of such a call to **exactDecider** by $O(N)$. Thus, the overall running time of line 8 - 11 is $O((n + N)\log(N/\varepsilon))$.

Now, 3α and β' are both inside the interval $(2\alpha, \beta/2)$, and as such, by Lemma 3.3.1, we have that

$$X' = \mathbf{simpl}(X, 3\alpha) = \mathbf{simpl}(X, \beta')$$

and similarly $Y' = \mathbf{simpl}(Y, 3\alpha) = \mathbf{simpl}(Y, \beta')$. Therefore, we have that

$$N = \mathcal{N}_{\le\beta'}(X', Y') = \mathcal{N}_{\le\beta'}(\mathbf{simpl}(X, \beta'), \mathbf{simpl}(Y, \beta')) = O(cn),$$

by Lemma 3.1.6 (with $\delta' = \beta'$ and $\varepsilon' = 1$). Thus, line 8 - 11 take overall $T = O(cn \log(cn/\varepsilon)) = O(cn \log n)$ time, since $c \in O(n)$ and $\varepsilon > 1/n$.

Finally, in order to compute the resulting matching in line 13, we compute the matching $X \Leftrightarrow X'$ and $Y \Leftrightarrow Y'$ as described in the proof of Lemma 3.1.3 and chain[1]

[1]Note that these matchings are not necessarily one-to-one, but they can be perturbed to obtain one-to-one matchings, see Section 2.3.

them with the matching of the simplified curves $X' \Leftrightarrow Y'$. Clearly, this and computing the resulting width takes $O(n)$ time. $\square$

The running time of Lemma 3.3.3 can be slightly improved.

Lemma 3.3.4 *The algorithm* **approxFréchet** *in Algorithm 3.2.9 can be modified to run in time* $O(cn/\varepsilon + cn\log n)$

Proof: Use Lemma 3.3.3, with $\varepsilon_0 = 1/2$, to get a 2-approximation ζ for the Fréchet distance between X and Y. This takes $O(cn \log n)$ time. time. Let $\mathcal{I}_0 = [\zeta, 2\zeta]$ be the corresponding interval that contains the distance. We could call **searchInterval**$(X, Y, \mathcal{I}_0, \varepsilon)$ and get a $(1+\varepsilon)$-approximation spending an additional $O((cn/\varepsilon)\log 1/\varepsilon)$ amount of time. One can do better by starting with a "large" ε and decreasing it during the binary search for the right value performed by **searchInterval**. This is a standard idea and it was also used by Aronov and Har-Peled [19].

Indeed, assume that in the beginning of the ith step, we know that the required Fréchet distance lies in an interval $\mathcal{I}_{i-1} = [\alpha_{i-1}, \beta_{i-1}]$ and $\beta_{i-1} - \alpha_{i-1} = \|\mathcal{I}_0\|\,\varepsilon_{i-1}$, where $\varepsilon_{i-1} = 1/2^{i-1}$.

Let $\Delta_{i-1} = \|\mathcal{I}_{i-1}\| = \beta_{i-1} - \alpha_{i-1}$, and let $x_{i,j} = \alpha_{i-1} + j\Delta_{i-1}/4$, for $j = 0, 1, 2, 3, 4$. Call the procedure **Decider** on three values $x_{i,1}$, $x_{i,2}$, and $x_{i,3}$, with the approximation parameter being $c_1\varepsilon_i$, for $c_1 > 0$ being a sufficiently small constant. Based on the outcome of these three calls, we can determine in constant time which of the three intervals $\mathcal{J}_{i,1} = [x_{i,0}, x_{i,2}]$, $\mathcal{J}_{i,2} = [x_{i,1}, x_{i,3}]$, or $\mathcal{J}_{i,3} = [x_{i,2}, x_{i,4}]$ must contain the Fréchet distance. We set this interval to be $\mathcal{I}_i$.

We repeat this process for M steps, where $M = \lceil \lg 1/\varepsilon \rceil$. It is easy to verify that the final interval now provides the required approximation. The running time of the second part of this algorithm is in

$$O\left(\sum_{i=1}^{M} cn/\varepsilon_i + M\right) = O\left(cn\sum_{i=1}^{M} 2^i + M\right) = O(cn/\varepsilon)\,,$$

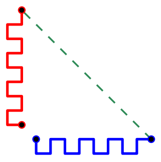
and this implies the claim. $\square$

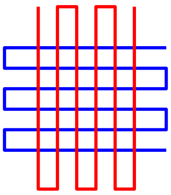
3.3.3 The result

Putting the above together, we get the following result.

Theorem 3.3.5 *Given two c-packed polygonal curves X and Y with a total of n vertices in $\mathbb{R}^d$, and a parameter $1 > \varepsilon > 0$, one can $(1+\varepsilon)$-approximate the Fréchet distance between X and Y in $O(cn/\varepsilon + cn\log n)$ time.*

Interestingly, simplification is critical for the efficiency of the above algorithm. Indeed, consider the two nicely behaved curves depicted on the right. The reachable portion of the free-space diagram of these two curves, for the distance realizing the Fréchet distance, covers a quadratic number of cells.



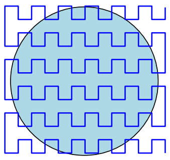


The use of simplification by itself is not sufficient to guarantee that the presented algorithm is efficient. Indeed, it might not be possible to simplify the input curves at all without losing too much information. In such contrived worst case examples, the free-space diagram still has quadratic complexity due to the inherent structure of the curves. See the figure to the left for one such example. Thus, we need the input curves to be c-packed or to satisfy some other realistic input model. In the next section we will analyze the relative free space complexity using existing realistic input models and prove the efficiency of the above algorithm also for these known classes of curves.

3.4 Extension to low-density curves

Definition 3.4.1 A polygonal curve X in $\mathbb{R}^d$ is ***ϕ-low-density*** if any ball $\mathbf{b}(\mathsf{p}, r)$ intersects at most ϕ segments of X that are longer than r.

First, observe that this input model is less restrictive than the input model which describes c-packed curves. It can be easily seen by a simple packing argument that a polygonal c-packed curve is ϕ-low-density, for $\phi = 2c$. For any ball $\mathbf{b} = \mathbf{b}(\mathsf{p}, r)$, consider the ball with the same center that has radius $r' = 2r$. Any edge intersecting $\mathbf{b}$ that is longer than r must contribute at least r to the length of the intersection of the curve with the larger ball, which is bounded by cr'. There can be at most $cr'/r = 2c$ edges of this type.



A curve that is low-density, however, is not necessarily c-packed for a small value of c. Indeed, a low-density curve X might have an arbitrarily long intersection with a ball by having sufficiently small segments, see the figure on the right. However, in this case X must have many vertices in the areas where its length cannot be bounded, as we will show in the following section.

3.4.1 Low density curves can be long only if they pay for it

Claim 3.4.2 *Let X be a ϕ-low-density polygonal curve, and let C be a hypercube in $\mathbb{R}^d$ with side length ℓ. Then, the number of edges of length $\geq \ell$ of X that intersect C is bounded by $c_d\phi$, where $c_d = \left\lceil \sqrt{d}/2 \right\rceil^d$.*

Proof: Partition the cube C into a $D \times D \times \cdots \times D$ grid, for $D = \left\lceil \sqrt{d}/2 \right\rceil$. Clearly, any edge that intersects C that has length $\geq \ell$ must intersect one of the hypercubes in this grid. A hypercube of this grid has diameter

$$\frac{\sqrt{d}\ell}{D} \leq \frac{\sqrt{d}\ell}{\sqrt{d}/2} \leq 2\ell,$$

and is included in a ball of radius ℓ. Thus, a hypercube in this grid intersects at most ϕ such long edges. We conclude that there can be at most ϕD^d long edges intersecting C. □

Lemma 3.4.3 *Let X be a ϕ-low-density curve in $\mathbb{R}^d$, which is not $O(\phi)$-packed and let C be a cube in $\mathbb{R}^d$ with side length r. Let $\alpha = \|X \cap \mathsf{C}\|$. There must be at least $\Omega((\alpha/r)^{1+1/(d-1)})$ vertices of X contained in $3\mathsf{C}$, where $3\mathsf{C}$ is the scaling of C by a factor of 3 around its center.*

Proof: [2] Let X be the set of edges of X that intersect C. Let X_0 be the set of segments of X of length at least $\operatorname{diam}(\mathsf{C})$. Similarly, let X_i be all the segments in X which have a length in the interval $[\operatorname{diam}(\mathsf{C})/2^i, \operatorname{diam}(\mathsf{C})/2^{i-1}]$, for $i = 1, \ldots, \infty$.

We have that

$$\alpha = \|X \cap \mathsf{C}\| \leq \sum_{i=0}^{\infty} \frac{|X_i| \cdot \operatorname{diam}(\mathsf{C})}{2^{i-1}} \leq r\sqrt{d} \sum_{i=0}^{\infty} \frac{|X_i|}{2^{i-1}}. \tag{3.1}$$

Similarly, let G_i be the partition of C into a grid by subcubes with side length $r/2^i$, for $i = 1, \ldots, \infty$. Associate each segment u of X_i with some cube of G_i that it intersects. By Claim 3.4.2, no cube can be associated with more than $c_d\phi$ cubes since X is ϕ-low-density. Therefore,

$$|X_i| \leq c_d \phi 2^{di}. \tag{3.2}$$

Now, let $c = c_d\phi\sqrt{d}$ and let $K = \left\lfloor \frac{1}{d-1} \lg \frac{\alpha}{cr} \right\rfloor - 3$. Since X is not $O(\phi)$-packed, we can assume that $K > 0$. We have by Eq. (3.1) and Eq. (3.2) that

$$r\sqrt{d} \sum_{i=0}^{K} \frac{|X_i|}{2^{i-1}} \leq cr \sum_{i=0}^{K} \frac{2^{di}}{2^{i-1}} \leq cr \sum_{i=0}^{K} 2 \cdot 2^{(d-1)i} \leq 4cr2^{(d-1)K} \leq \frac{\alpha}{2}.$$

This means that the edges of the curve associated with the first K levels of the grid partition can have a total length of at most $\alpha/2$. Therefore, at least half of the length of the curve is distributed over levels which lie beyond K. Note, however, that every segment of X_i, for $i > K$, can contribute at most $\operatorname{diam}(C)/2^{K+1}$ to the total length of $\|X \cap \mathsf{C}\|$. Setting $N = \sum_{i=K+1}^{\infty} |X_i|$ we conclude that

$$N \frac{\operatorname{diam}(C)}{2^{K+1}} \geq \frac{\alpha}{2} \implies N = \Omega\Big(\frac{\alpha}{r} \cdot 2^K\Big) = \Omega\bigg(\frac{\alpha}{r} \cdot \Big(\frac{\alpha}{r}\Big)^{1/(d-1)}\bigg) = \Omega\bigg(\Big(\frac{\alpha}{r}\Big)^{1+1/(d-1)}\bigg).$$

Now, each of these segments of X_i, for $i > K$, has its endpoints inside $3\mathsf{C}$, thus implying the claim. □

Observation 3.4.4 The bound in Lemma 3.4.3 is tight. For any $m > 0$ and any $d > 0$, consider the integer grid in $\mathbb{R}^d$ with coordinates in the range $1, \ldots, m$, and compute a path that visits all these grid points using only the grid edges of unit length, which is clearly possible.

Now, the resulting curve is 2^d-low-density and has length $\alpha = m^d - 1$ and its diameter is $r = \sqrt{d}m$. Lemma 3.4.3 implies that it has $\Omega\big((\alpha/r)^{d/(d-1)}\big) = \Omega\big(m^d\big)$ vertices. Since this grid has m^d vertices, this is tight.

[2]Note that an alternative proof to Lemma 3.4.3 can be found in [71].

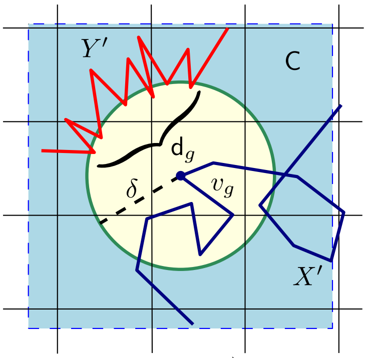
3.4.2 Accounting for many reachable free-space cells

If many columns of the free-space diagram of the two simplified low-density curves contain a linear number of reachable cells, then the curve must be "long" in the vicinity of the edges corresponding to those columns, since the simplification ensures a minimal edge length. A similar argument holds for the rows. Therefore, using Lemma 3.4.3, we can charge the additional reachable cells to vertices of the original curves. This yields the following result.

Lemma 3.4.5 *Given two low-density curves X and Y with a total number of n vertices in $\mathbb{R}^d$, and parameters $0 < \varepsilon \leq 1$ and $\delta > 0$. For $\mu = \varepsilon\delta$, let $X' =$ **simpl**(X, μ) and $Y' =$ **simpl**(Y, μ) denote their simplifications. Then $\mathcal{N}_{\leq\delta}(X', Y') = O\Big(\frac{n^{2(d-1)/d}}{\varepsilon^2}\Big)$.*

Proof: It suffices to bound the number of vertex-edge pairs (p, u), where p is a vertex of X', u is an edge of Y', and the distance between p and u is at most δ (naturally, we need to apply the same argument to pairs with vertices in Y' and edges in X'). The total number of such pairs bounds the total complexity of $\mathcal{R}_{\leq\delta} = \mathcal{R}_{\leq\delta}(X', Y')$.

To account for the number of such vertex-edge pairs, where the vertices are of X' and the edges are of Y', we charge vertex-edge pairs to vertices of *both* the original curves X and Y. To this end, let $M = O\big(n^{1-2/d}/\varepsilon^2\big)$. For every vertex-edge pair (p, u) that appears in the free-space diagram $\mathcal{R}_{\leq\delta}$ we associate u with p.



Consider a grid $\mathcal{G}$ of side length δ. For a grid cell g, consider the vertex of X' in g that is associated with the largest number of edges, and say it is being associated with d_g edges, and let v_g denote this "popular" vertex of X'. The total number of edges associated with vertices of X' inside g is bounded by $U_g = \mathsf{n}(X', g)\mathsf{d}_g$, where $\mathsf{n}(X', g)$ denotes the number of vertices of X' that lie inside g.

If $\mathsf{d}_g \leq M$ then $U_g \leq \mathsf{n}(X', g)M$, and we charge M vertex-edge pairs to each vertex of X inside g.

If $\mathsf{d}_g > M$ then we want to charge M vertex-edge pairs to each vertex of the original curve Y (or the original curve X) inside $3\mathsf{C}$, where C is a cube centered at v_g with side length $2(\delta + \mu) \leq 4\delta$ and $3\mathsf{C}$ is the same cube scaled by a factor of 3. We argue that this pays for the number of edges. For this we will use Lemma 3.4.3. First, observe that the length of the simplified curve Y' inside C is at least $\mathsf{d}_g\mu$. Indeed, the number of vertex-edge pairs d_g arise from different segments of Y' that are within distance at most δ from v_g, and each such segment has length at least μ. Thus, by Lemma 3.4.3, we have that Y must have at least

$$\Omega\Big((\mathsf{d}_g\mu/\delta)^{d/(d-1)}\Big) = \Omega\Big((\mathsf{d}_g\varepsilon)^{d/(d-1)}\Big)$$

vertices inside $3\mathsf{C}$, i.e., there is some constant c such that $c(\varepsilon\mathsf{d}_g)^{d/(d-1)} \leq \mathsf{n}(Y, 3\mathsf{C})$. Thus,

$$\mathsf{d}_g \leq \frac{1}{\varepsilon}\bigg(\frac{\mathsf{n}(Y, 3\mathsf{C})}{c}\bigg)^{(d-1)/d},$$

and we can derive

$$\mathsf{d}_g^2 \le \mathsf{n}(Y, 3\mathsf{C}) \frac{1}{c\varepsilon^2} \left(\frac{\mathsf{n}(Y, 3\mathsf{C})}{c} \right)^{1-2/d} \le \frac{1}{c\varepsilon^2} \left(\frac{n}{c} \right)^{1-2/d} \mathsf{n}(Y, 3\mathsf{C}) \le M\mathsf{n}(Y, 3\mathsf{C}),$$

for the constant in M chosen sufficiently large.

Now we can distinguish two cases. First assume that $\mathsf{n}(X', g) \le \mathsf{d}_g$ (i.e., the number of vertices of X' in the grid cell is small). In this case,

$$U_g = \mathsf{n}(X', g)\mathsf{d}_g \le \mathsf{d}_g^2 \le M\mathsf{n}(Y, 3\mathsf{C}).$$

Hence, we can charge M vertex-edge pairs to each vertex of Y inside the cube $3\mathsf{C}$.

Otherwise, $\mathsf{n}(X', g) > \mathsf{d}_g > M$ and the number of vertices of X' in the grid cell is large. But then, the length of X' inside C is at least $\mathsf{n}(X', g)\mu$, and by Lemma 3.4.3, we have that X must have at least $\Omega\big((\mathsf{n}(X', g)\varepsilon)^{d/(d-1)}\big)$ vertices inside $3\mathsf{C}$. Arguing as above, this implies that $\mathsf{n}(X', g)^2 \le M\mathsf{n}(X, 3\mathsf{C})$. Hence,

$$U_g = \mathsf{n}(X', g)\mathsf{d}_g \le \mathsf{n}(X', g)^2 \le M\mathsf{n}(X, 3\mathsf{C}).$$

Thus, we charge M vertex-edge pairs to each vertex of X inside the cube $3\mathsf{C}$. In other words, since the curve X is long, it can pay for the vertex-edge pairs itself.

Since $3\mathsf{C}$ intersects a constant number of cells of the grid, no vertex would get charged more than a constant number of times by the above scheme. Thus, every vertex, of either curve, gets charged $O(M)$ vertex-edge pairs overall, and the total number of vertex-edge pairs present in $\mathcal{R}_{\le\delta}$ is $O(nM)$, as claimed. □

Observation 3.4.6 One can extend the construction of Observation 3.4.4 to show that Lemma 3.4.5 is close to being tight. Indeed, consider the grid curve of Observation 3.4.4 in $d-1$ dimensions, for an integer m. We now lift it to d dimensions by considering the $[1, m]^d$ cube and placing two copies of the above curve on two opposite faces of the cube, denoted by f and f'. Let X_1 and X_2 denote these two copies.

Next, delete the even edges from X_1 and the odd edges from X_2. Connect every vertex v_1 of X_1 to its corresponding (copied) vertex v_2 in X_2 by a path made out of the $m-1$ unit edges along the grid line connecting the two vertices. This results in a curve X that is similar to the curve constructed in Observation 3.4.4, but has the advantage that when simplified for the distance $\mu = m$ it results in a curve with m^{d-1} segments of length $\ge m$ that connect points that lie on f and on f', respectively.

Let Y be a copy of X. For a fixed $\varepsilon > 0$, we can add a single segment to X such that the Fréchet distance between the resulting curves is exactly $\delta = m/\varepsilon$. Now, these two curves have $n = 2m^d + 2$ vertices overall, and furthermore, when we simplify them for the distance $\mu = \varepsilon\delta = m$, we end up with two curves such that every long edge of X' is going to be within distance $\le \delta = m/\varepsilon$ from a constant fraction of the edges of Y' (this would be all the edges if $1/\varepsilon > \sqrt{d}$). Therefore the complexity of the reachable free space is $\Omega(\mathsf{n}_{X'}\mathsf{n}_{Y'}) = \Omega\Big(\big(m^{d-1}\big)^2\Big) = \Omega\big(n^{2(d-1)/d}\big)$, where $\mathsf{n}_{X'}$ denotes the number of vertices of X'. The upper bound of Lemma 3.4.5 is (only) larger by a factor of $O(1/\varepsilon^2)$.

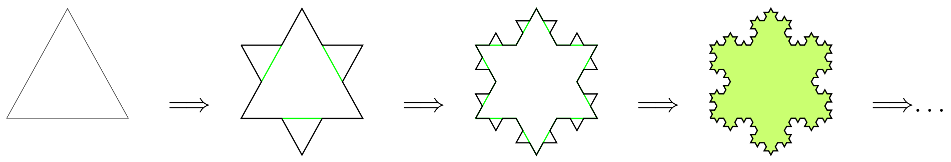


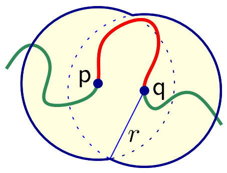
Figure 3.2: Koch's snowflake is an example of a κ-bounded curve that has infinite length but a finite diameter.

By using Lemma 3.4.5 instead of Lemma 3.1.6 in the proof of Lemma 3.3.3 we get the following result. Note that one can use the technique in the proof of Lemma 3.3.4 to get a slightly faster algorithm, see [71].

Theorem 3.4.7 *Given two low-density curves X and Y with a total of n vertices in $\mathbb{R}^d$, and a parameter $\varepsilon > 0$, there exists an algorithm which $(1+\varepsilon)$-approximates the Fréchet distance between X and Y in $O\left(\frac{n^{2(d-1)/d}}{\varepsilon^2}\log n\right)$ time.*

3.5 Extension to κ-bounded curves

We revisit the definitions of Alt *et al.* [14] of κ-bounded and κ-straight curves. Note that these definitions describe a very restricted class of curves while c-packed curves form a fairly general and natural class of curves. However, it is not true that any κ-bounded curve is $O(\kappa)$-packed. We therefore give a separate proof to bound the relative free space complexity of κ-bounded curves in order to improve upon the result in [14].



Definition 3.5.1 Let $\kappa \geq 1$ be a given parameter. A curve X is ***κ-straight*** if for any two points p and q on the curve, it holds that $\|X\langle \mathsf{p},\mathsf{q}\rangle\| \leq \kappa \|\mathsf{p}-\mathsf{q}\|$.

A curve X is a ***κ-bounded*** if for all $\mathsf{p},\mathsf{q} \in X$ it holds that the curve $X\langle \mathsf{p},\mathsf{q}\rangle$ is contained inside $\mathbf{b}(\mathsf{p},r) \cup \mathbf{b}(\mathsf{q},r)$, where $r = (\kappa/2)\|\mathsf{p}-\mathsf{q}\|$, see the figure on the right.

Lemma 3.5.2 *A κ-straight curve is 2κ-packed.*

Proof: Let X be a κ-straight curve in $\mathbb{R}^d$, and consider any ball $\mathbf{b}(\mathsf{p},r)$ that intersects it. Let q and s be the first and last points, respectively, along X that are in $\mathbf{b}(\mathsf{p},r)$. Clearly, $\|\mathsf{q}-\mathsf{s}\| \leq 2r$, and by the κ-straightness $\|X \cap \mathbf{b}(\mathsf{p},r)\| \leq \|X\langle \mathsf{q},\mathsf{s}\rangle\| \leq \kappa\|\mathsf{q}-\mathsf{s}\| \leq 2\kappa r$. □

Remark 3.5.3 It is easy to verify that a κ-straight curve is also κ-bounded. However, κ-bounded curves, counterintuitively, can have infinite length even when contained inside a finite domain. An example of this is ***Koch's snowflake***, which is a fractal curve depicted in Figure 3.2.

To see, intuitively, why Koch's snowflake is κ-bounded, let X_i be the ith polygonal curve generated by this process. There is a natural mapping between any point of X_i and X_{i+1}, for all i. In particular, consider two points p and q on the final curve X^*, and consider the two sequences of points $\mathsf{p}_i, \mathsf{q}_i \in X_i$, where $\mathsf{p}_{i+1} \in X_{i+1}$ (resp. $\mathsf{q}_{i+1} \in X_{i+1}$) is the natural image of p_i (resp. q_i); that is, $\lim_{i\to\infty} \mathsf{p}_i = \mathsf{p}$, and $\lim_{i\to\infty} \mathsf{q}_i = \mathsf{q}$.

Now, assume that $r = \|\mathsf{p} - \mathsf{q}\|$. Observe that, for all i, the polygonal curve X_i is made out of segments that are all of the same length. In particular, consider the first index k, such that this segment length of X_k is of length $\leq r/20$. It is easy to argue that $\|\mathsf{p}_k - \mathsf{p}\| \leq r/5$ and $\|\mathsf{q}_k - \mathsf{q}\| \leq r/5$. In fact, one can argue that no point of X_k moves more than a distance larger than $r/5$ to its final location on X^*.

Now, a tedious argument shows that there are $O(1)$ segments of X_k separating p_k from q_k. Therefore this portion of the curve X_k is covered by a disk of radius $O(r)$, and the corresponding portion of the final curve between p and q is also covered by a disk of radius $O(r)$. This implies that Koch's snowflake is κ-bounded.

A formal proof of this fact is considerably more tedious and is omitted.

Lemma 3.5.4 *Let X be a κ-bounded polygonal curve in $\mathbb{R}^d$, and let $\mu \leq \delta$ be parameters. Let $X' = \mathbf{simpl}(X, \mu)$. Then the number of segments of X' intersecting $\mathbf{b}(\mathsf{p}, \delta)$ is bounded by $O\big(\kappa^d (1 + \delta/\mu)^d\big)$, for any $\mathsf{p} \in \mathbb{R}^d$.*

Proof: For $X' = \mathsf{u}_1\mathsf{u}_2\ldots\mathsf{u}_k$, let $S_O = \{\mathsf{u}_1, \mathsf{u}_3, \ldots\}$ and $S_E = \{\mathsf{u}_2, \mathsf{u}_4, \ldots\}$ be the sets of odd and even segments of X', respectively.

Let $A_O \subseteq S_O$ be the set of odd segments of X' intersecting $\mathbf{b}(\mathsf{p}, \delta)$. For all i, pick an arbitrary point p_i on the ith segment of A_O that lies inside $\mathbf{b}(\mathsf{p}, \delta)$. Next, pick an original point q_i of X within distance at most μ from p_i, for $i = 1, \ldots, M = |A_O|$. Observe, that for all i we have that $\|\mathsf{p} - \mathsf{q}_i\| \leq \delta + \mu$. Furthermore, between any two distinct points p_i and p_j on the simplified curve X' there must lie an even segment of S_E in between them along the curve, and the length of this segment is at least μ (because the simplification algorithm generates segments of length at least μ). Also, the endpoints of this even segment lie on the original curve X.

We claim that no two points of $\mathsf{A} = \{\mathsf{q}_1, \ldots, \mathsf{q}_M\}$ can be too close to each other; that is, there are no two points $\mathsf{q}', \mathsf{q}'' \in \mathsf{A}$, such that $r = \|\mathsf{q}' - \mathsf{q}''\| \leq \mu/(4\kappa)$. Indeed, assume for the sake of contradiction, that there exist two such points. Then, by the above, the portion of X connecting them contains two points $\mathsf{v}', \mathsf{v}''$ that are at least μ apart. Let $R = \mathbf{b}(\mathsf{q}', (\kappa/2)r) \cup \mathbf{b}(\mathsf{q}'', (\kappa/2)r)$ and observe that $X\langle \mathsf{v}', \mathsf{v}'' \rangle \subseteq R$. However, the maximum distance between two points that are included inside R is bounded by its diameter. We have that

$$\mu \leq \|\mathsf{v} - \mathsf{v}'\| \leq \mathrm{diam}(R) = 2(\kappa/2)r + \|\mathsf{q}' - \mathsf{q}''\| \leq \frac{\mu}{4} + \frac{\mu}{4\kappa} \leq \frac{\mu}{2},$$

since $\kappa > 1$. A contradiction.

However, all the points of A lie inside a ball of radius $\delta + \mu$ centered at p. Now, placing a ball of radius $\mu' = \mu/(8\kappa)$ around each point of A, results in a set of interior disjoint balls. This implies, by a standard packing argument, that the number of points of A is bounded by $\mathrm{vol}(\mathbf{b}(\mathsf{p}, \delta + \mu)) / \mathrm{vol}(\mathbf{b}(\cdot, \mu')) = O\big((\delta + \mu)^d / (\mu/\kappa)^d\big) = \big((1 + \delta/\mu)^d \kappa^d\big)$.

This bounds the number of odd segments of X' intersecting the ball $\mathbf{b}(\mathsf{p}, \delta)$, and a similar argument holds for the even segments intersecting this ball. $\square$

Lemma 3.5.5 *Given two κ-bounded curves X and Y with a total number of n vertices in $\mathbb{R}^d$, and parameters $0 < \varepsilon \leq 1$ and $\delta > 0$. For $\mu = \varepsilon\delta$, let $X' = \mathbf{simpl}(X, \mu)$ and $Y' = \mathbf{simpl}(Y, \mu)$ denote their simplifications. Then $\mathcal{N}_{\leq\delta}(X', Y') = O\big((\kappa/\varepsilon)^d n\big)$.*

Proof: Let $\delta \geq 0$ be an arbitrary radius, and set $\mu = \varepsilon\delta$. Let $X' = \mathbf{simpl}(X, \mu)$ and $Y' = \mathbf{simpl}(Y, \mu)$. We need to show that the complexity of the reachable free space $\mathcal{R}_{\leq\delta}(X', Y')$ is $O\big(\kappa^d (1 + \delta/\mu)^d n\big) = O\big((\kappa/\varepsilon)^d n\big)$.

The boundary of a reachable cell in the free-space diagram has a non-empty intersection with $\mathcal{D}_{\leq\delta}(X', Y')$. Otherwise its interior could not be reached by a monotone path from $(0, 0)$. Therefore, using an argument similar to the proof of Lemma 3.1.6, Lemma 3.5.4 implies the desired bound. $\square$

By using Lemma 3.5.5 instead of Lemma 3.1.6 in the proof of Lemma 3.3.3 we get the following result. Note that one can use the technique in the proof of Lemma 3.3.4 to get a slightly faster algorithm, see [71].

Theorem 3.5.6 *Given two κ-bounded polygonal curves X and Y with a total of n vertices in $\mathbb{R}^d$, and a parameter $0 < \varepsilon \leq 1$, there exists an algorithm which $(1+\varepsilon)$-approximates the Fréchet distance between X and Y in $O\big((\kappa/\varepsilon)^d n \log n\big)$ time.*

3.6 Extension to closed packed curves

The Fréchet distance for closed curves is defined as in Section 2.1 with the altered condition that the reparameterization f is an orientation-preserving homeomorphism on the one-dimensional sphere. Note that in this case the constraint that the endpoints of the curves have to be matched to each other is dropped and the set of reparameterizations one has to consider is larger.

To apply our simplification algorithm (Algorithm 3.1.2) to a closed curve X, we pick an arbitrary vertex of X and run the algorithm as written. In the end, we connect the vertices in their circular order along X.

The decision problem for closed curves can be reduced to the previously considered case of open curves. Given two closed c-packed curves X and Y and a parameter δ. Pick a vertex p of the curve X, and assume that we know a point q on Y that is being matched to p by a matching of X and Y of width at most δ. Clearly, if we break X open at p, and Y at q, we retrieve two open curves $\widehat{X}$ and $\widehat{Y}$, and we can use the previous method to decide if $d_{\mathcal{F}}\left(\widehat{X}, \widehat{Y}\right) \leq \delta$. Hence we only need to generate a suitable set of candidates for q to determine if the Fréchet distance between X and Y is at most δ within a certain approximation error.

Lemma 3.6.1 *Let X be a closed c-packed polygonal curve in $\mathbb{R}^d$, and let $\mu \leq \delta$ be parameters. Let $X' = \mathbf{simpl}(X, \mu)$. Then the number of edges of X' intersecting $\mathbf{b}(\mathsf{p}, \delta)$ is bounded by $O(c\delta/\mu)$, for any $\mathsf{p} \in \mathbb{R}^d$.*

Proof: Consider the ball $\mathbf{b} = \mathbf{b}(\mathsf{p}, r)$ of radius $r = \mu + \delta$. Any edge u of X' that intersects $\mathbf{b}(\mathsf{p}, \delta)$ has to contribute at least μ to the length of the intersection with $\mathbf{b}$, as the simplification guarantees that every edge of X' is of length at least μ. Since X' is $6c$-packed, by Lemma 3.1.5, we have that $\|\mathbf{b} \cap X'\| \leq 6cr$, and the number of intersections of X' with $\mathbf{b}(\mathsf{p}, \delta)$ is $N \leq \|\mathbf{b} \cap X'\| / \mu \leq 6cr/\mu = 6c(\mu + \delta)/\mu = O(c + c\delta/\mu)$, which implies the claim. $\square$

Lemma 3.6.2 *Given two closed c-packed polygonal curves X and Y with a total number of n vertices and parameters δ and $0 < \varepsilon < 1$. Let $X' = \mathbf{simpl}(X, \mu)$ and $Y' = \mathbf{simpl}(Y, \mu)$ denote the curves simplified with $\mu \leq \varepsilon\delta$. For any point $\mathsf{p} \in X'$, we can compute a set of points $\mathcal{K} \subseteq Y'$ of size $O(c/\varepsilon)$, in $O(n + c/\varepsilon)$ time, such that if $d_{\mathcal{F}}(X', Y') \leq \delta$ then there exists a matching of width at most $(1 + \varepsilon)\delta$ that matches p to an element of $\mathcal{K}$.*

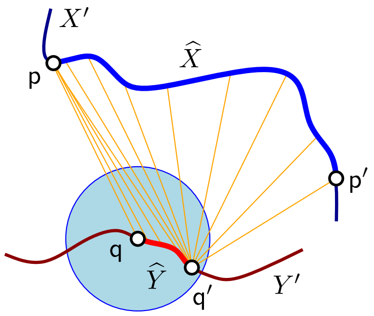
Proof: We walk along the curve Y' starting from an arbitrary point. If the starting point is within distance δ from p, then we add it to the candidate set $\mathcal{K}$. As we follow along the curve we create a candidate if we

(a) (re-)enter the ball $\mathbf{b}(\mathsf{p}, \delta)$, or

(b) have traveled a distance $\varepsilon\delta$ along Y' since the last creation of a candidate, unless we have exited the ball $\mathbf{b}(\mathsf{p}, \delta)$ in the meantime.

Clearly this takes $O(n + |\mathcal{K}|)$ time.

The number of events of type (a) is bounded (up to a factor of 2) by the number of intersections of Y' with the sphere $\mathbb{S}(\mathsf{p}, \delta)$, and by Lemma 3.6.1, this number is bounded by $O(c\delta/\mu) = O(c/\varepsilon)$. By Lemma 3.1.5 the simplified curve Y' is $6c$-packed and therefore the length of its intersection with $\mathbf{b}(\mathsf{p}, \delta)$ is at most $6c\delta$. This implies that we can have at most $O(6c\delta/\mu) = O(6c/\varepsilon)$ candidates that were created at events of type (b).



Consider a matching of X' and Y' of width at most δ under this matching. Next, consider a point $\mathsf{q} \in Y'$ that is matched to $\mathsf{p} \in X'$. Observe that $\mathsf{q} \in \mathbf{b}(\mathsf{p}, \delta)$ and there exists, by construction, a point $\mathsf{q}' \in \mathcal{K}$ such that the entire curve segment $Y'\langle \mathsf{q}, \mathsf{q}' \rangle$ is within distance δ from q.

Secondly, let p' be a point on X' that is matched to q' under this matching. We match the curve segment $\widehat{Y}$ between q and q' to p and the curve segment $\widehat{X}$ between p and p' to q, see the figure to the right. Clearly this preserves the monotonicity of the matching. By the triangle inequality, any point on $\widehat{Y}$ has distance at most $(1+\varepsilon)\delta$ to p. Similarly, for any point on $\widehat{X}$ there is a point on $\widehat{Y}$ that is within distance δ, therefore q' is within distance $(1 + \varepsilon)\delta$ from $\widehat{X}$.

We conclude that the Fréchet distance between X' and Y' is at most $(1+\varepsilon)\delta$ when restricted to matchings that associate p to q'. $\square$

One can adapt Lemma 3.2.3 to the closed curves case, by considering the $O(cn/\varepsilon)$ open curves that result from breaking Y' at any point of $\mathcal{K}$. The details of the adaption are straightforward, and we only state the result.

Lemma 3.6.3 *Given two closed polygonal c-packed curves X and Y with a total of n vertices in $\mathbb{R}^d$, and parameters δ and $1 > \varepsilon > 0$. Then, there exists an algorithm which, in $O\big((c/\varepsilon)^2 n\big)$ time, correctly outputs one of the following:*
(A) If $d_{\mathcal{F}}(X,Y) \le \delta$ then the algorithm outputs "$\le (1+\varepsilon)\delta$".
(B) If $d_{\mathcal{F}}(X,Y) > (1+\varepsilon)\delta$ then the algorithm outputs "$d_{\mathcal{F}}(X,Y) > \delta$".
(C) If $d_{\mathcal{F}}(X,Y) \in [\delta, (1+\varepsilon)\delta]$ then the algorithm outputs either of the above outcomes.

Plugging Lemma 3.6.3 into the algorithm of Theorem 3.3.5, we get the following result.

Theorem 3.6.4 *Given two closed polygonal c-packed curves X and Y with a total of n vertices in $\mathbb{R}^d$, and a parameter $1 > \varepsilon > 0$, one can $(1+\varepsilon)$-approximate the Fréchet distance between X and Y in $O\big(c^2 n\big(\varepsilon^{-2} + \log n\big)\big)$ time.*

3.7 Fatness implies packedness

We show that the boundary of an (α,β)-covered shape is c-packed even if the shape does not have a finite descriptive complexity. A somewhat similar result (which however is too weak to prove this result) is the packing lemma of de Berg [50] that shows that the boundary of the union of γ-fat shapes has low density. This implies that a connected component of this boundary has low density.

As mentioned before, since Koch's snowflake is γ-fat, if the finiteness requirement is removed, it follows that the boundary of γ-fat shapes with unbounded descriptive complexity are not c-packed, for any finite c.

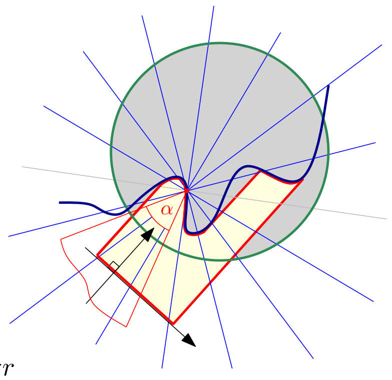
Definition 3.7.1 A bounded simply connected region P in the plane is (α,β)***-covered*** if for each point $\mathsf{p} \in \partial P$, there exists a triangle T_{p}, called a ***good triangle*** of p, such that: (i) p is a vertex of T_{p}, (ii) $T_{\mathsf{p}} \subseteq P$, (iii) all the angles of T_{p} are at least α, and (iv) the length of all the edges of T_{p} is at least $\beta \mathrm{diam}(P)$.

Note, that our definition is different from the standard definition of (α,β)-covered shapes, since we do not require that the region P has a finite descriptive complexity.

Lemma 3.7.2 *Let S be a set of segments contained inside a disk with radius r, such that for any point p lying on a segment of S, there is an infinite cone $\mathcal{V}$ of angle at least $\alpha \le \pi$ with an apex at p, such that the intersection of the interior of $\mathcal{V}$ with S is empty. Then, $\|S\| \le 10\pi r/(\alpha \sin(\alpha/4))$.*

Proof: Let $\mathcal{F}$ be a family of $\lceil 2\pi/(\alpha/2) \rceil$ cones, centered at the origin, such that they cover all directions, and each cone has angle $\alpha/2$. Clearly, for any point p lying on a segment of S, there must be a cone $\mathcal{V} \in \mathcal{F}$, such that the interior of $\mathsf{p} + \mathcal{V}$ does not intersect S. We will say that p is ***exposed*** by $\mathcal{V}$.

So, fix such a cone $\mathcal{V} \in \mathcal{F}$ and consider the direction $\vec{v}$ that splits the angle of $\mathcal{V}$ into two. Rotate the plane such that $\vec{v}$ is the direction of the negative y axis, and observe that any point of S that is exposed by (the rotated) $\mathcal{V}$ lies on the lower envelope of the segments of S.



Furthermore, the segment $\mathsf{u} \in S$ that contains this point must have an angle in the range $(-\pi/2 + \alpha/4, \pi/2 - \alpha/4)$ with the positive direction of the x-axis (we assume u is oriented from left to right).

Now, since the projection of S on the x-axis has length at most $2r$, it follows that the total length of the segments exposed by $\mathcal{V}$ is at most $2r/\sin(\alpha/4)$.

Hence, the total length of segments of S is bounded by

$$|\mathcal{F}| \left(\frac{2r}{\sin(\alpha/4)}\right) = \left(\frac{4\pi}{\alpha} + 1\right)\left(\frac{2r}{\sin(\alpha/4)}\right) \leq \frac{10\pi r}{\alpha \sin(\alpha/4)}.$$

□

Lemma 3.7.3 *If P is an (α, β)-covered polygon in the plane then it is c-packed, for*

$$c = O\left(\frac{1}{\alpha\beta \sin(\alpha/4)\tan(\alpha)}\right).$$

Proof: Let $S = \partial P$, and consider any disk D of radius r in the plane. Observe that the height of a good triangle is at least $\rho = (s/2)\tan(\alpha)$, for $s = \beta \text{diam}(P)$, and this also bounds the distance of any vertex of a good triangle to its facing edge. If $r \leq \rho/2$, then any good triangle for a point of S behaves like a cone as far as $S \cap D$ is concerned, and Lemma 3.7.2 implies that $\|S \cap D\| \leq 10\pi r/(\alpha \sin(\alpha/4))$ as desired.

If $r \leq \text{diam}(P)$, then cover D by $m = (2\sqrt{2}r/\rho + 1)^2$ disks of radius $\rho/2$. Clearly, for each such disk, the total length of segments of S inside it, by Lemma 3.7.2, is at most $5\pi\rho/(\alpha \sin(\alpha/4))$. Therefore the total length of S inside D is

$$\frac{5\pi\rho}{(\alpha \sin(\alpha/4))}\left(\frac{2\sqrt{2}r}{\rho} + 1\right)^2 \leq \frac{160\pi}{\alpha \sin(\alpha/4)} \cdot \frac{r}{\rho} \cdot r \leq \frac{320\pi}{\alpha\beta \sin(\alpha/4)\tan(\alpha)} r.$$

Observe that the total length of ∂P is bounded by the above expression, by taking D to be a disk of radius $r = \text{diam}(P)$ centered at some point of P. Therefore the claim trivially holds in the case $r \geq \text{diam}(P)$. □

3.8 Concluding remarks

In this chapter, we saw a new approximation algorithm for the Fréchet distance for polygonal curves in any fixed dimension. The algorithm is surprisingly simple and should be practical. Furthermore it works for any kind of polygonal curves. Since the algorithm simplifies the curves to the “right” resolution during the execution, we expect the algorithm to be fast in practice.

To analyze the complexity of the algorithm, we used realistic input models. In particular, we used the new concept of c-packedness. While not all curves are c-packed, it seems that many real-life curves are c-packed. The algorithm's analysis relies on the concept of the relative free space complexity of curves, which tries to capture the complexity of the free-space diagram when simplification is being used. We showed that the relative free space complexity of c-packed curves is linear and the algorithm therefore runs in near-linear time for these curves.

We extended the analysis of the algorithm to known families of curves, and in particular low-density curves. Here, we faced the challenge that, unlike for c-packed curves, simplifying a ϕ-low-density curve does not neccessarily yield an $O(\phi)$-low-density curve under our simplification algorithm. However, we showed that the relative free space complexity of low-density curves can be high only if the curve has many vertices in a small area. By using a sophisticated charging scheme, we were able to bound the relative free space complexity also of low density curves and as a result we showed that the algorithm runs in near-linear time for low-density curves in the plane. We also saw that in higher dimensions low-density curves are considerably less restriced than c-packed curves. Therefore, the relative free space complexity of two ϕ-low-density curves can be high even for a small value of ϕ.

Finally, we extended the analysis to κ-bounded curves, thereby improving upon a previous result and we also showed that the algorithm can be modified to handle closed curves efficiently. Despite the wealth of results many interesting open problems remain. We conclude by discussing a few of them in more detail.

Open Problem 3.1 It is an open problem if one can compute the Fréchet distance exactly in subquadratic time if the input curves are c-packed. In particular, it is an open problem if the running time of the algorithm by Buchin *et al.* [34] (and likewise the algorithm by Agarwal *et al.* [3] for the discrete Fréchet distance) can be improved under this assumption. Both algorithms use a look-up table to speed up the computations. The size of the table is determined by the number of combinatorially different partial solutions. In both cases, it is conceivable that the number of partial solutions that have to be considered by the algorithm is reduced by the packedness assumption.

Open Problem 3.2 The approximation algorithm described in this chapter has been extended to the case of map-matching a trajectory in a road network, see [43]. More recently, the so-called *highway dimension* was suggested as a realistic input model for continental-scale road networks by Abraham *et al.* [2]. This model restricts the minimal number of transit points needed to cover all shortest paths of length $O(r)$ in a ball of radius r. A low highway dimension of real-life road networks would explain the success of many seemingly different route-planning algorithms used in practice. However, computing the highway dimension for a given road-network is conjectured to be NP-hard. It would be interesting to study the relation between the highway dimension and the packedness of planar geometric graphs. Note that computing the packedness constant can be done in polynomial time.

Open Problem 3.3 Exact algorithms that minimize the Fréchet distance under translations in the plane run in $O(n^8 \log n)$ time [13] or in $O(n^{10}\text{polylog}(n))$ time [75].

An extension by Wenk to transformations in higher dimensions with k degrees of freedom runs in $O(n^{3k+2} \log n)$ time [156].

Wenk *et al.* show that it is sufficient to test the Fréchet distance for certain *critical transformations.* Their algorithm invokes the original decision procedure by Alt and Godau on the input curves for each such critical transformation. Thus, improving upon the quadratic-time decision algorithm immediately implies a better running time for this algorithm. However, the number of critical transformation is believed to be high. Even for translations in the plane, their number lies between $O(n^6)$ and $\Omega(n^2)$ in the worst case, see [156].

In order to design a faster algorithm for the Fréchet distance under transformations, one could try to approximate the critical transformations. For translations in the plane, it is easy to see that the optimal translation has to lie within a constant factor of the two translations that map the endpoints onto one another. Furthermore, assuming c-packedness of the input curves should have a positive effect on the number and the distribution of the critical transformations.

Approximation algorithms known in the literature only work for the case of translations in the plane. A $(1+\varepsilon)$-approximation algorithm that runs in $O(n^2/\varepsilon^2)$ time was presented by Wenk *et al.* [13]. A constant factor approximation algorithm which runs in $O(n^2 \log n)$ was presented by Efrat *et al.* [75].

Open Problem 3.4 The discrete Fréchet distance has been studied under imprecision by Ahn *et al.* [7]. In this setting the exact location of the input vertices is not known. Instead their possible locations are assumed to lie within given ball-shaped regions. Now the objective is to compute conservative upper and lower bounds on the Fréchet distance of the two curves. Ahn *et al.* give an $O(n^2 \log^2 n)$-time algorithm for the planar case and constant factor approximation algorithms that run in $O(n^2)$ time for fixed dimensions. It is an open problem how to combine this imprecise input model with realistic input models, such as c-packedness. It is conceivable that one can improve the running times of these algorithms by making such realistic input assumptions.

CHAPTER 4

Data structures for Fréchet-distance queries

In this chapter we describe several data structures to preprocess a polygonal curve Z in $\mathbb{R}^d$ to answer different types of Fréchet-distance queries. For the first type of query, we are given a query curve Q and two values $u, v \in [0, 1]$, which we interpret as points $Z(u)$ and $Z(v)$ on the curve, and we return the Fréchet distance $d_{\mathcal{F}}(Q, Z[u, v])$. The second type is a query for a simplification of Z. More specifically, we are given a parameter k and we want to extract a polygonal curve with k segments which has the smallest Fréchet distance to Z over all such k-segment curves and thus serves as a simplification of Z. Both types of queries are answered approximately. The chapter is organized as follows. We first establish some basic facts in Section 4.1. Then, in Section 4.2 we describe data structures for queries of the first type where Q is a single line segment. The data structure is described in several stages. In Section 4.3 we extend this data structure to support queries with polygonal curves of k segments. For this, we first define an ordering of the vertices of an input curve that allows to extract a simplification quickly. We call this ordering a *universal vertex permutation.* Then we show how to use this ordering to extend the data structure of Section 4.2 to support queries with polygonal curves of low complexity.

4.1 Useful lemmas for curves and segments

Definition 4.1.1 For a curve Z, the segment connecting its endpoints is its ***spine***, denoted by spine(Z).

The following is a sequence of technical lemmas that we need later on. These lemmas testify that:

(A) The spine of a curve is, up to a factor of two, the closest segment to this curve with respect to the Fréchet distance, see Lemma 4.1.2.

(B) The Fréchet distance between a curve and its spine is monotone, up to a factor of two, with respect to subcurves, see Lemma 4.1.3.

(C) Shortcutting a curve cannot increase the Fréchet distance of the curve to a line segment, see Lemma 4.1.4 or [36].

Lemma 4.1.2 *Let* pq *be a segment and* Z *be a curve. Then, (i)* $d_{\mathcal{F}}(\mathsf{pq}, Z) \geq d_{\mathcal{F}}(\mathsf{pq}, \mathrm{spine}(Z))$*, and (ii)* $d_{\mathcal{F}}(\mathsf{pq}, Z) \geq d_{\mathcal{F}}(\mathrm{spine}(Z), Z)/2$*.*

Proof: Let r and u be the endpoints of Z; that is $\mathrm{spine}(Z) = \mathsf{ru}$.

(i) Since in any matching of pq with Z it must be that p is matched to r, and q is matched to u, it follows that $d_{\mathcal{F}}(\mathsf{pq}, Z) \geq \max(\|\mathsf{p} - u\|, \|\mathsf{q} - v\|) = d_{\mathcal{F}}(\mathsf{pq}, \mathsf{ru}) = d_{\mathcal{F}}(\mathsf{pq}, \mathrm{spine}(Z))$, by Observation 2.3.1.

(ii) By (i) and the triangle inequality, we have that

$$d_{\mathcal{F}}(\mathrm{spine}(Z), Z) \leq d_{\mathcal{F}}(\mathrm{spine}(Z), \mathsf{pq}) + d_{\mathcal{F}}(\mathsf{pq}, Z) \leq 2d_{\mathcal{F}}(\mathsf{pq}, Z),$$

which implies the claim. □

Lemma 4.1.3 *Given two curves* Z *and* $\widehat{Z}$*, such that* $\widehat{Z}$ *is a subcurve of* Z*. Then, we have that* $d_{\mathcal{F}}\left(\mathrm{spine}\left(\widehat{Z}\right), \widehat{Z}\right) \leq 2d_{\mathcal{F}}(\mathrm{spine}(Z), Z)$*.*

Proof: Consider the matching that realizes the Fréchet distance between Z and $\mathrm{spine}(Z)$. It has to match the endpoints of $\widehat{Z}$ to points q and r on $\mathrm{spine}(Z)$. We have that $d_{\mathcal{F}}\left(\widehat{Z}, \mathsf{qr}\right) \leq d_{\mathcal{F}}(Z, \mathrm{spine}(Z))$. By Lemma 4.1.2 (i), we have $d_{\mathcal{F}}\left(\mathrm{spine}\left(\widehat{Z}\right), \mathsf{qr}\right) \leq d_{\mathcal{F}}\left(\widehat{Z}, \mathsf{qr}\right) \leq d_{\mathcal{F}}(Z, \mathrm{spine}(Z))$. Now, by the triangle inequality, we have that

$$d_{\mathcal{F}}\left(\widehat{Z}, \mathrm{spine}\left(\widehat{Z}\right)\right) \leq d_{\mathcal{F}}\left(\widehat{Z}, \mathsf{qr}\right) + d_{\mathcal{F}}\left(\mathsf{qr}, \mathrm{spine}\left(\widehat{Z}\right)\right) \leq 2d_{\mathcal{F}}(Z, \mathrm{spine}(Z)).$$

□

Lemma 4.1.4 *Let* $Z = u_1 u_2 \ldots u_n$ *be a polygonal curve,* pq *be a segment, and let* $i < j$ *be any two indices. Then, for* $Z' = Z\langle u_1, u_i\rangle \oplus u_i u_j \oplus Z\langle u_j, u_n\rangle$*, we have* $d_{\mathcal{F}}(Z', \mathsf{pq}) \leq d_{\mathcal{F}}(Z, \mathsf{pq})$*.*

Proof: Consider the matching realizing $d_{\mathcal{F}}(Z, \mathsf{pq})$, and break it into three portions:

- the portion matching $Z\langle u_1, u_i\rangle$ with a "prefix" $\mathsf{pp}' \subseteq \mathsf{pq}$,
- the portion matching $Z\langle u_i, u_j\rangle$ with a subsegment $\mathsf{p}'\mathsf{q}' \subseteq \mathsf{pq}$, and
- the portion matching $Z\langle u_j, u_n\rangle$ with a "suffix" $\mathsf{q}'\mathsf{q} \subseteq \mathsf{pq}$.

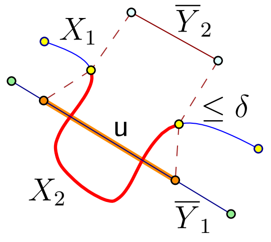
Now, by Lemma 4.1.2 (i), we have that

$$\begin{aligned} d_{\mathcal{F}}(Z, \mathsf{pq}) &= \max\Big(d_{\mathcal{F}}(Z\langle u_1, u_i\rangle, \mathsf{pp}'),\ d_{\mathcal{F}}(Z\langle u_i, u_j\rangle, \mathsf{p}'\mathsf{q}'),\ d_{\mathcal{F}}(Z\langle u_j, u_n\rangle, \mathsf{q}'\mathsf{q}) \Big) \\ &\geq \max\Big(d_{\mathcal{F}}(Z\langle u_1, u_i\rangle, \mathsf{pp}'),\ d_{\mathcal{F}}(u_i u_j, \mathsf{p}'\mathsf{q}'),\ d_{\mathcal{F}}(Z\langle u_j, u_n\rangle, \mathsf{q}'\mathsf{q}) \Big) \\ &\geq d_{\mathcal{F}}(Z', \mathsf{pq}). \end{aligned}$$

□

The following lemma is an extension of Lemma 4.1.3. Here, the spine is replaced by an arbitrary line segment. It will also be used in Chapter 5.

Lemma 4.1.5 *Given a value $\delta > 0$ and two curves X_1 and X_2, such that X_2 is a subcurve of X_1, and given two line segments $\overline{Y}_1$ and $\overline{Y}_2$, such that $d_{\mathcal{F}}\big(X_1, \overline{Y}_1\big) \leq \delta$ and the start (resp., end) point of X_2 is in distance δ to the start (resp., end) point of $\overline{Y}_2$, then $d_{\mathcal{F}}\big(X_2, \overline{Y}_2\big) \leq 3\delta$.*



Proof: Let u denote the subsegment of $\overline{Y}_1$ that is matched to X_2 under a matching realizing the Fréchet distance between X_1 and $\overline{Y}_1$. We know that $d_{\mathcal{F}}(X_2, \mathsf{u}) \leq \delta$ by this mapping. The start point of $\overline{Y}_2$ is in distance 2δ to the start point of u, since they are both in distance δ to the start point of X_2 and the same holds for the end points. This implies that $d_{\mathcal{F}}\big(\mathsf{u}, \overline{Y}_2\big) \leq 2\delta$. Now, by the triangle inequality, $d_{\mathcal{F}}\big(X_2, \overline{Y}_2\big) \leq d_{\mathcal{F}}(X_2, \mathsf{u}) + d_{\mathcal{F}}\big(\mathsf{u}, \overline{Y}_2\big) \leq 3\delta$. $\square$

4.2 Data structure for queries with single segments

Given a polygonal curve Z in $\mathbb{R}^d$, we want to build a data structure that supports queries for the Fréchet distance of subcurves of Z to query segments pq. We describe the data structure in three stages. We first describe a data structure that achieves a constant factor approximation in Section 4.2.1. We proceed by describing a data structure that answers queries for the Fréchet distance of the entire curve to a query segment up to an approximation factor of $(1+\varepsilon)$ in Section 4.2.2. Finally, we describe how to combine these two results to obtain the final data structure for segment queries in Section 4.2.3.

4.2.1 Stage 1: Achieving a constant-factor approximation

In this section we describe a data structure that preprocesses a curve Z to answer queries for the Fréchet distance of a subcurve of Z to a query segment up to a constant approximation factor. This data structure will be the basis for later extensions.

A query is specified by points u, v, p and q. Here u and v are points on Z (and we are also given the edges of Z containing these two points), and the points p and q define the query segment. Our goal is to approximate $d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.

4.2.1.1 The data structure

Preprocessing. Build a balanced binary tree T on the *edges* of Z. Every internal node ν of T corresponds to a subcurve of Z, denoted by $\mathrm{cr}(\nu)$. Let $\mathrm{seg}(\nu)$ denote the spine of $\mathrm{cr}(\nu)$ (Definition 4.1.1). For every node, we precompute its Fréchet distance of the curve $\mathrm{cr}(\nu)$ to the segment $\mathrm{seg}(\nu)$. Let d_ν denote this distance.

Answering a query. For the time being, assume that u and v are vertices of Z. In this case, one can compute, in $O(\log n)$ time, $k = O(\log n)$ nodes $\nu_1, \ldots, \nu_k$ of T, such that $Z\langle u, v\rangle = \mathrm{cr}(\nu_1) \oplus \mathrm{cr}(\nu_2) \oplus \cdots \oplus \mathrm{cr}(\nu_k)$. We compute the polygonal curve $Y = \mathrm{seg}(\nu_1) \oplus \cdots \oplus \mathrm{seg}(\nu_k)$, and compute its Fréchet distance from the segment pq.

We denote this distance by $d = d_{\mathcal{F}}(\mathsf{pq}, Y)$. We return

$$\Delta = d + \max_{i=1}^{k} \mathrm{d}_{\nu_i}$$

as the approximate distance between pq and the subcurve $Z\langle u, v\rangle$.

4.2.1.2 Analysis

Lemma 4.2.1 *Given a polygonal curve Z with n edges, one can preprocess it in $O(n \log^2 n)$ time, such that for any pair u, v of vertices of Z and a segment pq, one can compute, in $O(\log n \log\log n)$ time, a 3-approximation to $d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.*

Proof: The construction of the data structure and how to answer a query is described above. For the preprocessing time, observe that computing the Fréchet distance of a segment to a polygonal curve with k segments takes $O(k \log k)$ time [12]. Hence, the distance computations in each level of the tree T take $O(n \log n)$ time, and $O(n \log^2 n)$ time overall.

As for the query time, computing Y takes $O(\log n)$ time, and computing its Fréchet distance from pq takes $O(\log n \log\log n)$ time [12].

Finally, observe that the returned distance Δ is a realizable Fréchet distance, as we can take the matching between pq and Y, and chain it with the matching of every edge of Y with its corresponding subcurve of Z. Clearly, the resulting matching has width at most Δ.

Let t be the index realizing $\max_{i=1}^{k} \mathrm{d}_{\nu_i}$. Then, by repeated application of Lemma 4.1.4, we have that $d = d_{\mathcal{F}}(\mathsf{pq}, Y) \leq d_{\mathcal{F}}(\mathsf{pq}, Z)$. Thus,

$$\begin{aligned}\Delta = d + \max_{i=1}^{k} \mathrm{d}_{\nu_i} &= d_{\mathcal{F}}(\mathsf{pq}, Y) + \mathrm{d}_{\nu_t} \leq d_{\mathcal{F}}(\mathsf{pq}, Z) + d_{\mathcal{F}}(\mathrm{seg}(v_t), \mathrm{cr}(v_t)) \\ &\leq d_{\mathcal{F}}(\mathsf{pq}, Z) + 2 \min_{\mathsf{p'q'} \subseteq \mathsf{pq}} d_{\mathcal{F}}(\mathsf{p'q'}, \mathrm{cr}(v_t)) \leq 3 d_{\mathcal{F}}(\mathsf{pq}, Z).\end{aligned}$$

To see the last step, consider the matching realizing $d_{\mathcal{F}}(\mathsf{pq}, Z)$, and consider the subsegment $\mathsf{p'q'}$ of pq that is being matched to $\mathrm{cr}(v_t) \subseteq Z$. Clearly, $d_{\mathcal{F}}(\mathsf{p'q'}, \mathrm{cr}(v_t)) \leq d_{\mathcal{F}}(\mathsf{pq}, Z)$. □

Theorem 4.2.2 *Given a polygonal curve Z with n edges, one can preprocess it in $O(n \log^2 n)$ time and using $O(n)$ space, such that, given a query specified by*

(i) a pair of points u and v on the curve Z,

(ii) the edges containing these two points, and

(iii) a pair of points p and q,

one can compute, in $O(\log n \log\log n)$ time, a 3-approximation to $d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.

Proof: This follows by a relatively minor modification of the above algorithm and analysis. Indeed, given u and v (and the edges containing them), the data structure computes the two vertices u', v' that are endpoints of these edges that lie between u and v on the curve. The data structure then concatenates the segments uu' and $v'v$ to the approximation Y (here Y is computed for the vertices u' and v'). The remaining details are as described above. □

4.2.2 Stage 2: A segment query to the entire curve

In this section we describe a data structure that preprocesses a curve to answer queries for the Fréchet distance of the entire curve to a query segment up to an approximation factor of $(1+\varepsilon)$. We will use this data structure as a component in our later extensions.

4.2.2.1 The data structure

We use the following lemma for constructing an exponential grid.

Lemma 4.2.3 *Given a point $u \in \mathbb{R}^d$, a parameter $0 < \varepsilon \leq 1$ and an interval $[\alpha, \beta] \subseteq \mathbb{R}$ one can compute in $O\bigl(\varepsilon^{-d} \log(\beta/\alpha)\bigr)$ time and space an exponential grid of points $G(u)$, such that for any point $\mathsf{p} \in \mathbb{R}^d$ with $\|\mathsf{p} - u\| \in [\alpha, \beta]$, one can compute in constant time a grid point $\mathsf{p}' \in G(u)$ with $\|\mathsf{p} - \mathsf{p}'\| \leq (\varepsilon/2) \|\mathsf{p} - u\|$.*

Proof: We construct a series of axis-parallel hypercubes C_i centered at u and of side lengths $\lambda_i = 2^{i+2}\alpha$ for $i = 0, \ldots, m = \lceil \log(\beta/\alpha) \rceil$. Let V_i be the set of vertices of the grid $\mathcal{G}_i$ of side length $\varepsilon\lambda_i/c_1$, inside $\mathsf{C}_i \setminus \mathsf{C}_{i-1}$ (resp., C_0), where $c_1 = 4\sqrt{d}$.

Let $G(u) = \bigcup_{i=0}^{m} V_i$. The grid $G(u)$ satisfies the claimed properties. First observe that its size is bounded by

$$|G(u)| = \sum_{i=0}^{m} |V_i| = \sum_{i=0}^{m} O(1/\varepsilon^d) = O\bigl(\varepsilon^{-d} \log(\beta/\alpha)\bigr).$$

Secondly, for any point $\mathsf{p} \in \mathbb{R}^d$, such that $\alpha \leq \|\mathsf{p} - u\| \leq \beta$, let i be the maximal index such that $\mathsf{p} \in \mathsf{C}_i$ (since, C_m covers all such points, i is well defined).

By assumption $\|\mathsf{p} - u\| \geq \alpha$. Furthermore, if $i \geq 1$, then $\mathsf{p} \in \mathsf{C}_i \setminus \mathsf{C}_{i-1}$ and thus

$$\|\mathsf{p} - u\| \geq \lambda_{i-1}/2 = 2^i\alpha.$$

So, consider the grid cell C' of $\mathcal{G}_i$ that contains p, and let p' be the closest vertex of this grid to p. We have that

$$\|\mathsf{p} - \mathsf{p}'\| \leq \frac{\operatorname{diam}(\mathsf{C}')}{2} \leq \frac{\sqrt{d}}{2} \cdot \frac{\varepsilon\lambda_i}{c_1} = \frac{\sqrt{d}}{2} \cdot \frac{\varepsilon 2^{i+2}\alpha}{4\sqrt{d}} \leq \frac{\varepsilon}{2} \|\mathsf{p} - u\|,$$

as claimed.

As for computing p', observe that one can easily compute, in constant time, the index i such that $\mathsf{p} \in \mathsf{C}_i \setminus \mathsf{C}_{i-1}$ using the floor function. Then, one can look-up the grid cell that contains p in constant time, and scan the vertices of this cell to find the closest to p in constant time. □

Preprocessing. We are given a polygonal curve Z in $\mathbb{R}^d$ with n segments, and we would like to preprocess it for $(1 + \varepsilon)$-approximate Fréchet distance queries against a query segment. To this end, let $L = d_{\mathcal{F}}(uv, Z)$, where uv is the spine of Z. We construct an exponential grid $G(u)$ of points around u with the range $[\alpha, \beta] = [\varepsilon L/2, L/\varepsilon]$ as described in Lemma 4.2.3 and illustrated in Figure 4.1. We construct the same grid $G(v)$ around the vertex v.

Now, for every pair of points $(\mathsf{p}', \mathsf{q}') \in G(u) \times G(v)$ we compute the Fréchet distance $D[\mathsf{p}', \mathsf{q}'] = d_{\mathcal{F}}(\mathsf{p}'\mathsf{q}', Z)$ and store it. Thus, we take $O\bigl(\chi^2 n \log n\bigr)$ time to build a data structure that requires $O\bigl(\chi^2\bigr)$ space, where $\chi = \varepsilon^{-d} \log(1/\varepsilon)$.

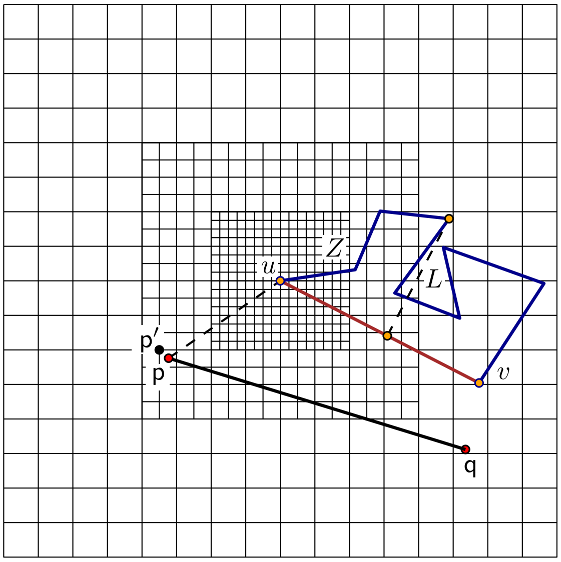


Figure 4.1: We build an exponential grid around each endpoint of the curve, such that for any point p, which has distance to the endpoint in the range $[\varepsilon L/2, L/\varepsilon]$, there exists a grid point p' which is relatively close by.

Answering a query. Given a query segment pq, we compute the distance

$$r = \max\Big(\|\mathsf{p} - u\|, \|\mathsf{q} - v\| \Big).$$

If $r \leq \varepsilon L/2$, then we return $L - r$ as the approximation to the distance $d_{\mathcal{F}}(\mathsf{pq}, Z)$. If $r \geq L/\varepsilon$ then we return r as the approximation. Otherwise, let p' (resp., q') be the nearest neighbor to p in $G(u)$ (resp., $G(v)$). We return the distance

$$\Delta = D[\mathsf{p}', \mathsf{q}'] - \max(\|\mathsf{p} - \mathsf{p}'\|, \|\mathsf{q} - \mathsf{q}'\|)$$

as the approximation.

4.2.2.2 Analysis

Lemma 4.2.4 *Given a polygonal curve Z with n vertices in $\mathbb{R}^d$, one can build a data structure, in $O(\chi^2 n \log n)$ time, that uses $O(\chi^2)$ space, such that given a query segment pq one can $(1+\varepsilon)$-approximate $d_{\mathcal{F}}(\mathsf{pq}, Z)$ in $O(1)$ time, where $\chi = \varepsilon^{-d}\log(1/\varepsilon)$.*

Proof: The data structure is described above. Given pq we compute the distance of the endpoints of this segment from the endpoints of Z. If they are too close, or if one of them is too far away, then we are done since in this case the Fréchet distance is dominated either by these distances or by the precomputed value L. Otherwise, we find the two cells in the exponential grid that contain p and q (that is, the indices of the grid points that are close to them) as described above. Using the indices of the grid points, we can directly look-up the approximation of the Fréchet distance in constant time.

Now, we argue about the quality of the approximation using the notation which is also used above. There are three cases: either (i) $r \leq \varepsilon L/2$, or (ii) $L \leq \varepsilon r$, or (iii) $\varepsilon L/2 \leq r \leq L/\varepsilon$. Let Δ be the returned value. We claim that in all three cases, it holds that

$$\Delta \leq d_{\mathcal{F}}(\mathsf{pq}, Z) \leq (1+\varepsilon)\Delta. \tag{4.1}$$

First note that by the triangle inequality,

$$L - r \leq d_{\mathcal{F}}(\mathsf{pq}, Z) \leq L + r. \tag{4.2}$$

Now, in case (i) above, L dominates the distance value and we return $\Delta = L - r$. Thus, Eq. (4.1) follows from Eq. (4.2).

In case (ii), r dominates the distance value and we return $\Delta = r$. Since r is at most $d_{\mathcal{F}}(\mathsf{pq}, Z)$, again Eq. (4.1) follows from Eq. (4.2).

In case (iii), the precomputed Fréchet distance of $\mathsf{p'q'}$ to Z dominates the distance. Recall that we return $\Delta = d_{\mathcal{F}}(\mathsf{p'q'}, Z) - d_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq})$ in this case. Again, by the triangle inequality, it holds that

$$d_{\mathcal{F}}(\mathsf{p'q'}, Z) - d_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}) \leq d_{\mathcal{F}}(\mathsf{pq}, Z) \leq d_{\mathcal{F}}(\mathsf{p'q'}, Z) + d_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq})\,. \tag{4.3}$$

Since r is at least $\varepsilon L/2$ and by Observation 2.3.1, Lemma 4.2.3 implies that

$$d_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}) \leq \max(\|\mathsf{p} - \mathsf{p'}\|, \|\mathsf{q} - \mathsf{q'}\|) \leq (\varepsilon/2)r,$$

thus, since also r is at most $d_{\mathcal{F}}(\mathsf{pq}, Z)$ it follows by Eq. (4.3) that

$$\Delta \leq d_{\mathcal{F}}(\mathsf{pq}, Z) \leq \Delta + 2d_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}) \leq \Delta + \varepsilon r \leq (1+\varepsilon)\Delta.$$

This implies the claim.

□

4.2.3 Stage 3: A segment query to a subcurve

In this section we describe a data structure that preprocesses a curve Z to answer queries for the Fréchet distance of a subcurve of Z to a query segment up to an approximation factor of $(1+\varepsilon)$. For this we combine the data structures developed in the previous sections.

As in Section 4.2.1, a query is defined by two points u and v on Z and a segment with endpoints p and q. The goal is now a $(1+\varepsilon)$-approximation to $d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.

4.2.3.1 The data structure

Preprocessing. Let Z be a given polygonal curve with n vertices. We build the data structure of Theorem 4.2.2. Next, for each node of the resulting tree T, we build for its subcurve the data structure of Lemma 4.2.4 using $\varepsilon' = \varepsilon/3$.

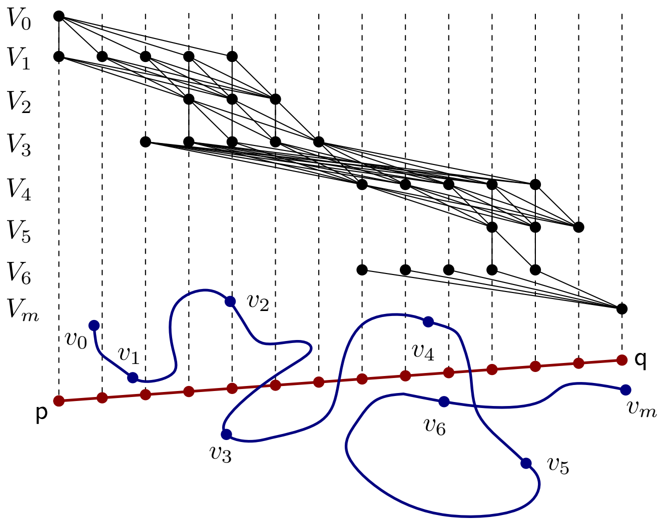


Figure 4.2: Schematic illustration of the graph G on the vertex set $\bigcup V_i$.

Answering a query. Using the data structure of Theorem 4.2.2 we first compute a 3-approximation r to $d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$; that is, $d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle) \leq r \leq 3d_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$. This query also results in a decomposition of $Z\langle u, v\rangle$ into $m = O(\log n)$ subcurves. Let $u = v_0, v_1, \ldots, v_{m-1}, v_m = v$ be the vertices of these subcurves, where v_0v_1 and $v_{m-1}v_m$ are subsegments of Z.

We want to find points on pq that can be matched to $v_1, \ldots, v_{m-1}$ under a $(1+\varepsilon)$-approximate Fréchet matching. To this end, we uniformly partition the segment pq into segments of length at most $\varepsilon r/c_1$, where c_1 is a sufficiently large constant which we define later. Let Π be the set of vertices of this implicit partition. For each vertex v_i, for $i = 1, \ldots, m-1$, we compute its nearest point on pq, and let $V_i \subseteq \Pi$ be the set of all vertices in Π that are in distance at most $2r$ from v_i. The set V_i is the set of candidate points to match v_i in the matching that realizes the Fréchet distance.

Now, we build a graph G where $\bigcup_i V_i$ is the multiset of vertices. Two points $x \in V_i$ and $y \in V_{i+1}$ are connected by a direct edge in this graph if and only if y is after x in the oriented segment pq. See Figure 4.2 for a schematic illustration. The price of such an edge (x, y) is a $(1+\varepsilon/4)$-approximation to the Fréchet distance between $Z\langle v_i, v_{i+1}\rangle$ and xy. The portion $Z\langle v_i, v_{i+1}\rangle$ of the curve corresponds to a node in T, and this node has an associated data structure that can answer such queries in constant time (see Lemma 4.2.4). For any point $x \in V_1$, we directly compute the Fréchet distance v_0v_1 with $\mathsf{p}x$. Similarly, we compute, for each $y \in V_{m-1}$, the Fréchet distance of the segment $v_{m-1}v_m$ to the segment $y\mathsf{q}$. We add the corresponding edges to G together with the vertices p and q.

Using a variant of Dijkstra's algorithm for bottleneck shortest paths, we now compute a path in this graph which minimizes the maximum cost of any single edge visited by the path, connecting p with q. The cost of this path is returned as the approximation to the Fréchet distance between $Z\langle u, v\rangle$ and pq. Intuitively, this path corresponds to the cheapest matching of $Z\langle u, v\rangle$ (broken into subcurves by the vertices

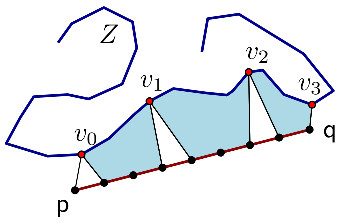


Figure 4.3: Illustration of the error introduced by snapping.

$v_0, \ldots, v_m)$ with $V_0 \times V_1 \times \cdots V_{m-1} \times V_m$, where $V_0 = \{\mathsf{p}\}$, $V_m = \{\mathsf{q}\}$, and every subcurve $Z\langle v_i, v_{i+1}\rangle$ is matched with two points in the corresponding sets V_i and V_{i+1}.

4.2.3.2 Analysis

Query time. Computing the set of vertices $v_0, v_1, \ldots, v_m$ takes $O(m) = O(\log n)$ time. The graph G has $N = O(m/\varepsilon)$ vertices and they can be computed in $O(m/\varepsilon)$ time. In particular, the number of vertices in V_i is bounded by $O(1/\varepsilon)$, since they are spread apart on a line segment by $\varepsilon r/c_1$ and contained inside a ball of radius $2r$. Thus, the graph has $O\Big((1/\varepsilon)^{-2}\Big)$ edges connecting V_i with V_{i+1} and $M = O\big(m/\varepsilon^2\big)$ edges in total. The cost of each edge can be computed in constant time, see Lemma 4.2.4. Computing the cheapest path between p and q in G can be done in $O(N \log N + M) = O\big((m/\varepsilon)\log(m/\varepsilon) + m/\varepsilon^2\big)$ time, using Dijkstra's algorithm for bottleneck shortest paths. Overall, the query time is

$$O\big(m + (m/\varepsilon)\log(m/\varepsilon) + m/\varepsilon^2\big) = O\big(\varepsilon^{-2}\log n \log\log n\big)\,.$$

Quality of approximation. Consider the matching that realizes the Fréchet distance between the query segment pq and the subcurve $Z\langle u, v\rangle$, and break it at the vertices of $v_0, \ldots, v_m$. Now, snap the matching such that the endpoints of $Z\langle v_i, v_{i+1}\rangle$ are mapped to their closest vertices in V_i and V_{i+1}, respectively, for all i. This introduces an error of at most $\varepsilon r/c_1 \leq (\varepsilon/3) d_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq})$, if we choose $c_1 \geq 9$, see Figure 4.3 for an illustration. We get another factor of $(1 + \varepsilon') = (1 + \varepsilon/3)$ error since we are approximating the price of these portions using Lemma 4.2.4. Therefore, the approximation has price at most

$$(1 + \varepsilon/3)(1 + \varepsilon/3) \cdot d_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq}) \leq (1 + \varepsilon) \cdot d_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq})\,.$$

Preprocessing time and space. Building the data structure described in Theorem 4.2.2 takes $O(n \log^2 n)$ time. For each node v of this tree, building the data structure of Lemma 4.2.4 takes $O\big(\chi^2 n_v \log n_v\big)$ time per node, where n_v is the number of vertices of the curve stored in the subtree of v. As such, overall, the preprocessing time is $O\big(\chi^2 n \log^2 n\big)$. For each node, this data structure requires $O\big(\chi^2\big)$ space and thus the overall space usage is $O\big(\chi^2 n\big)$, where $\chi = \varepsilon^{-d}\log(1/\varepsilon)$.

Putting the above together, we get the following result.

Theorem 4.2.5 *Given a polygonal curve Z with n vertices in $\mathbb{R}^d$, one can build a data structure, in $O(\chi^2 n \log^2 n)$ time, that uses $O(n\chi^2)$ space, such that for a query segment pq, and any two points u and v on the curve (and the segments of the curve that contain them), one can $(1+\varepsilon)$-approximate the distance $d_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq})$ in $O(\varepsilon^{-2} \log n \log\log n)$ time, and $\chi = \varepsilon^{-d} \log(1/\varepsilon)$.*

We emphasize that the result of Theorem 4.2.5 assumed nothing on the input curve Z. In particular, the curve Z is not necessarily c-packed.

4.3 Data structure for queries with polygonal curves of low complexity

We would like to extend the data structure described in Section 4.2.1 to support queries with curves of more than one segment. For this, we first introduce a new method to represent a polygonal curve in a way such that we can extract a simplification with a small number of segments quickly. We describe this method in Section 4.3.1 and we describe the extension of the data structure in Section 4.3.2.

4.3.1 Universal vertex permutation

We use the data structure described in Section 4.2.3 to preprocess Z, such that, given a number of vertices $k \in \mathbb{N}$, we can quickly return a simplification of Z which has

(i) $2k-1$ vertices of the original curve and
(ii) minimal Fréchet distance to Z, up to a constant factor, compared to any simplification of Z with only k vertices.

The idea is to compute a permutation of the vertices, such that the curve formed by the first k vertices in this permutation is a good approximation to the optimal simplification of a curve using (roughly) k vertices.

Related Work. There is a large body of literature on curve simplification. Since this is not the main subject of this chapter, we only discuss a selection of results which we consider most relevant, since they use the Fréchet distance as a quality measure. Agarwal *et al.* [4] give a near-linear time approximation algorithm to compute a simplification which is has Fréchet distance at most ε to the original curve and whose size is at most the size of the optimal simplification with error $\varepsilon/2$. Abam *et al.* [1] study the problem in the streaming setting, where one wishes to maintain a simplification of the prefix seen so far. Their algorithm achieves an $O(1)$ competitive ratio using $O(k^2)$ additional storage and maintains a curve with $2k$ vertices which has a smaller Fréchet distance to the prefix than the optimal Fréchet simplification with k vertices. Bereg *et al.* [23] give an exact $O(n \log n)$ algorithm that minimizes the number of vertices in the simplification, but using the discrete Fréchet distance, where only distances between the vertices of the curves are considered. Simplification under the Fréchet distance has also been studied by Guibas *et al.* [84].

Definition 4.3.1 Let Z be a polygonal curve with vertices V_Z. Let $V \subseteq V_Z$ be a subset of the vertices that contains the endpoints of Z. We call the polygonal curve obtained by connecting the vertices in V in their order along Z a ***spine curve*** of Z and we denote it with Z_V. Additionally we may call Z_V a ***k-spine curve*** of Z if it has k vertices.

Definition 4.3.2 Given a polygonal curve Z and a permutation $\Phi = \langle v_1, \ldots, v_n \rangle$ of the vertices of Z, where v_1 and v_2 are the endpoints of Z, let V_i be the subset $\{v_j \mid 1 \leq j \leq i\}$ of the vertices for any $2 \leq i \leq n$. We call Φ a ***universal vertex permutation*** if it holds that

(i) $c_1 d_{\mathcal{F}}(Z_{V_i}, Z) \geq d_{\mathcal{F}}\big(Z_{V_{i+1}}, Z\big)$, for any $2 \leq i < n$, and

(ii) $d_{\mathcal{F}}(Z_{V_i}, Z) \leq c_2 d_{\mathcal{F}}(Y, Z)$, for any polygonal curve Y with $\lceil i/c_3 \rceil$ vertices,

where c_1, c_2 and c_3 are constants larger than one which do not depend on n.

4.3.1.1 Construction of the permutation

We compute a universal vertex permutation of Z. The idea of the algorithm is to estimate for each vertex the error introduced by removing it, and repeatedly remove the vertex with the lowest error in a greedy fashion.

Specifically, for each vertex v that is not an endpoint of Z, let v^- be its predecessor on Z and let v^+ be its successor on Z. Let ϕ_v be a (11/10)-approximation of $d_{\mathcal{F}}(Z\langle v^-, v^+\rangle, v^-v^+)$. Insert the vertex v with weight ϕ_v into a min-heap $\mathcal{H}$. Repeat this for all the internal vertices of Z.

At each step, the algorithm extracts the vertex v from the heap $\mathcal{H}$ having minimum weight. Let $u = v^-(Z_{\mathcal{H}})$ and $w = v^+(Z_{\mathcal{H}})$ be the predecessor and successor of v in the curve $Z_{\mathcal{H}}$, respectively, where $\mathcal{H}$ denotes the set of vertices currently in the heap with the addition of the two endpoints of Z.

The algorithm removes v from $\mathcal{H}$ and updates the weight of u and w in $\mathcal{H}$ (if the vertex being updated is an endpoint of Z its weight is $+\infty$ and its weight is not being updated). Updating the weight of a vertex u is done by computing its predecessor and successor vertices in the current curve $Z_{\mathcal{H}}$ (i.e., $u^- = u^-(Z_{\mathcal{H}})$ and $u^+ = u^+(Z_{\mathcal{H}})$) and approximating the Fréchet distance of the subcurve of (the *original* curve) Z between these two vertices and the segment u^-u^+. Formally, the updated weight of u is ϕ_u, which is a (11/10)-approximation to

$$d_{\mathcal{F}}\big(Z\big\langle u^-, u^+\big\rangle, u^-u^+\big).$$

The updated weight of w is computed in a similar fashion.

The algorithm stops when $\mathcal{H}$ is empty. Reversing the order of the handled vertices, results in a permutation $\langle v_1, \ldots, v_n\rangle$, where v_1 and v_2 are the two endpoints of Z.

Implementation details. Using Theorem 4.2.5, the initialization takes $O\big(n \log^2 n\big)$ time overall, using $\varepsilon = 1/10$. In addition, the algorithm keeps the current set of vertices of $\mathcal{H}$ in a doubly linked list in the order in which the vertices appear along the original curve Z. In each iteration, the algorithm performs one extract-min from the min-heap $\mathcal{H}$, and calls the data structure of Theorem 4.2.5 twice to update the weight of the two neighbors of the extracted vertex. As such, overall, the running time of this algorithm is $O\big(n \log^2 n\big)$.

Extracting a spine curve quickly. Given a parameter K, we would like to be able to quickly compute the spine curve Z_{V_K}, where $V_K = \{v_1, \ldots, v_K\}$. To this end, we compute for $i = 1, \ldots \lfloor \log_2 n \rfloor$, the spine curve $Z_{V_{2^i}}$ by removing the unused vertices from $Z_{V_{2^{i+1}}}$. Naturally, we also store the original curve Z. Clearly, one can store these $O(\log n)$ curves in $O(n)$ space, and compute them in linear time. Now, given K, one can find the first curve in this collection that has more vertices than K, copy it, and remove from it all the unused vertices. Clearly, this query can be answered in $O(K)$ time.

4.3.1.2 Analysis

Lemma 4.3.3 *Let $\langle v_1, \ldots, v_n \rangle$ be the permutation computed above. Consider a value k, and let $V_k = \{u_1, \ldots, u_k\}$ be an ordering of the vertices of $v_1, \ldots, v_k$ by their order along Z. Then, it holds that $d_{\mathcal{F}}(Z, Z_{V_k}) \leq \max_{1 \leq i \leq k-1} d_{\mathcal{F}}(Z\langle u_i, u_{i+1}\rangle, u_i u_{i+1})$.*

Proof: This is immediate as one can combine for $i = 1, \ldots, k-1$, the matchings realizing $d_{\mathcal{F}}(Z\langle u_i, u_{i+1}\rangle, v_i u_{i+1})$ to obtain matchings of Z_{V_k} and Z, and such that the Fréchet distance is the maximum used in any of these matchings. □

Let $v_1, \ldots, v_n$ be the permutation of the vertices of Z as computed in the preprocessing stage, and let $\phi(v_i)$ denote weight of vertex v_i at the time of its extraction. We have the following three lemmas to prove that the computed permutation is universal.

Lemma 4.3.4 *For any $1 \leq i \leq n$, it holds that $\max_{i \leq j \leq n} \phi(v_j) \leq 4\phi(v_i)$.*

Proof: We show that the weight of a vertex at the time of extraction is at most 4 times smaller than the final weight of any of the vertices extracted before this vertex. Let v_i be a vertex and let $\phi_j(v_i)$ be the weight of this vertex at the time of extraction of some other vertex v_j, with $j > i$. Clearly, $\phi(v_j) = \phi_j(v_j) \leq \phi_j(v_i)$, since the algorithm extracted v_j with the minimum weight at the time. If $\phi(v_i) = \phi_i(v_i) \geq \phi_j(v_i)$ then the claim holds.

Otherwise, if $\phi(v_i) = \phi_i(v_i) < \phi_j(v_i)$, then there must be a vertex which caused the weight of v_i to be updated. Let k be the minimum index such that $j \geq k > i$ and $\phi_j(v_i) = \phi_k(v_i)$. We have that $\phi(v_i)$ is a $\frac{11}{10}$-approximation of the Fréchet distance $d_{\mathcal{F}}\big(u^i w^i, Z\big\langle u^i, w^i\big\rangle\big)$ for two vertices u^i and w^i. Similarly, we have that $\phi_k(v_i)$ is a $\frac{11}{10}$-approximation of the Fréchet distance $d_{\mathcal{F}}\big(u^k w^k, Z\big\langle u^k, w^k\big\rangle\big)$ for two vertices u^k and w^k. Observe that since the extraction of v_k caused the weight of v_i to be updated, it must be that $Z\big\langle u^k, w^k\big\rangle$ is a subcurve of $Z\big\langle u^i, w^i\big\rangle$. Hence, by Lemma 4.1.5, we have that

$$\frac{10}{11} \cdot \phi_k(v_i) \leq d_{\mathcal{F}}\big(u^k w^k, Z\big\langle u^k, w^k\big\rangle\big) \leq 3 d_{\mathcal{F}}\big(u^i w^i, Z\big\langle u^i, w^i\big\rangle\big) \leq 3 \cdot \frac{11}{10} \cdot \phi(v_i).$$

Now it follows that $\phi(v_j) \leq \phi_j(v_i) = \phi_k(v_i) \leq 4\phi(v_i)$, which proves the claim. □

Lemma 4.3.5 *For any $3 \leq i \leq n$ it holds that $d_{\mathcal{F}}(Z_{V_i}, Z) \leq 5\phi(v_{i+1})$.*

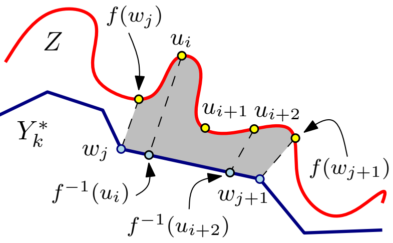
Proof: Let $u_1, \ldots, u_i$ be the vertices in V_i in the order in which they appear on Z_{V_i}. Consider the mapping between Z and this spine curve, which associates every edge $u_j u_{j+1}$ of Z_{V_i} with the subcurve $Z\langle u_j, u_{j+1}\rangle$. Clearly, it holds that

$$d_{\mathcal{F}}(Z, Z_{V_i}) \le \max_{1 \le j < i} d_{\mathcal{F}}(Z\langle u_j, u_{j+1}\rangle, u_j u_{j+1}) \le \frac{11}{10} \max_{i < j \le n} \phi(v_j).$$

Indeed, if u_{j+1} is the successor of u_j on Z, then $d_{\mathcal{F}}(Z\langle u_j, u_{j+1}\rangle, u_j u_{j+1}) = 0$, otherwise, there must be a vertex which appears on Z in between u_j and u_{j+1}, which is contained in $V_n \setminus V_i$ and the weight of this vertex is the approximation of this distance at the time of extraction. Now it follows by Lemma 4.3.4 that $d_{\mathcal{F}}(Z, Z_{V_i}) \le 5\phi(v_{i+1})$. □

Lemma 4.3.6 *For any $2 \le k \le n/2-1$, let Y_k^* be the curve with the smallest Fréchet distance from Z with k vertices (note, that Y_k^* is not restricted to have its vertices lying on Z). We have that $d_{\mathcal{F}}(Z, Y_k^*) \ge (5/11)\phi(v_{K+1})$, where $K = 2k-1$.*

Proof: Let $f : Y_k^* \to Z$ be the mapping realizing the Fréchet distance between Y_k^* and Z. Let $V_i = \langle v_1, \ldots, v_i\rangle$, for $i = 1, \ldots, n$.



Since Y_k^* has only k vertices, it breaks Z into $k-1$ subcurves. Since, $K \ge 2(k-1)+1$, there must be three consecutive vertices u_i, u_{i+1}, u_{i+2} on Z_{V_K} and two vertices w_j, w_{j+1} of Y_k^*, such that the vertices u_i, u_{i+1}, u_{i+2} appear on the subcurve $Z' = Z\langle f(w_j), f(w_{j+1})\rangle$, see the figure on the right.

Now, $f^{-1}(u_i)f^{-1}(u_{i+2}) \subseteq w_j w_{j+1}$ and by Lemma 4.1.2, we have

$$\begin{aligned} d_{\mathcal{F}}(Z, Y_k^*) &\ge d_{\mathcal{F}}\Big(Z\Big\langle f(w_j), f(w_{j+1})\Big\rangle, w_j w_{j+1}\Big) \ge d_{\mathcal{F}}(Z\langle u_i, u_{i+2}\rangle, f^{-1}(u_i)f^{-1}(u_{i+2})) \\ &\ge d_{\mathcal{F}}(Z\langle u_i, u_{i+2}\rangle, f^{-1}(u_i)f^{-1}(u_{i+2})) \ge \frac{1}{2} d_{\mathcal{F}}\Big(Z\langle u_i, u_{i+2}\rangle, \mathrm{spine}(Z\langle u_i, u_{i+2}\rangle)\Big) \\ &\ge \frac{1}{2} \cdot \frac{10}{11}\phi_{K+1}(u_{i+1}) \ge \frac{5}{11}\phi_{K+1}(v_{K+1}) = \frac{5}{11}\phi(v_{K+1}), \end{aligned}$$

as the simplification algorithm removed the minimum weight vertex at time $K+1$ (i.e., v_{K+1}). □

4.3.1.3 The result

Theorem 4.3.7 *Given a polygonal curve Z with n edges, we can preprocess it using $O(n)$ space and $O\big(n \log^2 n\big)$ time, such that, given a parameter $k \in \mathbb{N}$, we can output in $O(k)$ time a $(2k-1)$-spine curve Z' of Z and a value δ, such that*

(i) $\delta/11 \le d_{\mathcal{F}}(Y_k^*, Z)$, *and*

(ii) $d_{\mathcal{F}}(Z', Z) \le \delta$,

where Y_k^* is the polygonal curve with k vertices with minimal Fréchet distance from Z. (For $k \ge n/2$ we output Z and $\delta = 0$).

Proof: The algorithm computing the universal vertex permutation and its associated data structure is described above, for $K = 2k-1$. Specifically, it returns the spine curve $Z' = Z_{V_K}$ as the required approximation, with the value $\delta = 5\phi(v_{K+1})$. Computing Z' takes $O(k)$ time. By Lemma 4.3.5 and Lemma 4.3.6, we have that Z' and δ satisfy the claim.

Building the data structure takes $O\big(n \log^2 n\big)$ time, and it uses $O(n)$ space using $\varepsilon = 1/10$. Each query to this data structure takes $O(\log n \log\log n)$ time. We perform a constant number of these queries to the data structure per extraction from the heap, thus getting the claimed preprocessing time. □

4.3.2 Extending the data structure

We use the universal vertex permutation described in the previous section to extend our data structure of Section 4.2.1 to support queries with more than one segment.

4.3.2.1 The data structure

The input is a polygonal curve $Z \in \mathbb{R}^d$ with n vertices.

Preprocessing. Similar to the algorithm of Section 4.2.1, build a balanced binary tree T on Z. For every internal node ν of T construct the data structure of Theorem 4.3.7 for $\mathrm{cr}(\nu)$, denoted by $\mathcal{D}_\nu$, and store it at ν.

Answering a query. Given any two vertices u and v of Z, and a query polygonal curve Q with k segments, the task is to approximate $d_{\mathcal{F}}(Q, Z\langle u, v\rangle)$. We initially proceed as in Section 4.2.1, computing in $O(\log n)$ time, $m = O(\log n)$ nodes $\nu_1, \ldots, \nu_m$ of T, such that $Z\langle u, v\rangle = \mathrm{cr}(\nu_1) \oplus \mathrm{cr}(\nu_2) \oplus \cdots \oplus \mathrm{cr}(\nu_k)$. Now, extract a simplified curve with K vertices from $\mathcal{D}_{\nu_i}$, denoted by $\mathrm{simpl}_{\mathrm{K}}(\nu_i)$, for $i = 1, \ldots, m$, where $K = 2k-1$. For $i = 1, \ldots, m$, let δ_i denote the simplification error (as returned by $\mathcal{D}_{\nu_i}$), where $d_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i), \mathrm{cr}(\nu_i)) \leq \delta_i$ and $\delta_i/11$ is a lower bound to the Fréchet distance of *any* curve with at most k vertices from $\mathrm{cr}(\nu_i)$, for $i = 1, \ldots, m$ (see Theorem 4.3.7).

Next, compute the polygonal curve $S = \mathrm{simpl}_{\mathrm{K}}(\nu_1) \oplus \cdots \oplus \mathrm{simpl}_{\mathrm{K}}(\nu_m)$, and its Fréchet distance from Q; that is, $d = d_{\mathcal{F}}(S, Q)$. We return

$$\Delta = d + \max_{1 \leq i}^{m} \delta_i, \tag{4.4}$$

as the approximate distance between Q and $Z\langle u, v\rangle$.

4.3.2.2 Analysis

Query time. Extracting the $m = O(\log n)$ relevant nodes takes $O(\log n)$ time. Querying these m data structures for the simplification of the respective subcurves, takes $O(km)$ overall, by Theorem 4.3.7. Computing the Fréchet distance between the resulting simplification S of $Z\langle u, v\rangle$, which has $O(mk)$ edges, and Q takes $O(k^2 m \log(k^2 m))$ time [12]. Thus the overall time used for answering a query is bounded by

$$O\big(m + km + k^2 m \log(k^2 m)\big) = O\big(k^2 m \log(km)\big) = O\big(k^2 \log n \log(k \log n)\big).$$

Preprocessing time and space. Building the initial tree T takes $O(n)$ time and it requires $O(n)$ space. Let n_ν denote the number of vertices of $\mathrm{cr}(\nu)$. For each node ν, computing the additional information and storing it requires $O(n_\nu)$ space and $O\bigl(n_\nu \log^2 n_\nu\bigr)$ time. Recall that T is a balanced binary tree and for the nodes $\nu_1, \ldots, \nu_t$ contained in one level of the tree it holds that $\sum_{1\le i}^{t} n_{\nu_i} = n$. Thus, computing and storing the additional information takes an additional $O\bigl(n \log^3 n\bigr)$ time and $O(n \log n)$ space by Theorem 4.3.7.

Quality of approximation. By the following lemma the data structure achieves a constant-factor approximation.

Lemma 4.3.8 *Given a polygonal curve Z and a query curve Q with k segments, the value Δ (see Eq. (4.4)) returned by the above data structure is a constant-factor approximation to $d_{\mathcal{F}}(Q, Z\langle u, v\rangle)$.*

Proof: Clearly, Δ bounds the required distance from above, as one can extract a matching of Q and $Z\langle u, v\rangle$ realizing Δ. As such, we need to prove that $\Delta = O(r)$, where $r = d_{\mathcal{F}}(Q, Z\langle u, v\rangle)$.

So, let $f : Q \to Z\langle u, v\rangle$ be the mapping realizing $r = d_{\mathcal{F}}(Q, Z\langle u, v\rangle)$, and let $Q_i = f^{-1}(\mathrm{cr}(\nu_i))$, for $i = 1, \ldots, m$. Clearly, $r = \max_i d_{\mathcal{F}}(Q_i, \mathrm{cr}(\nu_i))$. Since Q_i has at most k vertices, by Theorem 4.3.7, we have

$$\frac{\delta_i}{11} \le d_{\mathcal{F}}(Q_i, \mathrm{cr}(\nu_i)) \le r, \qquad \text{and} \qquad d_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i)\,, \mathrm{cr}(\nu_i)) \le \delta_i, \tag{4.5}$$

for $i = 1, \ldots, m$. In particular, we have $\delta_i \le 11r$. Now, by the triangle inequality, we have that

$$d_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i)\,, Q_i) \le d_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i)\,, \mathrm{cr}(\nu_i)) + d_{\mathcal{F}}(\mathrm{cr}(\nu_i)\,, Q_i) \le \delta_i + r \le 12r.$$

As such, $d = d_{\mathcal{F}}(S, Q) \le \max_i d_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i)\,, Q_i) \le 12r$. Now, $\Delta = d + \max_i \delta_i \le 12r + 11r = 23r$. □

The result. Putting the above together, we get the following result. We emphasize that k is being specified together with the query curve, and the data structure works for any value of k.

Theorem 4.3.9 *Given a polygonal curve Z with n edges, we can preprocess it in $O(n \log^3 n)$ time and $O(n \log n)$ space, such that, given a query specified by*

(i) a pair of points u and v on the curve Z,

(ii) the edges containing these two points, and

(iii) a query curve Q with k segments,

one can approximate $d_{\mathcal{F}}(Q, Z\langle u, v\rangle)$ up to a constant factor in $O\bigl(k^2 \log n \log(k \log n)\bigr)$ time.

Proof: The preprocessing is described and analyzed above. The query procedure needs to be modified slightly since the u and v are not necessarily vertices of Z. However, this can be done the same way as for the initial data structure in Theorem 4.2.2.

Let u', v' be the first and last vertices of Z contained in $Z\langle u, v\rangle$. We now extract the $m = O(\log n)$ nodes $\nu_1, \ldots, \nu_m$ of T, such that

$$X = uu' \oplus \mathrm{cr}(\nu_1) \oplus \ldots \oplus \mathrm{cr}(\nu_m) \oplus v'v = Z\langle u, v\rangle \, .$$

We continue with the procedure as described above using this node set. The analysis of Lemma 4.3.8 applies with minor modifications. □

4.4 Concluding remarks

In this chapter we saw several data structures to preprocess a polygonal curve Z in $\mathbb{R}^d$ to answer different types of Fréchet-distance queries between subcurves of Z and a query curve. In this setting, we assumed that the endpoints of the subcurve of Z are given with the query. This will be useful in the next chapter, where we discuss a variant of the Fréchet distance that allows shortcuts on the input curves, which have to be matched under the Fréchet distance. However one can also imagine other settings. For example,

(i) Given a query curve Q, report the subcurve of Z that minimizes the Fréchet distance.

(ii) Given a query curve Q and a distance threshold δ, report all subcurves that are within distance δ, or report the number of those curves.

(iii) Given a query curve Q and a distance threshold δ, report all subcurves that are within distance δ to a transformed copy of Q.

There is little previous work on this topic. The problem variant in (ii) has been studied by de Berg *et al.* [53] and recently also by Gudmundsson and Smid under the c-packedness assumption [83]. To our knowledge theirs and ours are the first data structures of this kind.

One can also imagine Z to be a geometric graph. In this case, the problem variant (i) becomes a map-matching problem. There is more work on map-matching curves under the Fréchet distance, see [43] and references therein. In particular, the problem has first been studied by Alt *et al.* [10].

Furthermore, we studied the new concept of a universal vertex permutation, which defines an ordering of the vertices of Z by their "Fréchet error". By removing the vertices in this order, one obtains a series of simplifications of Z which are approximately optimal with respect to their Fréchet distance to Z. This enables a fast extraction of a simplification of Z of a given resolution. We expect this concept to be used in the design of future data structures for Fréchet-distance queries.

Finally, it would be interesting to extend the data structures in this chapter to other settings as discussed above. We conclude by outlining some ideas in this direction.

Open Problem 4.1 In order to extend our data structure to a query setting, where the subcurve of Z is not fixed, in particular (ii) described above, one needs to compute candidate endpoints for such subcurves of Z. If Z can be assumed to be c-packed, then the maximal number of such candidate endpoints is considerably restricted compared to the general case. Naively, one would have to consider one point per component of

Z inside the two balls of radius δ centered at the endpoints of Q. Even under the c-packedness assumption there might be many such components. However, to achieve a $(1+\varepsilon)$-approximation, it suffices to consider one point per component in the ball of radius $(1+\varepsilon)\delta$ and only of those components that intersect the ball of radius δ. Refer to Figure 4.4 for an example of such a candidate set. In this case, the number of query candidates can be bounded by $O(c/\varepsilon)$. If we can preprocess the curve for queries of this type, we can use the data structures in this chapter to approximate the Fréchet distance for every pair of candidate endpoints, one from each ball, to obtain a data structure with polylogarithmic query time.

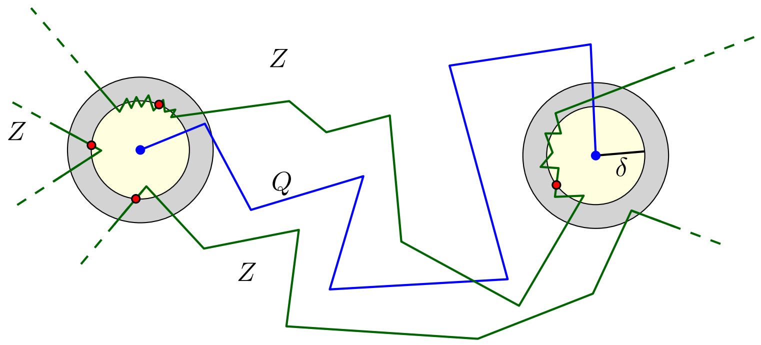


Figure 4.4: A small candidate set of endpoints of subcurves of Z which may be within Fréchet distance 2δ to the query curve Q.

CHAPTER 5

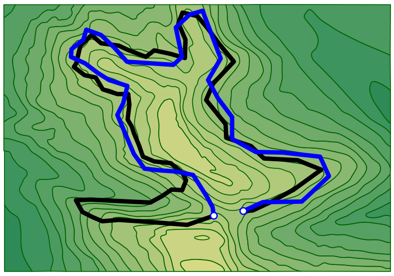
The Fréchet distance with shortcuts

In this chapter, we introduce a variant of the Fréchet distance, where one is allowed to take shortcuts on the input curves in order to decrease the Fréchet distance. The goal is to define a partial distance measure which is more robust against noise than the standard Fréchet distance. We motivate the problem in Section 5.1 and give the definition of the *directed k-shortcut Fréchet distance* in Section 5.2. In Section 5.3 we study the problem of computing this distance measure for two given polygonal curves in the case where k shortcuts are allowed between vertices of one of the two input curves. We study the case that k is bounded and the unbounded case. We develop polynomial-time exact algorithms and faster approximation algorithms for these cases. In Section 5.4 we study the more general case where shortcuts can start and end at any point along one of the two curves. We show that computing this version of the shortcut Fréchet distance is weakly NP-complete if an unbounded number of shortcuts is allowed. Furthermore, we extend some of our approximation results from the previous sections to this more general case. We conclude with discussion and some open problems in Section 5.5. In particular we discuss the *undirected* case, where shortcuts are allowed on both curves.

5.1 Introduction

A major drawback of the Fréchet distance is its sensitivity to local noise, which is frequent in real data. Unlike similarity measures such as the root-mean-square deviation (RMSD), which averages over a set of similarity values, and dynamic time warping, which minimizes the sum of distances along the curves, the Fréchet distance is a so-called *bottleneck measure* and can therefore be affected to an extent which is generally unrelated to the relative amount of noise across the curves. In practice, curves might be generated by physical tracking devices, such as GPS, which is known to be inaccurate when the connection to the satellites is temporarily disturbed due to atmospheric conditions or reflections of the positioning signal on high buildings. Such inaccurate data points are commonly referred to as "outliers". Note that outliers come in batches if they are due to such a temporary external condition. Similarly, in computer vision applications, the silhouette of an object could be partially occluded, and in sound recordings, outliers may be introduced due to background sounds or breathing.

Detecting outliers in time series has been studied extensively in the literature [113]. One may also be interested in outliers as a deviation from a certain expected behavior or because they carry some meaning.



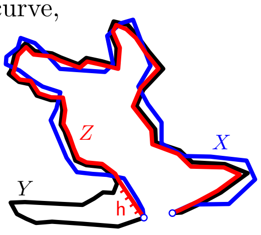
It could be, for instance, that trajectories of two hikers deviate locally, because one hiker chose to take a detour to a panoramic view point, see the figure depicted to the right. Outlier detection is inherently non-trivial if not much is known about the underlying probability distributions and the data is sparse [6]. We circumvent this problem in the computation of the Fréchet distance by minimizing over all possibilities for outlier-removal. In a sense, our approach is similar to computing a certain notion of partial similarity. Unlike other partial distance measures, the distance measure we propose is parameter-free. For comparison, in the *partial Fréchet distance*, as it was studied by Buchin *et al.* [35], one is interested in maximizing the portions of the curves which can be matched within a certain Fréchet distance (the parameter). In this case, the dissimilar portions of the curves are ignored. In our case, they are replaced by *shortcuts*, which have to be matched under the Fréchet distance. Buchin *et al.* [35] showed how to compute the partial Fréchet distance under the L_1 and L_∞ metric in roughly $O\big(n^3 \log n\big)$ time. To the best of our knowledge the problem of computing the Fréchet distance when one is allowed to introduce shortcuts has not been studied before.

The task at hand. We are given two polygonal curves X and Y in $\mathbb{R}^d$, which we perceive as a sequence of linearly interpolated measurement points. We believe that Y is similar to X but it might contain considerable noise that is occluding this similarity. That is, it might contain erroneous measurement points (outliers), which need to be ignored when assessing the similarity. We would like to apply a few edit operations to Y so that it becomes as similar to X as possible, in the process hopefully removing the noise in Y and judging how similar it really is to X. To this end, we conceptually remove subsequences of measurement points, which we suspect to be outliers, and minimize over all possibilities for such a removal.

Shortcut Fréchet distance. A shortcut replaces a subcurve between two vertices by a straight segment that connects these vertices. The part being shortcut is not ignored, but rather the new curve with the shortcuts has to be matched entirely to the other curve under the Fréchet distance. As a concrete example, consider the figure below. The Fréchet distance between X and Y is quite large, but after we shortcut the outlier "bump" in Y, the resulting new curve Z has a considerably smaller Fréchet distance to X. We are interested in computing the minimum such distance allowing an unbounded number of shortcuts.

Naturally, there are many other possibilities to try and tackle the task at hand, for example:

(i) bounding the number of shortcuts by a parameter k
(ii) allowing shortcuts on both curves,
(iii) allowing only shortcuts between vertices that are close-by along the curve,
(iv) ignoring the part being shortcut and maximizing the length of the remaining portions,
(v) allowing shortcuts to start and end anywhere along the curve,
(vi) allowing curved shortcuts, etc.



If one is interested in (iii) then the problem turns into a map-matching problem, where the start and end points are fixed and the graph is formed by the curve and its eligible shortcuts. For this problem, results can be found in the literature [43, 10]. A recent result by Har-Peled and Raichel [87] is applicable to the variant where one allows such shortcuts on both curves, i.e. (ii)+(iii). The version in (iv) has been studied under the name of *partial Fréchet distance* [35].

In this chapter, we first concentrate on the vertex-restricted shortcut Fréchet distance (see Section 5.2.1 for the exact definition) because computing it efficiently seems like a first step in understanding how to solve some of the more difficult variants, e.g., (v). Surprisingly, computing this simpler version of the shortcut Fréchet distance is already quite challenging, especially if one is interested in an efficient algorithm, see Section 5.3. As it turns out, the problem is weakly NP-hard for exact computations for variant (v), where we allow shortcuts to start and end anywhere along the curve. We will see this later in Section 5.4. Furthermore, we also discuss efficient solutions for variant (i), i.e., where at most k shortcuts are allowed.

Note that allowing shortcuts on both curves does not always yield a meaningful measure, especially if shortcuts on both curves may be matched to each other. In particular, if one of the two curves is more accurately sampled and can act as a model curve, allowing shortcuts on only one of the two curves seems reasonable.

Informal restatement of the problem. In the parametric space of the two input curves, we are given a terrain defined over a grid partitioning $[0,1]^2$, where the height at each point is defined as the distance between the two associated points on the two curves and the grid lines correspond to the vertices of the two curves. As in the regular Fréchet distance, we are interested in finding a path between $(0,0)$ and $(1,1)$ on the terrain, such that the maximum height on the path does not exceed some δ (the minimum such δ is the desired distance). This might not be possible as there might be "mountain chains" blocking the way. To overcome this, we are allowed to introduce tunnels that go through such obstacles. Each of these tunnels connects two points that lie on the horizontal lines of the grid, as these correspond to the vertices of one curve. Naturally, we require that the starting and ending points of such a tunnel have height at most δ (the current distance threshold being considered), and furthermore, the price of such a tunnel (i.e., the Fréchet distance between the corresponding shortcut and subcurve) is smaller than δ. Once we introduce these tunnels, we need to compute a *monotone* path from $(0,0)$ to $(1,1)$ in the grid which uses at most k tunnels. Finally, we need to search for the minimum δ for which there is a feasible solution.

Challenge and ideas. Let n be the total number of vertices of the input curves. A priori there are potentially $O(n^2)$ horizontal edges of the grid that might contain endpoints of a tunnel, and as such, there are potentially $O(n^4)$ different families of tunnels that the algorithm might have to consider. A careful analysis of the structure of these families shows that, in general, it is sufficient to consider one (canonical) tunnel per family. Using c-packedness and simplification, we can reduce the number of relevant grid edges to near linear. This in turn reduces the number of potential tunnels that need to be inspected to $O(n^2)$. This is still insufficient to get a near-linear time algorithm. Surprisingly, we prove that if we are interested only in a constant factor approximation, for every horizontal edge of the grid we need to inspect only a constant number of tunnels. Thus, we reduce the number of tunnels that the algorithm needs to inspect to near-linear.

And yet we are not done, as naively computing the price of a tunnel requires time near-linear in the size of the associated subcurve. To overcome this, we develop a new data structure, so that after preprocessing we can compute the price of a tunnel in polylogarithmic time per tunnel. Now, carefully putting all these insights together, we get a near-linear time algorithm for the approximate decision version of the problem.

However, to compute the minimum δ, for which the decision version returns true—which is the shortcut Fréchet distance—we need to search over the critical values of δ. To this end, we investigate and characterize the critical values introduced by the shortcut version of the problem. Using the decision procedure, we perform a binary search of several stages over these values, in the spirit of Chapter 3, to get the required approximation.

5.2 Preliminaries

5.2.1 The k-shortcut Fréchet distance

For the definitions of the standard Fréchet distance and standard concepts such as the free space diagram, refer to Section 2.1. For a polygonal curve Y, we use $\overline{Y}[y, y']$ to denote the line segment between the points $Y(y)$ to $Y(y')$ and we call this a ***shortcut*** of Y. We refer to any order-preserving concatenation of $k + 1$ non-overlapping (possibly empty) subcurves of Y with k shortcuts connecting the endpoints of the subcurves in the order along the curve, as a ***k-shortcut curve*** of Y. Formally, for values $0 \leq y_1 \leq y_2 \leq \cdots \leq y_{2k} \leq 1$, the shortcut curve is defined as $Y[0, y_1] + \overline{Y}[y_1, y_2] + Y[y_2, y_3] + \cdots + \overline{Y}[y_{2k-1}, y_{2k}] + Y[y_{2k}, 1]$. If each $Y(y_i)$ is a vertex of Y, we refer to the shortcut curve as being ***vertex-restricted***, otherwise we say it is ***unrestricted***.

Given two polygonal curves X and Y, we define their ***continuous k-shortcut Fréchet distance*** as the minimal Fréchet distance between the curve X and any unrestricted k-shortcut curve of Y. We denote it with $d_S(k, X, Y)$. If we do not want to bound the number of shortcuts, we omit the parameter k and denote it with $d_S(X, Y)$. The ***vertex-restricted k-shortcut Fréchet distance*** is defined as above using only vertex-restricted shortcut curves of Y. Furthermore, note that in all cases we allow only one of the input curves to be shortcut, namely Y, thus we call the distance measure ***directed***.

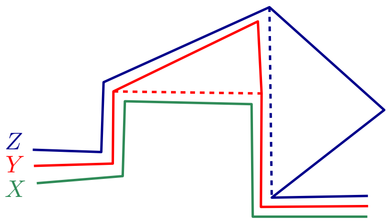


Figure 5.1: The directed k-shortcut Fréchet does not satisfy the triangle inequality. In the depicted counter-example it holds that $d_{\mathcal{S}}(k, X, Z) > d_{\mathcal{S}}(k, X, Y) + d_{\mathcal{S}}(k, Y, Z)$ for any value of k and for k unbounded. This holds true in the vertex-restricted and in the continuous case.

Unlike the standard Fréchet distance, the directed k-shortcut Fréchet distance does not satisfy the triangle inequality, see Figure 5.1. In the following, we will omit the predicates directed, vertex-restricted and continuous when it is clear from the context.

5.2.2 Tunnels in the free space diagram

In the parametric space, a shortcut $\overline{Y}[y_\mathsf{p}, y_\mathsf{q}]$ and the subcurve $X[x_\mathsf{p}, x_\mathsf{q}]$, that it is being matched to, correspond to a the rectangle with corners p and q, where $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p})$ and $\mathsf{q} = (x_\mathsf{q}, y_\mathsf{q})$. By shortcutting the curve on the vertical axis, we are collapsing this rectangle to a single row, see Example 5.2. More precisely, this is the free space diagram of the shortcut and the subcurve. We call this row a ***tunnel*** and denote it by $\tau(\mathsf{p}, \mathsf{q})$. We require $x_\mathsf{p} \leq x_\mathsf{q}$ and $y_\mathsf{p} \leq y_\mathsf{q}$ for monotonicity. Example 5.2 shows the full example of a tunnel. We call the Fréchet distance of the shortcut segment to the subcurve the ***price*** of this tunnel and denote it with $\mathrm{prc}(\tau(\mathsf{p}, \mathsf{q})) = d_{\mathcal{F}}\big(X[x_\mathsf{p}, x_\mathsf{q}], \overline{Y}[y_\mathsf{p}, y_\mathsf{q}]\big)$. A tunnel $\tau(\mathsf{p}, \mathsf{q})$ is ***feasible*** for δ if it holds that $\mathrm{dist}(\mathsf{p}) \leq \delta$ and $\mathrm{dist}(\mathsf{q}) \leq \delta$, i.e., if $\mathsf{p}, \mathsf{q} \in \mathcal{D}_{\leq\delta}(X, Y)$. Note that the feasibility of a tunnel is not equivalent with the feasibility of a monotone path in the free space of the tunnel.

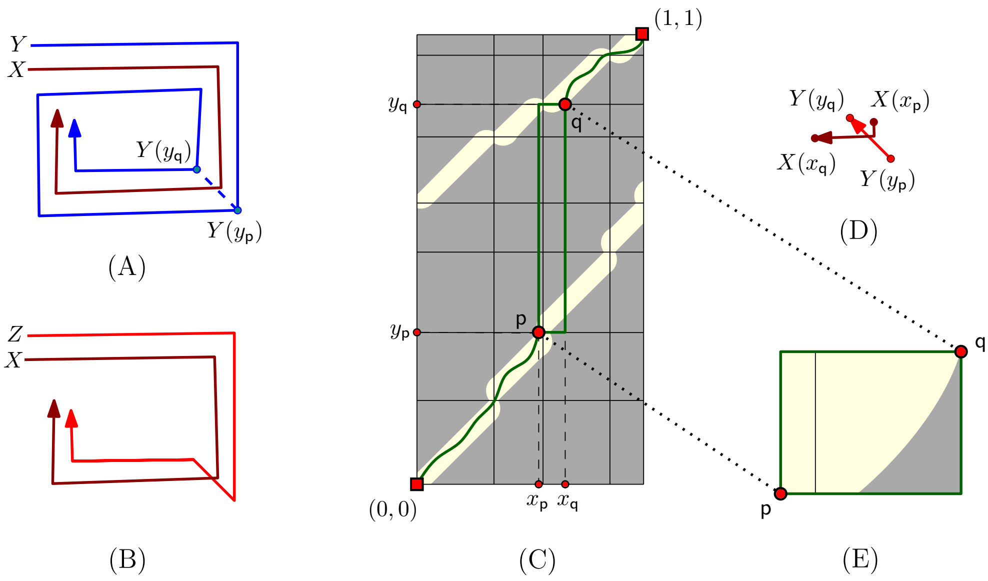
We define the ***k-reachable free space*** $\mathcal{R}^k_{\leq\delta}(X, Y)$ as

$$\mathcal{R}^k_{\leq\delta}(X, Y) = \left\{\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p}) \in [0,1]^2 \;\middle|\; d_{\mathcal{S}}(k, X[0, x_\mathsf{p}], Y[0, y_\mathsf{p}]) \leq \delta\right\}.$$

This is the set of points that have an (x, y)-monotone path from $(0, 0)$ that stays inside the free space and otherwise uses at most k tunnels.

Horizontal, vertical and diagonal tunnels. We can distinguish three types of tunnels. We call a tunnel that stays within a column of the grid, a ***vertical tunnel***. Likewise, a tunnel that stays within a row is called a ***horizontal tunnel***. Tunnels that span across rows and columns are ***diagonal tunnels***. Note that vertical tunnels that are feasible are always affordable by Observation 2.3.1, since the corresponding rectangle in the free space diagram collapses to a single free space cell in this case. Furthermore, the shortcut which corresponds to a horizontal tunnel lies within an edge of the input

Example 5.2 Shortcuts and tunnels



Two dissimilar curves X and Y are depicted in (A) that can be made similar by shortcutting one of them. The curve Z resulting from shortcutting Y is depicted in (B). Its (regular) Fréchet distance from X is dramatically reduced. The free space diagram is shown in (C). The tunnel $\tau(\mathsf{p}, \mathsf{q})$ connects previously disconnected components of the free space and corresponds to the shortcut and the subcurve matched to each other depicted in (D) and their free space diagram (E).

curve. Thus, shortcutting the curve does not have any effect in this case and we can safely ignore such horizontal tunnels.

5.2.2.1 Monotonicity of the prices of tunnels

Lemma 4.1.5 which we used in Chapter 4 implies readily that under certain conditions the prices of tunnels which share an endpoint are approximately monotone with respect to the x-coordinate of their starting point. We will exploit this in the approximation algorithm that computes the reachability in the free-space diagram. We will see in Section 5.3.5.1 that this drastically reduces the number of tunnels that need to be inspected in order to decide if a particular cell is reachable.

Lemma 5.2.1 *Given two polygonal curves X and Y and a value of $\delta \geq 0$, for any three points p, q and r in the δ-free space such that $x_\mathsf{p} \leq x_\mathsf{q} \leq x_\mathsf{r}$ it holds that* $\mathrm{prc}(\tau(\mathsf{q}, \mathsf{r})) \leq 3 \max\Big(\delta, \mathrm{prc}(\tau(\mathsf{p}, \mathsf{r}))\Big)$.

Proof: Let X_1 be the subcurve $X[x_\mathsf{p}, x_\mathsf{r}]$, and let $X_2 = X[x_\mathsf{q}, x_\mathsf{r}]$. Similarly, let $\overline{Y}_1$ be the shortcut $\overline{Y}[y_\mathsf{p}, y_\mathsf{r}]$ and let $\overline{Y}_2 = \overline{Y}[y_\mathsf{q}, y_\mathsf{r}]$. By Lemma 4.1.5 $\mathrm{prc}(\tau(\mathsf{q}, \mathsf{r})) \leq 3\delta'$ for $\delta' = \max\big(d_{\mathcal{F}}\big(X_1, \overline{Y}_1\big), \delta\big)$. □

We will see in Section 5.3.5.5 that the monotonicity property of Lemma 5.2.1 also enables a faster search over tunnel events. The property holds even if the tunnels under consideration are not valid. For example if $x_{\mathsf{p}} < x_{\mathsf{r}}$ and $y_{\mathsf{p}} > y_{\mathsf{r}}$ then the tunnel $\tau(\mathsf{p}, \mathsf{r})$ is not a valid tunnel and it cannot be used by a valid solution. Nevertheless, $\tau(\mathsf{p}, \mathsf{r})$ has a well defined price, and these prices have the required monotonicity property. The following is an easy consequence of Lemma 5.2.1.

Lemma 5.2.2 *For a parameter $\delta \geq 0$, let $\mathsf{p}_1, \ldots, \mathsf{p}_m$ be m points in the δ-free space ordered ascendingly by their x-coordinates, and let $\psi_i = \operatorname{prc}(\tau(\mathsf{p}_i, \mathsf{p}_m))$ for any $1 \leq i \leq m$. Then, we have:*
(A) If $\psi_i \geq \delta$ then for all $j > i$, we have $\operatorname{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)) \leq 3\psi_i$.
(B) If $\psi_i > 3\delta$ then for all $j < i$, we have $\operatorname{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)) \geq \psi_i/3$.

Proof: To see the first part of the claim, note that by Lemma 5.2.1, $\operatorname{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)) \leq 3\max(\delta, \psi_i) \leq 3\psi_i$. As for the second part, we have by the same lemma that $\delta < \psi_i/3 \leq \max(\delta, \operatorname{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)))$, and thus $\psi_i/3 \leq \operatorname{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m))$. □

5.2.2.2 Curve simplification

During the course of the approximation algorithm we simplify the input curves using Algorithm 3.1.2 in order to reduce the complexity of the free space. The directed k-shortcut Fréchet distance does not satisfy the triangle inequality, as can be seen by the counter-example shown in Figure 5.1. Therefore, we need the next lemma to ensure that the computed distance between the simplified curves approximates the distance between the original curves.

Lemma 5.2.3 *Given a simplification parameter μ and two polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$, let $X = \mathbf{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathbf{simpl}(\boldsymbol{Y}, \mu)$ denote their μ-simplifications, respectively. For any $k \in \mathbb{N}$, it holds that $d_{\mathcal{S}}(k, X, Y) - 2\mu \leq d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) \leq d_{\mathcal{S}}(k, X, Y) + 2\mu$. Similarly, $d_{\mathcal{S}}(X, Y) - 2\mu \leq d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y}) \leq d_{\mathcal{S}}(X, Y) + 2\mu$.*

Proof: The proof is straightforward and is included for the sake of completeness.

First, we show that $d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) \leq d_{\mathcal{S}}(k, X, Y) + 2\mu$. Let $Y' = Y_1 \oplus \overline{Y}_2 \oplus \cdots \oplus Y_{2k'+1}$ be a k'-shortcut curve of Y, with $k' \leq k$, and such that $d_{\mathcal{F}}(X, Y') \leq d_{\mathcal{S}}(k, X, Y)$ and let $X = X_1 \oplus X_2 \oplus \cdots \oplus X_{2k'+1}$ be the decomposition[1] of X induced by the matching realizing the k-shortcut Fréchet distance between X and Y'. We have that

$$d_{\mathcal{S}}(k, X, Y) = \max\left(\max_{0 \leq i \leq k'} d_{\mathcal{F}}(X_{2i+1}, Y_{2i+1}) \,,\, \max_{1 \leq i \leq k'} d_{\mathcal{F}}\big(X_{2i}, \overline{Y}_{2i}\big) \right),$$

which implies that the Fréchet distance between Y_i (resp., $\overline{Y}_i$) and X_i is at most $d_{\mathcal{S}}(k, X, Y)$ for any $1 \leq i \leq 2k' + 1$.

Consider a matching between X and $\boldsymbol{X}$ that realizes the Fréchet distance between them and consider the decomposition of $\boldsymbol{X} = \boldsymbol{X}_1 \oplus \boldsymbol{X}_2 \oplus \cdots \oplus \boldsymbol{X}_{2k'+1}$, such that X_i is matched to $\boldsymbol{X}_i$ under this matching. It holds that $d_{\mathcal{F}}(X_i, \boldsymbol{X}_i) \leq \mu$, by Lemma 3.1.3.

Now, in a similar way, let $\boldsymbol{Y}' = \boldsymbol{Y}_1 \oplus \overline{\boldsymbol{Y}}_2 \oplus \cdots \oplus \boldsymbol{Y}_{2k'+1}$ be a k'-shortcut curve of $\boldsymbol{Y}$, such that for any $0 \leq i \leq k'$, the subcurve Y_{2i+1} is matched to $\boldsymbol{Y}_{2i+1}$ under a matching

[1]Note that the matching is not necessarily one-to-one but we can obtain a suitable decomposition by breaking ties arbitrarily.

that realizes the Fréchet distance between Y and $\boldsymbol{Y}$. We have that $d_{\mathcal{F}}\big(\overline{Y}_i, \overline{\boldsymbol{Y}}_i\big) \leq \mu$ for any of the shortcuts, since their endpoints are in distance μ. It holds that

$$d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) \leq d_{\mathcal{F}}(\boldsymbol{X}, \boldsymbol{Y}') = \max\left(\max_{0 \leq i \leq k'} d_{\mathcal{F}}(\boldsymbol{X}_{2i+1}, \boldsymbol{Y}_{2i+1}) \ , \ \max_{0 \leq i \leq k'} d_{\mathcal{F}}\big(\boldsymbol{X}_{2i}, \overline{\boldsymbol{Y}}_{2i}\big) \right).$$

By the triangle inequality, we have that

$$\begin{aligned} d_{\mathcal{F}}(\boldsymbol{X}_{2i+1}, \boldsymbol{Y}_{2i+1}) &\leq d_{\mathcal{F}}(\boldsymbol{X}_{2i+1}, X_{2i+1}) + d_{\mathcal{F}}(X_{2i+1}, Y_{2i+1}) + d_{\mathcal{F}}(Y_{2i+1}, \boldsymbol{Y}_{2i+1}) \\ &\leq d_{\mathcal{S}}(k, X, Y) + 2\mu. \end{aligned}$$

Similarly, we have

$$d_{\mathcal{F}}\big(\boldsymbol{X}_{2i}, \overline{\boldsymbol{Y}}_{2i}\big) \leq d_{\mathcal{F}}(\boldsymbol{X}_{2i}, X_{2i}) + d_{\mathcal{F}}\big(X_{2i}, \overline{Y}_{2i}\big) + d_{\mathcal{F}}\big(\overline{Y}_{2i}, \overline{\boldsymbol{Y}}_{2i}\big) \leq d_{\mathcal{S}}(k, X, Y) + 2\mu.$$

This implies the second inequality in the claim. We can argue in the same way that $d_{\mathcal{S}}(k, X, Y) \leq d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) + 2\mu$, which implies the first inequality. □

5.3 The vertex-restricted shortcut Fréchet distance

In this section we study the directed, vertex-restricted k-shortcut Fréchet distance for the case of bounded and unbounded k. We will further omit the predicates directed and vertex-restricted if it is clear from the context. We give several algorithmic results for computing this distance measure. We first assume that k is unbounded and give a polynomial-time exact algorithm in Section 5.3.2. The algorithm uses a decision procedure in a binary search over a set of candidate values. Since the shortcuts introduce a new set of candidate values, we provide an elaborate study of these new events in Section 5.3.3. Based on the algorithm described in Section 5.3.2, and by using techniques developed in Chapter 3 and Chapter 4, we give a considerably faster approximation algorithm in Section 5.3.5, which runs in near-linear time if the input curves are c-packed. In Section 5.3.7 we extend the algorithm to the case where k is bounded by a given value. We begin by establishing some basic concepts and notation.

5.3.1 Canonical tunnels and gates

Let $u = Y(y_{\mathsf{p}})$ and $v = Y(y_{\mathsf{q}})$ and let e be the edge of X that contains $X(x_{\mathsf{p}})$ (resp., e' the edge that contains $X(x_{\mathsf{q}})$) for the tunnel $\tau(\mathsf{p}, \mathsf{q})$. We denote with $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$ the ***family of tunnels*** that $\tau(\mathsf{p}, \mathsf{q})$ belongs to. Furthermore, let $\mathcal{T}_{\leq\delta}(\mathsf{e}, \mathsf{e}', u, v)$ denote the subset of these tunnels that are feasible for δ.

The ***canonical tunnel*** of the tunnel family $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$, denoted by $\tau_{\min}(\mathsf{e}, \mathsf{e}', u, v)$, is the tunnel that matches the shortcut uv to the subcurve $X[s, t]$, such that s and t are the values realizing

$$\mathrm{r}_{\min}(\mathsf{e}, \mathsf{e}', u, v) = \min_{\substack{X(s) \in \mathsf{e}, X(t) \in \mathsf{e}', \\ s \leq t}} \max(\|X(s) - u\|, \|X(t) - v\|). \tag{5.1}$$

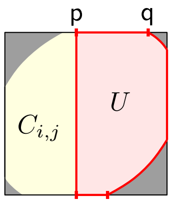
We refer to $\mathrm{r}_{\min}(\mathsf{e}, \mathsf{e}', u, v)$ as the ***minimum radius*** of this family. The canonical tunnel may not be uniquely defined if only one of the two values s or t determines the minimum radius. In this case, we define s and t as the values minimizing $\|X(s) - u\|$ and $\|X(t) - v\|$ for $X(s) \in \mathsf{e}$ and $X(t) \in \mathsf{e}'$, individually. We call the price of the canonical tunnel the ***canonical price*** of this tunnel family.

Clearly, one can compute the canonical tunnel $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$ in constant time. In particular, the price of this canonical tunnel is

$$\operatorname{prc}(\tau_{\min}(\mathsf{e}, \mathsf{e}', u, v)) = d_{\mathcal{F}}(X[s,t]\,, uv)\,. \tag{5.2}$$

We emphasize that a shortcut is always a segment connecting two vertices of the curve Y, and a tunnel always lies in the parametric space; that is, they exist in two completely different domains.

Observation 5.3.1 The minimum radius of a tunnel family $\mathrm{r}_{\min}(\mathsf{e}, \mathsf{e}', u, v)$ corresponds to either (i) the distance of u to its closest point on e, (ii) the distance of v to its closest point on e', or (iii) the common distance of u and v to the intersection of their bisector with the edge e (i.e., a vertex-vertex-edge distance, see Section 2.1). Note that the event in case (iii) can only happen if $\mathsf{e} = \mathsf{e}'$.



Gates. Let U be a subset of the parametric space that is convex in every cell. Let $I^h_{i,j}$ be a free space interval. We call the left and right endpoints of $U \cap I^h_{i,j}$ the left and right ***gates*** of U in the cell $C_{i,j}$. The figure to the right shows an example of gates p and q. The ***set of gates*** of U are the gates with respect to all cells in the free space diagram. We define the ***canonical gate*** of a vertex-edge pair as the point in parametric space that minimizes the vertex-edge distance. Note that canonical gates serve as endpoints of diagonal canonical tunnels.

5.3.2 A polynomial-time exact algorithm

Here we describe a polynomial-time exact algorithm for the shortcut Fréchet distance for case that the number of shortcuts is unbounded. Following the approach of Alt and Godau's algorithm (see Section 2.4), we devise a decision procedure which is used in a search for the optimal distance value. We first describe the algorithm. The analysis can be found in Section 5.3.4.

5.3.2.1 The xTunnel procedure

A key element in the decision procedure is the **xTunnel** procedure (Algorithm 5.3.2). Intuitively, this procedure receives as input a set of reachable points in the parametric space and a free space interval (in the form of the left gate) and we are asking if there exists an ***affordable*** tunnel connecting a reachable point to the interval. Here, affordable means that its price is at most δ. Furthermore, we are interested in the leftmost possible endpoint of such a tunnel. More precisely, the procedure receives a set of gates R and a gate p as input and returns the endpoint of an affordable tunnel that starts at a gate of R and ends either at p or the closest point to the right of p in the same free space interval. During the decision procedure, we will repeatedly invoke the **xTunnel** procedure with a set of gates R, for which we already know that they are contained in the reachable free space $\mathcal{R}^{\infty}_{\leq\delta}(X,Y)$, and the left gate associated

Algorithm 5.3.2 xTunnel(R, p, δ)

```
Input: set of gates $R$; gate $\mathsf{p} \in [0,1]^2$; distance $\delta \in \mathbb{R}$
 1: Let $x_{\min} = \mathsf{null}$      // leftmost reachable $x$-coordinate
 2: for $\mathsf{q} = (x_\mathsf{q}, y_\mathsf{q}) \in R$ do
 3:    if $x_\mathsf{q} \leq x_\mathsf{p}$ and $y_\mathsf{q} \leq y_\mathsf{p}$ then
 4:       if $\mathrm{prc}(\tau(\mathsf{q},\mathsf{p})) \leq \delta$ then
 5:          Return $\mathsf{p}$      // tunnel $\tau(\mathsf{q},\mathsf{p})$
 6:    if $x_\mathsf{q} \geq x_\mathsf{p}$ and $y_\mathsf{q} \leq y_\mathsf{p}$ and not$(x_{\min} \leq x_\mathsf{q})$ then
 7:       $\mathsf{v} = (x_\mathsf{q}, y_\mathsf{p})$      // vertical tunnel $\tau(\mathsf{q},\mathsf{v})$
 8:       if $\mathrm{dist}(\mathsf{v}) \leq \delta$ then
 9:          $x_{\min} \leftarrow x_\mathsf{q}$
10: if $x_{\min} \neq \mathsf{null}$ then
11:    Return $(x_{\min}, y_\mathsf{p})$
12: else
13:    Return $\mathsf{null}$
```

with a horizontal free space interval of $\mathcal{D}_{\leq\delta}(X,Y)$, in order to determine, if and to which extent this interval is reachable.

The **xTunnel** procedure can be described as follows. We simply test all tunnels that connect a gate q of R to p. This can be done by testing the Fréchet distance between the shortcut $\overline{Y}[y_\mathsf{q}, y_\mathsf{p}]$ and the subcurve $X[x_\mathsf{q}, x_\mathsf{p}]$ using the algorithm by Alt and Godau, see Section 2.4. For gates that lie to the right of p, we check if the corresponding vertical tunnel ends inside the free space interval connected to p. We ignore gates that lie above p. We return the leftmost point in the free space interval connected to p that can be reached by a tunnel of price at most δ, if such a point exists. Otherwise, we return null. The resulting procedure is layed out in Algorithm 5.3.2.

5.3.2.2 The decision algorithm

In the decision problem we want to know whether the shortcut Fréchet distance between two curves, X and Y, is at most a given value δ. The free space diagram $\mathcal{D}_{\leq\delta}(X,Y)$ may consist of a certain number of disconnected components and our task is to find a monotone path from $(0,0)$ to $(1,1)$ that traverses these components while using shortcuts between vertices of Y to "bridge" between points in different components or where there is no monotone path connecting them (see Example 5.2). The procedure is layed out in Algorithm 5.3.3 and described in detail below.

Detailed description of the decision procedure. The algorithm uses a directed graph G that has a node $v(i,j)$ for every free space cell $C_{i,j}$. For any path along the edges of the graph G from $v(1,1)$ to $v(i,j)$, there exists a monotone path that traverses the corresponding cells of $\mathcal{D}_{\leq\delta}(X,Y)$ while using zero or more affordable tunnels. A node $v(i,j)$ can have an incoming edge from another node $v(i',j')$, if $i' \leq i$ and $j' \leq j$ and either $v(i',j')$ is a neighboring node, or the two cells can be connected by an affordable tunnel which starts at the lower boundary of the cell corresponding to $v(i',j')$ and ends at the upper boundary of the cell corresponding to $v(i,j)$. Note that a node in this graph may have up to a quadratic number of incoming edges, one from

Algorithm 5.3.3 xDecider(X, Y, δ)

```
Input: polygonal curves $X$ and $Y$; distance $\delta \in \mathbb{R}$
 1: Assert that $\mathrm{dist}(0,0) = \|X(0) - Y(0)\| \leq \delta$ and $\mathrm{dist}(1,1) \leq \delta$
 2: Let $\mathcal{A}$ be an array with $\mathcal{A}[i] = \emptyset$ for $0 \leq i \leq n_1$
 3: Let $R = \{(0,0)\}$
 4: for $j = 1, \ldots, n_2$ do
 5:   for $i = 1, \ldots, n_1$ do
 6:     if $i = 1$ and $j = 1$ then
 7:       Let $R^h_{i,j} = I^h_{i,j}$ and $R^v_{i,j} = I^v_{i,j}$
 8:     else
 9:       Retrieve $R^v_{i-1,j}$ from $\mathcal{A}[i-1]$
10:       Retrieve $R^h_{i,j-1}$ from $\mathcal{A}[i]$
11:       Let $\mathsf{p}$ be the left gate of $I^h_{i,j}$
12:       $\mathsf{v}$ = xTunnel$(R, \mathsf{p}, \delta)$
13:       Compute $R^h_{i,j}$ and $R^v_{i,j}$ from $\mathsf{v}$, $R^v_{i-1,j}$, $R^h_{i,j-1}$, $I^v_{i,j}$ and $I^h_{i,j}$
14:     Store $R^h_{i,j}$ and $R^v_{i,j}$ in $\mathcal{A}[i]$
15:     if $R^h_{i,j} \neq \emptyset$ then
16:       Add gates of $R^h_{i,j}$ to $R$
17: if $(1,1) \in R$ then
18:   Return "$d_{\mathcal{S}}(X,Y) \leq \delta$"
19: else
20:   Return "$d_{\mathcal{S}}(X,Y) > \delta$"
```

each free space cell in its lower left quadrant, from which it could be reachable via a tunnel. The idea of the algorithm is to propagate reachability intervals $R^v_{i,j} \subseteq I^v_{i,j}$ and $R^h_{i,j} \subseteq I^h_{i,j}$ along the edges of the graph. To this end, the algorithm will implicitly traverse the entire graph.

The algorithm handles the nodes in the lexicographical order of the indices j and i of the nodes, thereby handling the corresponding cells of the free space diagram row by row from bottom to top and from left to right. During the traversal we store reachability intervals of the cells from the current and the previous row in an array $\mathcal{A}$ by the index i. When handling node $v(i,j)$, we can retrieve the reachability intervals of its direct neighbors $v(i, j-1)$ and $v(i-1, j)$ from $\mathcal{A}[i-1]$ and $\mathcal{A}[i]$. Furthermore we maintain the set of gates R, which are the endpoints of the horizontal reachability intervals computed so far. The cell $v(i,j)$ might also be reachable via a tunnel of price at most δ. Let p be the left endpoint of $I^h_{i,j}$. We invoke **xTunnel**(R, p, δ) to test if this is the case. If the call returns null, then there is no such affordable tunnel. Otherwise, if q is the returned point and r is the right endpoint of $I^h_{i,j}$, then we know that the line segment qr is reachable. Now we can compute the reachability intervals of the current cell as follows. We compute the set of points at the right side of $C_{i,j}$, which are reachable by a monotone path from $R^h_{i-1,j}$ and $R^v_{i,j-1}$. This is the interval $R^v_{i,j}$. We do the same for the top side of the cell and take the union of the obtained set of points with qr to obtain the interval $R^h_{i,j}$. Since the free space within a cell is convex and of constant complexity, computing the reachability intervals can be done in constant time. We store these intervals in $\mathcal{A}[i]$ and we add the endpoints of $R^h_{i,j}$ to R. After handling the last node, we can check if the top-right corner of the free space diagram is reachable by testing if $(1, 1)$ is contained in R.

5.3.2.3 The main algorithm

We are given two curves X and Y and we want to compute their shortcut Fréchet distance. We want to use the decision procedure described in Section 5.3.2.2 to perform a binary search for this distance. To this end, we need to compute a set of candidate values for the shortcut Fréchet distance. These are the values of δ, where structural changes happen in the free space diagram and a monotone path from $(0, 0)$ to $(1, 1)$ might become feasible. This could be a classical free space event, such as a vertex-vertex event, a vertex-edge event or a vertex-vertex-edge event (see Chapter 2). A second possibility is that a monotone path becomes feasible by using a tunnel. Thus, the shortcuts introduce a new type of event, which we need to analyze. This analysis can be found in the next section. The algorithm will simply compute all of these event values, perform a binary search over them using the decision algorithm and output the resulting value as the shortcut Fréchet distance between X and Y.

5.3.3 Tunnel events

The main algorithm uses the decision procedure to perform a binary search for the minimum δ for which the decision procedure returns "yes". In the problem at hand we are allowed to use tunnels to traverse the free space diagram, and it is possible that a path becomes feasible by introducing a tunnel. The algorithm has to consider this new type of critical events.

Consider the first time (i.e., the minimal value of δ) that a decision procedure would try to use a tunnel of a certain family.

Definition 5.3.4 Given a tunnel family $\mathcal{T}(\mathsf{e}_i, \mathsf{e}_j, u, v)$, we call the minimal value of δ such that $\mathcal{T}_{\leq\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ is non-empty the ***creation radius*** of the tunnel family and we denote it with $\mathrm{r}_{\mathrm{crt}}(\mathsf{e}_i, \mathsf{e}_j, u, v)$. (Note that the price of a tunnel might be considerably larger than its creation radius.)

Lemma 5.3.5 *The creation radius is equal to the minimum radius of a tunnel family; that is,* $\mathrm{r}_{\mathrm{crt}}(\mathsf{e}_i, \mathsf{e}_j, u, v) = \mathrm{r}_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v)$.

Proof: Recall that the creation radius of the tunnel family is the minimal value of δ such that any tunnel in this family is feasible. Let u' be the closest point of u on e_i and v' the closest point of v on e_j. If u' appears before v' on X, then the canonical tunnel is realized by $X(x_{\mathsf{q}}) = u'$ and $X(x_{\mathsf{q}}) = v'$ and the claim holds. In particular, this is the case if $i < j$.

Now, the only remaining possibility is that u' appears after v' on e. It must be that $i = j$, therefore let $\mathsf{e} = \mathsf{e}_i = \mathsf{e}_j$. Observe that in this case any tunnel in the family which is feasible for δ also has a price that is at most δ. Consider the point r realizing the quantity

$$\min_{\mathsf{r} \in \mathsf{e}} \max(\|\mathsf{r} - u\|, \|\mathsf{r} - v\|).$$

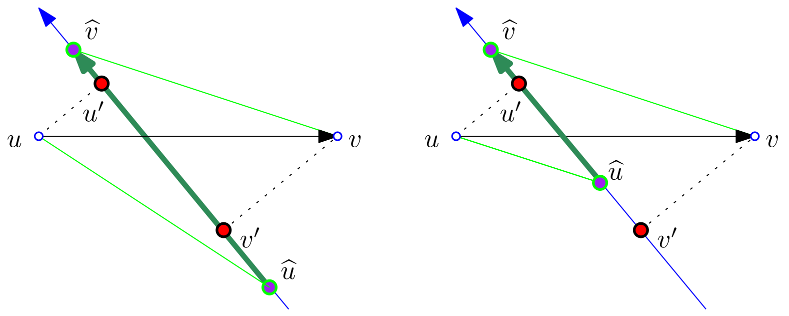


Figure 5.3: Two cases: v' appears either before or after $\widehat{u}$ along e, assuming that u' appears after v' on e.

Note that r is the subcurve of X corresponding to the (vertical) canonical tunnel in this case. We claim that for any subsegment $\widehat{u}\widehat{v} \subseteq \mathsf{e}$ (agreeing with the orientation of e) we have that $d_{\mathcal{F}}(\widehat{u}\widehat{v}, uv) \geq d_{\mathcal{F}}(\mathsf{r}, uv)$. If $\widehat{u} = \widehat{v}$ then the claim trivially holds.

Assume that v' appears after $\widehat{u}$ along e (the case depicted in Figure 5.3). Since u' appears after v' along e, we have that $\|v' - u\| \leq \|\widehat{u} - u\|$, as moving away from u' only increases the distance from u. Therefore,

$$d_{\mathcal{F}}(\mathsf{r}, uv) \leq d_{\mathcal{F}}(v', uv) = \max(\|v' - u\|, \|v' - v\|) \leq \max(\|\widehat{u} - u\|, \|\widehat{v} - v\|) = d_{\mathcal{F}}(\widehat{u}\widehat{v}, uv)\,.$$

Otherwise, if v' appears before $\widehat{u}$ along e, as depicted in Figure 5.3 on the right, then

$$d_{\mathcal{F}}(\mathsf{r}, uv) \leq d_{\mathcal{F}}(\widehat{u}, uv) = \max(\|\widehat{u} - u\|, \|\widehat{u} - v\|) \leq \max(\|\widehat{u} - u\|, \|\widehat{v} - v\|) = d_{\mathcal{F}}(\widehat{u}\widehat{v}, uv)\,,$$

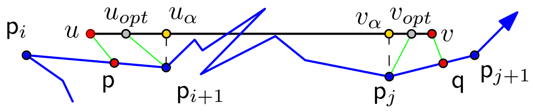
since moving away from v' only increases the distance from v.

This implies that the minimum δ for a tunnel in $\mathcal{T}(\mathsf{e}_i, \mathsf{e}_i, u, v)$ to be feasible is at least $d_{\mathcal{F}}(\mathsf{r}, uv) = \mathrm{r}_{\mathrm{crt}}(\mathsf{e}_i, \mathsf{e}_i, u, v)$. And r testifies that there is a tunnel in this family that is feasible for this value. □

The following lemma describes the behavior when δ rises above a tunnel price, such that the area in the free space that lies beyond this tunnel potentially becomes reachable by using this tunnel. More specifically, it implies that the first time (i.e., the minimal value of δ) that any tunnel of a family $\mathcal{T}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ is usable (i.e., its price is at most δ), any tunnel in the feasible set $\mathcal{T}_{\leq\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ associated with this family will be usable.

Lemma 5.3.6 *Given a value $\delta \geq 0$, we have for any tunnel $\tau(\mathsf{f}, \mathsf{g})$ in the feasible subset of a given tunnel family $\mathcal{T}_{\leq\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$, that*
(i) if $\delta \leq \mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v))$, then $\mathrm{prc}(\tau(\mathsf{f}, \mathsf{g})) = \mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v))$,
(ii) otherwise, $\mathrm{prc}(\tau(\mathsf{f}, \mathsf{g})) \leq \delta$.

Proof: We first handle the case that $i \neq j$. Let $\mathsf{e}_i = \mathsf{p}_i\mathsf{p}_{i+1}$ and $\mathsf{e}_j = \mathsf{p}_j\mathsf{p}_{j+1}$.



Let $\mathsf{p} \in \mathsf{e}_i$ and $\mathsf{q} \in \mathsf{e}_j$ be some points on these edges, that correspond to f and g, respectively. Observe that since this is a feasible tunnel in this family, we have that

$$\max(\|\mathsf{p} - u\|, \|\mathsf{q} - v\|) \leq \delta.$$

Consider the matching realizing the Fréchet distance of $X\langle \mathsf{p}, \mathsf{q}\rangle$ with uv, and let u_{opt} and v_{opt} be the points on uv that are matched to p_{i+1} and p_j by this matching. Let $\alpha = d_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_\alpha v_\alpha)$, where $u_\alpha v_\alpha$ is the subsegment of uv minimizing α.

We have, by Observation 2.3.1, that

$$\begin{aligned}
&d_{\mathcal{F}}(X\langle \mathsf{p}, \mathsf{q}\rangle, uv)\\
&= \max\Big(d_{\mathcal{F}}(\mathsf{p}\mathsf{p}_{i+1}, uu_{\mathrm{opt}}), d_{\mathcal{F}}\Big(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_{\mathrm{opt}}v_{\mathrm{opt}}\Big), d_{\mathcal{F}}(\mathsf{p}_j\mathsf{q}, v_{\mathrm{opt}}v)\Big)\\
&= \max\Big(\|\mathsf{p} - u\|, \|\mathsf{p}_{i+1} - u_{\mathrm{opt}}\|, d_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_{\mathrm{opt}}v_{\mathrm{opt}}), \|\mathsf{p}_j - v_{\mathrm{opt}}\|, \|\mathsf{q} - v\|\Big)\\
&= \max\Big(\|\mathsf{p} - u\|, d_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_{\mathrm{opt}}v_{\mathrm{opt}}), \|\mathsf{q} - v\|\Big)\\
&\geq \max(\|\mathsf{p} - u\|, d_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_\alpha v_\alpha), \|\mathsf{q} - v\|)\\
&= \max\Big(d_{\mathcal{F}}(\mathsf{p}\mathsf{p}_{i+1}, uu_\alpha), d_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_\alpha v_\alpha), d_{\mathcal{F}}(\mathsf{p}_j\mathsf{q}, v_1 v)\Big)\\
&\geq d_{\mathcal{F}}(X\langle \mathsf{p}, \mathsf{q}\rangle, uv).
\end{aligned}$$

For $\alpha = d_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_\alpha v_\alpha)$, this implies

$$d_{\mathcal{F}}(X\langle \mathsf{p}, \mathsf{q}\rangle, uv) = \max(\|\mathsf{p} - u\|, \alpha, \|\mathsf{q} - v\|),$$

where $\alpha \leq \max(\alpha, \delta)$ is equal for all tunnels in the family.

Now, if $\delta \leq \mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v))$ then we have $\mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v)) = \alpha \geq \delta$ and

$$\mathrm{prc}(\tau(\mathsf{f}, \mathsf{g})) = d_{\mathcal{F}}(X\langle \mathsf{p}, \mathsf{q}\rangle, uv) = \max(\|\mathsf{p} - u\|, \alpha, \|\mathsf{q} - v\|) \leq \max(\alpha, \delta) = \alpha.$$

This proves (i). Otherwise, we have $\mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v)) < \delta$. Which implies that $\alpha < \delta$, but then $\mathrm{prc}(\tau(\mathsf{f}, \mathsf{g})) \leq \delta$, implying (ii).

If $i = j$ then the Fréchet distance is between the shortcut segment and a subsegment of e_i. But this distance is the maximum distance between the corresponding endpoints, by Observation 2.3.1. As the distance between endpoints of shortcuts and subcurves corresponding to tunnels of $\mathcal{T}_{\leq\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ is at most δ, and by Lemma 5.3.5 the claim follows. □

5.3.4 Analysis of the exact algorithm

Lemma 5.3.7 *Given the left gate* p *of a free space interval* $I^h_{i,j}$ *and a set of gates* R*, and a parameter* $\delta > 0$*, the algorithm* **xTunnel** *(Algorithm 5.3.2) outputs the leftmost point* $\mathsf{v} \in I^h_{i,j}$ *that is the endpoint of a tunnel* $\tau(\mathsf{q}, \mathsf{v})$ *of price* $\mathrm{prc}(\tau(\mathsf{q}, \mathsf{v})) \leq \delta$ *from a gate* $\mathsf{q} \in R$*. If no such point exists, then the algorithm returns* null.

Proof: The algorithm returns a point v either in line 5 or in line 11 or it returns null. In the first case, it follows from the correctness of the algorithm by Alt and Godau that this is the endpoint of a tunnel of price at most δ. In the second case, this follows from the fact that vertical tunnels are always affordable if they are feasible. There can be no tunnel $\tau(\mathsf{q}, \mathsf{r})$ from a point $\mathsf{q} \in R$ which ends to the left of v and which has price at most δ. Indeed, either $\mathsf{v} = \mathsf{p}$, or q has to be considered in the main for-loop of the algorithm. If q lies in the lower left quadrant of p, then the tunnel would be tested in line 4 and returned in line 5. Otherwise it is a vertical tunnel and its feasibility is tested in line 8. Since the algorithm maintains the leftmost feasible vertical tunnel seen so far, v cannot lie to the left of the returned point. □

Lemma 5.3.8 *Given two curves X and Y and a parameter $\delta > 0$. The algorithm* **xDecider** *(Algorithm 5.3.3) outputs if $d_{\mathcal{S}}(X, Y) \leq \delta$.*

Proof: We claim that, after handling the cell $C_{i,j}$, it holds that
(i) the reachability intervals $R^v_{i,j} \subseteq I^v_{i,j}$ and $R^h_{i,j} \subseteq I^h_{i,j}$ are computed correctly, and
(ii) R contains all endpoints of $R^h_{i',j'}$ of all previously handled cells $C_{i',j'}$ and $\{(0,0)\}$.
We prove this by induction on the cells in the order in which they are handled by the algorithm. For $i = 1$ and $j = 1$, the claim trivially follows from the convexity of the free space inside the cell $C_{1,1}$ and by the assumption that $(0,0) \in \mathcal{D}_{\leq\delta}(X, Y)$. Now consider part (i) of the induction hypothesis for general i and j. Let v be the output of **xTunnel** and let r be the right endpoint of the corresponding free space interval. We first prove that the line segment vr is exactly the set of endpoints of monotone paths starting from $(0,0)$ and ending with a tunnel in $I^h_{i,j}$. Let π be such a monotone path. Let

$$\mathcal{U}_{i,j} = \bigcup_{\substack{1 \leq i' \leq i \\ 1 \leq j' \leq j}} C_{i',j'} \setminus C_{i,j}.$$

The set $\mathcal{U}_{i,j}$ contains all cells that are traversable by π before reaching $C_{i,j}$. Thus, any possible starting point of the tunnel of π that ends in $I^h_{i,j}$, has to be contained in a reachability interval of a cell of $\mathcal{U}_{i,j}$. Since we are only concerned with the directed, vertex-restricted shortcut Fréchet distance, tunnels are only possible between the horizontal free space intervals. Note that the algorithm has handled all cells in $\mathcal{U}_{i,j}$ before handling $C_{i,j}$ and by the induction hypothesis, the set R contains all endpoints of the horizontal reachability intervals of $\mathcal{U}_{i,j}$. Lemma 5.3.6 implies that, in order to determine if the tunnels of the feasible subset of a tunnel family are affordable (i.e., have price at most δ), it is sufficient to test any tunnel in this feasible subset.[2] The fact that π has its endpoint on vr now follows from Lemma 5.3.7. The second possibility for points in $I^h_{i,j}$ and $I^v_{i,j}$ to be reachable is via a monotone path which enters the cell $C_{i,j}$ through a reachability interval of one of its direct neighbors. By induction the two neighboring cells are also contained in the set $\mathcal{U}_{i,j}$ and thus their reachability intervals are correctly considered by the algorithm. This proves part (i) of the induction hypothesis. Now part (ii) follows from line 16 of the algorithm. Since the induction hypothesis also holds for the upper right cell of the free space diagram, the lemma follows. □

[2]Interestingly, the canonical price of the tunnel family which is independent of δ would suffice to perform this test. However, computing the canonical tunnel prices naively one by one takes $O(n^5 \log n)$ time and storing them explicitly takes $O(n^4)$ space.

Lemma 5.3.9 *Given two curves X and Y of total complexity $n = n_1 + n_2$ and a parameter $\delta > 0$. The algorithm* **xDecider** *(Algorithm 5.3.3) takes $O\big(n^5\big)$ time and $O(n)$ space.*

Proof: The algorithm uses the **xTunnel** procedure (Algorithm 5.3.2). This procedure iterates over the given set R, which has size at most $O(n^2)$, and invokes the decision algorithm of Alt and Godau once in every step of the iteration on a polygonal curve of complexity $O(n)$ and a line segment to test the price of the corresponding tunnel. This test takes $O(n)$ time, see Section 2.4. Thus, the running time for one call to **xTunnel** can be bounded by $O(n^3)$.

The algorithm invokes the **xTunnel** procedure $O(n^2)$ times leading to an overall running time in $O(n^5)$. The algorithm uses an array of length $O(n)$ and the algorithm of Alt and Godau uses space in $O(n)$. Thus the overall space requirement is in $O(n)$. □

Theorem 5.3.10 *Given two curves X and Y in $\mathbb{R}^d$ of total complexity n, one can compute the shortcut Fréchet distance $d_{\mathcal{S}}(X,Y)$ in $O\big(n^5 \log n\big)$ time and $O(n^4)$ space.*

Proof: The algorithm is described in Section 5.3.2.3. The correctness follows from Lemma 5.3.8 and from the analysis of the tunnel events in Section 5.3.3. Recall that there are two types of such tunnel events: (i) the event that the endpoints of a tunnel between two cells becomes feasible (i.e., a creation radius, Definition 5.3.4) and (ii) the event that the price of a tunnel on a monotone path becomes affordable (i.e., the price of a canonical tunnel, see Section 5.3.1). A creation radius is either a vertex-edge distance or a vertex-vertex-edge distance, by Observation 5.3.1. Computing the canonical price of a tunnel family (i.e., event (ii)), can be done in $O(n \log n)$ time by computing the price of the canonical tunnel using the algorithm by Alt and Godau, see Section 2.4. There are $O(n^2)$ vertex-vertex and vertex-edges events, there are $O(n^3)$ vertex-vertex-edge events and there are $O(n^4)$ tunnel families. Thus, the running time for computing the critical values is dominated by the computation of the canonical tunnel prices which takes $O(n^5 \log n)$ time. The algorithm performs a binary search on these critical values, thus it requires $O(n^4)$ space. The binary search uses the decision algorithm described in Section 5.3.2.2. By Lemma 5.3.8, this takes $O(n^5)$ time and $O(n)$ space per call. Thus, we have an overall running time of $O(n^5 \log n)$ and the theorem follows. □

5.3.5 A near-linear time approximation algorithm

Here, we describe the approximation algorithm for the case that the number of shortcuts used is unbounded. The algorithm runs in near-linear time if the input curves are c-packed. We use the following two non-trivial data structures.

Data Structure 5.3.11 *Given a polygonal curve Z with n vertices in $\mathbb{R}^d$, one can build a data structure, in $O\big(\varepsilon^{-2d} n \log^2(1/\varepsilon) \log^2 n\big)$ time, using $O\big(\varepsilon^{-2d} n \log^2(1/\varepsilon)\big)$ space, that supports a procedure* $\textbf{price}(\mathsf{p}, \mathsf{q}, \varepsilon)$ *which receives two points p and q in the parametric space of X and Y and returns in $O\big(\varepsilon^{-3} \log n \log\log n\big)$ time a value ϕ, such that $\phi \leq \operatorname{prc}(\tau(\mathsf{p},\mathsf{q})) \leq (1+\varepsilon)\phi$. See Section 4 and Theorem 4.2.5.*

Algorithm 5.3.13 tunnel$(R, \mathsf{p}, \varepsilon, \delta)$

```
Input: set of gates $R$; gate $\mathsf{p} \in [0,1]^2$; error $\varepsilon \in (0,1]$; distance $\delta \in \mathbb{R}$
 1: Let $\mathsf{q} = (x_\mathsf{q}, y_\mathsf{q})$ be a point in $R$ with max value of $x_\mathsf{q}$,
        such that $x_\mathsf{q} \le x_\mathsf{p}$ and $y_\mathsf{q} < y_\mathsf{p}$, where $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p})$.
 2: $\phi$ = price$(\mathsf{q}, \mathsf{p}, \varepsilon)$, see Data Structure 5.3.11.
 3: if $\phi \le 3\delta$ then
 4:     Return $\mathsf{p}$     // tunnel $\tau(\mathsf{q}, \mathsf{p})$
 5: Compute $j$ such that $x_\mathsf{p} \in \mathcal{I}_{\mathrm{edge}}(X, j) = [x_j, x_{j+1}]$
 6: Let $\mathsf{q} = (x_\mathsf{q}, y_\mathsf{q})$ be a point in $R$ with min value of $x_\mathsf{q}$,
        such that $x_\mathsf{q} \in \mathcal{I}_{\mathrm{edge}}(X, j)$, $x_\mathsf{q} \ge x_\mathsf{p}$, and $y_\mathsf{q} < y_\mathsf{p}$
 7: if $\mathsf{q}$ does not exist then
 8:     Return null.
 9: $\mathsf{v} = (x_\mathsf{q}, y_\mathsf{p})$
10: if dist$(\mathsf{v}) \le \delta$ then
11:     Return $\mathsf{v}$     // vertical tunnel $\tau(\mathsf{q}, \mathsf{v})$
12: else
13:     Return null.
```

Data Structure 5.3.12 *For given parameters ε and δ, and two c-packed curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^d$, let $X = \mathbf{simpl}(\boldsymbol{X}, \mu)$ and $X = \mathbf{simpl}(\boldsymbol{Y}, \mu)$, where $\mu = \varepsilon\delta$. One can compute all the vertex-edge pairs of the two simplified curves X and Y in distance at most δ from each other, in time $O(n \log n + c^2 n/\varepsilon)$. See below for details.*

We describe how to realize Data Structure 5.3.12. Observe that X and Y have density $\phi = O(c)$, see Definition 3.4.1. Now, we build the data structure of de Berg and Streppel [57] for the segments of Y (with $\varepsilon = 1/2$). For each vertex of X we compute all the segments of Y that are in distance at most δ from it, using the data structure [57]. Each query takes $O(\log n + k\phi)$ time, where k is the number of edges reported. Lemma 3.1.6 implies that the total sum of the k's is $O(cn/\varepsilon)$. We now repeat this for the other direction. This way, one can realize Data Structure 5.3.12.

5.3.5.1 The tunnel procedure

Like the exact **xTunnel** procedure described in Section 5.3.2.1, the **tunnel** procedure (see Algorithm 5.3.13) receives a set of gates R and a gate p as input and returns the endpoint of an affordable tunnel that starts at a gate of R and ends either at p or the closest point to the right of p in the same free space interval. However, we are allowed to answer this query approximately. More precisely, if a tunnel between a gate in R and the free space interval of p exists, which has price at most δ, then the algorithm will return the endpoint of a tunnel of price at most $(1+\varepsilon)3\delta$. If the algorithm returns null, then we know that no such tunnel of price at most δ exists.

The main idea of the **tunnel** procedure is the following. For a given tunnel, we can $(1+\varepsilon)$-approximate its price, using a data structure which answers these queries in polylogarithmic time, see Data Structure 5.3.11. The desired tunnel could be either a tunnel between a gate of R and p, or it could be a vertical tunnel from a gate of R which lies to the right of p. Naively, one could test all tunnels that start from

Algorithm 5.3.14 decider$(X, Y, \varepsilon, \delta)$

```
Input: polygonal curves $X$ and $Y$; error $\varepsilon \in (0,1]$; distance $\delta \in \mathbb{R}$
 1: Assert that $\mathrm{dist}(0,0) = \|X(0) - Y(0)\| \leq \delta$ and $\mathrm{dist}(1,1) \leq \delta$
 2: Let $Q$ be a min-priority queue for nodes $v(i,j)$ with keys $(jn+i)$
 3: Compute and enqueue the cells $C_{i,j}$ that have non-empty $I^h_{i,j}$ or $I^v_{i,j}$
 4: Let $R = \{(0,0)\}$ // set of gates
 5: Let $\overline{R}$ be an empty set and let $\overline{j} = 1$ // current row
 6: while $Q \neq \emptyset$ do
 7:     Dequeue node $v(i,j)$ and its copies from $Q$
 8:     if $\overline{j} < j$ then
 9:         Add all elements of $\overline{R}$ to $R$ and let $\overline{j} = j$
10:     Let $\mathsf{p}$ be the left gate of $I^h_{i,j}$
11:     $\mathsf{v} = \mathbf{tunnel}(R, \mathsf{p}, \varepsilon, \delta)$
12:     Compute $R^h_{i,j}$ and $R^v_{i,j}$ from $\mathsf{v}$, $R^v_{i-1,j}$, $R^h_{i,j-1}$, $I^v_{i,j}$ and $I^h_{i,j}$
13:     if $R^v_{i,j} \neq \emptyset$ then
14:         Enqueue $v(i+1,j)$ and insert edge between $v(i,j)$ and $v(i+1,j)$
15:     if $R^h_{i,j} \neq \emptyset$ then
16:         Enqueue $v(i,j+1)$ and insert edge between $v(i,j)$ and $v(i,j+1)$
17:         Add gates of $R^h_{i,j}$ to $\overline{R}$
18: if $(1,1) \in R$ then
19:     Return "$d_{\mathcal{S}}(X,Y) \leq (1+\varepsilon)3\delta$"
20: else
21:     Return "$d_{\mathcal{S}}(X,Y) > \delta$"
```

a gate in R and end in p, however, this takes time at least linear in the size of R. Since we are only interested in a constant factor approximation, it is sufficient, by Lemma 5.2.1, to test only the tunnel which corresponds to the shortest subcurve of X. The corresponding gates can be found in logarithmic time using a data structure, which is built on the set R and we assume is available to us. We can maintain this data structure during the decision procedure layed out in Algorithm 5.3.14. The technical details are described in the proof of Lemma 5.3.17.

5.3.5.2 The decision algorithm

Given two curves X and Y, and a parameter δ, we want to know if the shortcut Fréchet distance is at most δ. Like in the exact decision algorithm described in Section 5.3.2.2, we traverse the free space diagram in search for a monotone path. However, we will compute the reachable space only approximately and in turn achieve a better running time. The decision algorithm exploits the monotonicity of the tunnel prices shown in Lemma 5.2.1 and is based on a breadth first search in the free space diagram (a similar idea was used in Chapter 3, but here the details are more involved).

The decision algorithm answers the initial question with either

(i) "$d_{\mathcal{S}}(X,Y) \leq (3+\varepsilon)\delta$" (an equivalent to "yes") if a shortcut curve Y' of Y, such that $d_{\mathcal{F}}(X,Y') \leq (3+\varepsilon)\delta$ is found, or it answers with

(ii) $d_{\mathcal{S}}(X,Y) > \delta$ (an equivalent "no") if we conclude that no shortcut curve with $d_{\mathcal{F}}(X,Y') \leq \delta$ exists.

Algorithm 5.3.15 Decider($\boldsymbol{X}, \boldsymbol{Y}, \varepsilon, \delta$)

```
Input: polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$; error $\varepsilon \in (0,1]$; distance $\delta \in \mathbb{R}$
1: Let $\varepsilon' = \varepsilon/10$
2: Compute $X = \mathbf{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathbf{simpl}(\boldsymbol{Y}, \mu)$ with $\mu = \varepsilon'\delta$
3: Call $\mathbf{decider}(X, Y, \varepsilon', \delta')$ with $\delta' = (1+2\varepsilon')\delta$
4: Return either "$d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y}) \leq (1+\varepsilon)3\delta$" or "$d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y}) > \delta$"
```

Detailed description of the decision procedure. The decision algorithm is layed out in Algorithm 5.3.15 (and Algorithm 5.3.14). Like in the exact decision algorithm described in Section 5.3.2.2, we use a directed graph G, which stores the reachability within the free space diagram. The graph has a node $v(i,j)$ for every free space cell $C_{i,j}$ whose boundary has a non-empty intersection with the free space $\mathcal{D}_{\leq\delta}(X,Y)$. These intersections are defined as the free space intervals $I^h_{i,j}$, $I^v_{i,j}$, $I^h_{i-1,j}$ and $I^v_{i,j-1}$, see Chapter 2. The idea of the algorithm is to propagate reachability intervals $R^v_{i,j} \subseteq I^v_{i,j}$ and $R^h_{i,j} \subseteq I^h_{i,j}$ while traversing a sufficiently large subgraph starting from $v(1,1)$, and computing the necessary parts of this subgraph on the fly. We store these intervals with the cell $v(i,j)$ that has them on the top (resp., right) boundary. The reachability intervals $R^v_{i,j}$ being computed satisfy

$$\mathcal{R}^{\infty}_{\leq\delta}(X,Y) \cap I^v_{i,j} \subseteq R^v_{i,j} \subseteq \mathcal{R}^{\infty}_{\leq(1+\varepsilon)3\delta}(X,Y) \cap I^v_{i,j}, \tag{5.3}$$

and an analogous statement applies to $R^h_{i,j}$. The aim is to determine if either $(1,1) \in \mathcal{R}^{\infty}_{\leq(1+\varepsilon)3\delta}(X,Y)$ or $(1,1) \notin \mathcal{R}^{\infty}_{\leq\delta}(X,Y)$. Throughout the whole algorithm we also maintain a set of gates R, which represents the endpoints of the horizontal reachability intervals computed so far. (Technically, the gates which are computed in the current row are maintained in a separate set $\overline{R}$ and are only added to the set R when moving to the next row. This simplifies the range queries on R in the **tunnel** procedure.)

We will traverse the graph by handling the nodes in a row-by-row order, thereby handling any node $v(i,j)$ only after we handled the nodes $v(i',j')$, where $j' \leq j$, $i' \leq i$ and $(i'+j') < (i+j)$. To this end we keep the nodes in a min-priority queue where the node $v(i,j)$ has the key $(jn+i)$. The correctness of the computed reachability intervals will follow by induction on the order of these keys. Furthermore, it will ensure that we handle each node at most once and that we traverse at most three of the incoming edges to each node of the graph.

The queue is initialized with the entire node set at once. To compute this initial node set and the corresponding free space intervals we use Data Structure 5.3.12. The algorithm then proceeds by handling nodes in the order of extraction from this queue. When dequeuing nodes from the queue, the same node might appear three times (consecutively) in this queue. Once from each of its direct neighbors in the grid and once from the initial enqueuing.

In every iteration, the algorithm dequeues the one or more copies of the same node $v(i,j)$ and merges them into one node if necessary. Assume that $v(i,j)$ has an incoming edge that corresponds to an affordable tunnel. Let p be the left gate of $I^h_{i,j}$. We invoke $\mathbf{tunnel}(R, \mathsf{p}, \varepsilon, \delta)$ to test if this is the case. If the call returns null, then there is no such affordable tunnel. Otherwise, we know that the returned point v is contained in $R^h_{i,j}$. If there were more than one copies of this node in the queue, we also access the reachability intervals of the one or two neighboring vertices (i.e., $R^v_{i-1,j}$ and $R^h_{i,j-1}$). Using the reachability information from the at most three incoming edges obtained this way, we can determine if the cells $C_{i,j+1}$ and $C_{i+1,j}$ are reachable, by computing the resulting reachability intervals $R^h_{i,j}$ at the top side and $R^v_{i,j}$ the right side of the cell $C_{i,j}$. Since the free space within a cell is convex and of constant complexity, this can be done in constant time.

Now, if $R^h_{i,j} \neq \emptyset$ we create a node $v(i, j+1)$, connect it to $v(i, j)$ by an edge, we enqueue it, and add the gates of $R^h_{i,j}$ to R. If $R^v_{i,j} \neq \emptyset$ we create a node $v(i+1, j)$, connect it to $v(i, j)$ by an edge, and we enqueue it. If we discover that the top-right corner of the free space diagram is reachable this way, we output the equivalent to "yes" and the algorithm terminates. In this case we must have added $(1, 1)$ as a gate to R. The algorithm may also terminate before this happens if there are no more nodes in the queue, in this case we output that no suitable shortcut curve exists.

5.3.5.3 The main algorithm

The given input is two curves X and Y. We want to use the approximate decision procedure **Decider**, described above, in a binary search like fashion to compute the shortcut Fréchet distance. Conceptually, one can think of the decider as being exact. In particular, the algorithm would, for a given value of δ, call the decision procedure twice with parameters δ and $\delta' = \delta/4$ (using $\varepsilon = 1/3$). If the two calls agree, then we can make an exact decision, if the two calls disagree, then we can output a $O(1)$-approximation of the shortcut Fréchet distance.

The challenge is how to choose the right subset of candidate values to guide this binary search. Some of the techniques used for this search have been introduced in previous papers. In particular, this holds for the search over vertex-vertex, vertex-edge and vertex-vertex-edge distances which we describe as preliminary computations in Section 5.3.5.4. This first stage of the algorithm eliminates the candidate values that also need to be considered for the approximation of the standard Fréchet distance and it is almost identical to the algorithm presented in Chapter 3.

As mentioned before, a monotone path could also become usable by taking a tunnel. There are two types of events associated with a tunnel family: The first time such that any tunnel in this family is feasible, which is the ***creation radius***. Fortunately, the creation radii of all tunnels are approximated by the set of vertex-vertex and vertex-edge event radii, and our first stage search (see Section 5.3.5.4) would thus take care of such events.

Another event we need to consider is the first time that the feasible subset of a tunnel family becomes usable (i.e., the price of some tunnel in this family is below the distance threshold δ). Luckily, it turns out that it is sufficient to search over the price of the canonical tunnel associated with such a family. The price of a specific tunnel can be approximated quickly using Data Structure 5.3.11. However, there are $\Theta(n^4)$ tunnel families, and potentially the algorithm has to consider all of them. Fortunately, because of c-packedness, only $O(n^2)$ of these events are relevant. A further reduction in running time is achieved by using the monotonicity property of the prices of these tunnels (see Section 5.2.2.1) and our ability to represent them implicitly to search over them efficiently.

5.3.5.4 The algorithm – First stage

We are given two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ with total complexity n. We repeatedly compute sets of event values and perform binary searches on these values as follows.

We compute the set of vertices V of the two curves, and using well-separated pairs decomposition (Lemma 3.2.7), we compute, in $O(n \log n)$ time, a set U of $O(n)$ distances that, up to a factor of two, represents any distance between any two vertices of V.

Next, we use **Decider** (with fixed $\varepsilon = 1/3$) to perform a binary search for the atomic interval in U that contains the desired distance. Let $[\alpha, \beta]$ denote this interval. If $10\alpha \geq \beta/10$ then we are done, since we found a constant size interval that contains the Fréchet distance. Otherwise, we use the decision procedure to verify that the desired radius is not in the range $[\alpha, 10\alpha]$ and $[\beta/10, \beta]$. For $\alpha' = 3\alpha$, $\beta' = \beta/3$, let $\mathcal{I}' = [\alpha', \beta']$ denote the obtained interval.

We now continue the search using only **decider** and the simplified curves $X = \mathbf{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathbf{simpl}(\boldsymbol{Y}, \mu)$, where $\mu = \alpha'$. We extract the vertex-edge events of X and Y that are smaller than β', see Chapter 2. To this end, we compute all edges of X that are in distance at most β' of any vertex of Y and vice versa using Data Structure 5.3.12. Let U' be the set of resulting distances. We perform a binary search, using **decider** to find the atomic interval $\mathcal{I}'' = [\alpha'', \beta'']$ of $U' \cap \mathcal{I}'$ that contains the shortcut Fréchet distance between X and Y.

Finally, we again search the margins of this interval, so that either we found the desired approximation, or alternatively we output the interval $[10\alpha'', \beta''/10]$,

5.3.5.5 Second stage – Searching over tunnel prices

It remains to search over the canonical prices of tunnel families $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$, where $\mathsf{e} \neq \mathsf{e}'$ (since for the case where $\mathsf{e} = \mathsf{e}'$ the canonical price coincides with the creation event value). After the first stage, we have an interval $[\alpha, \beta] = [10\alpha'', \beta''/10]$, and simplified curves X and Y of which the shortcut Fréchet distance is contained in $[\alpha, \beta]$ and approximates $d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y})$. By Lemma 3.1.6, the number of vertex-edge pairs in distance β is bounded by $O(cn/\varepsilon)$. The corresponding horizontal grid edges in the parametric space contain the canonical gates which are feasible for any value in $[\alpha, \beta]$. Let P denote the $m = O(cn/\varepsilon)$ points in the parametric space that correspond to the canonical gates of these vertex-edge pairs; that is, for every feasible pair p (a vertex of Y) and e (an edge of X), we compute the closest point q on e to p, and place the point corresponding to (q, p) in the free space into P.

It is sufficient to consider the tunnel families between these vertex-edge pairs, since all other families are not feasible in the remaining search interval. Thus, if we did not care about the running time, we could compute and search over the prices of the tunnels $\mathsf{P} \times \mathsf{P}$, using Data Structure 5.3.11. Naively, this would take roughly quadratic time. Instead, we use a more involved implicit representation of these tunnels to carry out this task.

Implicit search over tunnel prices. Consider the implicit matrix of tunnel prices $M = \mathsf{P} \times \mathsf{P}$ where the entry $M(i, j)$ is a $(1+\varepsilon)$-approximation to the price of the canonical tunnel $\tau(\mathsf{p}_i, \mathsf{p}_j)$. By Lemma 5.2.2, the first j values of the jth row of this matrix are monotonically decreasing up to a constant factor, since they correspond to tunnels that share the same endpoint p_j and are ordered by their starting points p_i (we ignore the values in this matrix above the diagonal). Using Data Structure 5.3.11 we can $(1+\varepsilon)$-approximate a value in the matrix in polylogarithmic time per entry. Similarly, the lower triangle of this matrix is sorted in increasing order in each column. As such, this matrix is sorted in both rows and columns and one can apply the algorithm of Frederickson and Johnson [78] to find the desired value. This requires $O(\log m)$ calls to **Decider**, the evaluation of $O(m)$ entries in the matrix, and takes $O(m)$ time otherwise. Here, we are using **Decider** as an exact decision procedure. The algorithm will terminate this search with the desired constant factor approximation to the shortcut Fréchet distance.

5.3.6 Analysis of the approximation algorithm

5.3.6.1 Analysis of the tunnel procedure

Lemma 5.3.16 *Given the left gate* p *of a free space interval* $I^h_{i,j}$ *and a set of gates* R*, and parameters* $0 < \varepsilon \leq 1$ *and* $\delta > 0$*, the algorithm* **tunnel** *(Algorithm 5.3.13) outputs one of the following:*

(i) A point $\mathsf{v} \in I^h_{i,j}$*, such that there exists a tunnel* $\tau(\mathsf{q},\mathsf{v})$ *of price* $\mathrm{prc}(\tau(\mathsf{q},\mathsf{v})) \leq (1+\varepsilon)3\delta$ *from a gate* $\mathsf{q} \in R$*, or*

(ii) null*, in this case, there exists no tunnel of price at most* δ *between a gate of* R *and a point in* $I^h_{i,j}$*.*

Furthermore, in case (i), there exists no other point $\mathsf{r} \in [\mathsf{p},\mathsf{v}]$ *that is the endpoint of a tunnel from* R *with price at most* δ*.*

Proof: The correctness of this procedure follows from the monotonicity of the tunnel prices, which is testified by Lemma 5.2.1. Let ϕ be the $(1+\varepsilon)$-approximation to the price of the tunnel, that we compute in line 3. This tunnel starts at a point in R and ends in p and it corresponds to the shortest subcurve $\widehat{X}$ of X over any such tunnel. Lemma 5.2.1 implies that if $\phi > 3\delta$ then there can be no other tunnel of price at most δ that corresponds to a subcurve of X that contains $\widehat{X}$. Therefore, the price of any tunnel from a point $\mathsf{q} \in R$ that lies in the lower left quadrant of p, to a point that lies in the upper right quadrant of p has a price larger than δ. In particular, this holds for those tunnels that end to the right of p in the same free space interval. The only other possibility for a tunnel from R to $I^h_{i,j}$ is a vertical tunnel that lies to the right of p. Observe that a vertical tunnel that is feasible for δ always has price at most δ, since it corresponds to a subcurve of X that is equal to a point that is in distance δ to the shortcut edge. In line 5 and line 6 we compute the leftmost gate of R in the lower right quadrant of p that lies in the same column as p. If there exists such a point with a vertical tunnel that ends in the free space interval $I^h_{i,j}$, then we return the endpoint of this tunnel. Otherwise we can correctly output the equivalent to the answer that there exists no tunnel of price at most δ. $\square$

5.3.6.2 Analysis of the decision algorithm

Clearly, the priority queue operations take time in $O(N \log N)$ and space in $O(N)$, where $N = \mathcal{N}_{\leq\delta}(X,Y)$ is the size of the node set, which corresponds to the complexity of the free space diagram. We invoke the **tunnel** procedure once for each node. Since we add at most a constant number of gates for every cell to R, the size of this set is also bounded by $O(N)$. Therefore, after the initialization, the algorithm takes time near linear in the complexity of the free space diagram. We can reduce this complexity by first simplifying the input curves with $\mu = \Theta(\varepsilon\delta)$ before invoking the **decider** procedure, thereby paying another approximation factor. We denote the resulting wrapper algorithm with **Decider**, it is layed out in Algorithm 5.3.15. Now, the initial computation of the nodes takes near-linear time by Data Structure 5.3.12 and therefore the overall running time is near-linear. A more detailed analysis of the running time can be found in the following lemma.

Lemma 5.3.17 *Given parameters $\delta > 0$ and $0 < \varepsilon \leq 1$ and two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^d$ of total complexity n. The algorithm* **Decider** *(Algorithm 5.3.15) outputs one of the following:*

(i) "$d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y}) \leq (1+\varepsilon)3\delta$", or

(ii) "$d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y}) > \delta$".

In any case, the output returned is correct.

The running time is $O\big(Cn \log^2 n\big)$, where $C = c^2\varepsilon^{-2d}\log(1/\varepsilon)$.

Proof: The algorithm **Decider** computes the simplified curves $X = \mathbf{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathbf{simpl}(\boldsymbol{Y}, \mu)$ with $\mu = \Theta(\varepsilon\delta)$, before invoking the algorithm **decider** (Algorithm 5.3.14) on these curves. By the correctness of the **tunnel** procedure (i.e., Lemma 5.3.16), one can argue by induction that the subsets of points of $\mathcal{R}^{\infty}_{\leq\delta}(X, Y)$ intersecting a grid edge are sufficiently approximated by the reachable intervals computed by **decider** (see Eq. (5.3)). By Lemma 5.2.3, this approximates the decision with respect to the original curves sufficiently.

It remains to analyze the running time. By Lemma 3.1.6, the size of the node set of the traversed subgraph of G is bounded by $N = O(cn/\varepsilon)$. This also bounds the size of the point set R and the number of calls to the **tunnel** procedure, since there are at most a constant number of those per node. During the **tunnel** procedure (Algorithm 5.3.13) we

(A) approximate the price of one tunnel in line 3, and

(B) invoke two orthogonal range queries on the set R in line 1 and line 6.

As for (A), building the data structure that supports this kind of queries takes $T_1 = O\big(n\varepsilon^{-2d}\log^2(1/\varepsilon)\log^2 n\big)$ time by Data Structure 5.3.11. Since we perform $O(N)$ such queries, this takes $T_2 = O\big(N\varepsilon^{-3}\log n \log\log n\big) = O\big(cn\varepsilon^{-4}\log n \log\log n\big)$ time overall. As for (B) since we traverse the free space diagram in a row-by-row order from bottom to top, the entire set of points stored in R lies inside the vertical range which is queried. Therefore, for answering the queries for the leftmost and rightmost points within a quadrant, it is sufficient to store the points in a balanced binary search tree ordered by their x-coordinate. Overall, maintaining this data structure and answering the orthogonal range queries takes $T_3 = O(N \log N)$ time. During the algorithm, we maintain a priority queue, where each node is added and extracted at most three times. As such, the priority queue operations take time in $O(N \log N)$. The initial computation of the node set takes $T_4 = O(n \log n + c^2 n/\varepsilon)$ by Data Structure 5.3.12.

Therefore, the overall running time is $T_1 + T_2 + T_3 + T_4$, which is

$$O\bigl(n\varepsilon^{-2d}\log^2(1/\varepsilon)\log^2 n + cn\varepsilon^{-4}\log n\log\log n + cn\log n + n\log n + c^2 n/\varepsilon\bigr)$$
$$= O\bigl(Cn\log^2 n\bigr)\,,$$

where $C = c^2\varepsilon^{-2d}\log(1/\varepsilon)$. □

Observation 5.3.18 It is easy to modify the **decider** algorithm such that it also outputs the respective shortcut curve and matching which realizes the Fréchet distance. We would modify the tunnel procedure such that it returns not only the endpoint, but also the starting point of the computed tunnel. During the algorithm, we then insert an edge for each computed tunnel, thereby creating at most three incoming edges to each node. After the algorithm terminates, we can trace any path backwards from $(1,1)$ to $(0,0)$ in the subgraph computed this way. This path encodes the shortcut curves as well as the matchings.

5.3.6.3 Analysis of the main algorithm

The following lemma can be obtained using similar arguments as in the analysis of the main algorithm in Chapter 3. We provide a simplified proof for the case here, where we are only interested in a constant factor approximation.

Lemma 5.3.19 *Given two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^d$ with total complexity n, the first stage of the algorithm (see Section 5.3.5.4) outputs one of the following:*
(A) a $O(1)$-approximation to the shortcut Fréchet distance between $\boldsymbol{X}$ and $\boldsymbol{Y}$;
(B) an interval $\widehat{\mathcal{I}}$, and curves X and Y with the following properties:
(i) $d_{\mathbb{S}}(X,Y)$ is contained in $\widehat{\mathcal{I}}$ and $d_{\mathbb{S}}(X,Y)/3 \le d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) \le 3d_{\mathbb{S}}(X,Y)$,
(ii) $\widehat{\mathcal{I}}$ contains no vertex-edge, vertex-vertex, or vertex-vertex-edge distance and no tunnel creation radii (as defined in Section 5.3.3) of X and Y.
The running time is $O\bigl(c^2 n\log^3 n\bigr)$.

Proof: We first prove the correctness of the algorithm as stated in the claim. The set U approximates the vertex-vertex distances of the vertices of $\boldsymbol{X}$ and $\boldsymbol{Y}$ up to a factor of two. Therefore, the interval $\mathcal{I} = [\alpha,\beta]$, which we obtain from the first binary search, contains no vertex-vertex distance of $\boldsymbol{X}$ that is more than a factor of two away from its boundary. This implies that the simplification $X = \mathbf{simpl}(\boldsymbol{X},\mu)$ results in the same curve for any $\mu \in [3\alpha,\beta/3]$. An analogous statement holds for $\boldsymbol{Y}$. Unless, a constant factor approximation is found either in the interval $[\alpha,10\alpha]$ or the interval $[\beta/10,\beta]$, the algorithm continues the search using the procedure **decider** and the curves simplified with $\mu = 3\alpha$.

It is now sufficient to search for a constant factor approximation to $d_{\mathbb{S}}(X,Y)$ in the interval $\mathcal{I}' = [3\alpha,\beta/3]$, since this will approximate the desired Fréchet distance by a constant factor. Indeed, by the result of the initial searches, we have that $3\mu \le 10\alpha \le d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y})$. Lemma 5.2.3 implies that $d_{\mathbb{S}}(X,Y) \le d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) + 2\mu \le 3d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y})$. On the other hand, the same lemma implies that $d_{\mathbb{S}}(X,Y) \ge d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) - 2\mu \ge d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y})/3$. This implies, that $d_{\mathbb{S}}(X,Y) \in \mathcal{I}' = [3\alpha,\beta/3]$, since $d_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) \in [10\alpha,\beta/10]$. Note that this also proves the correctness of (i), since the returned interval is contained in $\mathcal{I}'$.

Observe that the set of vertex-vertex distances of X and Y is contained in the set of vertex-vertex distances of $\boldsymbol{X}$ and $\boldsymbol{Y}$. Clearly, $\mathcal{I}'$ cannot contain any vertex-vertex distances of X and Y. The algorithm therefore extracts the remaining vertex-edge events U' from the free space diagram and performs a binary search on them. We obtain the atomic interval $\mathcal{I}'' = [\alpha'', \beta'']$, which contains no vertex-edge events of X and Y. By Lemma 5.3.5 the tunnel creation radius of a tunnel family corresponds to the minimum radius of the family and by Observation 5.3.1 a minimum radius is either a vertex-edge distance or a vertex-vertex-edge distance. By Lemma 3.2.8 these event values would have to lie within a factor two of the boundaries of the interval $\mathcal{I}''$. Therefore, we again search the margins of this interval, so that either we found the desired approximation, or alternatively, it must be in the interval $\mathcal{I}''' = [10\alpha'', \beta''/10]$, which now contains no vertex-vertex, vertex-edge, vertex-vertex-edge distance or tunnel creation event of X and Y. Since $\mathcal{I}'''$ is the interval that the algorithm returns, unless it finds a constant factor approximation to the desired Fréchet distance, the above argumentation implies (i) and (ii).

As for the running time, computing the set U using well-separated pairs decomposition can be done in $O(n \log n)$ time, see Chapter 3. Computing the set U' takes time in $O(n \log n + c^2 n)$, by Data Structure 5.3.12 with $\mu = \beta/3$ and $\delta = \beta$. The algorithm invokes the decision procedure $O(\log n)$ times, and this dominates the overall running time, see Lemma 5.3.17. $\square$

Lemma 5.3.20 *Given two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^d$ of total complexity n, one can compute a constant factor approximation to $d_{\mathcal{S}}(\boldsymbol{X}, \boldsymbol{Y})$. The running time is $O\big(c^2 n \log^3 n\big)$.*

Proof: First, the algorithm performs the preliminary computations as described in Section 5.3.5.4. By Lemma 5.3.19, we either find a constant factor approximation, or we obtain an interval $[\alpha, \beta]$ and simplified curves X and Y. Furthermore, the interval $[\alpha, \beta]$ does not contain any vertex-vertex, vertex-edge, vertex-vertex-edge distance, or tunnel creation event of X and Y. Let P be the canonical gates that are feasible in the β-free space of X and Y. We have that $m = |\mathsf{P}| = O(n)$ and we can compute them using Data Structure 5.3.12 in $O(n \log n + c^2 n)$ time, for $\varepsilon = 1/3$. Thus, the running time up to this stage is bounded by $O\big(c^2 n \log^3 n\big)$, by Lemma 5.3.19.

Now, we invoke the second stage of the algorithm described in Section 5.3.5.5 on the matrix of implicit tunnel prices defined by P and return the output as our solution.

Consider a monotone path in the parametric space that corresponds to the optimal solution. If the price of this path is determined by either a vertex-vertex, a vertex-edge or a vertex-vertex-edge event then we have found an approximation to the shortcut Fréchet distance already in the first stage of the search algorithm. If it is dominated by a tunnel price and this tunnel has both endpoints in the same column of the free space, then by Observation 2.3.1 it is a creation radius. By Lemma 5.3.5 this is equivalent to the minimum radius of the corresponding tunnel family and by Observation 5.3.1 the price of such a tunnel is outside the interval $[\alpha, \beta]$, since by Lemma 5.3.19 these critical values were eliminated in the first stage. Otherwise, this critical tunnel has to be between two columns. Let δ be the price of this tunnel (which is also the price of the whole solution). Consider what happens to this path if we slightly decrease δ. Since δ is optimal, then the critical tunnel either ceases to be feasible or its price is not affordable anymore.

If the critical tunnel is no longer feasible, then one of its endpoints is also an endpoint of the free space interval it lies on. Consider the modified path in the free space, which uses the new endpoint of the free space interval. If the free space interval is empty, then this corresponds to a vertex edge event, and this is not possible inside the interval $[\alpha, \beta]$. The other possibility is that the path is no longer monotone. However, this corresponds to a vertex-vertex-edge event, which again we already handled because of Lemma 5.3.19.

If the tunnel is still feasible, then it must be that the endpoints of this tunnel are contained in the interior of the free space interval and not on its boundary. Now Lemma 5.3.6 (i) implies that the price of this tunnel is equal to the price of the canonical tunnel. As such, the price of the optimal solution is being approximated correctly in this case.

Observe that in the second stage we are searching over all tunnel events that lie in the remaining search interval (whether they are relevant in our case or not). Hence, the search would find the correct critical value, as it is one of the values considered in the search.

The running time of second stage is bounded by:

(A) $O(n \log n \log \log n)$ time to compute the needed entries in the matrix, using Data Structure 5.3.11.

(B) $O\big((c^2 n \log^2 n) \log n\big)$ time for the $O(\log n)$ calls to **Decider**.

(C) $O(n)$ for other computations.

Therefore, the overall running time of the algorithm is $O\big(c^2 n \log^3 n\big)$. □

5.3.6.4 The result

Theorem 5.3.21 *Given two c-packed polygonal curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *in* $\mathbb{R}^d$ *with total complexity n, and a parameter* $\varepsilon > 0$*, the algorithm of Section 5.3.5 computes a* $(3+\varepsilon)$*-approximation to the shortcut Fréchet distance between X and Y in time in* $O\big(Cn \log^3 n\big)$*, where* $C = c^2 \varepsilon^{-2d} \log(1/\varepsilon)$*. The algorithm also outputs the shortcut curve of* $\boldsymbol{Y}$ *and the matchings that realize the approximated shortcut Fréchet distance.*

Proof: The result follows from Lemma 5.3.20. We can turn any constant factor approximation into a $(3+\varepsilon)$-approximation, using **Decider** with $\varepsilon' = \varepsilon/3$ in a binary search over a constant number of subintervals $[\alpha, \beta]$ where $\beta = (3+\varepsilon)\alpha$. It is easy to modify the algorithm, such that it also outputs the shortcut curve and the matchings realizing the approximate Fréchet distance, see Observation 5.3.18. □

5.3.7 Extension to the k-shortcut Fréchet distance

In this section, we describe an algorithm that can be used if the number of shortcuts desired is bounded by a prespecified integer k. The running time of this algorithm is also near-linear in n and has linear dependency on k.

The main algorithm is identical to the algorithm used in the unbounded case (see Section 5.3.5.3), except that it uses k-**Decider** (Algorithm 5.3.24) instead of **Decider** (Algorithm 5.3.15). Therefore, we only describe and analyze the decision procedure.

Algorithm 5.3.23 **k-decider**$(k, X, Y, \varepsilon, \delta)$

```
Input: $k \in \mathbb{N}$; polygonal curves $X$ and $Y$; error $\varepsilon \in (0,1]$; distance $\delta \in \mathbb{R}$
 1: Assert that $\mathrm{dist}(0,0) = \|X(0) - Y(0)\| \le \delta$ and $\mathrm{dist}(1,1) \le \delta$
 2: Let $L_0$ be the set of gates of $\mathcal{D}_{\le\delta}(X,Y)$
 3: Let $S_0 = \{(0,0)\}$
 4: for $i = 1, \ldots, k$ do
 5:    Compute $\mathcal{R}_{\le\delta}(S_{i-1})$
 6:    Let $L_i = L_{i-1} \setminus \mathcal{R}_{\le\delta}(S_{i-1})$
 7:    Let $R_i$ be the set of gates of $\mathcal{R}_{\le\delta}(S_{i-1})$
 8:    for $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p}) \in L_i$ do
 9:       $\mathsf{v} = \mathbf{tunnel}(R_i, \mathsf{p}, \varepsilon, \delta)$
10:       if $\mathsf{v} \ne \mathsf{null}$ then
11:          Add $\mathsf{v}$ to $S_i$
12: if $(1,1)$ is contained in $\mathcal{R}_{\le\delta}(S_0) \cup \cdots \cup \mathcal{R}_{\le\delta}(S_k)$ then
13:    Return "$d_\mathcal{S}(k, X, Y) \le (1+\varepsilon)3\delta$"
14: else
15:    Return "$d_\mathcal{S}(k, X, Y) > \delta$"
```

5.3.7.1 Basic tools

Given a finite set of points P in the parametric space of X and Y, the points that are reachable by an (x,y)-monotone path from a point in P that stays inside the free space is the ***locally reachable free space*** from P (denoted by $\mathcal{R}_{\le\delta}(\mathsf{P})$).

Lemma 5.3.22 *Given a finite set of points P in the δ-free space diagram of two polygonal curves X and Y in $\mathbb{R}^d$ of total complexity n (such that no cell in the free space contains more than $O(1)$ points of P), one can compute $\mathcal{R}_{\le\delta}(\mathsf{P})$ in time $O(|\mathsf{P}| + \mathcal{N}_{\le\delta}(X,Y))$.*

Proof: For each point of P we know the cell of the free space that contains it. So, consider the subset of P contained in a particular cell $C_{i,j}$. Out of this subset, we only need to consider the leftmost and lowermost point that is contained in $\mathcal{D}_{\le\delta}(X,Y)$ for the computation of the reachability intervals at the top and right boundary of this cell. The other points have no effect on the outcome. Recall that the complexity of the free space inside a cell is constant. We know for each $\mathsf{p} \in \mathsf{P}$ the cell $C_{i,j}$ that contains it on the inside or has it on its lower or left boundary. We can in linear time filter out the irrelevant points. As such, assume that P contains only relevant points.

We sort the points P by the indices of the cells that contain them (i.e., a point associated with $C_{i,j}$ appears before a point associated with $C_{i',j'}$, if $j < j'$ or $j = j'$ and $i < i'$). This sorting can be done in linear time using radix-sort. Now, we deploy the BFS approach, as used in Section 5.3.5.2, to compute the reachable free space. The BFS uses two queues, one for the currently visited cells, and the precomputed one (i.e., the ordering computed by the radix sort), to figure out which cells needs to be explored. In each iteration it takes the minimum cell of the two queues. □

Algorithm 5.3.24 k-**Decider**$(k, \boldsymbol{X}, \boldsymbol{Y}, \varepsilon, \delta)$

```
Input: $k \in \mathbb{N}$; polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$; error $\varepsilon \in (0,1]$; distance $\delta \in \mathbb{R}$
1: Let $\varepsilon' = \varepsilon/10$
2: Compute $X = \mathbf{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathbf{simpl}(\boldsymbol{Y}, \mu)$ with $\mu = \varepsilon'\delta$
3: $k$-decider$(k, X, Y, \varepsilon', \delta')$ with $\delta' = (1+2\varepsilon')\delta$
4: Return either "$d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) \leq (1+\varepsilon)3\delta$" or "$d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) > \delta$"
```

5.3.7.2 The decision algorithm

We now describe an approximate decision procedure for the k-shortcut Fréchet distance, where k is a prespecified number of shortcuts allowed. As in the previous algorithm, we will use the **tunnel** procedure described in Section 5.3.5.1. The new decision procedure k-**decider** is layed out in Algorithm 5.3.23. The idea of this algorithm is to incrementally approximate the i-reachable free space, for $i \leq k$. In each step of the iteration we compute the free space that is locally reachable from the endpoints of the tunnels that were computed in the previous iteration (line 6). This can be done efficiently using Lemma 5.3.22. We extract the set of gates R_i from this reachable free space. Let L_i denote the set of gates in $\mathcal{D}_{\leq\delta}(X, Y)$ that have not been reached so far. Now we would like to connect from gates in R_i to gates in L_i via affordable tunnels using the **tunnel** procedure (Algorithm 5.3.13) and which we described and analyzed already. By Lemma 5.3.16, this procedure returns a tunnel of price at most $(1+\varepsilon)3\delta$, given that there exists such a tunnel of price at most δ. The quadrant queries are performed on a static two-dimensional range tree, which we build on the set of discovered gates R_i for each iteration. The initial set of "undiscovered" gates L_0 can be computed using Data Structure 5.3.12. If, after k steps of this algorithm, the point $(1,1)$ has not been reached, then we know that $d_{\mathcal{S}}(k, X, Y) > \delta$. Otherwise, we know that there exists a k'-shortcut curve Y' for $0 \leq k' \leq k$, such that $d_{\mathcal{F}}(X, Y') \leq (1+\varepsilon)3\delta$. As in the previous algorithm, we simplify the curves with $\mu = \Theta(\varepsilon\delta)$ before invoking the k-**decider** procedure in order to ensure that the complexity of the free space diagram is near-linear. The resulting wrapper algorithm is called k-**Decider** and layed out in Algorithm 5.3.24.

5.3.7.3 Analysis

Here, we analyze the decision algorithm. The analysis of the main algorithm is presented in Section 5.3.6.3.

Lemma 5.3.25 *For every* $\mathsf{r} = (x_{\mathsf{r}}, y_{\mathsf{r}}) \in \mathcal{R}_{\leq\delta}(S_i)$, *as computed by* k-**decider** *(Algorithm 5.3.23), it holds that* $\mathsf{r} \in \mathcal{R}^i_{\leq(1+\varepsilon)3\delta}(X, Y)$.

Proof: We prove that for any such r, it holds that $d_{\mathcal{S}}(i, X[0, x_{\mathsf{r}}], Y[0, y_{\mathsf{r}}]) \leq (1+\varepsilon)3\delta$. This is equivalent to showing that there exists a decomposition $X[0,x] = X_0 \oplus X_1 \oplus X_2 \oplus \cdots \oplus X_{2i}$ and a decomposition $Y[0,y] = Y_0 \oplus Y_1 \oplus Y_2 \oplus \cdots \oplus Y_{2i}$, such that for $j = 0, \ldots, i$, it holds that $d_{\mathcal{F}}(X_{2j-1}, Y_{2j-1}) \leq (1+\varepsilon)3\delta$ and $d_{\mathcal{F}}\big(X_{2j}, \overline{Y}_{2j}\big) \leq (1+\varepsilon)3\delta$. By the definition of the locally reachable free space, $d_{\mathcal{F}}(X[0, x_{\mathsf{r}}], Y[0, y_{\mathsf{r}}]) \leq \delta \leq (1+\varepsilon)3\delta$ for any point r in $\mathcal{R}_{\leq\delta}(S_0)$. Therefore, the claim is clearly true for $i = 0$.

For $i > 0$, we inductively decompose the curves such that the pieces satisfy the above condition. For any point r in $\mathcal{R}_{\leq\delta}(S_i)$ there exists a point $\mathsf{u} = (x_\mathsf{u}, y_\mathsf{u}) \in S_i$, such that u is connected by a (x, y)-monotone path to r. This is ensured by Lemma 5.3.22. Let $X_{2i} = X[x_\mathsf{u}, x_\mathsf{r}]$ and $Y_{2i} = Y[y_\mathsf{u}, y_\mathsf{r}]$. We have that $d_\mathcal{F}(X_{2i}, Y_{2i}) \leq \delta \leq (1+\varepsilon)3\delta$, since this path is contained in $\mathcal{D}_{\leq\delta}(X_{2i}, Y_{2i})$.

The point u was added to S_i because it was returned as the endpoint of a tunnel by the **tunnel** procedure in line 9. By Lemma 5.2.1, this tunnel starts at a point $\mathsf{q} \in R_i \subseteq \mathcal{R}_{\leq\delta}(S_{i-1})$ and has the property that $d_\mathcal{F}\big(X[x_\mathsf{q}, x_\mathsf{u}], \overline{Y}[y_\mathsf{q}, y_\mathsf{u}]\big) \leq (1+\varepsilon)3\delta$. We set $X_{2i-1} = X[x_\mathsf{q}, x_\mathsf{u}]$ and $\overline{Y}_{2i-1} = \overline{Y}[y_\mathsf{q}, y_\mathsf{u}]$. By induction, there exists an $(i-1)$-shortcut curve of $Y[0, y_\mathsf{q}]$ that has Fréchet distance at most $(1+\varepsilon)3\delta$ to $X[0, x_\mathsf{q}]$. Concatenating these curves with the tunnel $\tau(\mathsf{q}, \mathsf{u})$, and the monotone path from u to r implies the claim. □

Lemma 5.3.26 *After each iteration of the outer for-loop as executed by the algorithm* k**-decider** *(Algorithm 5.3.23) it holds that the i-reachable free space* $C_i = \mathcal{R}^i_{\leq\delta}(X, Y)$ *is contained in* $\bigcup_{0\leq j\leq i} \mathcal{R}_{\leq\delta}(S_j)$.

Proof: By the definition of the locally reachable free space, the claim is clearly true for $i = 0$. For $i > 0$, assume for the sake of contradiction, that the claim was false, and furthermore, that this is the minimal i for which the claim fails. Therefore, we can assume that the claim is true for any smaller value of i. That is, for any q, such that $d_\mathcal{S}(i-1, X[0, x_\mathsf{q}], Y[0, y_\mathsf{q}]) \leq \delta$, we have that $\mathsf{q} \in \mathcal{R}_{\leq\delta}(S_j)$ for some $j \leq (i-1)$, in other words, the $(i-1)$-reachable free space is computed correctly in previous iterations of the loop. Now, failing the claim means that the algorithm missed a point $\mathsf{p} \in C_i \setminus \mathcal{R}_{\leq\delta}(S_i)$, such that there exists a i-shortcut curve of $Y[0, x_\mathsf{p}]$ which is within Fréchet distance δ to $X[0, y_\mathsf{p}]$. Let P denote the path in the parametric space realizing this distance. We will show that the existence of such a path P implies that $\mathsf{p} \in \mathcal{R}_{\leq\delta}(S_i)$ as computed by the algorithm, which then implies the claim.

Let u and r be the points in the parametric space, such that ur is the last tunnel taken by P, and P connects r to p by a monotone path inside $\mathcal{D}_{\leq\delta}(X, Y)$. We have that $\mathrm{prc}(\tau(\mathsf{u}, \mathsf{r})) \leq \delta$. Furthermore, it must be that $\mathsf{u} \in \mathcal{R}_{\leq\delta}(S_{i-1})$, since u is part of the $(i-1)$-reachable free space which was computed correctly. So, let u' and u'' be the gates of the reachability interval of $\mathcal{R}_{\leq\delta}(S_{i-1})$, which contains u (as computed by Lemma 5.3.22). Similarly, let r' be the left gate of the edge containing r. Observe that $\mathsf{u}', \mathsf{u}'' \in R_i$ and $\mathsf{r}' \in L_i$. Consider the tunnel $\tau(\mathsf{u}', \mathsf{r})$. It must be that its price is at most δ, since $\mathrm{prc}(\tau(\mathsf{u}, \mathsf{r})) \leq \delta$. The algorithm will invoke the **tunnel** procedure with the parameters R_i and r'. By Lemma 5.3.16, and since $\mathrm{prc}(\tau(\mathsf{u}', \mathsf{r})) \leq \delta$, it must return the endpoint v of an affordable tunnel that lies in the free space interval of r. Furthermore, the returned point is the leftmost such point, that is, no other tunnel from R_i of price at most δ can end at a point in $[\mathsf{r}', \mathsf{v}]$. Therefore, v must lie to the left of r or be equal to r. The algorithm would then add v to the set S_i. This implies that r, and therefore also p, are contained in $\mathcal{R}_{\leq\delta}(S_i)$. □

We now prove the correctness and running time of k**-Decider** for approximating the k-shortcut Fréchet distance.

Lemma 5.3.27 *Given three parameters* $k \in \mathbb{N}$, $\delta > 0$ *and* $0 < \varepsilon \leq 1$ *and two c-packed polygonal curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *in* $\mathbb{R}^d$ *of total complexity n. The algorithm* k**-Decider**

(Algorithm 5.3.24) outputs one of the following: (i) "$d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) \leq (1+\varepsilon)3\delta$", or (ii) "$d_{\mathcal{S}}(k, \boldsymbol{X}, \boldsymbol{Y}) > \delta$". In any case, the output returned is correct. The running time is $O\bigl(Ckn \log^2 n\bigr)$, where $C = c^2 \varepsilon^{-2d} \log(1/\varepsilon)$.

Proof: The algorithm k**-Decider** computes the simplified curves $X = \textbf{simpl}(\boldsymbol{X}, \mu)$ and $Y = \textbf{simpl}(\boldsymbol{Y}, \mu)$ with $\mu = \Theta(\varepsilon\delta)$, before invoking the algorithm k**-decider** (Algorithm 5.3.23) on these curves. If, by the end of the computations, $(1,1) \in \mathcal{R}_{\leq\delta}(S_0) \cup \cdots \cup \mathcal{R}_{\leq\delta}(S_k)$, as tested in line 12, then Lemma 5.3.25 proves that the output returned in line 13 is correct with respect to the simplified curves. Otherwise, Lemma 5.3.26 shows that the output in line 15 is correct with respect to them. By Lemma 5.2.3, this approximates the decision with respect to the original curves sufficiently.

We assume that we have an annotated graph representation of the free space diagram as used in Lemma 5.3.22. In this case, we can easily extract the gates contained in $\mathcal{R}_{\leq\delta}(S_i)$ during its computation in line 6, as well as, to check whether it contains $(1,1)$.

The sets R_i, S_i and L_i are finite sets of two-dimensional points, and as such, we can use static two dimensional range trees for the orthogonal range queries in line 1 and line 6 in the **tunnel** procedure.

As for the running time, observe that by Lemma 3.1.6 there are at most $N = O(cn/\varepsilon)$ points in L_i, S_i and R_i, at any point in time. As such, building the range trees on them, in each iteration, takes $O(N \log N)$ time, and this also accounts for all the queries performed on these data structures. Note that computing the reachable free space in line 6 in each iteration takes time in $O(|S_i| \log n + N \log N) = O(N \log N)$ by Lemma 5.3.22. Therefore, maintaining the sets R_i, S_i and L_i and all operations on them takes $T_1 = O(kN \log N)$ time overall. Computing the initial set of gates L_0 in line 3, takes $T_2 = O(n \log n + c^2 n/\varepsilon)$ time by Data Structure 5.3.12.

Building the data structure that supports the queries for the price of a tunnel takes $T_3 = O\bigl(n\varepsilon^{-2d} \log^2(1/\varepsilon) \log^2 n\bigr)$ time by Data Structure 5.3.11. Throughout the algorithm execution, we perform $O(kN)$ such queries on this data structure, which takes

$$T_4 = O\bigl(kN\varepsilon^{-3} \log n \log\log n\bigr) = O\bigl(ckn\varepsilon^{-4} \log n \log\log n\bigr)$$

time overall.

As such, the overall running time is

$$\begin{aligned} T_1 + T_2 + T_3 + T_4 &= O\bigl(nck\varepsilon^{-4} \log n + kN \log N + n\varepsilon^{-2d} \log^2(1/\varepsilon) \log^2 n + c^2 n/\varepsilon\bigr) \\ &= O\bigl(Ckn \log^2 n\bigr), \end{aligned}$$

where $C = c^2 \varepsilon^{-2d} \log(1/\varepsilon)$. □

5.3.7.4 The result

Theorem 5.3.28 *Given an integer $k > 0$, a parameter $\varepsilon > 0$, and two c-packed polygonal curves X and Y in $\mathbb{R}^d$, with total complexity n, the algorithm described above computes a $(3+\varepsilon)$-approximation to the k-shortcut Fréchet distance between X and Y in time in $O\bigl(Ckn \log^2 n\bigr)$, where $C = c^2 \varepsilon^{-2d} \log(1/\varepsilon)$. The algorithm also* *outputs the shortcut curve of* $\boldsymbol{Y}$ *and the matchings that realize the respective* k*-shortcut Fréchet distance.*

Proof: The main algorithm is described in Section 5.3.5.3 and analyzed in Section 5.3.6.3. We use the decision procedure k-**Decider** (Section 5.3.7.2) instead of **Decider**. The running time and correctness therefore follows from the proof of Theorem 5.3.21, except that we use Lemma 5.3.27 for the decision procedure. □

5.4 The continuous shortcut Fréchet distance

In this section we study the directed shortcut Fréchet distance in the general case where shortcuts can start and end at any point along one of the two curves and the number of shortcuts is unbounded. We will further omit the predicates directed and continuous in this section and simply refer to it as the shortcut Fréchet distance. In Section 5.4.1 we give an NP-hardness reduction and prove its correctness in Section 5.4.2. In Section 5.4.3 we extend some of our approximation results from the previous sections to this more general case.

5.4.1 NP-hardness reduction

We prove that deciding if the shortcut Fréchet distance between two given curves is smaller or equal a given value is weakly NP-hard by a reduction from SUBSET-SUM. We first discuss the main ideas and challenges in Section 5.4.1.1, then we formally describe the reduction in Section 5.4.1.2. The correctness is proven in Section 5.4.2.

5.4.1.1 General idea

The SUBSET-SUM instance is given as a set of input values and a designated sum value. A solution to the instance is a subset of these values that has the specified sum. We describe how to construct a ***target curve*** X and a ***base curve*** Y, such that there exists a shortcut curve of Y which is in Fréchet distance 1 to X if and only if there exists a solution to the SUBSET-SUM instance. We call such a shortcut curve ***feasible*** if it lies within Fréchet distance 1. The construction of the curves is split into gadgets, each of them encoding one of the input values.

Idea of the reduction. We construct the target curve to lie on a horizontal line going mostly rightwards. The base curve has several horizontal edges which lie at distance 1 to the target curve and which go leftwards. The monotonicity requirement on the matchings considered by the Fréchet distance (see Section 2.1) forces a feasible shortcut curve to "jump" rightwards along the edges of the base curve using shortcuts in between and visiting each edge in exactly one point. We can further control the feasible shortcut curves by adding occasional "twists" to the target curve. The monotonicity requirement forces the shortcuts to go through the centers of these twists. See Figure 5.4 for an illustration. In our construction this mechanism is used as follows. Intuitively, an edge visited by the shortcut curve acts as a mirror and a shortcut matching to a twist acts as a projection onto such a mirror[3] where the center of the

[3]However, Snell's law does not hold in this analogy.

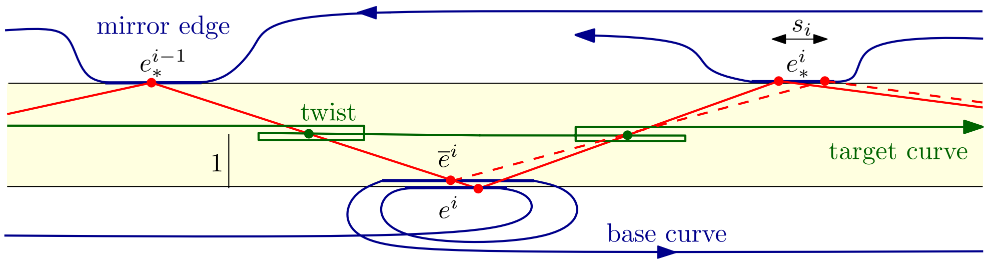


Figure 5.4: Simplistic version of the gadget encoding one of the input values s_i. A feasible shortcut curve entering from a position on the edge e_*^{i-1} has the choice to visit either e^i or $\overline{\mathsf{e}}^i$, resulting in different visiting positions on e_*^i.

twist corresponds to the projection center. A shortcut curve that enters a gadget via a projection center will have two different edges of the base curve available to visit. Depending on the choice of the edge, the two projections cascade through the next projection center until they hit the following mirror edge where their distance to each other encodes one of the input values. The choice of the initial edge (i.e., e^i vs. $\overline{\mathsf{e}}^i$ in Figure 5.4) encodes whether this input value is selected by the shortcut curve. If yes, then the shortcut curve will end up further to the right on the last mirror edge of the gadget (i.e., e_*^i in Figure 5.4), from which it enters the next gadget. Due to the resulting horizontal shift, the value is implicitly copied to the next gadget. By visiting a number of such gadgets, which have been threaded together, a shortcut curve accumulates the sum of the selected subset in the visiting position on the last edge of each gadget. We construct the terminal gadget such that only those shortcut curves that encode a subset which sums to the designated value can connect through the last projection center to the endpoint of the base curve.

Challenges. We would like to construct gadgets such as the one shown in Figure 5.4 which have the behavior as described above. A key element in the construction is the fact that a feasible shortcut curve visiting a mirror edge at distance exactly 1 to the target curve has to leave the mirror edge immediately. However, if we want to offer a choice of destination edges for a shortcut, then one of the two edges has to lie within distance 1. In this case, it may happen that a shortcut curve visits the edge in more than one point by moving leftward on the edge before leaving again in rightward direction. Therefore, the visiting position which encodes the current partial sum will be only approximated. We will use a scaling parameter to prevent influence of this approximation error on the solution. The second challenge is that we need to space the two edges e^i and $\overline{\mathsf{e}}^i$ horizontally apart to prevent the shortcut curve from visiting both of them. We will use more than two twists per gadget to realize the spacing and the correct distances of the projections. After selecting the initial edge, the projections of different shortcut curves will cascade through the remaining projection centers of the gadget while bouncing off from the mirrors in order to be bundled again on the last edge of the gadget, where their distance to each other encodes one of the input values.

5.4.1.2 Reduction

We describe how to construct the curves X and Y from an instance of SUBSET-SUM and how to extract a solution.

Input. We are given n positive integers $\{s_1, s_2, \ldots, s_n\}$ and a positive integer σ. The problem is to decide whether there exists an index set I, such that $\sum_{i \in I} s_i = \sigma$. For any index set I, we call $\sigma_i = \sum_{1 \leq j \leq i, j \in I} s_j$ the ith ***partial sum*** of the corresponding subset.

Layout and notation. Our construction consists of $n+2$ gadgets: an initialization gadget g_0, a terminal gadget g_{n+1}, and intermediate gadgets g_i for each value s_i, for $i = 1, \ldots, n$. A ***gadget*** g_i consists of curves X_i and Y_i. We concatenate these in the order of i to form X and Y. We denote with H_y the horizontal line at y. The edges of X_i lie on H_0 running in positive x-direction, except for occasional twists, which we define as follows. A ***twist*** centered at a point $\mathsf{p} = (p, 0)$ is a subcurve defined by the vertices $(p-1, 0)$,$(p+1, 0)$,$(p-1, 0)$,$(p+1, 0)$ which we connect by straight line segments in this order. We call p the ***projection center*** of the twist. Let ζ be a global parameter of the construction, we call the open rectangle of width ζ and height 3 and centered at p a ***buffer zone*** of the twist. The idea is that the base curve stays outside the buffer zones of the twists of the target curve. We call the locus of points that are within distance 1 to the target curve the ***hippodrome***. The curve Y_i is defined by placing leftward horizontal edges on H_{-1}, H_1 and H_α, where $0 < \alpha < 1$ is another global parameter of the construction. We call these edges ***mirror edges***. The remaining edges of Y_i, which are used to connect the mirror edges to each other, are called ***connector edges***. The mirror edges located on H_1 and H_{-1} can be connected using curves that lie outside the hippodrome. Since those connector edges cannot be visited by any feasible shortcut curve, their exact placement is irrelevant. The edges connecting to mirror edges located on H_α and which intersect the hippodrome are placed carefully such that no feasible shortcut curve can visit them. Since all relevant points of the construction lie on a small set of horizontal lines, we take the liberty to slightly abuse notation by denoting the x-coordinate of a point and the point itself with the same variable, albeit using a different font. For example, we denote with a^i_j the point that has x-coordinate a^i_j. In the figures, we omit the top index i of the variables of gadget g_i to make the description less cluttered. We use $\overline{\mathsf{ab}}$ to denote the line through the points a and b.

Global variables. The construction uses four global variables. The parameter $0 < \alpha < 1$ is besides 1 and -1 the y-coordinate of a horizontal line that supports mirror edges. The parameter $\beta > 0$ controls the minimal horizontal distance between mirror edges that lie between two consecutive buffer zones. The parameter $\delta > 0$ acts as a scaling parameter to ensure (i) that the projections stay inside the designated mirror edges and (ii) that the projections of two different partial sums are kept disjoint despite the approximation error. The fourth parameter ζ defines the width of a buffer zone, this is the minimum horizontal distance that a point on the base curve has to a projection center. How to choose the exact values of these variables will follow from Lemma 5.4.13.

Main gadgets. The construction of the gadgets is incremental. Given the endpoints of the last mirror edge of gadget g_{i-1} and the value s_i, we construct gadget g_i. The overall structure is depicted in Figure 5.6. The curve Y_i for $1 \leq i \leq n$ has seven mirror edges. These are $\mathsf{e}^i_j = \mathsf{a}^i_j\mathsf{b}^i_j$, and $\bar{\mathsf{e}}^i_j = \mathsf{c}^i_j\mathsf{d}^i_j$, for $1 \leq j \leq 3$, and the edge $\mathsf{e}^i_* = \mathsf{a}^i_*\mathsf{b}^i_*$. We connect the mirror edges using additional edges to define the following order along the base curve: $\mathsf{e}^{i-1}_*, \mathsf{e}^i_1, \bar{\mathsf{e}}^i_1, \bar{\mathsf{e}}^i_2, \mathsf{e}^i_2, \mathsf{e}^i_3, \bar{\mathsf{e}}^i_3, \mathsf{e}^i_*$. The mirror edges lie on the horizontal lines H_1, H_{-1} and H_α. We use vertical connector edges which run in positive y-direction and additional connector edges which lie completely outside the hippodrome to connect the mirror edges on H_α. The curve X_i for $1 \leq i \leq n$ consists of four twists centered at the projection centers p^i_j for $1 \leq j \leq 4$ which are connected in the order of j by rightward edges on H_0. To choose the exact coordinates of these points, we go through several rounds of fixing the position of the next projection center and then projecting the endpoints of mirror edges to obtain the endpoints of the next set of mirror edges. The construction is defined in four steps in Table 5.5 and illustrated in Figure 5.6 (bottom right and bottom left). The intuition behind this choice of projection centers is the following. In every step we make sure that the base curve stays out of the buffer zones. Furthermore, in Step 1 we choose the projection center far enough to the right such that two mirror edges located between two consecutive buffer zones have distance at least β. In Step 4 we align the projections of the two edges e^i_3 and $\bar{\mathsf{e}}^i_3$. In this alignment, the visiting position that represents "0" on e^i_3 (i.e., in its distance to a^i_3) and the visiting position that represents "s_i" on $\bar{\mathsf{e}}^i_3$ (i.e., in its distance to c^i_3) both project to the same point on e^i_* (i.e., the visiting position that represents "s_i" in its distance to b^i_*). In this way, the projections from e^i_3 are horizontally shifted by s_i (scaled by δ) with respect to the projections from $\bar{\mathsf{e}}^i_3$.

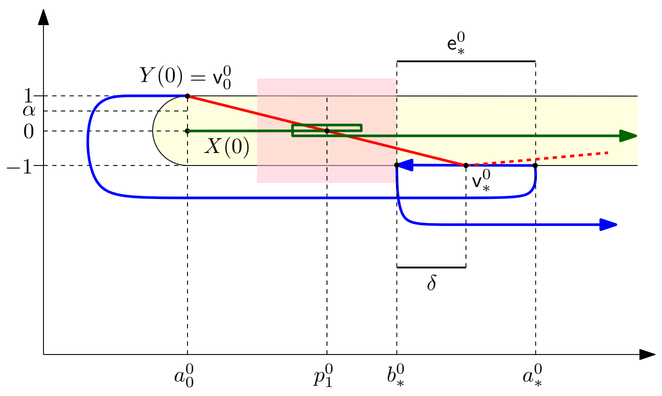
Initialization gadget. We let both curves start on the vertical line at $a^0_0 = 0$ by placing the first vertex of X_0 at $(a^0_0, 0)$ and the first vertex of the Y_0 at $(a^0_0, 1)$. The base curve Y_0 then continues to the left on H_1 while the target curve X_0 continues to the right on H_0. See Figure 5.6 (top left) for an illustration. The curve X_0 has one twist centered at $p^0_1 = a^0_0 + \delta + \zeta/2$. The curve Y_0 has one mirror edge $\mathsf{e}^0_* = \mathsf{a}^0_*\mathsf{b}^0_*$, which we define by setting $b^0_* = p^0_1 + \zeta/2$ and $a^0_* = p^0_1 + 2\delta + \zeta/2$.

Terminal gadget. Assume we have constructed the gadgets $g_0,\ldots,g_n$ and now want to construct gadget g_{n+1} from e^*_n. The curve X_{n+1} has one twist centered at $p^{n+1}_1 = a^n_* + \zeta/2$. Let $p_\sigma = (b^n_* + \delta(\sigma + 1))$ and project the point $(p_\sigma, -1)$ through p^{n+1}_1 onto H_1 to obtain a point $(a_\sigma, 1)$. We finish the construction by letting both the target curve X_{n+1} and the base curve Y_{n+1} end on a vertical line at a_σ. The curve X_{n+1} ends on H_0 approaching from the left, while the curve Y_{n+1} ends on H_1 approaching from the right. Figure 5.6 (top right) shows an illustration.

Encoding of a subset. In our reduction, any shortcut curve of the base curve encodes a subset of the SUBSET-SUM instance. We say that the value s_i is included in the encoded subset if and only if the shortcut curve visits the edge e^i_1. The ith partial sum of the encoded subset will be represented by the point where the shortcut curve visits the edge e^*_i. In particular, the distance of the visiting point to the endpoint of the edge represents this value, scaled by δ and up to a small additive error.

Step 1:	$\mathsf{p}_1 = \overline{\mathsf{a}_*^{i-1}\,(\phi_i, -\alpha)} \cap H_0$	with $\phi_i = a_*^{i-1} + \beta + \lambda_i$ $\lambda_i = a_*^{i-1} - b_*^{i-1}$
	$\mathsf{a}_1 = \overline{\mathsf{b}_*^{i-1}\mathsf{p}_1} \cap H_1$	$\mathsf{b}_1 = \overline{\mathsf{a}_*^{i-1}\mathsf{p}_1} \cap H_1$
	$\mathsf{c}_1 = \overline{\mathsf{b}_*^{i-1}\mathsf{p}_1} \cap H_\alpha$	$\mathsf{d}_1 = \overline{\mathsf{a}_*^{i-1}\mathsf{p}_1} \cap H_\alpha$
Step 2:	$\mathsf{p}_2 = (a_1 + \zeta/2, 0)$	
	$\mathsf{a}_2 = \overline{\mathsf{b}_1\mathsf{p}_2} \cap H_{-1}$	$\mathsf{b}_2 = \overline{\mathsf{a}_1\mathsf{p}_2} \cap H_{-1}$
	$\mathsf{c}_2 = \overline{\mathsf{d}_1\mathsf{p}_2} \cap H_{-1}$	$\mathsf{d}_2 = \overline{\mathsf{c}_1\mathsf{p}_2} \cap H_{-1}$
Step 3:	$\mathsf{p}_3 = (c_2 + \zeta/2, 0)$	
	$\mathsf{a}_3 = \overline{\mathsf{b}_2\mathsf{p}_3} \cap H_\alpha$	$\mathsf{b}_3 = \overline{\mathsf{a}_2\mathsf{p}_3} \cap H_\alpha$
	$\mathsf{c}_3 = \overline{\mathsf{d}_2\mathsf{p}_3} \cap H_1$	$\mathsf{d}_3 = \overline{\mathsf{c}_2\mathsf{p}_3} \cap H_1$
Step 4:	$\mathsf{p}_4 = \overline{(c_3 - \delta s_i, 1)\mathsf{a}_3} \cap H_0$	
	$\mathsf{a}_4 = \overline{\mathsf{b}_3\mathsf{p}_4} \cap H_{-1}$	$\mathsf{b}_4 = \overline{\mathsf{a}_3\mathsf{p}_4} \cap H_{-1}$
	$\mathsf{c}_4 = \overline{\mathsf{d}_3\mathsf{p}_4} \cap H_{-1}$	$\mathsf{d}_4 = \overline{\mathsf{c}_3\mathsf{p}_4} \cap H_{-1}$
	$\mathsf{a}_* = (\max(a_4, b_4, c_4, d_4), -1)$	$\mathsf{b}_* = (\min(a_4, b_4, c_4, d_4), -1)$

Table 5.5: Construction of the main gadgets g_i for $1 \leq i \leq n$. We omitted the top index i of the variables. Thus, b_* stands for b_*^i, etc. H_1, H_{-1} and H_α denote the horizontal lines at $1, -1$ and α respectively.



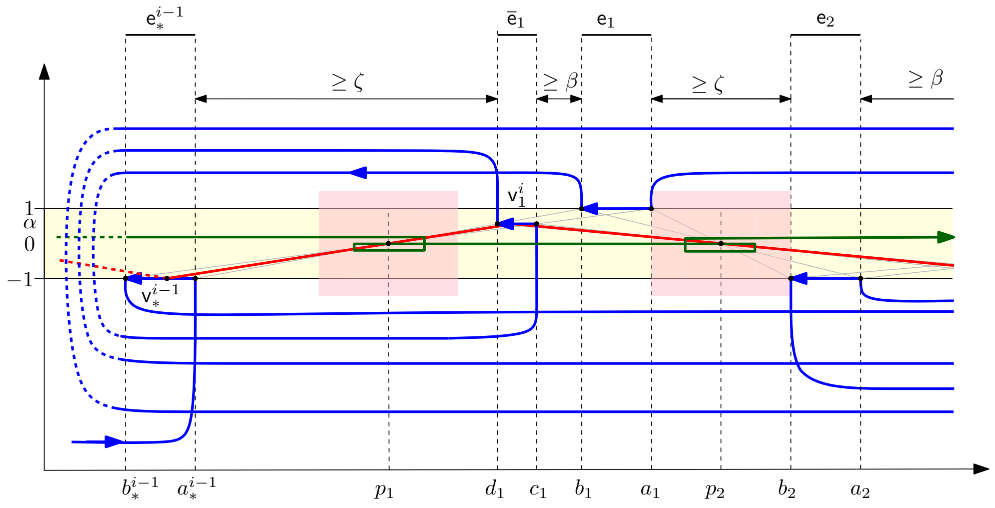
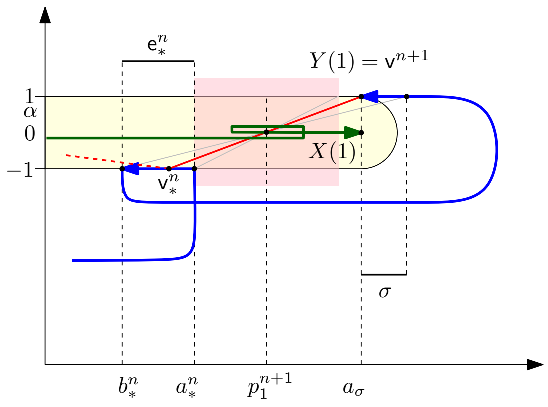


Figure 5.6: top: initialization gadget g_0; bottom: the left part of gadget g_i. The target curve is shown in green, the base curve in blue. An example shortcut curve is shown in red. Buffer zones and the hippodrome are shown as shaded regions. For the sake of presentation, the target curve is distorted to show its topology, and the lengths of the mirror edges have been assumed smaller. The top index i has been omitted from the variables.



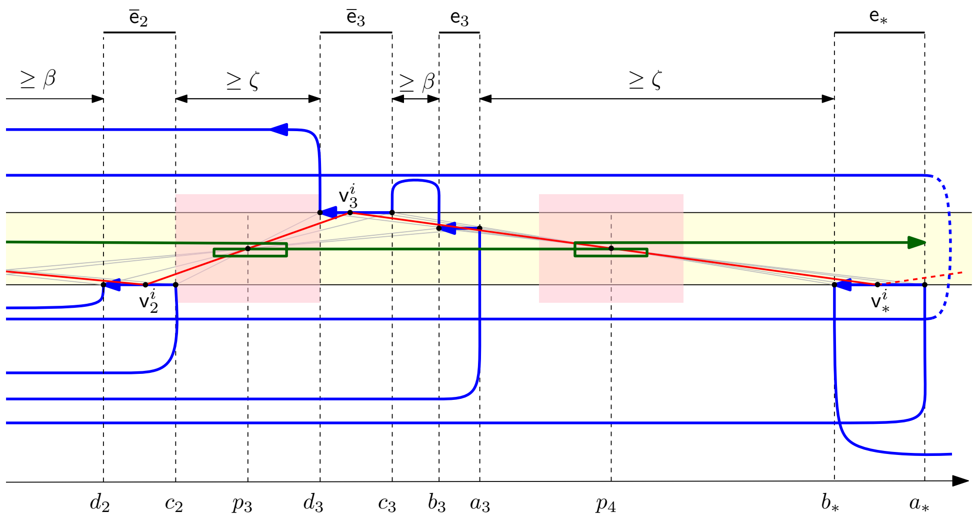


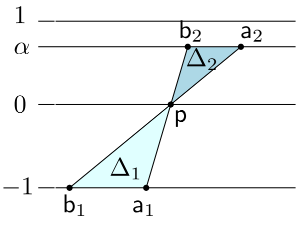
Figure 5.6: (continued) top: terminal gadget g_{n+1}; bottom: the right part of gadget g_i with example shortcut curve.

5.4.2 Correctness of the reduction

Now we prove that the construction has the desired behavior. That is, we prove that any feasible shortcut curve encodes a subset that constitutes a solution to the SUBSET-SUM instance (Lemma 5.4.13) and for any solution of the SUBSET-SUM instance, we can construct a feasible shortcut curve (Lemma 5.4.11).

We call a shortcut curve ***one-touch*** if it visits any edge of the base curve in at most one point. Intuitively, for any feasible shortcut curve of Y, there exists a one-touch shortcut curve that "approximates" it. We first prove the correctness of the reduction for this restricted type of shortcut curve (Lemma 5.4.11), before we turn to general shortcut curves. In Definition 5.4.2 we define a ***one-touch encoding***, which is a one-touch shortcut curve that is feasible if and only if the encoded subset constitutes a valid solution. For such curves, Lemma 5.4.4 describes the correspondence of the current partial sum with the visiting position on the last edge of each gadget. It readily follows that we can construct a feasible shortcut curve from a valid solution (Lemma 5.4.11). For the other direction of the correctness proof we need some lemmas to testify that any feasible shortcut curve is approximately monotone (Lemma 5.4.7), has its vertices outside the buffer zones (Lemma 5.4.6), and therefore has to go through all projection centers (Lemma 5.4.8). We generalize Lemma 5.4.4 to bound the approximation error in the representation of the current partial sum (Lemma 5.4.12). As a result, Lemma 5.4.13 implies that any feasible shortcut curve encodes a valid solution.

In the proofs to these lemmas we often reason using projections of distances between H_1,H_{-1} and H_α. The common argument is captured in the following observation.



Observation 5.4.1 (Distance projection) If Δ_1 is a triangle defined by two points a_1 and b_1 that lie on H_{-1} and a point p that lies on H_0, and if Δ_2 is a triangle defined by the points p, and the two points $\mathsf{b}_2 = \overline{\mathsf{a}_1\mathsf{p}} \cap H_\alpha$ and $\mathsf{a}_2 = \overline{\mathsf{b}_1\mathsf{p}} \cap H_\alpha$, which are the projections onto H_α, then it holds that $\alpha(a_1 - b_1) = a_2 - b_2$, where a_1, b_1, a_2 and b_2 are the respective x-coordinates of the points.

5.4.2.1 Correctness for one-touch shortcut curves

We first do the analysis under the simplifying assumption that only shortcut curves are allowed that are one-touch, i.e., shortcut curves can visit the base curve in at most one point per edge. In the next section we will build upon this analysis for the general case.

Definition 5.4.2 (One-touch encoding) Let I be an index set of a subset $S' \subseteq S$. We construct a one-touch shortcut curve Y_I of the base curve incrementally. The first two vertices on the initial gadget are defined as follows. We choose the first vertex of the base curve $Y(0)$ for v_0^0, then we project it through the first projection center p_1^0 onto e_*^0 to obtain v_*^0. Now for $i > 0$, if $i \in I$, then we project v_*^{i-1} through p_1^i onto e_1^i, otherwise onto $\bar{\mathsf{e}}_1^i$ to obtain v_1^i. We continue by projecting v_j^i through p_{j+1}^i onto Y_i to obtain v_{j+1}^i, for $1 \leq j \leq 4$. Let $\mathsf{v}_*^i = \mathsf{v}_4^i$. We continue this construction throughout

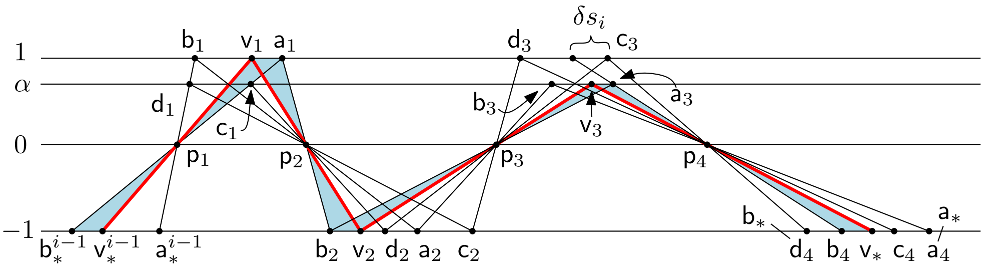


Figure 5.7: The path of a shortcut curve through the gadget g_i in the case where s_i is included in the selected set (Lemma 5.4.4). For presentation purposes, we allowed the mirror edges to overlap horizontally and we omitted the top index i of the variables.

all gadgets in the order of i. Finally, we choose $Y(1) = (a_\sigma, 1)$ as the last vertex of our shortcut curve. Figure 5.6 shows an example.

Lemma 5.4.3 *For any $1 \leq i \leq n$, it holds that $b_4^i = b_*^i + \delta s_i$ and that $d_4^i = b_*^i$.*

Proof: By construction, the projection center p_4^i lies on a common line with a_3^i and b_4^i. Recall that we chose p_4^i such that this line intersects H_1 at the x-coordinate $c_3^i - \delta s_i$ (see Table 5.5 and Figure 5.7). Thus, by Observation 5.4.1, it holds that $b_4^i - d_4^i = \delta s_i$. Furthermore, d_4^i is the point with minimum x-coordinate out of the projections of a_3^i, b_3^i, c_3^i, and d_3^i through p_4^i onto H_{-1}, since $\delta s_i \geq 0$. Since we chose b_*^i as such point with minimum x-coordinate, the claim is implied. □

Lemma 5.4.4 *Given a shortcut curve Y_I which is a one-touch encoding (Definition 5.4.2), let v_*^i be the vertex of Y_I on e_*^i, for any $0 \leq i \leq n$. It holds for the distance of v_*^i to the endpoint of this edge that $\left\|\mathsf{v}_*^i - \mathsf{b}_*^i\right\| = \delta(\sigma_i + 1)$, where σ_i is the ith partial sum of the subset encoded by Y_I.*

Proof: We prove the claim by induction on i. For $i = 0$, the claim is true by the construction of the initialization gadget, since the partial sum $\sigma_0 = 0$ and $\left\|\mathsf{v}_*^0 - \mathsf{b}_*^0\right\| = \delta$. For $i > 0$, there are two possibilities, either $i \in I$ or $i \notin I$. Consider the case that s_i is included in the set encoded by Y_I. In that case, the curve has to visit the edge e_1^i.

By a repeated application of Observation 5.4.1 we can derive that

$$\left\|\mathsf{v}_*^{i-1} - \mathsf{b}_*^{i-1}\right\| = \left\|\mathsf{v}_1^i - \mathsf{a}_1^i\right\| = \left\|\mathsf{v}_2^i - \mathsf{b}_2^i\right\| = \frac{\left\|\mathsf{v}_3^i - \mathsf{a}_3^i\right\|}{\alpha} = \left\|\mathsf{v}_*^i - \mathsf{b}_4^i\right\|.$$

Refer to Figure 5.7 for an illustration of the geometry of the path through the gadget. The shaded region shows the triangles that transport the distances. Therefore, by induction and by Lemma 5.4.3,

$$\left\|\mathsf{v}_*^i - \mathsf{b}_*^i\right\| = \left\|\mathsf{v}_*^i - \mathsf{b}_4^i\right\| + \left\|\mathsf{b}_4^i - \mathsf{d}_4^i\right\| = \delta(\sigma_{i-1} + 1) + \delta s_i = \delta(\sigma_i + 1).$$

Thus, the claim follows for the case that s_i is selected. For the second case, the curve has to visit the edge $\bar{\mathsf{e}}_1^i$. Again, by Observation 5.4.1 it holds that

$$\left\|\mathsf{v}_*^{i-1} - \mathsf{b}_*^{i-1}\right\| = \frac{\left\|\mathsf{v}_1^i - \mathsf{c}_1^i\right\|}{\alpha} = \left\|\mathsf{v}_2^i - \mathsf{d}_2^i\right\| = \left\|\mathsf{v}_3^i - \mathsf{c}_3^i\right\| = \left\|\mathsf{v}_*^i - \mathsf{d}_4^i\right\|.$$

Thus $\left\|\mathsf{v}_*^i - \mathsf{d}_4^i\right\| = \delta(\sigma_{i-1} + 1) = \delta(\sigma_i + 1)$. By Lemma 5.4.3, $\mathsf{d}_4^i = \mathsf{b}_*^i$, thus the claim is implied also in this case. □

Using the arguments from the proof of Lemma 5.4.4 one can derive the following corollary.

Corollary 5.4.5 *For any $0 \le i \le n$ the length of the edge e_*^i is equal to*

$$\delta\left(\sum_{1\le j\le i} s_j + 2\right).$$

Lemma 5.4.6 *If $\alpha \in [1/2, 1)$ and $\beta \ge \zeta/2$, then the base curve does not enter any of the buffer zones.*

Proof: For the buffer zones centered at p_2^i and p_3^i for $1 \le i \le n$ the claim is implied by construction. The same holds for the projection centers of the initialization gadget and the terminal gadget. Thus, we only need to argue about the first and the last projection center of the intermediate gadgets g_i for $1 \le i \le n$. Consider p_1^i, by construction and since $\alpha \ge 1/2$, it holds that

$$d_1^i - p_1^i = p_1^i - \phi_i \ge \phi_i - a_*^{i-1} = \beta + \lambda_i > \beta \ge \zeta/2,$$

where λ_i and ϕ_i are defined as in the construction of the gadgets. Since d_1^i is the closest x-coordinate of the base curve to p_1^i, the claim follows for the first projection center of a gadget.

For the last projection center, p_4^i, we use the fact that it lies to the right of the point where the line through a_3^i and c_3^i passes through H_0. Let the x-coordinate of this point be denoted c_i. Now, let Δ_1 be the triangle defined by p_2^i, d_1^i and c_1^i and let Δ_2 be the triangle defined by p_3^i, a_3^i and b_3^i. By the symmetry of the construction, the two triangles are the same up to reflection at the bisector between p_2^i and p_3^i. Therefore,

$$p_4^i - a_3^i \ge c_i - a_3^i = d_1^i - p_1^i > \zeta/2,$$

and this implies the claim. □

Lemma 5.4.7 *Any feasible shortcut curve is rightwards 4-monotone. That is, if x_1 and x_2 are the x-coordinates of two points that appear on the shortcut curve in that order, then $x_2 + 4 \ge x_1$. Furthermore, it lies inside or on the boundary of the hippodrome.*

Proof: Any point on the feasible shortcut curve has to lie within distance 1 to some point of the target curve, thus the curve cannot leave the hippodrome. As for the montonicity, assume for the sake of contradiction, that there exist two points such that

$x_2+4 < x_1$. Let $\widehat{x}_1$ be the x-coordinate of the point on target curve matched to x_1 and let $\widehat{x}_2$ be the one for x_2. By the Fréchet matching it follows that $\widehat{x}_2 - 1 + 4 < \widehat{x}_1 + 1$. This would imply that the target curve is not 2-monotone, which contradicts the way we constructed it. □

Lemma 5.4.8 *If $\alpha \in [1/2, 1)$, $\zeta > 4$ and if $\beta \geq \zeta/2$, then a feasible shortcut curve passes through every buffer zone of the target curve via its projection center and furthermore it does so from left to right.*

Proof: Any feasible shortcut curve has to start at $Y(0)$ and end at $Y(1)$, and all of its vertices must lie in the hippodrome or on its boundary. By Lemma 5.4.6 the base curve does not enter any of the buffer zones and therefore the feasible shortcut curve has to pass through the buffer zone by using a shortcut. If we choose the width of a buffer zone $\zeta > 4$, then the only manner possible to do this while matching to the two associated vertices of the target curve in their respective order, is to go through the intersection of their unit disks that lies at the center of the buffer zone. This is the projection center associated with the buffer zone. By the order in which the mirror edges are connected to form the base curve, it must do so in positive x-direction and it must do so exactly once. □

Lemma 5.4.9 *For any $1 \leq i \leq n$ it holds that $b_1^i - c_1^i \geq \beta$, $d_2^i - a_2^i \geq \beta$, and $b_3^i - c_3^i \geq \beta$.*

Proof: Recall that we chose p_1^i by constructing the point $(\phi_i, -\alpha)$, where $\phi_i = a_*^{i-1} + \lambda_i + \beta$ and $\lambda_i = a_*^{i-1} - b_*^{i-1}$. The construction is such that $(\phi_i, -\alpha)$, p_1^i and a_*^{i-1} lie on the same line. Consider the point $(r_i, -\alpha)$ that lies on the line through b_*^{i-1} and p_1^i. We have that the triangle Δ_1 defined by $(r_i, -1)$, p_1^i and a_*^{i-1} is the same up to rotation as the triangle Δ_2 defined by $(c_1^i, 1)$, p_1^i and b_1^i. Refer to Figure 5.8 for an illustration. By Observation 5.4.1,

$$b_1^i - c_1^i = r_i - a_*^{i-1} = \beta + \lambda_i - (\phi_i - r_i) \geq \beta,$$

since $(\phi_i - r_i) = \alpha\lambda_i$. This proves the first part of the claim. Now it readily follows that also $d_2^i - a_2^i \geq \beta$. Indeed, it follows by Observation 5.4.1 that $q_i - a_2^i = b_1^i - c_1^i$, where q_i is the x-coordinate of the projection of $(c_1^i, 1)$ through p_2^i onto H_{-1}, and this projection lies between d_2^i and a_2^i. Again, refer to Figure 5.8 and in particular to triangles Δ_1' and Δ_2'.

The claim $b_3^i - c_3^i \geq \beta$ follows from the symmetry of the middle part of the gadget. Consider the triangle Δ_3 defined by $(c_1^i, 1)$, p_2^i and $(b_1^i, 1)$ and the triangle Δ_4 defined by $(c_3^i, 1)$, p_3^i and $(b_3^i, 1)$. By construction Δ_3 is a reflected version of Δ_4, where the axis of reflection is the bisector of the two projection centers. Thus, by the above argument we have that $b_3^i - c_3^i = b_1^i - c_1^i \geq \beta$.

□

Lemma 5.4.10 *If $\alpha \in [1/2, 1)$, $\zeta > 4$ and if $\beta > 4$, then a feasible shortcut curve that is one-touch visits either e_j^i or $\overline{\mathsf{e}}_j^i$ for any $1 \leq i \leq n$ and $1 \leq j \leq 3$. Furthermore, it visits all edges e_*^i for $0 \leq i \leq n$.*

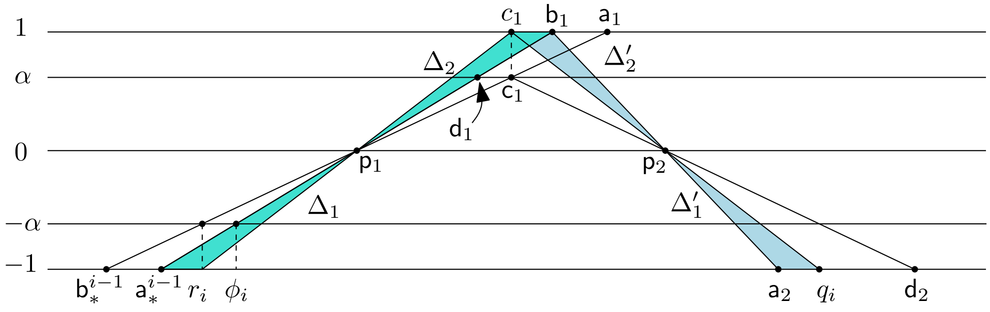


Figure 5.8: The geometry used in the proof of Lemma 5.4.9. We omitted the top index i of the variables.

Proof: By Lemma 5.4.7, any feasible shortcut curve is 4-monotone. Furthermore, it starts at $Y(0)$ and ends at $Y(1)$ and by Lemma 5.4.8, it goes through all projection centers of the target curve from left to right. We first want to argue that it visits at least one mirror edge between two projection centers, i.e., that it cannot "skip" such a mirror edge by matching to two twists in one shortcut. Such a shortcut would have to lie on H_0, since it has to go through the two corresponding projection centers lying on H_0. By construction, the only possible endpoints of such a shortcut lie on the connector edges that connect to mirror edges on H_α. Assume such a shortcut could be taken by a shortcut curve starting from $Y(0)$. Then, there must be a connector edge which intersects a line from a point on a mirror edge through a projection center. In particular, since the curve has to go through all projection centers, one or more of the following must be true for some $1 \le i \le n$: (i) there exists a line through p_1^i intersecting a mirror edge e_*^{i-1} and a connector edge of $\bar{\mathsf{e}}_1^i$, or (ii) there exists a line through p_3^i intersecting a mirror edge e_2^i or $\bar{\mathsf{e}}_2^i$ and a connector edge of e_3^i. However, this was prevented by the careful placement of these connector edges.

It remains to prove that the shortcut curve cannot visit both e_j^i and $\bar{\mathsf{e}}_j^i$ for any i and j, and therefore visits at most one mirror edge between two projection centers. First of all, the shortcut curve has to lie inside or on the boundary of the hippodrome and is 4-monotone (Lemma 5.4.7). At the same time, we constructed the gadget such that the mirror edges between two consecutive projection centers have distance at least β by Lemma 5.4.9 and that the left mirror edge comes after the right mirror edge along their order of Y. Since we chose $\beta > 4$, the shortcut curve cannot visit both mirror edges. □

Putting the above lemmas together implies the correctness of the reduction for shortcut curves that are one-touch, i.e., which visit every edge in at most one point.

Lemma 5.4.11 *If $\alpha \in [1/2, 1)$, $\zeta > 4$ and if $\beta > 4$, then for any feasible one-touch shortcut curve $Y_\diamondsuit$, it holds that the subset encoded by $Y_\diamondsuit$ sums to σ. Furthermore, for any subset of s that sums to σ, there exists a feasible one-touch shortcut curve that encodes it.*

Proof: Lemma 5.4.10 and Lemma 5.4.8 imply that $Y_\Diamond$ must be a one-touch encoding as defined in Definition 5.4.2 if it is feasible. By Lemma 5.4.4, the second last vertex of $Y_\Diamond$ is the point on the edge e_*^n, which is in distance $\delta(\sigma_\Diamond + 1)$ to b_*^n, where $\sigma_\Diamond$ is the sum encoded by the subset selected by $Y_\Diamond$. The last vertex of $Y_\Diamond$ is equal to $Y(1)$, which we placed in distance $\delta(\sigma + 1)$ to the projection of b_*^n through p_1^{n+1}. Thus, if and ony if $\sigma_\Diamond = \sigma$, then the last shortcut of $Y_\Diamond$ passes through the last projection center of the target curve. It follows that if $\sigma_\Diamond \neq \sigma$, then $Y_\Diamond$ cannot be feasible. For the second part of the claim, we construct a one-touch encoding as defined in Definition 5.4.2. By the above analysis, it will be feasible if the subset sums to σ, since the curve visits every edge of Y in at most one point and in between uses shortcuts which pass through every buffer zone from left to right and via the buffer zone's projection center. □

5.4.2.2 Correctness for general shortcut curves

Next, we generalize Lemma 5.4.4 to bound the incremental approximation error of the visiting positions on the last edge of each gadget for general shortcut curves, that is, assuming shortcut curves are not necessarily one-touch.

Lemma 5.4.12 *Choose $\alpha \in [1/2, 1)$, $\zeta > 4$ and $\beta > 4$. Given a feasible shortcut curve $Y_\Diamond$, let v_*^i be any point of $Y_\Diamond$ on e_*^i and let v_*^i denote its x-coordinate. For any $0 \leq i \leq n$ let σ_i denote the ith partial sum of the subset encoded by $Y_\Diamond$. If we choose $\delta > \varepsilon_i$, then it holds that*

$$b_*^i + \delta_i - \varepsilon_i \leq v_*^i \leq b_*^i + \delta_i + \varepsilon_i$$

where $\delta_i = \delta(\sigma_i + 1)$ and $\varepsilon_i = \frac{8i+4}{\alpha}$.

Proof: We prove the claim by induction on i. For $i = 0$ the claim follows by the construction of the initialization gadget. Indeed, the curve $Y_\Diamond$ has to start at $Y(0) = \mathsf{a}_0^0$ and by Lemma 5.4.8 it has to pass through both p_1^0 and p_1^1. Since the three points $\mathsf{a}_0^0, \mathsf{p}_1^0$, and p_1^1 do not lie on a common line, there must be an edge of the base curve in between the two projection centers visited by $Y_\Diamond$. By Lemma 5.4.7, the shortcut curve cannot leave the hippodrome. However, the only edge available in the hippodrome is e_*^0. By construction, the only possible shortcut to this edge ends at the center of the edge in distance δ to b_*^0. Since the mirror edge runs leftwards, the only other points that can be visited by $Y_\Diamond$ lie therefore in this direction. However, by Lemma 5.4.7, $Y_\Diamond$ is rightwards 4-monotone. It follows that

$$b_*^0 + \delta - 4 \leq v_*^0 \leq b_*^i + \delta.$$

Since $\varepsilon_0 = 4/\alpha > 4$ and $\sigma_0 = 0$, this implies the claim for $i = 0$.

For $i > 0$, the curve $Y_\Diamond$ entering gadget g_i from the edge e_*^{i-1} has to pass through the first buffer zone via the projection center p_1^i. By induction,

$$b_*^{i-1} + \delta_{\min} \ \leq \ v_*^{i-1} \ \leq \ b_*^{i-1} + \delta_{\max},$$

where $\delta_{\min} = \delta_{i-1} - \varepsilon_{i-1}$ and $\delta_{\max} = \delta_{i-1} + \varepsilon_{i-1}$ denote the minimal and maximal distances of the visiting position to the left endpoint on the edge e_*^{i-1}. Since $\delta > \varepsilon_i = \varepsilon_{i-1} + 8/\alpha$, and $\sigma_{i-1} \geq 0$, it follows that

$$\delta_{\min} = \delta(\sigma_{i-1} + 1) - \varepsilon_{i-1} \geq \delta - \varepsilon_{i-1} > 8/\alpha. \tag{5.4}$$

Furthermore, by Corollary 5.4.5,

$$\delta_{\max} \leq \delta\left(\sum_{1\leq j\leq i-1} s_j + 1\right) + \varepsilon_{i-1} \leq (\lambda_i - \delta) + \varepsilon_{i-1} \leq \lambda_i - 8/\alpha, \tag{5.5}$$

where $\lambda_i = a_*^{i-1} - b_*^{i-1}$ is the length of edge e_*^{i-1}. Thus, v_*^{i-1} lies at distance at least $8/\alpha$ from each endpoint of e_*^{i-1}. Therefore, the only two edges of the base curve which intersect the line $\overline{\mathsf{v}_*^{i-1}\mathsf{p}_1^i}$ within the hippodrome are e_1^i and $\bar{\mathsf{e}}_1^i$. Note that also the vertical connector edges at $\bar{\mathsf{e}}_1^i$ do not intersect any such line within the hippodrome.

Now, there are two cases, either the shortcut ends on e_1^i or on $\bar{\mathsf{e}}_1^i$. Assume the latter case. By Observation 5.4.1 the x-coordinate of the endpoint of the shortcut lies in the interval

$$\left[c_1^i - \alpha\delta_{\max}\ ,\ c_1^i - \alpha\delta_{\min}\right].$$

By the same observation, the length of the edge $\bar{\mathsf{e}}_1^i$ is equal to $\alpha\lambda_i$. Thus, the endpoint of the shortcut lies inside the edge.

We now argue that the shortcut curve has to leave the edge by using a shortcut, i.e., the shortcut curve cannot "walk" out of the edge by using a subcurve of Y. The mirror edges are oriented leftwards. Since the shortcut curve has to be rightwards 4-monotone (Lemma 5.4.7), it can only walk by a distance 4 on each such edge. Let $\mathcal{I}_j^i$ denote the range of x-coordinates of $Y_\diamondsuit$ on $\bar{\mathsf{e}}_j^i$. By the above,

$$\mathcal{I}_1^i \subseteq \left[c_1^i - \alpha\delta_{\max} - 4\ ,\ c_1^i - \alpha\delta_{\min}\right]$$

Thus, by Eq. (5.5) and Eq. (5.4) and since $c_1^i - d_1^i = \alpha\lambda_i$, it holds that $\mathcal{I}_1^i \subseteq [d_1^i, c_1^i]$, i.e., the shortcut curve must leave the edge $\bar{\mathsf{e}}_1^i$ by using a shortcut. A shortcut to e_1^i would violate the order along the base curve Y. Since the shortcut curve is rightwards 4-monotone (Lemma 5.4.7) and must pass a through a buffer zone via its projection center (Lemma 5.4.8), the only way to leave the edge is to take a shortcut through p_2^i. The only edge intersecting a line through p_2^i and a point on $\bar{\mathsf{e}}_1^i$ is $\bar{\mathsf{e}}_2^i$. Thus, $\bar{\mathsf{e}}_2^i$ must be the next edge visited. Now we can again use Observation 5.4.1 to project the set of visiting points onto the next edge and use the fact that the shortcut curve can only walk rightwards and only by a distance at most 4 on a mirror edge to derive that

$$\mathcal{I}_2^i \subseteq \left[d_2^i + \delta_{\min} - 4\ ,\ d_2^i + (\delta_{\max} + 4/\alpha)\right] \subseteq [d_2^i, c_2^i].$$

By repeated application of the above arguments, we obtain that $\bar{\mathsf{e}}_3^i$ is visited within

$$\mathcal{I}_3^i \subseteq \left[c_3^i - (\delta_{\max} + 4/\alpha) - 4\ ,\ c_3^i - (\delta_{\min} - 4)\right] \subseteq [d_3^i, c_3^i],$$

and that $\bar{\mathsf{e}}_*^i$ is visited within

$$\mathcal{I}_*^i \subseteq \left[d_4^i + (\delta_{\min} - 4) - 4\ ,\ d_4^i + (\delta_{\max} + 4/\alpha + 4)\right] \subseteq [d_4^i, c_4^i] \subseteq [b_*^i, a_*^i]$$

For each visited edge, it follows by Eq. (5.5) and Eq. (5.4) that the shortcut curve visits the edge in the interior.

Now, since the shortcut curve did not visit e_1^i, the input value s_i is not included in the selected subset, therefore $\delta_i = \delta_{i-1}$. Using the interval $\mathcal{I}_*^i$ derived above, and the fact that $d_4^i = b_*^i$ (Lemma 5.4.3) it follows that

$$v_*^i \;\geq\; d_4^i + \delta_{\min} - 8 \;\geq\; d_4^i + (\delta_{i-1} - \varepsilon_{i-1}) - 8 \geq b_*^i + \delta_i - \varepsilon_i,$$

and similarly,

$$v_*^i \leq d_4^i + \delta_{\max} + 8/\alpha \leq d_4^i + (\delta_{i-1} + \varepsilon_{i-1}) + 8/\alpha \leq b_*^i + \delta_i + \varepsilon_i$$

Thus, the claim follows in the case that $Y_\Diamond$ visits $\overline{\mathsf{e}}_1^i$.

The case that $Y_\Diamond$ visits e_1^i can be proven along the same lines. However, now s_i is included in the selected subset, and therefore $\delta_i = \delta_{i-1} + \delta s_i$. Using the arguments above we can derive

$$\mathcal{I}_*^i \subseteq \Big[b_4^i + \delta_{\min} - 8/\alpha \;,\; b_4^i + \delta_{\max} + 8/\alpha \Big]$$

By Lemma 5.4.3, $b_4^i = b_*^i + \delta s_i$. (Note that the same argument was used in Lemma 5.4.4). Thus, analogous to the above

$$v_*^i \geq b_4^i + \delta_{i-1} - \varepsilon_i \geq b_*^i + \delta s_i + \delta_{i-1} - \varepsilon_i \geq b_*^i + \delta_i - \varepsilon_i$$

and similarly,

$$v_*^i \leq b_4^i + \delta_{i-1} + \varepsilon_i \leq b_*^i + \delta s_i + \delta_{i-1} + \varepsilon_i \leq b_*^i + \delta_i + \varepsilon_i$$

Therefore the claim is implied also in this case. □

Lemma 5.4.13 *If we choose* $\alpha \in [1/2, 1), \beta > 4, \zeta > 4,$ *and* $\delta \geq 25$, *then any feasible shortcut curve* $Y_\Diamond$ *encodes a subset* $S' \subseteq S$ *that sums to* σ.

Proof: Since $Y_\Diamond$ is feasible, it must be that it visits e_*^n at distance $\delta(\sigma+1)$ to b_*^n, since this is the only point to connect via a shortcut through the last projection center to the endpoint of Y and by Lemma 5.4.8 all projection centers have to be visited. So let $v_*^n = b_*^n + \delta(\sigma+1)$ be the x-coordinate of this visiting point (the starting point of the last shortcut), and let σ_n be the sum of the subset encoded by $Y_\Diamond$. Since we chose $\delta > \varepsilon_n$, Lemma 5.4.12 implies that

$$b_*^n + \delta(\sigma_n + 1) - \varepsilon_n \leq b_*^n + \delta(\sigma + 1) \leq b_*^n + \delta(\sigma_n + 1) + \varepsilon_n.$$

Therefore,

$$\sigma_n - \varepsilon_n/\delta \leq \sigma \leq \sigma_n + \varepsilon_n/\delta$$

For our choice of parameters $\varepsilon_n/\delta < 1$. Therefore, it must be that $\sigma = \sigma_n$, since both values are integers.

□

5.4.2.3 The result

Given a shortcut curve and target curve, one can compute their Fréchet distance in polynomial time using the algorithm by Alt and Godau (see Section 2.4). Thus, the problem of deciding if the shortcut Fréchet distance between two given polygonal curves is smaller or equal a given value is in NP. Each of the constructed gadgets has constant size and the sizes of coordinates are polynomial, and thus the overall construction is of polynomial size. Now, together with Lemma 5.4.11 and the fact that the reduction is polynomial, Lemma 5.4.13 implies the NP-completeness of the problem.

Theorem 5.4.14 *The problem of deciding whether the shortcut Fréchet distance between two given curves is less or equal a given distance is NP-complete.*

5.4.3 Algorithmic results

We describe an approximate decision algorithm for the directed continuous shortcut Fréchet distance. Given a value of δ and two polygonal curves X and Y in $\mathbb{R}^2$ of total complexity $n = n_1 + n_2$, the algorithm outputs either (i) "$d_{\mathcal{S}}(X, Y) \leq 3\delta$", or (ii) "$d_{\mathcal{S}}(X, Y) > \delta$". The algorithm runs in $O(n^3 \log n)$ time and $O(n)$ space regardless if the curves are c-packed. We first describe the main idea of the algorithm in Section 5.4.3.1. The algorithmic layout is based on the algorithms described in Section 5.3.2.2 and Section 5.3.5.2. As the decision algorithms in the previous sections, the algorithm invokes a subroutine to test for the reachability via tunnels. Here, we split the subroutine into two procedures **verticalTunnel** and **diagonalTunnel**, which we describe in Section 5.4.3.2 and Section 5.4.3.3. Furthermore, the range queries from Section 5.3.5.2 are handled by storing the extremal points within the columns in an array, instead of a range tree. The resulting decision algorithm is described in Section 5.4.3.4 and analyzed in Section 5.4.3.5.

5.4.3.1 Challenge and ideas

The standard way to solve the decision problem for the Fréchet distance and its variants is to search for monotone paths in the free space diagram, see Section 2.1. In the case of the shortcut Fréchet distance, this path can now use shortcuts on Y, if the matched subcurve of X is within Fréchet distance. To formalize this, we introduced the concept of tunnels in Section 5.2.2. In the general version of the shortcut Fréchet distance, the tunnels can now start and end anywhere inside the free space cells, while in the vertex-restricted case they are constrained to the grid of the parametric space. In order to extend the algorithms from the previous sections to this case, we need a new method to compute the space which is reachable within a free space cell. For this we use the known concept of line-stabbing. Guibas et al. [84] study the problem of stabbing an ordered set of unit radius disks with a line. In particular, one of the problems studied is the following. Given a series of disks $D_1, \ldots, D_n$, of the same radius does there exist a directed line ℓ, with n points p_i that lie along ℓ in the order of i, such that $\mathsf{p}_i \in D_i$? It is known that this relates to the Fréchet distance as follows: an *ordered line stabbing* of disks of radius δ is equivalent to a line segment with Fréchet distance δ to the polygonal curves through the centers of the disks. The

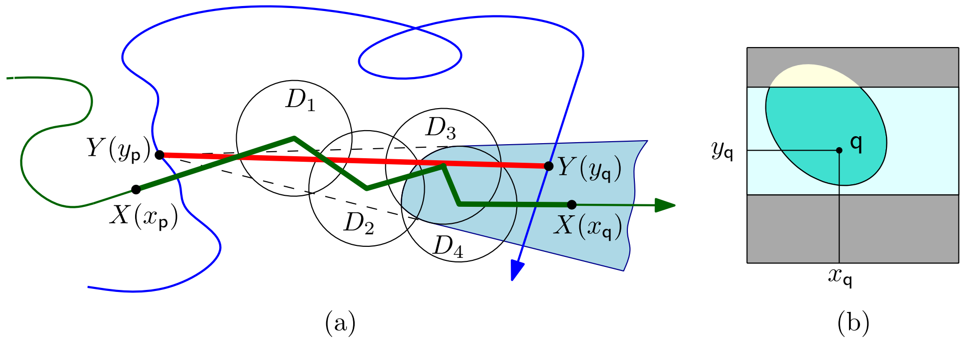


Figure 5.9: (a) Example of a tunnel $\tau(\mathsf{p}, \mathsf{q})$ computed by the **diagonalTunnel** procedure. The shaded area shows the line-stabbing wedge. (b) The free space cell which contains the endpoint of the tunnel.

algorithm described by Guibas *et al.* maintains a so called *line-stabbing wedge* that contains all points b, such that there is a line through b that visits the first i disks before visiting b. We will use this algorithm to compute all tunnels $\tau(\mathsf{p}, \mathsf{q})$ of price at most δ starting from a particular point p in the parametric space and ending in a particular cell. By intersecting the line-stabbing wedge with the edge of Y that corresponds to the cell, we obtain a horizontal strip which represents the set of such tunnel endpoints q. See Figure 5.9 for an illustration.

The second challenge is that the reachable free space can fragment into exponentially many of such horizontal strips. However, we can exploit the monotonicity of the tunnel prices to approximate the reachable free space as done in the previous algorithms. In this approximation scheme, the combinatorial complexity of the reachable space is constant per cell.

5.4.3.2 The diagonalTunnel procedure

This procedure receives as input a cell $C_{i,i}$ and a point $\mathsf{p} \in \mathcal{D}_{\leq\delta}$, such that p lies in the lower left quadrant of the lower left corner of the cell $C_{i,i}$ and a parameter $\delta > 0$. The output will be a set of points $\mathsf{P} \subseteq C_{i,i}$, such that $\mathrm{prc}(\tau(\mathsf{p}, \mathsf{q})) \leq \delta$ if and only if $\mathsf{q} \in \mathsf{P}$. We use the line-stabbing algorithm mentioned above with minor modifications. Let $x_\mathsf{p} = x_-$ be the x-coordinate of p and let x_+ be the x-coordinate of some point in the interior of $C_{i,i}$. Let $D_1, \ldots, D_k$ be the disks of radius δ centered at the vertices of X that are spanned by the subcurve $X[x_-, x_+]$. There are two cases, either $Y(y_\mathsf{p})$ is contained in D_i for all $1 \leq i \leq k$, or there exists some i, such that $Y(y_\mathsf{p})$ lies outside of D_i. In the first case, we return $\mathsf{P} = C_{i,i} \cap \mathcal{D}_{\leq\delta}$. In the second case we initialize the line-stabbing wedge of [84] with the tangents of $Y(y_\mathsf{p})$ to the disk D_i, where i is the smallest index such that $Y(y_\mathsf{p}) \notin D_i$. We then proceed with the algorithm as written by handling the disks $D_{i+1}, \ldots, D_k$. Finally, we intersect the line-stabbing wedge with the edge of Y that corresponds to $C_{i,i}$. Refer to Figure 5.9 for an illustration. This yields a horizontal slab of points that lie in $C_{i,i}$ which we then intersect with the δ-free space and return as our set P.

Algorithm 5.4.15 Decider(X, Y, δ)

```
Input: polygonal curves $X$ and $Y$; distance $\delta \in \mathbb{R}$
 1: Assert that $\mathrm{dist}(0,0) = \|X(0) - Y(0)\| \le \delta$ and $\mathrm{dist}(1,1) \le \delta$
 2: Let $\mathcal{A}$, $\overline{\mathcal{A}}$, $g^\ell$, $\overline{g}^\ell$, $g^r$, and $\overline{g}^r$ be arrays of size $n_1$
 3: for $j = 1, \dots, n_2$ do
 4:    Update $\overline{\mathcal{A}} \leftarrow \mathcal{A}$, $\overline{g}^\ell \leftarrow g^\ell$, and $\overline{g}^r \leftarrow g^r$
 5:    for $i = 1, \dots, n_1$ do
 6:       if $i = 1$ and $j = 1$ then
 7:          Let $\mathsf{P}_{i,j} = \mathcal{D}_{\le\delta}^{(i,j)}$
 8:       else
 9:          Retrieve $R^v_{i-1,j}$ and $R^h_{i,j-1}$ from $\mathcal{A}[i-1]$ and $\overline{\mathcal{A}}[i]$
10:          Step 1: Compute $\mathsf{P}^1_{i,j}$ from $R^v_{i-1,j}$ and $R^h_{i,j-1}$
11:          Step 2: Let $\mathsf{P}^2_{i,j} =$ verticalTunnel$(\overline{g}^\ell[i], C_{i,j}, \delta)$.
12:          Step 3: Let $\mathsf{P}^3_{i,j} =$ diagonalTunnel$(\overline{g}^r[i-1], C_{i,j}, 3\delta)$.
13:          Let $\mathsf{P}_{i,j} = Q(\mathsf{P}^1_{i,j} \cup \mathsf{P}^2_{i,j} \cup \mathsf{P}^3_{i,j}) \cap \mathcal{D}_{\le\delta}^{(i,j)}$
14:       if $\mathsf{P}_{i,j} \neq \emptyset$ then
15:          Update $g^\ell[i]$ and $g^r[i]$ using the gates of $\mathsf{P}_{i,j}$
16:          Compute $R^v_{i,j}$ and $R^h_{i,j}$ from $\mathsf{P}_{i,j}$ and store them in $\mathcal{A}[i]$
17:       else
18:          Update $g^r[i]$ using $g^r[i-1]$
19: if $(1,1) \in \mathcal{A}[n_1]$ then
20:    Return "$d_{\mathcal{S}}(X,Y) \le 3\delta$"
21: else
22:    Return "$d_{\mathcal{S}}(X,Y) > \delta$"
```

5.4.3.3 The verticalTunnel procedure

This procedure receives as input a cell $C_{i,i}$ and a point p which lies below this cell in the same column and a parameter $\delta \ge 0$. Let H_{p} be the closed halfplane which lies to the right of the vertical line through p. The procedure returns the intersection of H_{p} with the δ-free space in $C_{i,i}$.

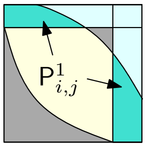
5.4.3.4 The decision algorithm

The algorithm is layed out in Algorithm 5.4.15. We traverse the free space diagram as done in Section 5.3.2.2 and approximate the reachable free space as done in Section 5.3.5.2. That is, we traverse the free space diagram in a row-by-row order from bottom to top and from left to right. For every cell, we compute a set of reachable points $\mathsf{P}_{i,j} \subseteq C_{i,j}$, such that

$$\mathcal{R}_{\le\delta}^{(i,j)}(X,Y) \subseteq \mathsf{P}_{i,j} \subseteq \mathcal{R}_{\le 3\delta}^{(i,j)}(X,Y)\,.$$

Thus, the set of computed points approximates the reachable free space. From $\mathsf{P}_{i,j}$ we compute reachability intervals $R^v_{i,j}$ and $R^h_{i,j}$, which we define as the intersections of $\mathsf{P}_{i,j}$ with the top and right cell boundary. Furthermore we compute the ***gates*** of $\mathsf{P}_{i,j}$, which we now define as the two points of the set with minimum and maximum

x-coordinates. (Recall that in Section 5.3, where tunnels were confined to the horizontal edges of the grid, gates were defined as the extremal points of the reachability intervals.) We keep this information for the cells in the current and previous row in one-dimensional arrays by the index i. We use three arrays $\mathcal{A}$, g^ℓ and g^r to write the information of the current row and three arrays $\overline{\mathcal{A}}$, $\overline{g}^\ell$ and $\overline{g}^r$ to store the information from the previous row. Here, $\mathcal{A}$ and $\overline{\mathcal{A}}$ are used to store the reachability intervals, and g^ℓ, $\overline{g}^\ell$, $\overline{g}^r$ g^r are used to store extremal points (i.e., gates) of the computed reachable space. In particular, $g^\ell[i]$ and $\overline{g}^\ell[i]$ store the leftmost reachable point (i.e., gate) discovered so far that lies inside column i and in $g^r[i]$ and $\overline{g}^r[i]$ we maintain the rightmost reachable point (i.e., gate) discovered so far that lies to the left of column $i+1$. During the traversal, we can update this information in constant time per cell using the gates of $\mathsf{P}_{i,j}$, and the gates stored in $g^r[i-1]$, $g^r[i]$ and $g^\ell[i]$. (Recall that in Section 5.3.5.2 the gates were maintained in a search tree to enable queries for the leftmost and rightmost gates. The same approach would work here, however it would require superlinear space since the number of gates might be quadratic in the worst case.)



We handle a cell $C_{i,j}$ in three steps. We first compute the set of points $\mathsf{P}^1_{i,j}$ in this cell that are reachable by a monotone path via $R^v_{i,j-1}$ or $R^h_{i-1,j}$. Since these reachability intervals have been computed in previous steps, they can be retrieved from $\mathcal{A}[i-1]$ and $\overline{\mathcal{A}}[i]$. More specifically, to compute $\mathsf{P}^1_{i,j}$, we take the closed halfplane above the horizontal line at the lower endpoint of $R^v_{i,j-1}$ and intersect it with the δ-free space inside the cell, which we can compute ad-hoc from the two corresponding edges. Similarly, we take the closed halfplane to the right of the left endpoint of $R^h_{i-1,j}$ and intersect it with the δ-free space. The union of those two sets is $\mathsf{P}^1_{i,j}$. See the figure to the right for an example. In a second step, we compute the set of points $\mathsf{P}^2_{i,j}$ in $C_{i,j}$ that are reachable by a vertical tunnel from below. For this, we retrieve the leftmost reachable point in the current column by probing $\overline{g}^\ell[i]$. Assume there exists such a point and denote it by p_2. We invoke **verticalTunnel** $(\mathsf{p}_2, C_{i,j}, \delta)$ and let $\mathsf{P}^2_{i,j}$ be the output of this procedure. In the third step, we compute the set of points $\mathsf{P}^3_{i,j}$ in $C_{i,j}$ that are reachable by a diagonal tunnel. For this, we retrieve the rightmost reachable point in the cells that are spanned by the lower left quadrant of the lower left corner of $C_{i,j}$. This point is stored in $\overline{g}^r[i-1]$. Let this point be p_3, if it exists. We invoke **diagonalTunnel** $(\mathsf{p}_3, C_{i,j}, 3\delta)$ and let $\mathsf{P}^3_{i,j}$ be the output of this procedure. Now, we compute

$$\mathsf{P}_{i,j} = Q\big(\mathsf{P}^1_{i,j} \cup \mathsf{P}^2_{i,j} \cup \mathsf{P}^3_{i,j}\big) \cap \mathcal{D}^{(i,j)}_{\leq\delta},$$

where $Q(\mathsf{P})$ is defined as the union of the upper right quadrants of the points of P. We store the intersection of $\mathsf{P}_{i,j}$ with the top and right side of the cell in $\mathcal{A}[i]$ and update the gates stored in $g^r[i]$ and $g^\ell[i]$. After handling the last cell, we can check if the upper right corner of the parametric space is reachable by probing $\mathcal{A}[n_1]$ and output the corresponding answer.

Computation of the gates. The gates of $\mathsf{P}_{i,j}$ can be computed in constant time. A gate of this set either lies on the grid of the parametric space, or it may be internal to the free space cell. The endpoints of a free space interval can be computed using the intersection of the corresponding edge and a disk of radius δ centered at the

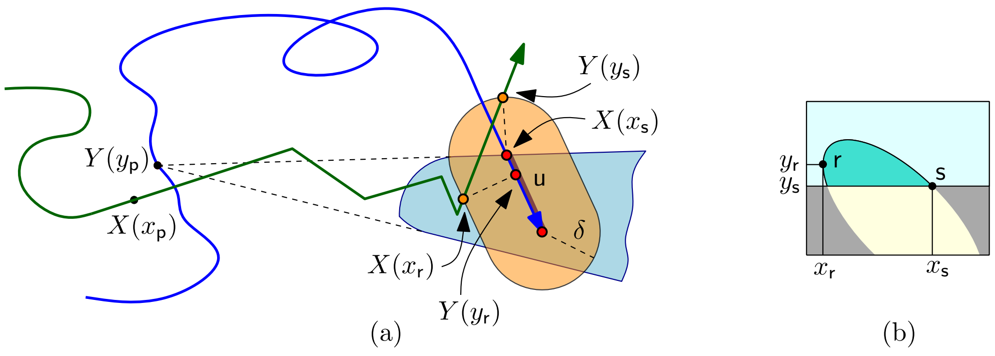


Figure 5.10: Computation of the gates $\mathsf{r} = (x_\mathsf{r}, y_\mathsf{r})$ and $\mathsf{s} = (x_\mathsf{s}, y_\mathsf{s})$ of $\mathsf{P}^3_{i,j}$.

corresponding vertex. Internal gates of the free space can be computed in a similar way. One can use the Minkowski sum of the edge of Y with a disk of radius δ. The intersection points of the resulting hippodrome with the edge of X correspond to the x-coordinates of the gates, while we can obtain the y-coordinates by projecting the intersection point back onto the edge of Y. A gate might also be the intersection point of a horizontal line with the free space as computed in Step 1 and Step 3 of the decision algorithm. Consider the **diagonalTunnel** procedure which we use to compute $\mathsf{P}^3_{i,j}$. The procedure computes a portion u of the edge e_j of Y by intersection with the line-stabbing wedge. In order to obtain the extremal points of the returned set in parametric space, we can take the Minkowski sum of u with a disk of radius δ and intersect the resulting hippodrome with the edge e_i of X. See Figure 5.10 for an illustration. We can a similar method for $\mathsf{P}^1_{i,j}$. The actual gates of $\mathsf{P}_{i,j}$ can then be computed using a simple case distinction.

5.4.3.5 Analysis

We now analyze the correctness and running time of the algorithm described above.

Lemma 5.4.16 *Given a cell $C_{i,i}$, a point $\mathsf{p} \in \mathcal{D}_{\leq\delta}$ and a parameter $\delta \geq 0$, the* **diagonalTunnel** *procedure described in Section 5.4.3.2 returns a set of points $\mathsf{P} \subseteq C_{i,i}$, such that for any $\mathsf{q} \in C_{i,i}$, it holds that $\mathrm{prc}(\tau(\mathsf{p},\mathsf{q})) \leq \delta$ if and only if $\mathsf{q} \in \mathsf{P}$.*

Proof: The correctness of the procedure follows from the correctness of the line-stabbing algorithm as analyzed in [84]. Recall that we intersect the line-stabbing wedge of $Y(y_\mathsf{p})$ and the disks $D_1, \ldots, D_k$ with the edge of Y that corresponds to $C_{i,i}$ to retrieve the horizontal slab in $C_{i,i}$ that defines P. Refer to Figure 5.9 for an illustration. It follows that any directed line segment $\overline{Y(y_\mathsf{p})Y(y_\mathsf{q})}$, where y_q is the y-coordinate of a point $\mathsf{q} \in \mathsf{P}$, contains points p_i for $1 \leq i \leq k$ in the order of i along the segment, such that $\mathsf{p}_i \in D_i$. (For the case that $Y(y_\mathsf{p})$ is contained in each of the disks $D_1, \ldots, D_k$, any line through $Y(y_\mathsf{p})$ stabs the disks in any order, by choosing $\mathsf{p}_i = Y(y_\mathsf{p})$ for all $1 \leq i \leq k$.) Thus, we can match the shortcut $\overline{Y}[y_\mathsf{p}, y_\mathsf{q}]$ to the subcurve $X[x_\mathsf{p}, x_\mathsf{q}]$ within Fréchet distance δ as follows. For any two inner vertices

$\mathsf{v}_i, \mathsf{v}_{i+1}$ of $X[x_{\mathsf{p}}, x_{\mathsf{q}}]$, we can match the edge connecting them to the line segment $\overline{\mathsf{p}_i\mathsf{p}_{i+1}}$ by Observation 2.3.1. For the first segment, note that we required $\mathsf{p} \in \mathcal{D}_{\leq\delta}$. For the last segment, we ensured that $\mathsf{P} \subseteq \mathcal{D}_{\leq\delta}$ by construction. Thus, also here we can apply Observation 2.3.1. As for the other direction, let $\mathsf{q} \in C_{i,i}$, such that $\operatorname{prc}(\tau(\mathsf{p},\mathsf{q})) \leq \delta$. It must be, that the line segment from $Y(y_{\mathsf{p}})$ to $Y(y_{\mathsf{q}})$ stabs the disks $D_1, \ldots, D_k$ in the correct order. Thus, $Y(y_{\mathsf{q}})$ would be included in the computed line-stabbing wedge and subsequently, q would be included in P. □

Lemma 5.4.17 *For two polygonal curves X and Y in $\mathbb{R}^2$ of total complexity n, the* **diagonalTunnel** *procedure described in Section 5.4.3.2 takes $O(n \log n)$ time and $O(n)$ space.*

Proof: Our modification of the line-stabbing algorithm does not increase the running time and space requirements of the algorithm, which is $O(k \log k)$ with k being the number of disks handled. Intersecting the line-stabbing wedge with a line segment can be done in time $O(\log k)$, since the complexity of the wedge is $O(k)$. Thus, the claim follows directly from the analysis of the line-stabbing algorithm in [84] and by the fact that the algorithm handles at most n disks. □

Lemma 5.4.18 *For any $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$, let $\mathsf{P}^3_{i,j}$ be the set computed in Step 3 of the decision algorithm layed out in Algorithm 5.4.15 and let $R = \bigcup_{k=1}^{i-1} \bigcup_{\ell=1}^{i-1} \mathsf{P}_{k,\ell}$, i.e., the reachable points computed in the lower left quadrant of the cell. It holds that:*

(i) There exists a point $\mathsf{p} \in R$, such that for any $\mathsf{q} \in C_{i,j}$, the diagonal tunnel $\tau(\mathsf{p},\mathsf{q})$ has price $\operatorname{prc}(\tau(\mathsf{p},\mathsf{q})) \leq 3\delta$ if and only if $\mathsf{q} \in \mathsf{P}^3_{i,j}$.

(ii) There exists no other point $\mathsf{b} \in C_{i,j} \setminus \mathsf{P}^3_{i,j}$ that is the endpoint of a diagonal tunnel from R with price at most δ.

Proof: The lemma follows from the monotonicity of the tunnel prices, which is testified by Lemma 5.2.1 and from the correctness of the **diagonalTunnel** procedure (Lemma 5.4.16). Note that the algorithm computes the gates of R within every cell. Furthermore, the gates are maintained in the arrays $\overline{g}^r$ and g^r, such that, when handling the cell $C_{i,j}$, we can retrieve the rightmost gate in the lower left quadrant of the lower left corner of $C_{i,j}$ from $\overline{g}^r[i-1]$. (This can be easily shown by induction on the cells in the order in which they are handled.) Let p be the point stored in $\overline{g}^r[i-1]$. Part (i) of the claim follows from Lemma 5.4.16, since **diagonalTunnel** is called with the parameter p to obtain $\mathsf{P}^3_{i,j}$. Part (ii) of the claim follows from Lemma 5.2.1, since p is the rightmost point in R that could serve as a starting point for a diagonal tunnel ending in $C_{i,j}$. Indeed, assume that there would exist such points $\mathsf{b} \in C_{i,j} \setminus \mathsf{P}^3_{i,j}$ and $\mathsf{c} \in R$ with tunnel price $\operatorname{prc}(\tau(\mathsf{c},\mathsf{b})) \leq \delta$. It must be that b lies to the left of p, since p was the rightmost possible gate. By (i), $\operatorname{prc}(\tau(\mathsf{p},\mathsf{b})) > 3\delta$ and therefore Lemma 5.2.1 implies that $\operatorname{prc}(\tau(\mathsf{c},\mathsf{b})) > \delta$, a contradiction. □

Lemma 5.4.19 *For any $1 \leq i \leq n_1$ and $1 \leq j \leq n_2$, let $\mathsf{P}^2_{i,j}$ be the set computed in Step 2 of the decision algorithm layed out in Algorithm 5.4.15. and let $R = \bigcup_{\ell=1}^{j-1} \mathsf{P}_{i,\ell}$, i.e., the reachable points computed in column i below the cell. For any $\mathsf{q} \in C_{i,j}$, the vertical tunnel $\tau(\mathsf{p},\mathsf{q})$ has price $\operatorname{prc}(\tau(\mathsf{p},\mathsf{q})) \leq \delta$ for some $\mathsf{p} \in R$ if and only if $\mathsf{q} \in \mathsf{P}^2_{i,j}$.*

Proof: Note that vertical tunnels are always affordable if they are feasible by Observation 2.3.1. As in the proof of Lemma 5.4.18, we note that the algorithm computes the gates of R within every cell. Furthermore, gates are maintained in the arrays $\overline{g}^{\ell}$ and g^{ℓ}, such that, when handling the cell $C_{i,j}$, we can retrieve the leftmost gate below $C_{i,j}$ in the same column from $\overline{g}^{\ell}[i]$. (Again, this can be easily shown by induction on the cells in the order in which they are handled.) Let p be the point stored in $\overline{g}^{\ell}[i]$ when handling the cell $C_{i,j}$. Since $\mathsf{P}^2_{i,j}$ is computed by calling **verticalTunnel** on p, the claim follows. □

Lemma 5.4.20 *The output of* **Decider** *(Algorithm 5.4.15) is correct.*

Proof: The proof goes by induction on the order of the handled cells. We claim that for any point $\mathsf{q} \in C_{i,j}$ it holds that (a) if $\mathsf{q} \in \mathsf{P}_{i,j}$, then $\mathsf{q} \in \mathcal{R}_{\leq 3\delta}$, and (b) if $\mathsf{q} \in \mathcal{R}_{\leq \delta}$ then $\mathsf{q} \in \mathsf{P}_{i,j}$. For the first cell $C_{1,1}$, this is clearly true. Indeed, a shortcut from $Y(0)$ to any point on the first edge of Y, results in a shortcut curve that has Fréchet distance zero to Y. By the convexity of the free space in a single cell, it follows that $\mathcal{R}^{(1,1)}_{\leq\delta} = \mathcal{D}^{(1,1)}_{\leq\delta} = \mathsf{P}_{1,1} \subseteq \mathcal{R}^{(1,1)}_{\leq 3\delta}$ given that $(0,0) \in \mathcal{D}_{\leq\delta}$.

Now, consider a cell $C_{i,j}$ that is handled by the algorithm. We argue that part (a) of the induction hypothesis holds. It must be that either (i) $\mathsf{q} \in \mathsf{P}^1_{i,j}$, (ii) $\mathsf{q} \in \mathsf{P}^2_{i,j}$, (iii) $\mathsf{q} \in \mathsf{P}^3_{i,j}$, or (iv) q is in the upper right quadrant of some point q' in one of $\mathsf{P}^1_{i,j}, \mathsf{P}^2_{i,j}$ or $\mathsf{P}^3_{i,j}$. In cases (i), the claim follows by induction since $\mathsf{P}_{i-1,j}$ and $\mathsf{P}_{i,j-1}$ are computed before $\mathsf{P}_{i,j}$. In case (ii) the claim follows by induction, since the rows are handled from bottom to top and by Lemma 5.4.19. In case (iii) the claim follows by Lemma 5.4.18 and by induction, since the algorithm traverses the free space diagram in a row-by-row manner from bottom to top and in every row from left to right. Now, in case (iv), the claim follows from (i),(ii), or (iii). Indeed, we can always connect q' with q by a straight line segment, and since $\mathcal{D}_{\leq\delta}$ is convex inside any cell, these straight monotone paths are preserved in the intersection with the free space.

It remains to prove part (b). Let $\mathsf{q} \in C_{i,j}$ be the endpoint of a monotone path from $(0,0)$ that stays inside the δ-free space and otherwise uses tunnels of price at most δ. There are three possibilities for π to enter $C_{i,j}$: (i) via the boundary with its direct neighbors, (ii) via a vertical tunnel, or (iii) via a diagonal tunnel. (As for horizontal tunnels, we can always replace such a horizontal tunnel by the corresponding monotone path through the free space.) We can show in each of these cases that q should be included $\mathsf{P}_{i,j}$. In case (i) we can apply the induction hypothesis for $\mathsf{P}_{i-1,j}$ and $\mathsf{P}_{i,j-1}$, in case (ii) we can apply Lemma 5.4.19 and the induction hypothesis for cells below $\mathsf{P}_{i,j}$ in the same column and in case (iii) we can apply Lemma 5.4.18 and the induction hypothesis for cells in the lower left quadrant of the cell. □

Lemma 5.4.21 *Given two polygonal curves X and Y in $\mathbb{R}^2$ of complexity $n = n_1 + n_2$, the algorithm* **Decider** *(Algorithm 5.4.15) takes time in $O(n^3 \log n)$ and space in $O(n)$.*

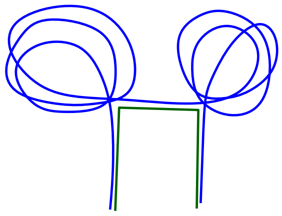
Proof: The algorithm keeps six arrays of length n_1, which store objects of constant complexity. The tunnel procedure takes space in $O(n)$, by Lemma 5.4.17. Thus, overall, the algorithm requires $O(n)$ space. As for the running time, the algorithm handles $O(n^2)$ cells. Each cell is handled in three steps of which the first and second step take constant time each and the third step takes time in $O(n \log n)$ by Lemma 5.4.17. The computation of the gates can be done in constant time per cell. Furthermore, the algorithm takes $O(n)$ time per row to update the arrays. Overall, the running time can be bounded by $O(n^3 \log n)$ time. $\square$

5.4.3.6 The result

Theorem 5.4.22 *Given two curves X and Y of complexity $n = n_1 + n_2$ and a value of δ, the decision algorithm outputs one of the following, either*
(i) $d_{\mathbb{S}}(X, Y) \leq 3\delta$, *or*
(ii) $d_{\mathbb{S}}(X, Y) > \delta$.
In any case, the output is correct. The algorithm runs in $O(n^3 \log n)$ time and $O(n)$ space.

5.5 Concluding remarks

In this chapter, we studied the problem of how to introduce shortcuts on one of two given polygonal curves in order to minimize the Fréchet distance between the two curves. There are many different variants of this problem. We showed that it is possible to compute a $(3+\varepsilon)$-approximation in a running time which is near-linear in the complexity of the input curves if
(i) the number of shortcuts is unbounded (or small),
(ii) shortcuts have to start and end at input vertices, and
(iii) the input curves are c-packed.
More specifically, the algorithm runs in $O(c^2 n \log^3 n)$ time, where n is the total number of vertices of the two input curves. The exact version of the algorithm runs in $O(n^5 \log n)$ time. We extended the approximate decision procedure used by this algorithm to the more general case where shortcuts can start and end anywhere along one of the input curves by combining it with a line-stabbing algorithm. The resulting algorithm runs in $O(n^3 \log n)$ time also for non-packed curves.[4] Extending the decision procedure to the computation problem in the general case is still open (see Problem 5.2 below). For exact computations we showed NP-hardness in this case.



In light of the algorithmic results, the NP-hardness comes as a surprise. Note that the vertex-restricted and the continuous variant of the problem are closely related. One way to compute an approximation to the continuous variant (with additive error) is to sample the input curve along the edges and to use the vertex-restricted exact algorithm on the resulting curves. Thus, we can get arbitrarily close to the correct distance value by using a pseudo-polynomial algorithm. However, it is unclear if this helps in finding an exact solution. Note that there may be many combinatorially different shortcut curves which are close to the target curve under the Fréchet distance, as demonstrated by the example depicted to the right.

[4]For a comparison of the bounds, observe that any polygonal curve is c-packed for $c \in O(n)$. For the exact algorithm we also did not use the c-packedness assumption in the analysis.

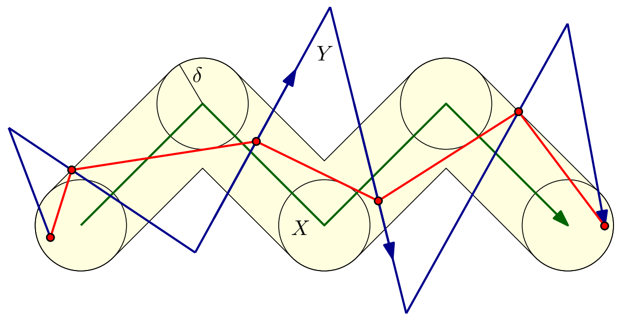


Figure 5.11: Example of a geometric configuration of the curves X and Y that determines their directed continuous shortcut Fréchet distance. The realizing shortcut curve of Y is shown in red.

There are many open questions for further research. The most immediate questions are how to extend our results to other definitions of a shortcut Fréchet distance as mentioned in the introduction, and how to improve the approximation factor. We conclude by discussing a few of these problems in more detail.

Open Problem 5.1 In Section 5.4 we studied the problem of computing the directed continuous shortcut Fréchet distance. We described an approximate decision algorithm which is based on the near-linear-time decision algorithm described in Section 5.3.5 and which runs in $O(n^3 \log n)$ time (see Theorem 5.4.22). The increase in running time stems from the fact that the algorithm computes the line-stabbing wedge for every free space cell. Assuming the input curves are c-packed, and deploying the simplification technique from Chapter 3, one can achieve a better running time. For this, the algorithm first needs to compute all edge-edge pairs that are in distance at most δ from each other. These correspond to the relevant free space cells. If the input curves lie in the plane, then the distance between two edge is either attained at a vertex, or the two edges intersect each other. Since the curves are c-packed, there can be at most $O(c)$ intersections. Thus, one would use Data Structure 5.3.12 to compute all vertex-edge pairs and in addition compute the intersecting edge pairs. In this way, the whole framework of the algorithm described in Section 5.3.5 could be used and lead to a much faster algorithm. At the expense of a higher approximation factor, one might also be able to design a near-linear-time decision algorithm.

Open Problem 5.2 An important open question is how to compute (or even approximate) the directed continuous shortcut Fréchet distance. The standard way to compute the Fréchet distance is to use a decision procedure in a binary search over critical values. As described in Chapter 2, these are determined by local geometric configurations such as the distance between a vertex and an edge. Also for the vertex-restricted shortcut Fréchet distance, there are at most a polynomial number of critical values that need to be considered in the search. The situation becomes more intricate in the general case where shortcuts are not confined to input vertices. In the example depicted in Figure 5.11, the shortcut Fréchet distance coincides with the minimum value of δ such that three tunnels can be connected monotonically along the base curve. The realizing shortcut curve is also shown. For any input size one can construct an example of this type where the critical value depends on the geometric configuration of a linear-size subset of edges. Thus, in order to compute all critical values of this type one would have to consider an exponential number of geometric configurations. A full characterization of the events and algorithms to compute or approximate the critical values is subject for further research. Figure 5.12 shows other relevant examples of events that can determine the continuous shortcut Fréchet distance and which cannot happen in the vertex-restricted case.

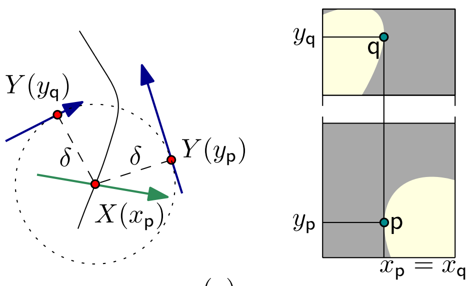
Open Problem 5.3 Another open problem is to determine whether the shortcut Fréchet distance is fixed-parameter tractable in the number of shortcuts k. Our reduction does not readily translate to a reduction from k-SUM: In our construction, shortcuts are taken in each gadget, regardless of whether a value of the SUBSET instance is chosen or not. In a reduction from k-SUM, a shortcut curve encoding a valid solution must use at most k shortcuts. It seems more likely, that the problem is fixed-parameter tractable in k, since there are at most n^k combinatorially different shortcut curves in this case.

Open Problem 5.4 The base curve in our NP-hardness reduction self-intersects and is not c-packed. In fact, it cannot be c-packed for any placement of the connector edges for any constant c. Whether the problem is NP-hard or polynomial time computable for c-packed, non-intersecting, or even monotone curves is currently unclear.

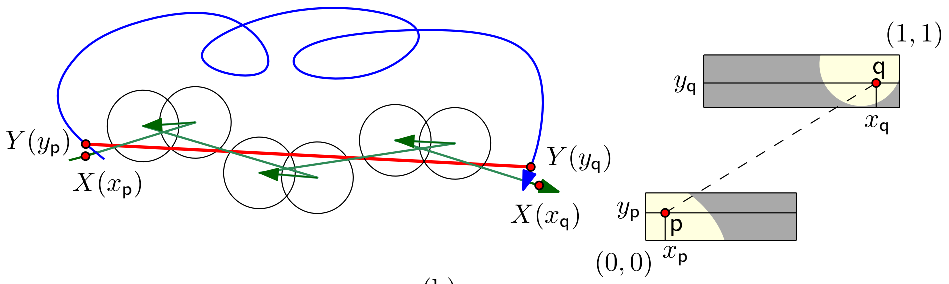
Open Problem 5.5 It seems natural to allow shortcuts on both curves and hence to define an undirected version of the shortcut Fréchet distance. It is an open problem how to define this in a reasonable way and how to compute it. One needs to restrict the set of eligible shortcuts, otherwise the optimization algorithm would simply shortcut both curves from start to end and this does not yield a meaningful distance measure.

A reasonable restriction could be to disallow shortcuts to be matched to each other under the Fréchet distance. Note that for such a definition of the undirected shortcut Fréchet distance the presented NP-hardness proof also applies. Intuitively, shortcuts can only affect the target curve by either shortening or eliminating one or more twists. However, any feasible shortcut curve of the base curve has to pass through the buffer zones corresponding to these twists by using a shortcut. As a result, any shortcut on the target curve has to be matched at least partially to a shortcut of the base curve in order to affect the feasible solutions and this is prevented by definition. Thus, we believe that the problem of computing an undirected shortcut Fréchet distance is also NP-hard for this definition.

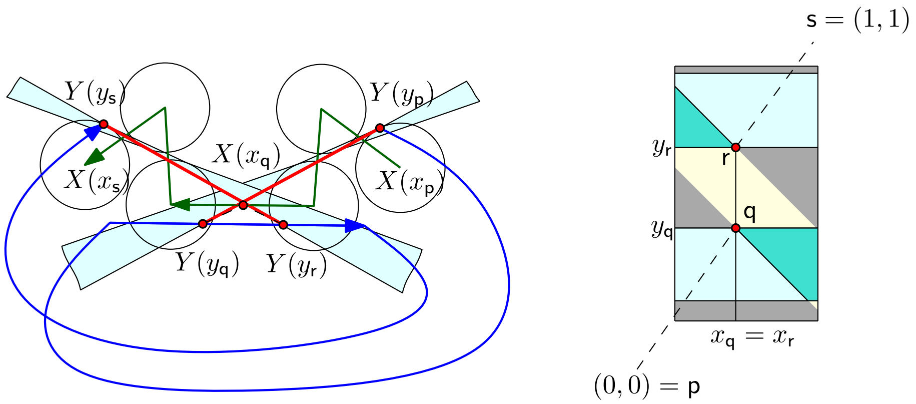
Open Problem 5.6 Another direction of research would be to study the computational complexity of a discrete variant of the shortcut Fréchet distance. The discrete Fréchet distance only considers matchings between the vertices of the curves and can be computed using dynamic programming. In practice, such a discrete shortcut Fréchet distance might approximate the continuous version and it might be easier to compute. Thus it deserves further attention.



(a)



(b)



(c)

Figure 5.12: (a) The common distance of two edges to the intersection of their bisector with another edge; (b) Example of the event that the line stabbing wedge of disks centered at vertices of the target curve becomes non-empty; (c) Example of the event that two tunnels can be connected monotonically along the target curve.

CHAPTER 6

Flow computations on imprecise terrains

In this chapter we study the computation of water flow on terrains in which the elevations are imprecise. We motivate the problem and discuss the basic terminology in Sections 6.1. In Section 6.2 we show that the problem of deciding whether water can flow between two given points on a polyhedral terrain is NP-hard. In the remaining sections we study the problem in a different setting, namely where water can flow along the edges of a graph. We will see various results using this model in Section 6.3. We devise an algorithm to compute the maximal upstream area of a set of points, which we call the *potential watershed*. We extend this concept and our techniques to the maximal downstream area and the minimal upstream area (the *persistent watershed*) of a set of points. In Section 6.4 we study these concepts for a certain class of imprecise terrains which we call *regular*. We prove that persistent watersheds satisfy certain nesting conditions on regular terrains. This leads to efficient computations of fuzzy watershed boundaries and to the definition of the *potential ridge*, which delineates the persistent watersheds of the "main" minima of a regular terrain. We conclude with a critical discussion of the introduced concepts and open problems in Section 6.5.

6.1 Introduction

Simulating the flow of water on a terrain is a problem that has been studied for a long time in geographic information science (GIS), and has received considerable attention from the computational-geometry community [51, 52, 116]. It can be an important tool in analyzing flash floods for risk management [28], for stream flow forecasting [103], and in the general study of geomorphological processes [46], and it could contribute to obtaining more reliable climate change predictions [145].

When modeling the flow of water across a terrain, it is generally assumed that water flows downward in the direction of steepest descent. It is common practice to compute drainage networks and catchment areas directly from a digital elevation

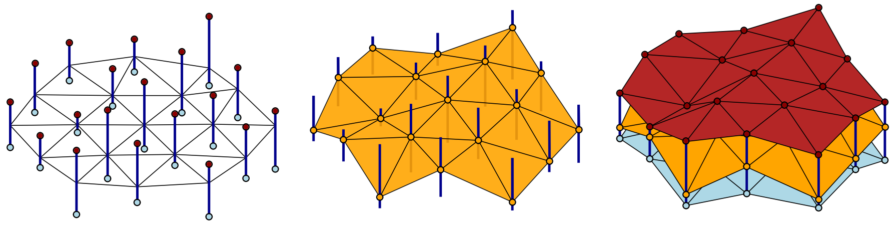


Figure 6.1: Left: An imprecise terrain. Each vertex of the triangulation has a elevation interval (gray). Center: a realization of the imprecise terrain. Right: the same realization together with the *highest* and *lowest* possible realizations of the imprecise terrain.

model of the terrain. Most hydrological research in GIS models the terrain surface with a grid in which each cell can drain to one or more of its eight neighbors (e.g. [143]). This can also be modeled as a computation on a graph, in which each node represents a grid cell and each edge represents the adjacency of two neighbors in the grid. Alternatively, one could use an irregular network in which each node drains to one or more of its neighbors, which may reduce the required storage space by allowing less interesting parts of the terrain to have a lower sampling density. We will refer to this as the *network model*, and we assume that, from every node, water flows down along the steepest incident edge. Assuming the elevation data is exact, drainage networks can be computed efficiently in this model (e.g. [49]). In computational geometry and topology, researchers have studied flow path and drainage network computations on triangulated polyhedral surfaces (e.g. [52, 54, 111]). In this model, which we call the *surface model*, the flow of water can be traced across the surface of a triangle. This avoids creating certain artifacts that arise when working with grid models. However, the computations on polyhedral surfaces may be significantly more difficult than on network models [58].

Naturally, all computations based on terrain data are subject to uncertainty, which comes from various sources like measurement, interpolation, and numerical errors. The GIS community has recognized the importance of dealing with uncertainty explicitly, in particular for hydrological modeling [154, 26]. A natural approach is to model the elevation at a point of the terrain using stochastic methods. However, the models available in the hydrology literature are unsatisfactory [40, 120, 136] and computationally expensive [150]. A particular challenge is posed by the fact that hydrological computations can be extremely sensitive to small elevation errors [91, 109]. While most of these studies have been done in the network model, we note that there also exists work on the behavior of watersheds under noise in the surface model [90].

A non-probabilistic model of imprecision that is often used in computational geometry consists in representing an imprecise attribute (such as location) by a region that is guaranteed to contain the true value. This approach has also been applied to polyhedral terrains (e.g. [82, 101]), replacing the exact elevation of each surface point by an imprecision interval (see Figure 6.1). In this way, each terrain vertex does not have one fixed elevation, but a whole range of possible elevations which includes the true elevation. Choosing a concrete elevation for each vertex results in a *realization* of the imprecise terrain. The realization is a (precise) polyhedral terrain. Since the set of all possible realizations is guaranteed to include the true (unknown) terrain, one can now obtain bounds on parameters of the true terrain by computing the best- and worst-case values of these parameters over the set of all possible realizations. Note that we assume that there is only an error in the z-coordinate (and not in the x, y-coordinates). This is partially motivated by the fact that commercial terrain data suppliers often only report elevation error [77]. However, it is also a natural simplification of the model.

In this chapter we apply this model of imprecise terrains to problems related to the simulation of water flow, both on terrains represented by surface models and on terrains represented by network models. One of the most fundamental questions one may ask about water flow on terrains is whether water flows from a point p to another point q. In the context of imprecise terrains, reasonable answers may be "definitely not", "possibly", and "definitely". The *watershed* of a point in a terrain is the part of the terrain that drains to this point. Phrasing the same question in terms of watersheds leads us to introduce the concepts of *potential* (maximal) and *persistent* (minimal) watersheds.

6.1.1 Basic definitions and notation

We define an ***imprecise terrain*** T as a possibly non-planar geometric graph G with nodes $V \subset \mathbb{R}^2$ and edges $E \subseteq V \times V$, where each node $v \in V$ has an imprecise third coordinate, which represents its ***elevation***. We denote the bounds of the elevation of v with $low(v)$ and $high(v)$. A ***realization*** R of an imprecise terrain T consists of the given graph together with an assignment of elevations to nodes, such that for each node v its elevation $elev_R(v)$ is at least $low(v)$ and at most $high(v)$. We denote with R^- the realization such that $elev_{R^-}(v) = low(v)$ for every node v and, similarly, we denote with R^+ the realization such that $elev_{R^+}(v) = high(v)$. The set of all realizations of an imprecise terrain T is denoted $\mathcal{R}_T$.

For any set of nodes $P \subseteq V$, we define the ***neighborhood*** of P as the set of nodes $N(P) = \{s : s \notin P \wedge \exists\, t \in P : (s,t) \in E\}$. Now, consider a realization R of an imprecise terrain as defined above. A set of nodes $P \subseteq V$ constitutes a ***local minimum*** in R if the following conditions are met: (i) the subgraph of G induced by P is connected, (ii) all nodes of P have the same elevation according to R, and (iii) their elevation is strictly lower than the elevation of any node in $N(P)$ according to R. Likewise, a local maximum is a set of nodes at the same elevation of which the neighborhood is strictly lower.

6.2 NP-hardness in the surface model

In the surface model water flows across the surface of a polyhedral terrain. In this section we prove that it is NP-hard to decide whether water potentially flows from a point s to another point t in this model. The reduction is from 3-SAT; the input is a 3-CNF formula with n variables and m clauses. We first define the details of the flow model in Section 6.2.1. Next, we describe the general idea of the proof in Section 6.2.2, then we proceed with a detailed description of the construction in Section 6.2.3, and finally we prove the correctness in Section 6.2.4 and Section 6.2.5.

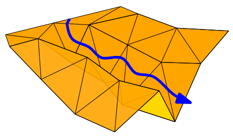
6.2.1 Flow model

Consider an imprecise terrain, where the graph that represents the terrain forms a planar triangulation in the (x, y)-domain. Any realization of this terrain is a polyhedral terrain with a triangulated surface. If we assume that the water which arrives at a particular point p on this surface will always flow in the true direction of steepest descent at p across the surface, possibly across the interior of a triangle, then we obtain a continuous model of water flow. Since the steepest-descent paths do not necessarily follow along the edges of the graph, but instead lead across the surface formed by the graph, we call this model the ***surface model***. This model has also been used before, for example in [52, 54, 111].

6.2.2 Overview of the construction

The main idea of the NP-hardness construction is to encode the variables and clauses of the 3-SAT instance in an imprecise terrain, such that a truth assignment to the variables corresponds to a realization—i.e., an assignment of elevations—of this terrain. If and only if all clauses are satisfied, water will flow from a certain starting vertex s to a certain target vertex t. We first introduce the basic elements of the construction: channels and switch gadgets.

Channels. We can mold channels in a fixed terrain surface to route water along any path, as long as the path is monotone in the direction of steepest descent on the terrain. We do this by giving vertices next to the path a higher elevation, thus building walls that force the water to stay in the channel. We can end a channel in a local minimum at any point, if needed.



Switch gadgets. The general idea of a *switch gadget* is that it provides a way for water to switch between channels. A simple switch gadget has one incoming channel, three outgoing channels, and two *control vertices* a and b, placed on the boundary of the switch. The water from the incoming channel has to flow across a central triangle, which is connected to a and b. Depending on their elevations, the two vertices a and b divert the water from the incoming channel to a particular outgoing channel and thereby "control" the behavior of the switch gadget. This is possible, since the slopes of the central triangles, which the water needs to pass, depend on the elevations of a and b and those two are the only vertices with imprecise elevations. The elevations of the remaining vertices which define the gadget are fixed. We lead the middle outgoing channel to a local minimum as shown in the examples. In this way, we ensure that, if any water can pass the switch, the elevations of its control vertices are at unambiguous extremal elevations. Depending on the particular construction of the switch, we may want the control vertices to be at opposite extremal elevations or at corresponding extremal elevations. Refer to Figure 6.2 for an illustration.

We can also build switches for multiple incoming channels. In this case, every incoming channel has its own dedicated set of outgoing channels, but all channels are controlled by the same two vertices, see Figure 6.3.

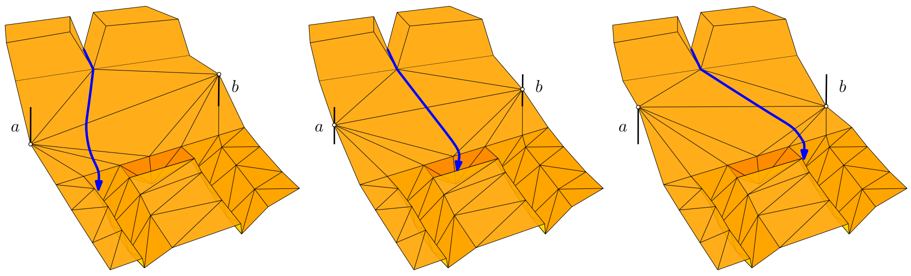


Figure 6.2: Three different states of a simplistic switch gadget.

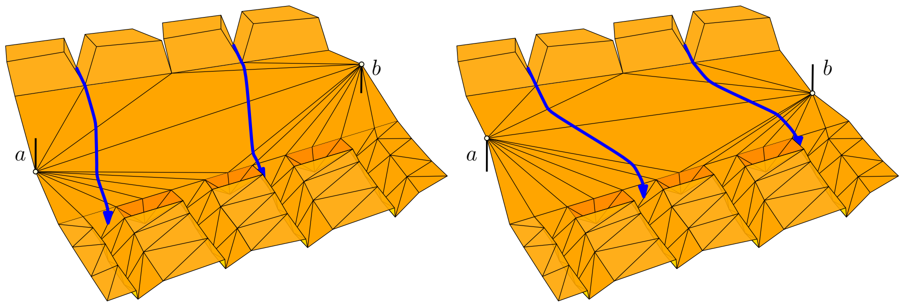


Figure 6.3: Sketch of a switch with multiple incoming channels.

Global layout. The global layout of the construction is depicted in Figure 6.4. The construction contains a grid of $m \times n$ cells, in which each clause corresponds to a column and each variable to a row of the grid. The grid is placed on the western slope of a “mountain”; columns are oriented north-south and rows are oriented east-west. We create a system of channels that spirals around this mountain, starting from s at the top and ending in t at the bottom of the mountain. We ensure that in no realization, water from s can escape this channel system and, if it reaches t, we know that it followed a strict course that passes through every cell of the grid exactly once, column by column from east to west, and within each column, from north to south. Embedded in this channel system, we place a switch gadget in every cell of the grid, which allows the water from s to “switch” from one channel to another within the current column depending on the elevations of the vertices that control the gadget. In this way, the switch gadgets of a row encode the state of a variable. To ensure that the state of a variable is encoded consistently across a row of the grid, the switch gadgets in a row are linked by their control vertices. Every column has a dedicated entry point at its north end, and a dedicated exit point at its south end. If and only if water flows between these two points, the clause that is encoded in this column is satisfied by the corresponding truth assignment to the variables. The slope of the mountain is such that columns descend towards the south, and the exit point of each column (except the westernmost one) is higher than the entry point of the adjacent column to the west; water can flow between these points through a channel around the back of the mountain. The easternmost column’s entry point is the starting vertex s, and the westernmost column’s exit point is the target vertex t.

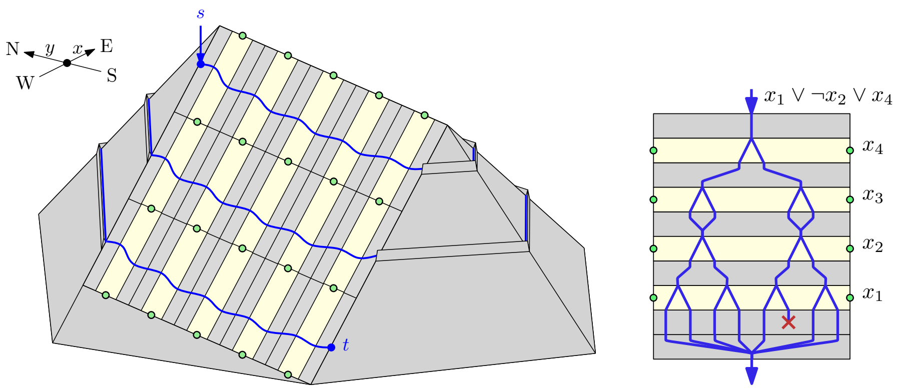


Figure 6.4: Left: Global view of the NP-hardness construction, showing the grid on the mountain slope. The fixed parts are shown in gray, the variable parts are shown light yellow and the imprecise vertices are filled light green; Right: Detail of a clause, which forms one of the columns of the grid.

Clause columns. To encode each clause, we connect the switch gadgets in a column of the global grid by channels in a tree-like manner. By construction, water will arrive in a different channel at the bottom of the column for each of the eight possible combinations of truth values for the variables in the clause. This is possible because a switch gadget can switch multiple channels simultaneously. We let the channel in which water would end up if the clause is not satisfied lead to a local minimum; the other seven channels merge into one channel that leads to the exit point of the clause. The possible courses that water can take will also cross switch gadgets of variables that are not part of the clause: in that case, each course splits into two courses, which are merged again immediately after emerging from the switch gadget. Figure 6.4 (right) shows an example.

Sloped switch gadgets. Since the grid is placed on the western slope of a mountain, water on the central triangle of a switch will veer off towards the west, regardless of the elevations of its control vertices. However, as we will see, we can still design a working switch gadget in this case. Recall that we link the switch gadgets of a variable row by their control vertices, such that each switch gadget shares one control vertex with its neighboring cell to the west and one with its neighboring cell to the east. As mentioned before, such a row encodes the state of a particular variable. We say that it is in a consistent state if either all control vertices of the switches are *high* or all control vertices are *low*. Thus, we will use the following assignment of truth values to the elevations of the control vertices of our switch gadgets: both vertices set to their highest elevation encodes *true*; both vertices set to their lowest elevation encodes *false*; other combinations encode *confused*. Depending on the truth value encoded by the elevations of the imprecise vertices, water that enters the gadget will flow to different channels. The channels in which the water ends up when the gadget reads *confused* always lead to a local minimum. For the other channels, their destination depends on the clause. In Figure 6.5 you can see a sketch of a sloped switch gadget which works the way described above.

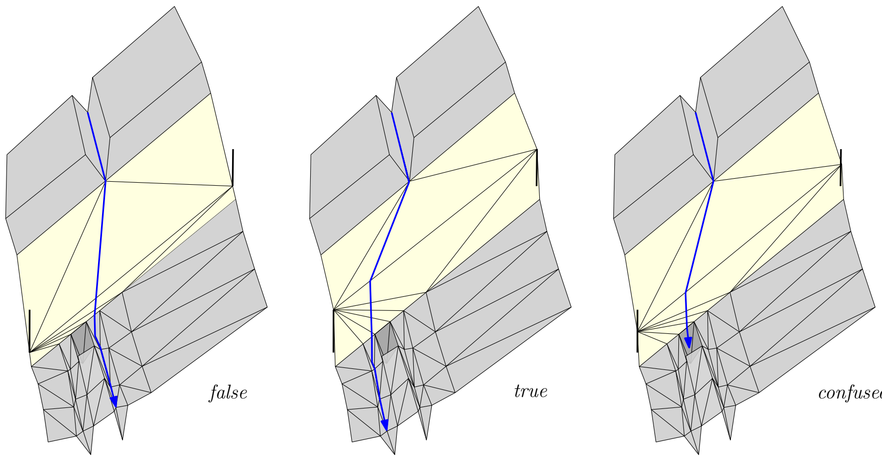


Figure 6.5: Illustration of a sloped switch gadget similar to the one used in the final construction. The final gadget has multiple incoming channels, which is not shown in this figure.

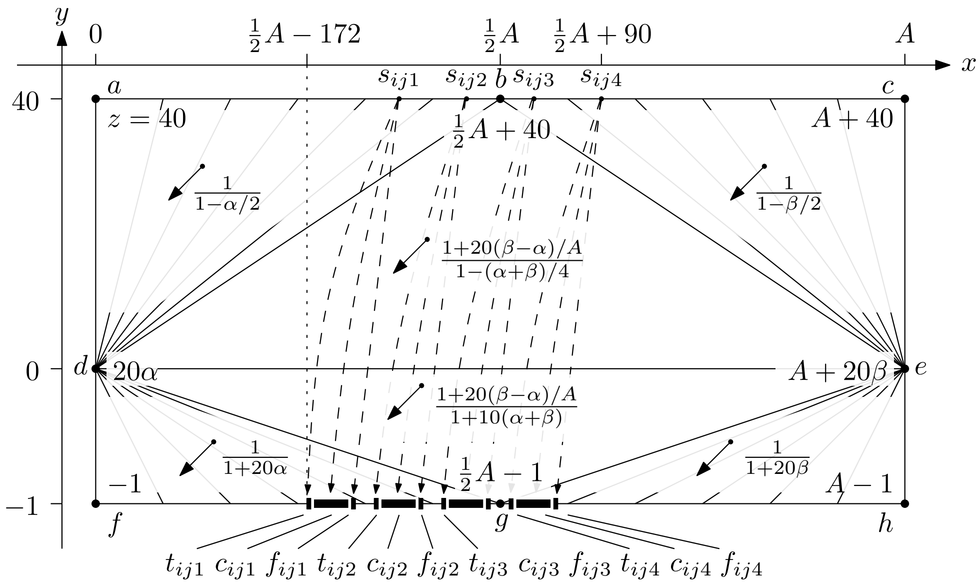
6.2.3 Details of the construction

Recall that we are given a 3-SAT instance with n variables and m clauses. The central part of the construction, which will contain the gadgets, consists of a grid of n rows—one for each variable—and m columns—one for each clause. We denote the width of each row, measured from north to south, by $B = 400$, and the width of a column, measured from west to east, by $A = \max((n+1) \cdot B, 4000)$. Ignoring local details, on any line from north to south in this part of the construction, the terrain descends at a rate of $dz/dy = 1$, and on any line from east to west, it descends at a rate of $dz/dx = 1$; thus we have $z = x + y$. Observe that each column measures $nB < A$ from north to south; thus the southern edge of each column is at a higher elevation than the northern edge of the next column to the west. The dedicated entrance and exit points of column $1 \leq j \leq m$ are placed at $(jA - \frac{1}{2}A, nB, jA - \frac{1}{2}A + nB)$ and $(jA - \frac{1}{2}A, 0, jA - \frac{1}{2}A)$, thus allowing the construction of a descending channel from each column's exit point to the entry point of the column to the west.

For every variable v_i, $1 \leq i \leq n$, we place $m+1$ imprecise vertices v_{ij}, for $0 \leq j \leq m$, in row i, on the boundaries of the columns corresponding to the m clauses. Vertex v_{ij} has x-coordinate jA, y-coordinate $iB - \frac{1}{2}B$, and an imprecise z-coordinate $[jA + iB - \frac{1}{2}B, jA + iB - \frac{1}{2}B + 20]$. On every pair of imprecise vertices $v_{i(j-1)}, v_{ij}$ we build a switch gadget G_{ij}; thus there is a switch gadget for each variable/clause pair. The coordinates of the vertices in each gadget, relative to the coordinates of $v_{i(j-1)}$, can be found in Box 6.6.

Switch gadget construction. We use the sloped switch gadget described above and illustrated in Figure 6.5. Our switch gadget occupies a rectangular area that is A wide from west to east, and 41 wide from north to south. Its key vertices and their coordinates, relative to each other, can be found in Box 6.6. There are two imprecise

Box 6.6 Distances and gradients on a connector gadget



All coordinates are relative to the lowermost position of the control vertex $d = v_{i(j-1)}$. The other control vertex is $e = v_{ij}$. Thus, e and d are the only imprecise vertices. The x- and y-coordinates of the vertices are indicated on the axes. The elevations of the key vertices are written next to the vertices. The elevations of the control vertices are expressed as a function of $\alpha, \beta \in [0, 1]$. The directions of steepest descent on the different faces of the gadget (marked with arrows) are expressed in the form dx/dy, as a function of α and β.

vertices, d and e, with elevation range $[0, 20]$ and $[A, A + 20]$, respectively—so in any realization, their elevations have the form 20α and $A + 20\beta$, respectively, where $\alpha, \beta \in [0, 1]$.

On the north edge of the gadget, there may be many more vertices, all collinear with a, b and c. The vertices on the western half of the north edge are connected to the western control vertex, and the vertices on the eastern half of the north edge are connected to the eastern control vertex. In particular, each gadget G_{ij} is designed to receive water from four channels that arrive at four points $s_{ij1}, s_{ij2}, s_{ij3}, s_{ij4}$ on the north edge, close to b; the coordinates of these points are $s_{ijk} = (\frac{1}{2}A - 150 + 60k, 40, \frac{1}{2}A - 110 + 60k)$.

On the south edge of the gadget, there is a similar row of vertices, all collinear with f, g and h, that are connected to the control vertices. To the south, the gadget is connected to twelve channels that catch all water that arrives at certain intervals on the south edge: for each $k \in \{1, 2, 3, 4\}$, there is a *western channel* t_{ijk} catching all water arriving between $s_{ijk} - (82, 41, 123)$ and $s_{ijk} - (77, 41, 118)$, a *middle channel* c_{ijk} catching all water arriving between $s_{ijk} - (77, 41, 118)$ and $s_{ijk} - (44, 41, 85)$, and an *eastern channel* f_{ijk} catching all water arriving between $s_{ijk} - (44, 41, 85)$ and $s_{ijk} - (39, 41, 80)$.

In a particular realization R, we define the switch gadget to be in a *false* state if $\alpha = \beta = 0$, in a *true* state if $\alpha = \beta = 1$, and in a *confused* state if $\alpha \leq \frac{1}{2}$ while $\beta \geq \frac{1}{2}$, or if $\alpha \geq \frac{1}{2}$ while $\beta \leq \frac{1}{2}$. As we will show below, in the true, false, and confused states the gadget leads any water that comes in at any point s_{ijk} into t_{ijk}, f_{ijk}, and c_{ijk}, respectively.

We model the fixed part of the terrain such that the middle channels all lead to local minima. The western and eastern channels correspond to a (partial) truth assignment of the variables of the clause that is represented by the column that contains the gadget; these channels lead to a local minimum or to the next row, as described below.

Constructing the clause columns. Each clause is modeled in a column j by making certain connections between the outgoing channels of each gadget to the dedicated entrance points of the gadget in the next row. Observe that by our choice of B, the entrance point of column j lies above all entrance points of G_{nj}, all outgoing channels of any gadget $G_{(i+1)j}$ start at higher elevations than all entrance points of G_{ij}, and all outgoing channels of G_{1j} start at an elevation higher than the exit point of the column. This ensures that all channels described below can indeed be built as monotonously descending channels, so that water can flow through it. We will now explain the connections which we use to build a clause.

Let $p > q > r$ be the indices of the variables that appear in the clause. The water courses modeling the clause start at the entry point of the column, from which any flow path is led through a channel to entry point s_{nj1} of gadget G_{nj}.

For $i \neq \{p, q, r\}, k \in \{1, 2, 3, 4\}$, we connect both t_{ijk} and f_{ijk} to $s_{(i-1)jk}$ (if $i > 1$) or to the exit point of the column (if $i = 1$).

We connect t_{pj1} and f_{pj1} to $s_{(p-1)j1}$ and $s_{(p-1)j2}$, respectively. Thus, for $i \in \{q, ..., p-1\}$, water that enters G_{ij} at s_{ij1} and s_{ij2} represents the cases that p is true and p is false, respectively.

We connect $t_{qj1}, f_{qj1}, t_{qj2}$ and f_{qj2} to $s_{(q-1)j1}, s_{(q-1)j2}, s_{(q-1)j3}$ and $s_{(q-1)j4}$, respectively. Thus, for $i \in \{r, ..., q-1\}$, water that enters G_{ij} at $s_{ij1}, s_{ij2}, s_{ij3}$ and s_{ij4} represents the four different possible combinations of truth assignments to p and q, respectively.

The eight channels $t_{rj1}, f_{rj1}, t_{rj2}, f_{rj2}, t_{rj3}, f_{rj3}, t_{rj4}, f_{rj4}$ now represent the eight different possible combinations of truth assignments to the variables of the clause. The channel that corresponds to the truth assignment that renders the clause false, is constructed such that it ends in a local minimum. The other seven channels all lead to $s_{(r-1)j1}$ (if $r > 1$) or to the exit point of the clause column (if $r = 1$).

6.2.4 Analysis of flow through a gadget

Below we will analyze where water may leave a gadget G_{ij} after entering the gadget at point s_{ijk}, with x-coordinate x_k. In the discussion below, all coordinates are relative to the lowermost position of the western control vertex of the gadget—refer to Box 6.6, which also shows the directions of steepest descent (i.e. the surface gradients, expressed as dx/dy) on each face of the gadget.

First observe that in any case, the directions of steepest descent on $\triangle abd$, $\triangle bce$ and $\triangle bed$ are at least $1 - 20/A \geq 199/200 = 0.995$ and at most $(1 + 20/A)/(1/2) \leq 201/100 = 2.01$. Thus, when the water reaches y-coordinate 38, it will be at x-coordinate at least $x_k - 4.02$ and at most $x_k - 1.99$.

Note that the line bd intersects the plane $y = 38$ at $x = \frac{1}{2}A - \frac{1}{40}A \leq \frac{1}{2}A - 100$, and the line be intersects the plane $y = 38$ at $x = \frac{1}{2}A + \frac{1}{40}A \geq \frac{1}{2}A + 100$. By our choice of coordinates for the entrance points s_{ijk}, we have $|x_k - \frac{1}{2}A| \leq 90$; therefore the water will be on $\triangle bed$ when it reaches $y = 38$. Let $g_{\max}$ and $g_{\min}$ be the maximum and minimum possible gradients dx/dy on $\triangle bed$, respectively. Thus, the water will reach the line de at x-coordinate at least $x_k - 4.02 - 38g_{\max}$ and at most $x_k - 1.99 - 38g_{\min}$.

Finally, the directions of steepest descent on $\triangle dgf$, $\triangle egh$ and $\triangle deg$ are more than 0 and less than $1 + 20/A \leq 201/200 < 1.01$. Thus, the water will reach the line fh at x-coordinate more than $x_k - 5.03 - 38g_{\max}$ and less than $x_k - 1.99 - 38g_{\min}$.

We will now consider five classes of configurations of the control vertices in the gadget, and compute the interval of x-coordinates where water may reach the line fh in each case.

- $\alpha = \beta = 1$ *(true state)* In this case we have $g_{\max} = g_{\min} = 2$, so water will reach the line fh within the x-coordinate interval $(x_k - 81.03, x_k - 77.99)$, and thus it will flow into channel t_{ijk}.

- $\alpha + \beta > \frac{3}{2}$ *(true-ish state)* In this case we have $g_{\max} \leq (1 + 20/A)/(1/2) \leq 2.01$ and $g_{\min} \geq (1 - 20/A)/(1 - 3/8) \geq 199/125 > 1.59$. Thus water will reach the line fh within the x-coordinate interval $(x_k - 81.41, x_k - 62.41)$, and thus it will flow into channel t_{ijk} or c_{ijk}.

- $\frac{1}{2} \leq \alpha + \beta \leq \frac{3}{2}$ *(this includes all proper confused states)* In this case we have $g_{\max} \leq (1+20/A)/(1-3/8) \leq 201/125 < 1.61$ and $g_{\min} \geq (1-20/A)/(1-1/8) \geq 199/175 > 1.13$. Thus water will reach the line fh within the x-coordinate interval $(x_k - 66.21, x_k - 44.93)$, and thus it will flow into channel c_{ijk}.

- $\alpha + \beta < \frac{1}{2}$ *(false-ish state)* In this case we have $g_{\max} \leq (1 + 20/A)/(1 - 1/8) \leq 201/175 < 1.15$ and $g_{\min} \geq (1 - 20/A) \geq 199/200 > 0.99$. Thus water will reach the line fh within the x-coordinate interval $(x_k - 48.73, x_k - 39.61)$, and thus it will flow into channel c_{ijk} or f_{ijk}.

- $\alpha = \beta = 0$ *(false state)* In this case we have $g_{\max} = g_{\min} = 1$, so water will reach the line fh within the x-coordinate interval $(x_k - 43.03, x_k - 39.99)$, and thus it will flow into channel f_{ijk}.

6.2.5 Correctness of the NP-hardness reduction

Lemma 6.2.1 *If water flows from s to t in some realization, then there is a truth assignment that satisfies the 3-CNF formula.*

Proof: Water that starts flowing from s, which is the entrance point of the clause column m, is immediately forced into a channel to entrance point s_{nm1} of gadget G_{nm}. As calculated above, any water that enters a gadget at one of its designated entrance points will leave the gadget in one of its designated channels, which leads either to a local minimum, or to a designated entrance point of the next gadget. Therefore, water from s can only reach t after flowing through all switch gadgets.

Since all middle outgoing channels c_{ijk} lead to local minima, we know that if there is a flow path from s to t, then the water from s is nowhere forced into a middle outgoing channel. It follows that no gadget is in a proper confused state. As a consequence, in any row, either all gadgets have their control vertices in the lower open half of their elevation range, or all gadgets have their control vertices in the upper open half of their elevation range. In the first case, all gadgets in the row are in a *false-ish* state, and any incoming water from s leaves those gadgets in the same channels as if the gadgets were in a proper *false* state. In the second case, all gadgets in the row are in a *true-ish* state, and any incoming water from s leaves those gadgets in the same channels as if the gadgets were in a proper *true* state.

We can now construct a truth assignment $\mathcal{A}$ to the variables, in which each variable is *true* if the control vertices in the corresponding row are in the upper halves of their elevation ranges, and *false* otherwise. It follows from the way in which channel networks in clause columns are constructed, that in each clause column, water will flow into one of the seven channels that corresponds to a truth assignment that satisfies the corresponding clause—otherwise the water would not reach t. Therefore, $\mathcal{A}$ satisfies each clause, and thus, the complete 3-CNF formula. □

Lemma 6.2.2 *If there is a truth assignment to the variables that satisfies the given 3-CNF formula, then there is a realization of the imprecise terrain in which water flows from s to t.*

Proof: We set all control vertices in rows corresponding to true variables to their highest positions and all control vertices in rows corresponding to false variables to their lowest positions. One may now verify that, by construction, in each clause column water from the column's entry point will flow into one of the seven channels that lead to the column's exit point, and thus, water from s reaches t. □

Thus, 3-SAT can be reduced, in polynomial time, to deciding whether there is a realization of T such that water can flow from s to t. We conclude that deciding whether there exists a realization of T such that water can flow from s to t is NP-hard.

Theorem 6.2.3 *Let T be an imprecise triangulated terrain, and let s and t be two points on the terrain. Deciding whether there exists a realization $R \in \mathcal{R}_T$ such that $p \xrightarrow[R]{} q$ is NP-hard.*

6.3 Watersheds in the network model

In the network model we assume that water flows only along the edges of a realization. More specifically, water that arrives in a node p continues to flow along the steepest-descent edges incident to p, unless p is a local minimum.

6.3.1 Flow model

Consider a realization R of an imprecise terrain as defined in Section 6.1.1. If water is only allowed to flow along the edges of the realization, then the realization represents a network. Therefore we refer to this model of water flow as the ***network model***. Below, we state more precisely how water flows in this model and give a proper definition of the watershed. This model or variations of it have been used before, for example in [49, 122, 143].

The steepness of descent (slope) of an edge $(p, q) \in E$ in realization R is defined as $\sigma_R(p, q) = (elev_R(p) - elev_R(q))/|pq|$, where $|pq|$ is the Euclidean distance between p and q in the xy-plane. The node q is a ***steepest-descent neighbor*** of p in R, if and only if $\sigma_R(p, q)$ is non-negative and maximal over all neighbors q of p. In the realization R, water that arrives in p will continue to flow to each of its steepest-descent neighbors, unless p constitutes a local minimum. If there exists a local minimum $P \ni p$, then the water that arrives in p will flow to the neighbors of p in P and eventually reach all the nodes of P, but it will not flow further to any node outside the set P. If water from p reaches a node $q \in V$ then we write $p \xrightarrow[R]{} q$ ("p flows to q in R"), and for technical reasons we define $p \xrightarrow[R]{} p$ for all p and R.

The ***discrete watershed*** of a node q in a realization R is defined as the union of nodes that flow to q in R, that is $\mathcal{W}_R(q) := \{p \ : \ p \xrightarrow[R]{} q\}$. Similarly, we define the discrete watershed of a set of nodes Q in this realization as $\mathcal{W}_R(Q) := \bigcup_{q \in Q} \mathcal{W}_R(q)$. To simplify the terminology we will refer to it as watershed, since this is unambigious in the context of the remaining sections.

Consider the graph G of the imprecise terrain. A path π in G is a ***flow path*** for a realization R if it does not self-intersect (any node appears on the path at most once) and each node on the path (except the first) is a steepest-descent neighbor of its predecessor on the path. For any pair of nodes p, q in π, we write $p \xrightarrow{\pi} q$ if π contains p and q in this order. For any set of realizations $\mathcal{S} \subseteq \mathcal{R}_T$, we denote with $\Pi(\mathcal{S})$ the set union of all flow paths induced by a realization in $\mathcal{S}$. We define a ***maximal flow path*** as a flow path that ends in a local minimum and cannot continue without intersecting itself.

6.3.2 Flow paths are stable

This subsection is a note on flow paths in the network model. We define when a flow path is stable and argue that any flow path induced by a realization in $\mathcal{R}_T$ is stable with respect to some ε-neighborhood of $\mathcal{R}_T$. Intuitively, the analysis in this section shows that the flow paths considered in our model are never the result of an isolated degenerate situation, but could also exist if the estimated elevation intervals of the vertices would be slightly different. This may serve as a justification or proof of soundness of the network model. It is not necessary to read this section in order to understand the results in the remaining part of the chapter.

For two realizations $R, R' \in \mathcal{R}_T$, we call R' an ***ε-perturbation*** of R if for all nodes $v \in V$ it holds that $|elev_R(v) - elev_{R'}(v)| \leq \varepsilon$. For a set of realizations $\mathcal{S}$, let $\mathcal{S}^\varepsilon$ denote the union of $\mathcal{S}$ with the ε-perturbations of elements of $\mathcal{S}$. We say that a flow path π is ***stable*** with respect to $\mathcal{S}$ if for some $\varepsilon > 0$ the flow path exists in any ε-perturbation of some $R \in \mathcal{S}$. In this context, we call R a ***perturbation center*** of π.

Lemma 6.3.1 *Given a set of realizations $\mathcal{S}$ and any value $\delta > 0$, it holds that any flow path π induced by a realization in $\mathcal{S}$ is stable with respect to $\mathcal{S}^\delta$.*

Proof: We call a realization which does not contain horizontal edges and in which any node has at most one steepest-descent neighbor ***non-ambiguous***, similarly, a realization for which any of these properties does not hold is called ambiguous. Any flow path π induced by a non-ambiguous realization $R \in \mathcal{S}$ is stable with perturbation center R, since we can pick ε small enough such that the order of the slopes of the edges does not change. Now, let $\pi = p_1, p_2, \ldots, p_k$ be a flow path from p_1 to p_k which is induced by an ambiguous realization $R \in \mathcal{S}$. We lower each node p_i by $\delta/2 + (i\delta)/(4k)$ and perturb the remaining vertices by some value smaller than $\varepsilon/4$. Since π is non-intersecting, we create a non-ambiguous realization $R' \in \mathcal{S}^\delta$ in this way which also induces π. This proves the claim. $\square$

6.3.3 Potential watersheds

The ***potential watershed*** of a set of nodes Q in a terrain T is defined as

$$\mathcal{W}_{\cup}(Q) := \bigcup_{R \in \mathcal{R}_T} \bigcup_{q \in Q} \mathcal{W}_R(q),$$

which is the union of the watersheds of Q over all realizations of T. This is the set of nodes from which there exists a flow path to a node of Q in some realization. With slight abuse of notation, we may also write $\mathcal{W}_{\cup}(q)$ to denote the potential watershed of a single node q.

6.3.3.1 Canonical realizations

We prove that for any given set of nodes Q in an imprecise terrain, there exists a realization R such that $\mathcal{W}_R(Q) = \mathcal{W}_{\cup}(Q)$. For this we introduce the notion of the overlay of a set of watersheds in different realizations of the terrain. Informally, the overlay is a realization that sets every node that is contained in one of these watersheds to the lowest elevation it has in any of these watersheds.

Definition 6.3.2 Given a sequence of realizations $R_1, ..., R_k$ and a sequence of nodes $q_1, ..., q_k$, the ***watershed-overlay*** of $\mathcal{W}_{R_1}(q_1), ..., \mathcal{W}_{R_k}(q_k)$ is the realization $\overline{R}$ such that for every node v, we have that $elev_{\overline{R}}(v) = high(v)$ if $v \notin \bigcup \mathcal{W}_{R_i}(q_i)$ and otherwise

$$elev_{\overline{R}}(v) = \min_{i: v \in \mathcal{W}_{R_i}(q_i)} elev_{R_i}(v).$$

Note that we allow ourselves a slight abuse of wording and notation here: the input to the watershed-overlay operation is not a set of watersheds, but a sequence of realizations and a sequence of nodes.

Lemma 6.3.3 *Let $\overline{R}$ be the watershed-overlay of $\mathcal{W}_{R_1}(q_1), \ldots, \mathcal{W}_{R_k}(q_k)$, and let $Q = \bigcup_{1 \le i \le k} q_i$, then $\mathcal{W}_{\overline{R}}(Q)$ contains $\mathcal{W}_{R_i}(q_i)$, for any $1 \le i \le k$.*

Proof: Let u be a node of the terrain. We prove the lemma by induction on increasing symbolic elevation to show that if u is contained in one of the given watersheds, then it is also contained in $\mathcal{W}_{\overline{R}}(Q)$. To this end, we define *level*(R_i, u) as the smallest number of edges on any path along which water flows from u to q_i in R_i; if there is no such path, then $\mathit{level}(R_i, u) = \infty$. Now we define the ***symbolic elevation*** of u, denoted $\mathit{elev}^*(u)$, as follows: if u is contained in any watershed $\mathcal{W}_{R_i}(q_i)$, then $\mathit{elev}^*(u)$ is the lexicographically smallest tuple $(\mathit{elev}_{R_i}(u), \mathit{level}(R_i, u))$ over all i such that $u \in \mathcal{W}_{R_i}(q_i)$; otherwise $\mathit{elev}^*(u) = (\mathit{high}(u), \infty)$.

Now consider a node u that is contained in one of the given watersheds. The base case is that u is contained in Q, and in this case the claim holds trivially. Otherwise, let R_i be a realization such that $u \in \mathcal{W}_{R_i}(q_i)$ and such that $(\mathit{elev}_{R_i}(u), \mathit{level}(R_i, u))$ is lexicographically smallest over all $1 \leq i \leq k$. By construction, we have that $\mathit{elev}_{R_i}(u) = \mathit{elev}_{\overline{R}}(u)$. Consider a neighbor v of u such that (u, v) is a steepest-descent edge incident on u in R_i, and $\mathit{level}(R_i, v)$ is minimal among all such neighbors v of u. Since $\mathit{elev}_{\overline{R}}(v) \leq \mathit{elev}_{R_i}(v) \leq \mathit{elev}_{R_i}(u) = \mathit{elev}_{\overline{R}}(u)$ and $\mathit{level}(R_i, v) = \mathit{level}(R_i, u) - 1$, it holds that v has smaller symbolic elevation than u. Therefore, by induction, $v \in \mathcal{W}_{\overline{R}}(Q)$. If v is still a steepest-descent neighbor of u in $\overline{R}$, then this implies $u \in \mathcal{W}_{\overline{R}}(Q)$. Otherwise, there is a node $\widehat{v}$ such that $\sigma_{\overline{R}}(u, \widehat{v}) > \sigma_{\overline{R}}(u, v) \geq 0$. There must be an R_j such that $\widehat{v} \in \mathcal{W}_{R_j}(q_j)$, since otherwise, by construction of the watershed-overlay, we have $\mathit{elev}_{\overline{R}}(\widehat{v}) = \mathit{high}(\widehat{v}) \geq \mathit{elev}_{R_i}(\widehat{v})$ and thus, $\sigma_{R_i}(u, \widehat{v}) \geq \sigma_{\overline{R}}(u, \widehat{v}) > \sigma_{\overline{R}}(u, v) \geq \sigma_{R_i}(u, v)$ and v would not be a steepest-descent neighbor of u in R_i. Moreover, we have $\sigma_{\overline{R}}(u, \widehat{v}) > 0$ and, therefore, $\mathit{elev}_{\overline{R}}(\widehat{v}) < \mathit{elev}_{\overline{R}}(u)$, so $\widehat{v}$ has smaller symbolic elevation than u. Therefore, by induction, also $\widehat{v} \in \mathcal{W}_{\overline{R}}(Q)$ and thus, $u \in \mathcal{W}_{\overline{R}}(Q)$. $\square$

The above lemma implies that for any set of nodes Q, the watershed-overlay $\overline{R}$ of the watersheds of the elements of Q in all possible realizations $\mathcal{R}_T$, would realize the potential watershed of Q. That is, we have that $\mathcal{W}_{\cup}(Q) \subseteq \mathcal{W}_{\overline{R}}(Q)$ and since $\mathcal{W}_{\cup}(Q)$ is the union of all watersheds of Q in all realizations, we also have that $\mathcal{W}_{\overline{R}}(Q) \subseteq \mathcal{W}_{\cup}(Q)$, which implies the equality of the two sets. Therefore, we call $\overline{R}$ the ***canonical realization*** of the potential watershed $\mathcal{W}_{\cup}(Q)$ and we denote it with $R_{\cup}(Q)$.

Note, however, that it is not immediately clear that the canonical realization always exists: the set of possible realizations is a non-discrete set, and thus the elevations in the canonical realization are defined as minima over a non-discrete set. Therefore, one may wonder if these minima always exist. Below, we will describe an algorithm that can actually compute the canonical realization of any set of nodes Q; from this we may conclude that it always exists.

6.3.3.2 Outline of the potential watershed algorithm

Next, we describe how to compute $\mathcal{W}_{\cup}(Q)$ and its canonical realization $R_{\cup}(Q)$ for a given set of nodes Q. Note that for all nodes $p \notin \mathcal{W}_{\cup}(Q)$, we have, by definition of the canonical realization, $\mathit{elev}_{R_{\cup}(Q)}(p) = \mathit{high}(p)$. The challenge is therefore to compute $\mathcal{W}_{\cup}(Q)$ and the elevations of the nodes of $\mathcal{W}_{\cup}(Q)$. Below we describe an algorithm that does this. The idea of the algorithm is to compute the nodes of $\mathcal{W}_{\cup}(Q)$ and their elevations in the canonical realization in increasing order of elevation, similar to the way in which Dijkstra's shortest path algorithm computes distances from the source. We denote the resulting algorithm with **PotentialWS**(Q) (Algorithm 6.3.5). The correctness and running time of the algorithm are proved in Theorem 6.3.9. A key ingredient of the algorithm is a subroutine, **Expand**(q', z'), which is defined as follows.

Algorithm 6.3.4 Let **Expand**(q', z') denote a function that returns for a node q' and an elevation $z' \in [low(q'), high(q')]$ a set of pairs of nodes and elevations, which includes the pair (p, z) if and only if $p \in N(q')$, there is a realization R with $elev_R(q') \in [z', high(q')]$ such that $p \underset{R}{\rightarrow} q'$, and z is the minimum elevation of p over all such realizations R.

Algorithm 6.3.5 PotentialWS(Q)

```
Input: set of nodes $Q$
1: for all $q \in Q$ do enqueue $(q, z)$ with key $z = low(q)$
2: while the queue is not empty do
3:    Extract a pair $(q', z')$ with minimum key $z'$ from the queue
4:    if $q'$ is not already in the output set then
5:       Output $q'$ with elevation $z'$
6:       Enqueue each $(p, z) \in$ Expand$(q', z')$
```

6.3.3.3 Expansion of a node using the slope diagram

Before presenting the algorithm for the expansion of a node, we discuss a data structure that allows us to do this efficiently.

Definition 6.3.6 For given elevations of the neighbors of a node p, we define the ***slope diagram*** of p as the set of points $\widehat{q_i} = (\delta_i, z_i)$ such that q_i is a neighbor of p, z_i is its elevation and δ_i is its distance to p.

The intuition behind the slope diagram is the following. For a given elevation z of p, let $\widehat{p} = (0, z)$ be a point on the vertical axis of the slope diagram. Note that for any neighbor q_i, the slope of the line through p and q_i is the same as the slope of the line through $\widehat{p}$ and $\widehat{q_i}$ in the slope diagram. If q_i is a steepest descent neighbor of p under the given assignment of elevations, then all other neighbors $\widehat{q_j}$ lie above or on the line through $\widehat{p}$ and $\widehat{q_i}$ in the slope diagram.

Now, let $q_1, q_2, ...$ be a subset of the neighbors of p indexed such that $\widehat{q_1}, \widehat{q_2}, ...$ appear in counter-clockwise order along the boundary of the convex hull of the slope diagram, starting from the leftmost point and continuing to the lowest point. We ignore neighbors that do not lie on this lower left chain. Let H_i be the halfplane in the slope diagram that lies above the line through $\widehat{q_i}$ and $\widehat{q_{i+1}}$. Let $U(p)$ be the intersection of these halfplanes $H_1, H_2, ...$ with the halfplane to the right of the vertical line through the leftmost point, and the halfplane above the horizontal line through the bottommost point of the convex chain; see the shaded area in Figure 6.7.

The tangent of $U(p)$ through $\widehat{p}$ in the slope diagram passes through exactly the neighbors of p which are steepest descent neighbors of p. If $U(p)$ does not have a tangent through p, then p is a local minimum.

For a neighbor p of q', we can now compute the elevation of p as it should be returned by **Expand**(q', z') as follows. We use the slope diagram of p with the neighbors of p set to their highest position (that is, for a neighbor q_i we use $high(q_i)$) and compute the tangents to $U(p)$ which pass through the point $\widehat{q'} = (\delta', z')$, where

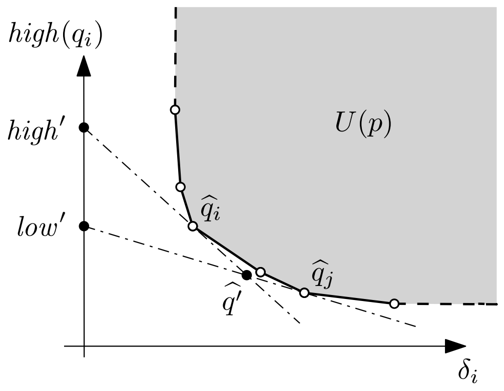


Figure 6.7: Querying the slope diagram.

δ' is the distance from q' to p. Assume for now that $U(p)$ has two tangents through $\widehat{q'}$ and let $[low', high']$ be the interval where the two tangents intersect the vertical axis of the slope diagram. Lemma 6.3.7 below implies that the lowest elevation that p can have in order to send flow via q' to the watershed is the lower endpoint of the interval $\mathcal{I} = [low(p), high(p)] \cap [low', high']$. This is the elevation which we return for p in the output of **Expand**(q', z'), unless the interval is empty. In the latter case we omit p from the output.

Lemma 6.3.7 *If q' is at elevation z', then the interval $\mathcal{I} = [low(p), high(p)] \cap [low', high']$ defines the elevations of p for which q' can be the steepest descent neighbor of p.*

Proof: Fix p at some arbitrary elevation z and let $\widehat{p} = (0, z)$ be its corresponding point in the slope diagram. If $z \in \mathcal{I}$, then all neighbors of p lie above or on the line through $\widehat{p}$ and $\widehat{q'}$ in the slope diagram. Thus, q' is steepest descent neighbor for this configuration of elevations. On the other hand, if there exists a configuration of the elevations of the other neighbors of p, such that they lie above or on the line through $\widehat{p}$ and $\widehat{q'}$, then they also lie above or on this line if we set them to their highest possible position. Thus, z would be included in $\mathcal{I}$ in this case. We conclude that if and only if $z \in [low', high']$ we can find a configuration of the elevations of the neighbors of p, in which q' is at elevation z' and at the same time q' is the steepest descent neighbor of p. □

We can compute the slope diagrams of all nodes with the neighbors set to their highest positions in a preprocessing phase. During the main algorithm the tangents can be computed via a binary search on the boundary of the convex hull in the slope diagram. In the proof of the following lemma we describe the technical details of this procedure more specifically. We also discuss the special cases where $U(p)$ does not have two tangents through $\widehat{q'}$.

Lemma 6.3.8 *Given the slope diagrams of the neighbors of q', we can compute the function* **Expand**(q', z') *in time $O(d \log d')$, where d is the node degree of q', and d' is the maximum node degree of a neighbor of q'.*

Proof: For each neighbor p of q' we proceed as follows. We are given the (precomputed) slope diagram of p with all neighbors of p set to their highest possible position.

First, determine by binary search on the boundary of $U(p)$, if the vertical line through $\widehat{q'}$ intersects the boundary of $U(p)$, and if it does, whether the intersection point lies below $\widehat{q'}$. If it does, $\widehat{q'}$ lies in the interior of $U(p)$. Then q' can never be a steepest-descent neighbor of p, and therefore p is not included in the result of **Expand**(q', z').

Otherwise $\widehat{q'}$ lies outside the interior of $U(p)$ and we continue as follows. If $\widehat{q'}$ lies on the vertical line that contains the left edge of $U(p)$, we define $high' = \infty$. Otherwise we find, by binary search on the boundary of $U(p)$, the corner $\widehat{q}_i$ that lies to left of the vertical line through $\widehat{q'}$, such that the line through $\widehat{q'}$ and $\widehat{q}_i$ is tangent on $U(p)$; let $high'$ be the vertical coordinate of the intersection of this tangent with the vertical axis. If $\widehat{q'}$ lies on or below the horizontal line that contains the bottom edge of $U(p)$, then we define $low' = z'$. Otherwise we find, by binary search on the boundary of $U(p)$, the corner $\widehat{q}_j$ that lies below the horizontal line through $\widehat{q'}$, such that the line through $\widehat{q'}$ and $\widehat{q}_j$ is tangent on $U(p)$; let low' be the vertical coordinate of the intersection of this tangent with the vertical axis.

If $low' > high(p)$ or $high' < low(p)$, then the set of elevations that p could have while having q' as a steepest-descent neighbor is empty, and we do not include p in the result of **Expand**(q', z'). Otherwise we include p with elevation $\max(low(p), low')$.

All computations for a single neighbor p of q' can be done in time logarithmic in the degree of p, and thus, the function **Expand**(q', z') can be computed in time $O(d \log d')$ in total. □

6.3.3.4 Correctness and running time of the complete algorithm

Theorem 6.3.9 *Algorithm* **PotentialWS**(Q) *computes the potential watershed* $\mathcal{W}_\cup(Q)$ *of a set of nodes* Q *and its canonical realization* $R_\cup(Q)$ *in* $O(n \log n)$ *time, where* n *is the number of edges in the terrain.*

Proof: The algorithm searches the graph starting from the nodes of Q. At each point in time we have three types of nodes. Nodes that have been extracted from the priority queue have a *finalized* elevation, a node that is currently in the priority queue but was never extracted (yet) has a *tentative* elevation, other nodes have not been reached.

We will show that when (p, z) is first extracted from the priority queue in Algorithm 6.3.5, p is indeed contained in the potential watershed of Q, and the elevation z is the lowest possible elevation of p such that water flows from p to any node in Q in any realization. To this end we use an induction on the points extracted, in the order in which they are extracted for the first time. The induction hypothesis consists of two parts:

(i) There exists a realization R and $q \in Q$ such that $elev_R(p) = z$ and R induces a flow path π from p to q which only visits nodes that have been extracted from the priority queue.

(ii) There exists no realization R and $q \in Q$ such that $elev_R(p) < z$ and $p \xrightarrow[R]{} q$.

If a node $p \in Q$ is extracted with $z = low(p)$, then the claims hold trivially. Note that the first extraction from the priority queue must be of this type.

If p is extracted from the priority queue for the first time and $p \notin Q$, then there must be at least one node p' that was extracted earlier, such that **Expand**(p', z'), for some elevation z', resulted in p having the tentative elevation z. By induction, there exists a realization R' and $q \in Q$, such that $elev_{R'}(p') = z'$, there is a flow path π from p' to q in R', and π does not include p.

To see that part (i) of the induction hypothesis holds for p, we construct a realization R by modifying R' as follows: we set $elev_R(p) = z$, and we set $elev_R(r) = high(r)$ for each neighbor r of p that does not lie on π. In comparison to R', only p and its neighbors may have a different elevation in R. Since $elev_R(p) = z \geq z'$ is still at least as high as the elevation of any node on π, water will still flow along the path π from p' to q. By the definition of **Expand**, none of the neighbors of p that are set at their highest elevation can out-compete p' as a steepest-descent neighbor of p. Therefore, in R, the node p must have p' or another node of π as a steepest-descent neighbor. Thus, water from p will flow onto π, and thus, to q.

Next we show (ii). Suppose, for the sake of contradiction, there is a realization R such that $elev_R(p) < z$ and there is a flow path from p to a node $q \in Q$. Consider two consecutive nodes r and s on this path, such that r has not been extracted before but s has been previously extracted (it may be that $r = p$ and/or $s \in Q$). Note that flow paths have to be monotone in the elevation. We argue that this path cannot stay below z in any realization. Since r is a neighbor of s, it has been added to the priority queue during the expansion of s. Let the tentative elevation of r that resulted from this expansion be z_r. By induction, since the elevation of s is finalized, z_r is a lower bound on the elevation of r for any flow path that follows the edge (r, s) and then continues to a node in Q in any realization. However, $z_r \geq z$, since r was not extracted from the priority queue before p. Therefore, a path from p to q that contains r with $elev_R(p) < z$ cannot exist. This proves (ii). It follows that the algorithm outputs all nodes of $\mathcal{W}_\cup(Q)$ together with their elevations in $R_\cup(Q)$.

As for the running time, computing and storing $U(p)$ for a node p of degree d takes $O(d \log d)$ time and $O(d)$ space. Since the sum of all node degrees is $2n$, computing and storing $U(p)$ for all nodes p thus takes $O(n \log d_{\max})$ time and $O(n)$ space in total, where $d_{\max}$ is the maximum node degree in the terrain. While running algorithm **PotentialWS**(Q), each node is expanded at most once. By Lemma 6.3.8, **Expand**(q', z') on a node q' of degree d takes time $O(d \log d_{\max})$. Thus, again using that all nodes together have total degree $2n$, the total time spent on expanding is $O(n \log d_{\max}) = O(n \log n)$. Each extraction from the priority queue takes time $O(\log n)$ and there are at most $O(n)$ nodes to extract. Therefore **PotentialWS** takes time $O(n \log n)$ overall. □

6.3.4 Computing potential watersheds in linear time

For grid terrains, the maximal node degree is constant. Thus, the slope diagram computations take only $O(1)$ time per expansion. In fact, since all nodes which are added to the priority queue will eventually become part of the computed watershed, we could actually compute $\mathcal{W}_\cup(Q)$ in $O(k \log k)$ time, where $k = |\mathcal{W}_\cup(Q)|$. Alternatively, we can use the techniques from Henzinger et al. [92] for shortest paths to overcome the priority queue bottleneck, as explained in the proof below.

Theorem 6.3.10 *The canonical realization of the potential watershed of a set of cells Q in an imprecise grid terrain of n cells can be computed in $O(n)$ time.*

Proof: The computation of potential watersheds in Section 6.3.3 has much in common with computing single-source shortest paths. In both cases, the goal is to compute a label $\delta(v)$ for each node v: in the case of potential watersheds it is the lowest elevation such that a flow path to a given destination q exists; in the case of shortest paths it is the distance from the given source q. During the computation, we maintain *tentative* labels $d[v]$ for each node v which are upper bounds on the labels to be computed. (The tentative label of a node that has not been discovered yet would be ∞.) The computations consist of a sequence of edge *relaxations*: when relaxing a directed edge (u, v), we try to improve (that is, lower) $d[v]$ based on the current value of $d[u]$, which is an upper bound on $\delta(u)$. Both problems share some crucial properties: for every node v that can be reached, there is a "shortest" path $\pi(v) = u_0, u_1, ..., u_k$ where $u_0 = q$ and $u_k = v$, the correct labels $\delta(u_0), \delta(u_1), ..., \delta(u_k)$ form a non-decreasing sequence, and when the edges on this path are relaxed in order from (u_0, u_1) to (u_{k-1}, u_k), the relaxation of (u_{i-1}, u_i) will correctly set $d[u_i]$ equal to $\delta(u_i)$. All that is necessary to compute all labels is that the sequence ρ of relaxations performed by the algorithm contains $\pi(v)$ as a subsequence, for each v. Note that the edges of $\pi(v)$ do not need to be consecutive in ρ: the labels along $\pi(v)$ are computed correctly even if the relaxations of $\pi(v)$ are interleaved with relaxations of other edges, or even with out-of-order relaxations of edges of $\pi(v)$.

There are several algorithms to find a sequence of relaxations ρ in the above setting, such that for every node v, the sequence ρ contains the relaxations of a shortest path $\pi(v)$ as a subsequence. These algorithms are usually known as algorithms to compute (single-source) shortest paths, but they can also be applied directly to the more general setting described above. Dijkstra's algorithm finds a sequence of relaxations that is optimal in the sense that it relaxes each edge only once. However, to achieve this, the algorithm needs $\Theta(n)$ operations on a priority queue of size $\Theta(n)$ in the worst case, where n is the number of nodes and edges in the graph [44].

An alternative is the algorithm of Henzinger et al. [92]. This algorithm uses a hierarchy of priority queues. Most priority queue operations in this algorithm are on small priority queues. The algorithm needs more relaxations than Dijkstra's algorithm, but still not more than $O(n)$. Provided the relaxations take constant time each, the whole algorithm runs in $O(n)$ time. However, the algorithm by Henzinger et al. only works if a recursive decomposition of the graph is provided that satisfies certain properties. Fortunately such decompositions can be found in $O(n)$ time for planar graphs, and also for certain other types of graphs. In particular, it is easy to construct such a decomposition for a graph that represents a grid terrain model, even in the model where each cell can drain to one or more of its eight neighbors, for which the adjacency graph is non-planar. Let $r_1 < r_2 < ...$ be a sequence of powers of four. Now we can easily make a decomposition of the graph into square regions of $\sqrt{r_1} \times \sqrt{r_1}$ nodes; we group these together into regions of $\sqrt{r_2} \times \sqrt{r_2}$ regions, etc., generally grouping regions of $\sqrt{r_i} \times \sqrt{r_i}$ nodes into regions of $\sqrt{r_{i+1}} \times \sqrt{r_{i+1}}$ nodes (some regions at the boundary of the whole input grid may be slightly smaller). On each level i, the regions have size $\Theta(r_i)$ and each region has $\Theta(\sqrt{r_i})$ nodes on its boundary, thus each level forms a so-called r_i-division. We choose the region sizes such that they satisfy Equation (19) from Henzinger et al.

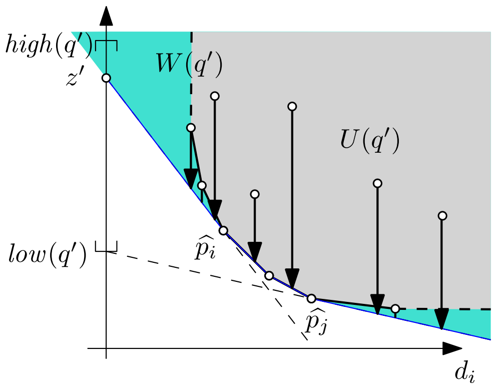


Figure 6.8: Computations in the slope diagram

With this decomposition, the structure of the single-source shortest paths algorithm by Henzinger et al. can also be applied to the computation of potential watersheds on grid terrains. For grid terrains, $d_{\max} = O(1)$, and thus, the computation of the slope diagrams and the $O(n)$ relaxation steps from the "shortest-paths" algorithm take only $O(n)$ time. Together with $O(n)$ time for priority queue operations, we get a total running time of $O(n)$. □

6.3.5 Potential downstream areas

Similar to the potential watershed of a set Q, we can define the set of points that potentially *receive* water from a node in Q. Let the ***potential downstream area*** of Q be defined as:

$$\mathcal{D}_\cup(Q) = \bigcup_{R \in \mathcal{R}_T} \bigcup_{q \in Q} \{p : q \xrightarrow[R]{} p\}.$$

Naturally, a canonical realization for this set does not necessarily exist. Nevertheless, the potential downstream area can be computed in a similar way as described in Section 6.3.3. The difference is that we will now process nodes in *decreasing* order of their *maximal* elevation such that they could still *receive* water from a node in Q. The algorithm is the same as Algorithm 6.3.5, except that in the first line the nodes are enqueued with their highest possible elevation, in line 3 we dequeue the current node with the largest key and we use the following subroutine in line 6:

Definition 6.3.11 Let **ExpandDown**(q', z') denote a function that returns for a node q' and an elevation $z' \in [low(q'), high(q')]$ a set of pairs of nodes and elevations, which includes the pair (p, z) if and only if $p \in N(q')$, there is a realization R with $elev_R(q') \in [low(q'), z']$ such that $q' \xrightarrow[R]{} p$, and z is the maximum elevation of p over all such realizations R.

Lemma 6.3.12 *We can compute the function* **ExpandDown**(q', z') *in* $O(d \log d)$ *time, where* d *is the node degree of* q'.

Proof: Consider the slope diagram of q' as defined in Section 6.3.3.3. Let z_0 be $\min \mathit{high}(p)$ over all neighbors p of q'; note that this is the vertical coordinate of the lowermost point of $U(q')$. Let $\widehat{q'} = (0, z')$ and consider its lower tangent to $U(q')$. Let $\widehat{p_i}$ be the corner of $U(q')$ that intersects the tangent. Similarly, let $\widehat{p_j}$ be the corner of $U(q')$ that intersects the tangent through $(0, \max(\mathit{low}(q'), z_0))$. Let $W(q')$ be the intersection of the halfplanes above these two tangents and the halfplanes $H_i, \ldots, H_j$ as defined in Section 6.3.3.3. Clearly, a neighbor of q' can have a steepest-descent edge from q', for some elevation of q' in $[\mathit{low}(q'), z']$, if and only if its representative in the slope diagram lies below $W(q')$ or on the boundary of $W(q')$. To compute the neighbors of q' and their elevations as they should be returned by **ExpandDown**(q', z'), we test each neighbor p of q' as follows. We find the point $\widehat{p'} = (|pq'|, z)$ that is the projection from $\widehat{p}$ down onto the boundary of $W(q')$. If $z \geq \mathit{low}(p)$, we return (p, z), otherwise we do not include p in the result.

The slope diagram with $W(q')$ can be computed $O(d \log d)$ time. The neighbors p of q' can be sorted by increasing distance from q' in the xy-projection in $O(d \log d)$ time; after that, the projections of all points $\widehat{p}$ can be computed in $O(d)$ time in total by handling them in order of increasing distance from q' and walking along the boundary of $W(q')$ simultaneously. □

Theorem 6.3.13 *Given a set of nodes Q of an imprecise terrain, we can compute the set $\mathcal{D}_\cup(Q)$ in time $O(n \log n)$, where n is the number of edges in the terrain.*

Proof: The algorithm searches the graph starting from the nodes of Q. As in the algorithm for potential watersheds, nodes that have been extracted from the priority queue have a *finalized* elevation; nodes that are currently in the priority queue but were never extracted (yet) have *tentative* elevations. However, this time these elevations are not to be understood as elevations of the nodes in a single realization, but simply as the highest known elevations so that the nodes may be reached from Q.

The induction hypothesis is symmetric to the hypothesis used for potential watersheds: we show that when (p, z) is first extracted from the priority queue, p is indeed contained in the potential downstream area of Q, and the elevation z is the highest possible elevation of p such that water flows from any node in Q to p in any realization. Again, the induction is on the points extracted, in the order in which they are extracted for the first time. The induction hypothesis consists of two parts:

(i) There exists a realization R and $q \in Q$ such that $\mathit{elev}_R(p) = z$, there is a flow path π from q to p in R, and π only visits nodes that have been extracted from the priority queue.

(ii) There exists no realization R and $q \in Q$ such that $\mathit{elev}_R(p) > z$ and $q \xrightarrow[R]{} p$.

If a node $p \in Q$ is extracted with $z = \mathit{high}(p)$, then the claims hold trivially. Note that the first extraction from the priority queue must be of this type.

If p is extracted from the priority queue for the first time and $p \notin Q$, then there must be at least one node p' that was extracted earlier, such that **ExpandDown**(p', z'), for some elevation z', resulted in p having the tentative elevation z. By induction, there exists a realization R' and $q \in Q$, such that $\mathit{elev}_{R'}(p') = z'$, there is a flow path π from q to p' in R', and π does not include p.

So far the proof is basically symmetric to that of Theorem 6.3.9. However, to see (i), we need a different construction. Let $z'' \leq z'$ be an elevation such that water flows from p' to p in the realization R'' with $elev_{R''}(p') = z''$, $elev_{R''}(p) = z$, and $elev_{R''}(p'') = high(p'')$ for all other nodes p''. Note that z'' exists by definition of **ExpandDown**. We now construct a realization R by modifying R' as follows: we set $elev_R(p') = z''$, we set $elev_R(p) = z$, and we set $elev_R(r) = high(r)$ for each neighbor r of p' such that $r \neq p$ and r does not lie on π. In comparison to R', only two nodes in R may have lower elevation, namely p and p'. Therefore, water will still flow along the path π from q until it either reaches p', or a node that now has p or p' as a new steepest-descent neighbor. Thus, in any case, there is a flow path from q to either p or p'. If the flow path reaches p', then, by definition of **ExpandDown**, none of the neighbors of p' that are set at their highest elevation can out-compete p as a steepest-descent neighbor of p'. Of course, the neighbors of p' that lie on π cannot out-compete p either, since these neighbors have elevation at least as high as p'. Therefore, p must be a steepest-descent neighbor of p' in R', and water from p' will flow to p. Thus, in any case, water from q will reach p in R' along a path that is a prefix of π, followed by an edge to p. This proves part (i) of the induction hypothesis.

The proof of part (ii) is completely analogous to the proof of Theorem 6.3.9.

It follows that the algorithm outputs all nodes of $\mathcal{D}_\cup(Q)$. The running time analysis is analogous to Theorem 6.3.9. □

6.3.6 Persistent watersheds

In this section we will give a definition of minimal watersheds, and explain how to compute them. Recall that the potential (maximal) watershed of a node set Q is defined as the set of nodes that have some flow path to a node in Q. We can write this as follows:

$$\mathcal{W}_\cup(Q) = \left\{p : \exists\, \pi \in \Pi(\mathcal{R}_T), \pi \ni p\ \exists\, q \in Q : p \xrightarrow{\pi} q\right\}.$$

An analogous definition that would be consistent with the intuitive idea of a minimal watershed would be:

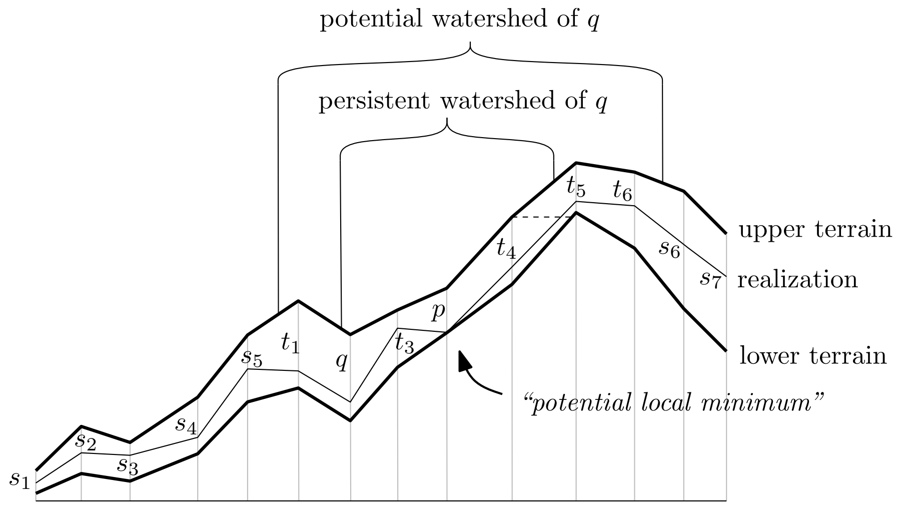
$$\mathcal{W}_\cap(Q) := \left\{p : \forall\, \pi \in \Pi^+(\mathcal{R}_T), \pi \ni p\ \exists\, q \in Q : p \xrightarrow{\pi} q\right\}, \tag{6.1}$$

where $\Pi^+(\mathcal{R}_T)$ is the set of all maximal flow paths. Intuitively, $\mathcal{W}_\cap(Q)$ is the set of nodes p from which water flows to Q via *any* induced maximal flow path that contains p. We call this the ***minimal watershed*** of Q.

However, this definition seems not very useful. Consider a measuring device with a constant elevation error, which is used to sample points in a gently descending valley. Increasing the density of measurement points has the effect that eventually all imprecision intervals of neighboring nodes overlap in the vertical dimension. Thus, each node could become a local minimum in some realization. Now, water flowing down the valley could, theoretically, "get stuck" at any point, and thus, the minimal watershed of any point q in this valley would contain nothing but q itself, see Example 6.9. [1]

[1]This phenomenon has also been observed in practice. Firstly, Hebeler et al. [91] observe that the watershed is more sensitive to elevation error in "flatlands". Secondly, simulations have shown that also potential local minima or "small sub-basins" can severely affect the outcome of hydrological computations [109].

Example 6.9 The persistent and potential watersheds of a node q.



An example of a 1.5 dimensional imprecise terrain, where the minimal watershed of q can be arbitrarily reduced by oversampling. In fact, the minimal watershed of q only contains q. Flow from any other node can get stuck in a potential local minimum. An example is the node p. Note that p cannot be in the minimal watershed of any other node. The complement of $\mathcal{W}_\cup(q)$ is the set $S = \{s_1, ..., s_7\}$. The q-avoiding potential watershed $\mathcal{W}_\cup^{\setminus q}(S)$ contains t_1 (because water from t_1 may flow directly to s_5) and t_5 and t_6 (because water from t_5 and t_6 may flow to s_6). The points q, t_3, p, t_4, t_5 are not in $\mathcal{W}_\cup^{\setminus q}(S)$, as water from there can only reach S by first flowing to q before reaching s_5. Thus, $\{q, t_3, p, t_4, t_5\}$ constitutes the persistent watershed $\mathcal{W}_\cap(q)$.

Nevertheless, it seems clear that any water flowing in the valley must eventually reach q (possibly after flooding some local minima in the valley), since the water has nowhere else to go. This leads to an alternative definition of a minimal watershed, after we rewrite the definition of the minimal watershed from Eq. (6.1) slightly. Observe that the following holds for the complement of the minimal watershed.

$$(\mathcal{W}_\cap(Q))^c = \left\{p : \exists\, \pi \in \Pi(\mathcal{R}_T), \pi \ni p\ \neg\exists\, q \in Q : p \xrightarrow{\pi} q\right\} \tag{6.2}$$

Thus, the minimal watershed of Q is the complement of the set of nodes p, for which it is possible that water follows a flow path from p that does *not* lead to Q. Assume there exists a suitable set of *alternative destinations* S, such that we can rewrite Eq. (6.2) as follows:

$$(\mathcal{W}_\cap(Q))^c = \left\{p : \exists\, \pi \in \Pi(\mathcal{R}_T), \pi \ni p\ \exists\, s \in S : (p \xrightarrow{\pi} s) \wedge (\pi \cap Q = \emptyset)\right\}. \tag{6.3}$$

Note that the right hand side of Eq. (6.3) is equivalent to the set:

$$\mathcal{W}_\cup^{\setminus Q}(S) := \bigcup_{\pi\in\Pi(\mathcal{R}_T)} \bigcup_{s\in S} \{p : (p \xrightarrow{\pi} s) \wedge (\pi \cap Q = \emptyset)\} \tag{6.4}$$

We call the set in Eq. (6.4) the ***Q-avoiding potential watershed*** of a set of nodes S and we denote it with $\mathcal{W}_{\cup}^{\setminus Q}(S)$. This is the set of nodes that have a potential flow path to a node $s \in S$ that does not pass through a node of Q before reaching s.

It remains to identify the set of alternative destinations S. Since each flow path can be extended until it reaches a local minimum, the set of potential local minima clearly serves as such a set of destinations. Let $V_{\min}^{\setminus Q}$ be the union of all sets $P \subseteq V$ such that there exists a realization in which P is a local minimum and $P \cap Q = \emptyset$. However, it is also safe to include the nodes that do not form local minima but that do not have any flow path to Q. This is the complement of the set $\mathcal{W}_{\cup}(Q)$. It follows for the minimal watershed:

$$\mathcal{W}_{\cap}(Q) = \Big(\mathcal{W}_{\cup}^{\setminus Q} \Big(V_{\min}^{\setminus Q} \cup (\mathcal{W}_{\cup}(Q))^c \Big) \Big)^c$$

Note that we can rewrite this as follows:

$$\mathcal{W}_{\cap}(Q) = \Big(\mathcal{W}_{\cup}^{\setminus Q} ((\mathcal{W}_{\cup}(Q))^c) \Big)^c \setminus \mathcal{W}_{\cup}^{\setminus Q} \Big(V_{\min}^{\setminus Q} \cap \mathcal{W}_{\cup}(Q) \Big)$$

Based on the above considerations we suggest the following alternative definition of a minimal watershed.

Definition 6.3.14 The ***persistent watershed*** of a set of nodes Q is defined as

$$\mathcal{W}_{\Cap}(Q) := \Big(\mathcal{W}_{\cup}^{\setminus Q} ((\mathcal{W}_{\cup}(Q))^c) \Big)^c .$$

This is the complement of the set of nodes that have a potential flow path to a node outside the potential watershed of Q without passing through Q. See Example 6.9: the persistent watershed of q consists of the nodes that can never be high enough so that water from those nodes could escape from the potential watershed of q on the right; water from these nodes can only escape from the potential watershed of q by first flowing down to q.

To compute the persistent watershed efficiently, all we need are efficient algorithms to compute potential watersheds and Q-avoiding potential watersheds. We have already seen how to compute $\mathcal{W}_{\cup}(Q)$ efficiently in Section 6.3.3. Note that the Q-avoiding potential watershed of S is different from the potential watershed of S in the terrain T' that is obtained by removing the nodes Q and their incident edges from T. The next lemma states that we can also compute Q-avoiding potential watersheds efficiently nonetheless.

Lemma 6.3.15 *There is an algorithm which outputs the Q-avoiding potential watershed of S and takes time $O(n \log n)$, where n is the number of edges of the terrain.*

Proof: We modify the algorithm to compute the potential watershed of S (Algorithm 6.3.5), such that, each time the algorithm extracts a node from the priority queue, this node is discarded if it is contained in Q. Instead, the algorithm continues with the next node from the priority queue. Clearly, this algorithm does not follow any potential flow paths that flow through Q. However, the nodes of Q are still being considered by the neighbors of its neighbors as a node they have to compete against for being the steepest-descent neighbor. It is easy to verify that the proof of Theorem 6.3.9 also holds for the computation of Q-avoiding potential watersheds. □

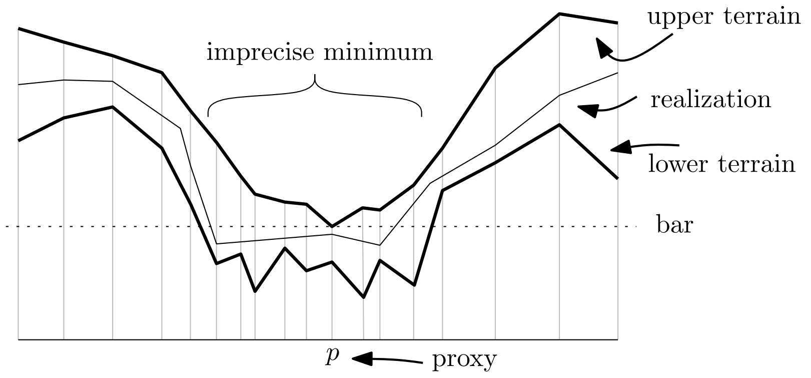


Figure 6.10: Example of an imprecise minimum with a proxy p in a non-regular terrain.

By applying Theorem 6.3.9 and Lemma 6.3.15, we obtain:

Theorem 6.3.16 *We can compute the persistent watershed $\mathcal{W}_{\cap}(Q)$ of Q in time $O(n \log n)$, where n is the number of edges of the terrain.*

6.4 Regular terrains

We can still extend the results on imprecise watersheds in the network model. However, our extensions only hold for a certain class of imprecise terrains, which we call "regular". We will first define this class and characterize it. To this end we will introduce the notion of imprecise minima (see Definition 6.4.1), which are the "stable" minima of an imprecise terrain, regular or non-regular. In Section 6.4.2 we will describe how to compute these minima and how to turn a non-regular terrain into a regular terrain. In the remaining sections, we discuss nesting properties and fuzzy boundaries of imprecise watersheds. Furthermore, we observe that regular terrains have a well-behaved ridge structure, which delineates the main watersheds.

The main focus of this section is on the extension of the results in Section 6.3. Some of the concepts introduced here could also be applied to the surface model, however, we confine our discussion to the network model.

6.4.1 Characterization of regular terrains

We first give a definition of a proper minimum in an imprecise terrain.

Definition 6.4.1 A set of nodes S in an imprecise terrain T is an ***imprecise minimum*** if and only if in every realization of T, the set S contains a local minimum, and no proper subset of S has this property.

Note that the local minima contained in S can vary from one realization to another. Now a regular imprecise terrain is defined as follows:

Definition 6.4.2 An imprecise terrain T is a ***regular imprecise terrain*** if and only if every local minimum of the lowermost realization R^- of T is an imprecise minimum of T.

Any imprecise minimum S on a regular terrain is a local minimum in R^-. Indeed, assume S would not be a minimum on R^-. Then, by Definition 6.4.1, it would still contain a proper subset S' that is a minimum on R^-. By Definition 6.4.2, S' must be an imprecise minimum of T, but this contradicts Definition 6.4.1. Now, we observe:

Lemma 6.4.3 *Let S be an imprecise minimum on a regular terrain. Then each node $s \in S$ has the same elevation lower bound $low(s)$. Furthermore, for each non-empty subset $S' \subset S$ we have $\mathcal{W}_{\cup}(S') = \mathcal{W}_{\cup}(S)$ and $\mathcal{W}_{R^-}(S') = \mathcal{W}_{R^-}(S)$.*

Proof: For the sake of contradiction, suppose not all nodes of S have the same elevation lower bound. Then there would be a proper subset S' of S that is a local minimum of R^-, and thus, by definition of a regular terrain, S' would be an imprecise minimum which is at the same time a proper subset of S. But this would, by Definition 6.4.1, contradict that S is an imprecise minimum. Therefore, each node $s \in S$ has the same elevation lower bound.

Now, in $R_{\cup}(S)$, all nodes $s \in S$ are at their lowermost elevation and thus S is a local minimum in $R_{\cup}(S)$. Thus, all nodes p that have a flow path to *any* node $s \in S$, have a flow path to *each* node $s \in S$, and thus each non-empty subset $S' \subset S$ has $\mathcal{W}_{\cup}(S') = \mathcal{W}_{\cup}(S)$. By the same argument, we have $\mathcal{W}_{R^-}(S') = \mathcal{W}_{R^-}(S)$. □

We will now derive a characterization of imprecise minima in general. For this, we introduce proxies.

Definition 6.4.4 A ***proxy*** of an imprecise minimum S is a node $p \in S$, such that there are no realizations R and nodes $q \notin S$ such that $p \underset{R}{\rightarrow} q$.

Thus, water that arrives in a proxy of an imprecise minimum S, can never leave S anymore. This implies that the proxy is not in the potential watershed of any set of nodes that lies entirely outside S. The following lemma states that every imprecise minimum contains a proxy.

Lemma 6.4.5 *Let the* **bar** *of a set S be $bar(S) = \min_{s \in S} high(s)$. A set S is an imprecise minimum if and only if (i) $bar(S) < \min_{t \in N(S)} low(t)$ and (ii) no proper subset S' of S has this property. Every imprecise minimum has a proxy.*

Proof: We first argue that if S is an imprecise minimum, then this implies (i) and (ii) for S.

To prove (i), consider the following realization R: For all nodes $r \in S$ we set $elev_R(r) = \max(bar(S), low(r))$, and for all nodes $t \in N(S)$ we set $elev_R(t) = low(t)$. Assume for the sake of contradiction that (i) would not hold. Then there exists a node $t \in N(S)$ which lies at elevation at most $bar(S)$ in R. Now, if all nodes of S would have the same elevation in R, then S would either be part of a local minimum that includes t, or S would have t as a lower neighbor: in either case, in the realization R the set S would neither be a local minimum by itself nor include a local minimum, contradicting the assumption that S is an imprecise minimum. Therefore, since S is an imprecise minimum, it must be that not all nodes of S have the same elevation, and there exists a proper subset $S' \subset S$ which is a local minimum in R. Like all nodes of S, the local minimum S' must have elevation at least $\mathit{bar}(S)$; each node $t \in N(S')$ must be set at a higher elevation $\mathit{low}(t)$. If we would remove the nodes of $N(S')$ from S, the imprecise minimum S would be separated into several components, including at least one component S'' that contains a node s with $\mathit{high}(s) = \mathit{bar}(S)$. This component S'' is a proper subset of S. Its neighborhood $N(S'')$ consists of nodes from $N(S)$ and $N(S')$, all of which have an elevation lower bound strictly above $\mathit{bar}(S) = \min_{s \in S''} \mathit{high}(s)$. Thus $S'' \subset S$ must contain a local minimum in any realization, contradicting the assumption that S is an imprecise minimum. Therefore the assumption that (i) would not hold must be wrong, and (i) must hold.

To prove (ii), assume, for the sake of contradiction, that S contains a proper subset S' such that $\mathit{bar}(S') < \min_{t \in N(S')} \mathit{low}(t)$. Thus, S' would contain a local minimum in any realization, and S would not be an imprecise minimum; hence (ii) must hold for S.

Now we argue that, if (i) and (ii) are met, then S is an imprecise minimum. Observe that condition (i) implies that S contains a local minimum in any realization. Now assume, for the sake of contradiction, that there exists a proper subset S' that always contains a local minimum. Let S' be a smallest such subset of S. We have that S' is an imprecise minimum, and therefore, as we proved above, it holds that $\mathit{bar}(S') < \min_{t \in N(S')} \mathit{low}(t)$, which contradicts that condition (ii) holds for S. Hence, there is no proper subset S' of S that always contains a local minimum; therefore S is an imprecise minimum.

As a proxy of an imprecise minimum S, we take any node s such that $\mathit{high}(s) < \min_{t \in N(S)} \mathit{low}(t)$. By condition (i) of the lemma, such a node s always exists. Since s lies below any node of $N(S)$ in any realization, there are no realizations R and nodes $q \notin S$ such that $s \xrightarrow[R]{} q$; thus s is a proxy of S. $\square$

6.4.2 Computing proxies and regular terrains

Any imprecise terrain can be turned into a regular imprecise terrain by raising the lower bounds on the elevations such that local minima that violate the regularity condition are removed from R^-. Indeed, in hydrological applications it is common practice to preprocess terrains by removing certain local minima before doing flow computations [143]. To do so while still respecting the given upper bounds on the elevations, we can make use of the algorithm by Gray et al. [81]. The original goal of this algorithm is to compute a realization of a surface model that minimizes the number of local minima in the realization, but the algorithm can also be applied to a network model. It can easily be modified to output a proxy for each imprecise minimum of a terrain. Moreover, the realization M computed by the algorithm has the following convenient property: if we change the imprecise terrain by setting $\mathit{low}(v)$ to $\mathit{elev}_M(v)$ for each node, we obtain a regular imprecise terrain. We denote the algorithm with **Regularize**.

The algorithm. The algorithm proceeds as follows. We will sweep a horizontal plane upwards. During the sweep, any node is in one of three states. Initially, each node is *undiscovered.* Once the sweep plane reaches $low(v)$, the state of the node changes to *pending.* Pending nodes are considered to be at the level of the sweep plane, but they may still be raised further. During the sweep, we will always maintain the connected components of the graph induced by the nodes that are currently pending; we call this graph G_P. As soon as it becomes clear that a node cannot be raised further or does not need to be raised further, its final elevation on or below the sweep plane is decided and the node becomes *final.* The algorithm is driven by two types of events: we may reach $low(v)$ for some node v, or we may reach $high(v)$ for some node v. These events are handled in order of increasing elevation; $low(v)$-events are handled before $high(v)$-events at the same elevation. The events are handled as follows:

- reaching $low(v)$: we make v pending, and find the component S of G_P that contains v. If v has a neighbor that is final, we make all nodes of S final at elevation $low(v)$.
- reaching $high(v)$: if v is final, nothing happens; otherwise we report v as a proxy, we find the connected component S of G_P that contains v, and we make all nodes of S final at elevation[2] $\max_{s\in S} low(s)$.

Gray et al. explain how to implement the algorithm to run in $O(n \log n)$ time [81].

Lemma 6.4.6 *Given an imprecise terrain T, (i) all nodes reported by the algorithm* **Regularize**(T) *described above are proxies of imprecise minima, and (ii) the algorithm reports exactly one proxy of each imprecise minimum of T.*

Proof: We first prove the second part, and then the first part of the lemma.

(ii) Let S be an imprecise minimum. Let v be the node in S which was the first to have its $high(v)$-event processed. By Lemma 6.4.5, v is a proxy of S and we have $high(v) < min_{t\in N(S)} low(t)$. Hence, when $high(v)$ is processed, the component of G_P that contains v does not contain any nodes outside S, and the $high(v)$-event is the first event to make any nodes in this component final. Thus, v is reported as a proxy. Furthermore, no node $s \in S$ can have $low(s) > high(v)$, otherwise $bar(S \setminus \{s\}) = high(v) < \min_{t\in\{s,N(S)\}} low(t) \leq \min_{t\in N(S\setminus\{s\})} low(t)$, and thus, by Lemma 6.4.5, S would not be an imprecise minimum. Hence, when the $high(v)$-event is about to be processed, all nodes of S have been discovered and are currently pending. The $high(v)$-event makes all nodes of S final; thus, any $high(s)$-events for other nodes $s \in S$ will remain without effect and no more proxies of S will be reported.

(i) Let v be a node that is reported as a proxy in a $high(v)$-event. We claim that the connected component S of G_P that contains v at that time, is an imprecise minimum. Indeed, by definition of G_P, all nodes of S are pending, and thus $high(v) = \min_{s\in S} high(s) = bar(S)$. Furthermore, because S is a connected component of G_P, all nodes $t \in N(S)$ must be either undiscovered or final. In fact, the algorithm maintains the invariant that no neighbor of a finalized node is pending; since all nodes in S are pending, all nodes $t \in N(S)$ must be undiscovered. Therefore $high(v) \leq \min_{t\in N(S)} low(t)$. Because all $low(t)$-events at the same elevation as

[2]This is a small variation: the algorithm as described originally by Gray et al. would make the elevations final at $high(v) = \min_{s\in S} high(s)$. However, in the current context we prefer to make the elevations final at $\max_{s\in S} low(s)$, to maintain as much of the imprecision in the original imprecise terrain as possible.

high(v) are processed before the *high*(v)-event is processed, we actually have a strict inequality: $high(v) < \min_{t \in N(S)} low(t)$. It follows that S satisfies condition (i) of Lemma 6.4.5. Furthermore, no proper subset S' of S has this property, otherwise, by the analysis given above, a proxy for S' would have been reported already and the nodes from S' would have been removed from G_P at that time. Hence, S also satisfies condition (ii) of Lemma 6.4.5, and S is an imprecise minimum, with v as a proxy. □

Lemma 6.4.7 *Let M be the realization of a terrain T as computed by the algorithm* **Regularize**(T)*. Let T' be the imprecise terrain that is obtained from T by setting $low(v) = elev_M(v)$ for each node v. The terrain T' is a regular imprecise terrain.*

Proof: Note that M is the lowermost realization of T'. Consider any local minimum S of M. Observe that the algorithm cannot have finalized the elevations of the last pending nodes of S in a *low*(v)-event, because then we would have $v \in S$ and v must have a neighbor $t \notin S$ that was finalized before v; hence $elev_M(t) \leq elev_M(v)$ and S would not be a local minimum. Therefore, the algorithm must have finalized the last elevations of the nodes of S in a *high*(v)-event for a node $v \in S$. Furthermore, each node $t \in N(S)$ must have been undiscovered at that time; otherwise t would have become part of the same component as the nodes of S before its elevations were finalized, or t would have been finalized before v: in both cases S would not be a local minimum. Hence we have $low(t) > high(v)$ for each node $t \in N(S)$, and thus, S is a local minimum in every realization of T or T'. Furthermore, no proper subset of S' of S contains a local minimum in every realization of T', since in particular, in M the set S is a local minimum and therefore no proper subset S' of S is a local minimum. Thus, by Definition 6.4.1 and Definition 6.4.2, T' is a regular terrain. □

6.4.3 Nesting properties of imprecise watersheds

To be able to design data structures that store imprecise watersheds and answer queries about the flow of water between nodes efficiently, it would be convenient if the watersheds satisfy the following ***nesting condition***: if p is contained in the watershed of q, then the watershed of p is contained in the watershed of q. Clearly, potential watersheds do not satisfy this nesting condition, while minimal watersheds do. However, in general, persistent watersheds are not nested in this way. We give a counter-example that uses a non-regular terrain in the next lemma, before proving the nesting condition for persistent watersheds in regular terrains later in this section.

Lemma 6.4.8 *There exists an imprecise terrain with two nodes p and q such that $p \in \mathcal{W}_\cap(q)$ and $\mathcal{W}_\cap(p) \not\subseteq \mathcal{W}_\cap(q)$.*

Proof: We give an example of a non-regular terrain that has this property. Refer to Figure 6.11. The persistent watershed of p as shown in red is not completely contained in the persistent watershed of q as shown in blue. The left figure gives a top-view. All edges have unit length, except for the edge between w and q. The right figure shows the fixed elevations of s, t, t', u, v and w, the elevation intervals of p, q and r, and the correct horizontal distances on all edges except $|pv|$ and $|qv|$. The red outline delimits $\mathcal{W}_\cup(p) = \{p, q, r, s, t, v, w\}$. The red dashed outline delimits $\mathcal{W}_\cap(p) = \{p, s, v\}$. The blue outline delimits $\mathcal{W}_\cup(q) = \{p, q, s, v, w\}$. The blue dashed outline delimits $\mathcal{W}_\cap(q) = \{p, q, v\}$. □

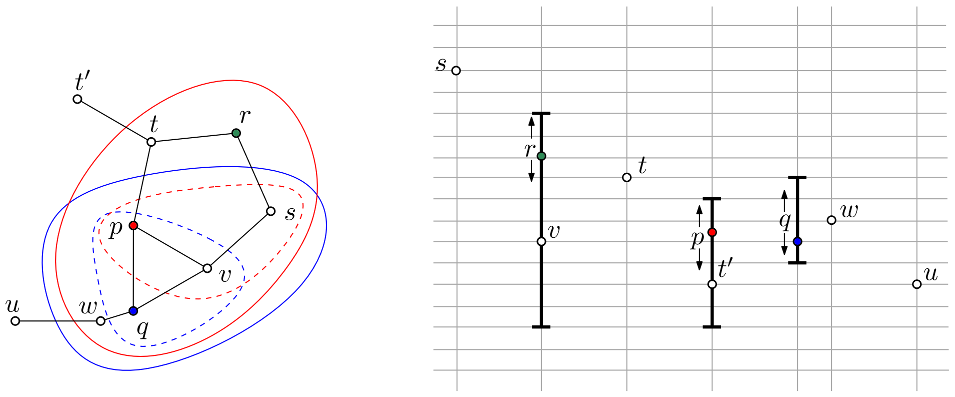


Figure 6.11: An example of a non-regular terrain where persistent watersheds are not properly nested.

The following lemmas will prove that on regular imprecise terrains persistent watersheds do satisfy the nesting condition.

Lemma 6.4.9 *Let Q be a set of nodes of a regular imprecise terrain, then $\mathcal{W}_{\Cap}(Q) \subseteq \mathcal{W}_{R^-}(Q)$.*

Proof: Assume for the sake of contradiction that there exists a maximal flow path π induced by R^- which starts at a node $p \in \mathcal{W}_{\Cap}(Q)$ and has the following properties:
(i) π does not flow to any node in Q, and
(ii) π flows to a local minimum S of R^-, where $S \cap Q = \emptyset$.

Because the terrain is regular, S must be an imprecise minimum (by Definition 6.4.2), and therefore S must have a proxy s by Lemma 6.4.5. Without loss of generality, assume that π flows to this proxy. As observed above, s is not in the potential watershed of any set of nodes outside S; in particular, s is not in $\mathcal{W}_{\cup}(Q)$, see Figure 6.12 (left). This implies that π leaves $\mathcal{W}_{\cup}(Q)$ without going through any node of Q. This contradicts the fact that $p \in \mathcal{W}_{\Cap}(Q)$, since by the definition of persistent watersheds, π cannot leave $\mathcal{W}_{\cup}(Q)$ without going through Q. Thus, π cannot exist. In particular, this implies that any flow path induced by R^-, which starts at a node $p \in \mathcal{W}_{\Cap}(Q)$ and ends in a local minimum of R^-, has to flow to a node of Q. Since we can extend any flow path until it reaches a local minimum, this implies that p is contained in $\mathcal{W}_{R^-}(Q)$ and thus $\mathcal{W}_{\Cap}(Q) \subseteq \mathcal{W}_{R^-}(Q)$. □

Lemma 6.4.10 *Let Q be a set of nodes of an imprecise terrain, and let $P \subseteq \mathcal{W}_{R^-}(Q)$. Then $\mathcal{W}_{\cup}(P) \subseteq \mathcal{W}_{\cup}(Q)$.*

Proof: Recall that $R_{\cup}(P)$ is the canonical realization of $\mathcal{W}_{\cup}(P)$ as defined in Section 6.3.3.1. Let $\overline{R}$ be the watershed-overlay of $\mathcal{W}_{R_{\cup}(P)}(P)$ and $\mathcal{W}_{R^-}(Q)$. Consider a node $r \in \mathcal{W}_{\cup}(P)$ and a flow path π from r to a node $p \in P$ in $R_{\cup}(P)$, see Figure 6.12 (center). Let π' be the maximal prefix of π such that the nodes of π' have the same elevation in $R_{\cup}(P)$ and $\overline{R}$, and let π'' be the maximal prefix of π' such that π'' is still a flow path in $\overline{R}$. We distinguish three cases:

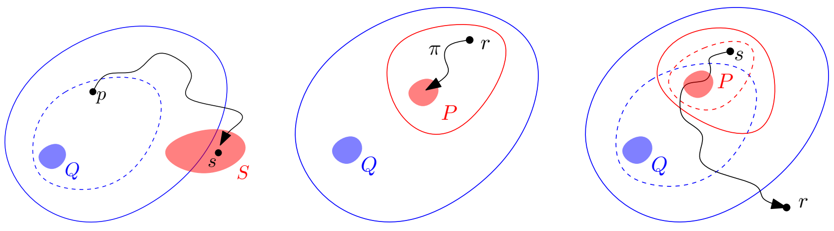


Figure 6.12: Schematic illustrations to the proofs of Lemmas 6.4.9 (left), 6.4.10 (center) and 6.4.11 (right). Potential watersheds are outlined with solid curves, persistent watersheds are outlined with dashed curves.

- If both π' and π'' are empty, then r has higher elevation in $R_{\cup}(P)$ than in $\overline{R}$, so r must be in $\mathcal{W}_{R^-}(Q)$.
- If $\pi' = \pi'' = \pi$, then flow from r reaches a node $p \in P \subseteq \mathcal{W}_{R^-}(Q)$ in $\overline{R}$.
- Otherwise, let (u, v) be the edge of π such that u is the last node of π''. Now v is not on π'', so in $\overline{R}$, flow from u either still follows (u, v) but $elev_{\overline{R}}(v) < elev_{R_{\cup}(P)}(v)$, or flow from u is diverted over an edge $(u, \widehat{v})$ to another node $\widehat{v}$ with $elev_{\overline{R}}(\widehat{v}) < elev_{R_{\cup}(P)}(\widehat{v})$. In either case, from u we follow an edge to a node of which the elevation in $\overline{R}$ is lower than in $R_{\cup}(P)$; therefore this must be a node of $\mathcal{W}_{R^-}(Q)$.

In any case, there is a flow path from r to a node of $\mathcal{W}_{R^-}(Q)$. From here, there must be a path to a node $q \in Q$, since every flow path within $\mathcal{W}_{R^-}(Q)$ in R^- is also a flow path in $\overline{R}$. Thus there is a flow path from r to q in $\overline{R}$, and thus, $r \in \mathcal{W}_{\cup}(Q)$. This proves the lemma. $\square$

Lemma 6.4.11 *(Persistent watersheds are nested) Let Q be a set of nodes of a regular imprecise terrain, and let $P \subseteq \mathcal{W}_{\Cap}(Q)$. Then $\mathcal{W}_{\Cap}(P) \subseteq \mathcal{W}_{\Cap}(Q)$.*

Proof: Assume for the sake of contradiction that there exists a node $s \in \mathcal{W}_{\Cap}(P)$, such that $s \notin \mathcal{W}_{\Cap}(Q)$. By definition, $s \in \mathcal{W}_{\cup}(P)$. By Lemma 6.4.9, $s \in \mathcal{W}_{R^-}(P)$ and by Lemma 6.4.10 $\mathcal{W}_{R^-}(P) \subseteq \mathcal{W}_{\cup}(Q)$. Thus, it also holds that $s \in \mathcal{W}_{\cup}(Q)$.

Furthermore, by assumption $s \notin \mathcal{W}_{\Cap}(Q)$ and therefore s must have a flow path π to a point $r \notin \mathcal{W}_{\cup}(Q)$, which does not pass through a node of Q, refer to Figure 6.12 (right). Since $s \in \mathcal{W}_{\Cap}(P)$, π must include a node $p \in P$. The existence of such a flow path from p to r which does not include a node of Q contradicts the fact that $p \in \mathcal{W}_{\Cap}(Q)$. $\square$

6.4.4 Fuzzy watershed boundaries

Lemma 6.4.9 and Lemma 6.4.10 also allow us to compute the set difference between the potential and the persistent watershed of a set of nodes Q efficiently, given only the boundary of the watershed of Q on the lowermost realization of the terrain. We first define these concepts more precisely.

Definition 6.4.12 Given a realization R, and a set of nodes Q, let $\mathcal{X}_R(Q)$ be the set of directed edges (u, v) such that $u \in \mathcal{W}_R(Q)$ and $v \notin \mathcal{W}_R(Q)$. We call $\mathcal{X}_R(Q)$ the ***watershed boundary*** of Q in R. Likewise, we define the ***fuzzy watershed boundary*** of Q as the directed set of edges (u, v) such that $u \in \mathcal{W}_{\cup}(Q)$ and $v \notin \mathcal{W}_{\Cap}(Q)$ and we denote it with $\mathcal{X}_{\cup}(Q)$. We call the set $\mathcal{W}_{\cup}(Q) \setminus \mathcal{W}_{\Cap}(Q)$ the ***uncertainty area*** of this boundary.

We will now discuss how we can compute the uncertainty area of any fuzzy watershed boundary efficiently.

Algorithm 6.4.13 Let **WSBoundary**(Q) denote an algorithm which computes the uncertainty area of the fuzzy watershed watershed boundary of Q as follows. Assume we are given $\mathcal{X}_{R^-}(Q)$. We will compute the set $\mathcal{W}_{\cup}(Q) \setminus \mathcal{W}_{\Cap}(Q)$ with the following modified version of the algorithm **PotentialWS** (Section 6.3.3.2). Instead of initializing the priority queue with the nodes of Q, we initialize in the following way. For each edge $(u, v) \in \mathcal{X}_{R^-}(Q)$, we use the slope diagram of u to determine the minimum elevation z_u of u, such that there is a realization in which water flows on the edge from u to v. If there exists such an elevation z_u, we enqueue u with elevation (and key) z_u. Similarly, we use the slope diagram of v to determine the minimum elevation z_v of v, such that water may flow on the edge from v to u. If z_v exists, we enqueue v with elevation (and key) z_v. After initializing the priority queue in this way, we run **PotentialWS** as written.

Lemma 6.4.14 *If the terrain is regular, the algorithm* **WSBoundary**(Q) *(Algorithm 6.4.13) computes* $\mathcal{W}_{\cup}(Q) \setminus \mathcal{W}_{\Cap}(Q)$*, which is the uncertainty area of the fuzzy watershed boundary of* Q*.*

Proof: Observe, following the proof of Theorem 6.3.9, that for any node p output by **WSBoundary**(Q) there are a realization R_s and a node s which was in the initial queue with elevation z_s, such that $elev_{R_s}(s) = z_s$ and R_s induces a flow path π from p to s. Let (u, v) be the edge of $\mathcal{X}_{R^-}(Q)$ which led to the insertion of $s \in \{u, v\}$ into Q with elevation z_s. Let t be the other node of (u, v), that is, $t \in \{u, v\} \setminus \{s\}$. Let R_t be the realization obtained from R_s by setting $elev_{R_t}(t) = low(t)$. Observe that, by our choice of z_s, the realization R_t now induces a flow path from p to t. We will now argue that (i) $p \in \mathcal{W}_{\cup}(Q)$, and (ii) $p \notin \mathcal{W}_{\Cap}(Q)$.

(i) The existence of R_u implies that $p \in \mathcal{W}_{\cup}(u)$; since $u \in \mathcal{W}_{R^-}(Q)$ (by definition of $\mathcal{X}_{R^-}(Q)$) this implies $p \in \mathcal{W}_{\cup}(Q)$ (by Lemma 6.4.10).

(ii) Let π be a flow path induced by R^- which starts at v and ends in a local minimum S of R^-. By definition of $\mathcal{X}_{R^-}(Q)$, there is no flow path from v to Q on R^-, thus π does not contain any nodes of Q. Furthermore this holds for any flow path π flowing from v to a node in S. Therefore, S and Q are disjoint. By Definition 6.4.2, S is an imprecise minimum and by Lemma 6.4.5 S contains a proxy s, which is, by Definition 6.4.4, not contained in $\mathcal{W}_{\cup}(Q)$. Hence, by Definition 6.3.14, $p \notin \mathcal{W}_{\Cap}(Q)$.

Next, we will argue that if $p \in \mathcal{W}_{\cup}(Q)$ and $p \notin \mathcal{W}_{\Cap}(Q)$, the algorithm will output p. We distinguish two cases.

If $p \in \mathcal{W}_{R^-}(Q)$, then, because $p \notin \mathcal{W}_{\Cap}(Q)$, there must be a flow path on R^- from p to a minimum S that does not contain any node of Q. By Definition 6.4.2, Lemma 6.4.5 and Definition 6.4.4, there will then be a flow path from p to a proxy $s \in S$ that lies outside $\mathcal{W}_{\cup}(Q)$, and thus, outside $\mathcal{W}_{R^-}(Q)$.

If $p \notin \mathcal{W}_{R^-}(Q)$, then, because $p \in \mathcal{W}_{\cup}(Q)$, there must be a realization in which there is a flow path from p to Q, and thus, from p to $\mathcal{W}_{R^-}(Q)$.

In both cases, there is a realization in which there is a flow path from p that traverses an edge $(u, v) \in \mathcal{X}_{R^-}(Q)$, either from u to v or from v to u. The algorithm reports at least all such points p.

This completes the proof of the lemma. □

Note that if k is the total size of the input $(\mathcal{X}_{R^-}(Q))$ and the output $(\mathcal{W}_{\cup}(Q) \setminus \mathcal{W}_{\cap}(Q))$ and d' is the maximum node degree, then **WSBoundary**(Q) runs in $O(k \log k + k \log d')$ time. When a data structure is given that stores the boundaries of watersheds on R^- so that they can be retrieved efficiently, and the imprecision is not too high, this would enable us to compute the boundaries and sizes of potential and persistent watersheds much faster than by computing them (or their complements) node by node with the algorithm **PotentialWS** described in Section 6.3.3.2.

We can use the same idea as above to compute an uncertain area of the watershed boundaries between a set of nodes Q. More precisely, given a collection of nodes Q such that no node $q \in Q$ is contained in the potential watershed of another node $q' \in Q$, we can compute the nodes that are in the potential watersheds of multiple nodes from Q.

Algorithm 6.4.15 Let Q be $\{q_1, ..., q_k\}$ and let G' be the graph induced by the potential watershed of Q. We denote with **PotentialRidge**$(q_1, \ldots, q_k)$ the following algorithm which computes the uncertainty area between the watersheds of the input nodes. The algorithm is essentially the same as the algorithm that computes the uncertainty area of a single watershed's boundary—the main difference is that now we have to start it with a suitable set of edges $\mathcal{X}$ on the fuzzy boundaries *between* the watersheds of the nodes of Q. More precisely, $\mathcal{X}$ should be an edge separator set of G', which separates the nodes of G' into k components $G'_1, ..., G'_k$ such that nodes of each component G'_i are completely contained in $\mathcal{W}_{\cup}(q_i)$.

We obtain $\mathcal{X}$ with the following modification of **PotentialWS**. For each node p we will maintain, in addition to a tentative elevation z, a tentative tag that identifies a node $q \in Q$ such that there is a realization R with $elev_R(p) = z$ and $p \xrightarrow[R]{} q$. We initialize the priority queue of **PotentialWS** with all nodes $q \in Q$, each with tentative elevation $low(q)$ and each tagged with itself. The first time any particular node q' is extracted from the priority queue, we obtain not only its final elevation but also its final tag q from the queue, and each pair $(p, z) \in$ **Expand**(q', z') is enqueued with that same tag q. At the end of **PotentialWS**, we obtain the set of nodes in $\mathcal{W}_{\cup}(Q)$ together with their elevations in the canonical realization $R_{\cup}(Q)$ and with tags, such that any set of nodes tagged with the same tag $q \in Q$ forms a connected subset of $\mathcal{W}_{\cup}(q)$. We now extract the separator set $\mathcal{X}$ by identifying the edges between nodes of different tags.

Having obtained $\mathcal{X}$, we compute the union of the pairwise intersections of the potential watersheds of $q_1, ..., q_k$ as follows. Again, we use **PotentialWS**. This time the priority queue is initialized as follows. For each edge $(u, v) \in \mathcal{X}$, we use the slope diagram of u to determine the minimum elevation z_u of u, such that there is a realization R *with* $elev_R(v) = elev_{R_{\cup}(Q)}(v)$ in which water flows on the edge from u

to v. If there exists such an elevation z_u, we enqueue u with elevation (and key) z_u. Similarly, we use the slope diagram of v to determine the minimum elevation z_v of v, such that water may flow on the edge from v to u *at elevation* $elev_{R_\cup(Q)}(u)$. If z_v exists, we enqueue v with elevation (and key) z_v. After initializing the priority queue in this way, we run **PotentialWS** as written, and output the result.

Lemma 6.4.16 *Given a set of nodes $q_1, \ldots, q_k$ of an imprecise terrain, such that $q_i \notin \mathcal{W}_\cup(q_j)$ for any $i \neq j$ and $1 \leq i, j \leq k$, we can use Algorithm 6.4.15 to compute the set $\bigcup_i \bigcup_{j \neq i} (\mathcal{W}_\cup(q_i) \cap \mathcal{W}_\cup(q_j))$ in $O(n \log n)$ time, where n is the number of edges of the imprecise terrain. The resulting algorithm is denoted with* **PotentialRidge**$(q_1, \ldots, q_k)$.

Proof: The algorithm is described above. The separator set $\mathcal{X}$ is obtained in $O(n \log n)$ time by running the modified version of **PotentialWS** and one scan over the graph to identify edges between nodes with different tags. Computing the union of the pairwise intersections of the potential watersheds of $q_1, ..., q_k$ takes $O(n \log n)$ time again.

By the same arguments as in the proof of Lemma 6.4.14, we can observe the following: for any node p output by the above algorithm, there are an edge $(u, v) \in \mathcal{X}$, a realization R_u with $elev_{R_u}(u) = elev_{R_\cup(Q)}(u)$ and $p \underset{R_u}{\rightarrow} u$, and a realization R_v with $elev_{R_v}(v) = elev_{R_\cup(Q)}(v)$ and $p \underset{R_v}{\rightarrow} v$. Let $q_u, q_v \in Q$ be the nodes of Q with which u and v were tagged, respectively. It follows that there is a flow path from p to q_u in the watershed overlay of $\mathcal{W}_{R_u}(u)$ and $\mathcal{W}_{R_\cup(Q)}(q_u)$, so $p \in \mathcal{W}_\cup(q_u)$. Analogously, $p \in \mathcal{W}_\cup(q_v)$. Since $(u, v) \in \mathcal{X}$, we have $q_u \neq q_v$, so any point p that is output by the algorithm lies in the intersection of the potential watersheds of two different nodes from Q.

Next, we will argue that if p lies in the intersection of the potential watersheds of two different nodes from Q, then the algorithm will output p. Let $q \in Q$ be the node with which p is tagged (hence, $p \in \mathcal{W}_\cup(q)$), and let $q' \in Q, q' \neq q$ be another node from Q such that $p \in \mathcal{W}_\cup(q')$. Consider a flow path π from p to q' in $R_\cup(q')$, and let (r, r') be the edge on π such that r is tagged with a node other than q' while : r' and all nodes following the last occurrence of (r, r') in π are tagged with q'. Note that (r, r') must exist because all nodes of π lie in $\mathcal{W}_\cup(Q)$ and have received a tag, p is tagged with another node than q', and, since none of the nodes of Q lie in each other's potential watersheds, q' is tagged with itself. Therefore (r, r') exists, and $(r, r') \in \mathcal{X}$. Moreover, we have $elev_{R_\cup(Q)}(r') = elev_{R_\cup(q')}(r')$. Therefore r was put in the priority queue with the minimum elevation such that there is a realization R with $elev_R(r') = elev_{R_\cup(q')}(r')$ in which water flows on the edge from r to r'. By induction on the nodes of π from r back to p, it follows that p must eventually be extracted from the priority queue and output.

This completes the proof of the lemma. □

6.4.5 The fuzzy watershed decomposition

In this section we further characterize the structure of imprecise terrains by considering the ridge lines that delineate the "main" watersheds. In fact, the fuzzy watershed boundaries (Definition 6.4.12) of the imprecise minima (Definition 6.4.1) possess a well-behaved ridge structure if the terrain is regular. Consider the following definition of an "imprecise" ridge.

Definition 6.4.17 Let $S_1, \ldots, S_k$ be the imprecise minima of an imprecise terrain. We call the union of the pairwise intersection of the potential watersheds of imprecise minima the ***potential ridge*** of the terrain.

Let S be an imprecise minimum of a regular imprecise terrain. The next lemma testifies that the persistent watershed of any proxy q of S is equal to the intersection of the persistent watersheds of all possible non-empty subsets of S. Therefore, we think of $\mathcal{W}_{\cap}(q)$ as the actual minimal watershed of S, or the minimum associated with S. By Lemma 6.4.3, the potential watersheds of all non-empty subsets of S are equal. Consequently, we think of the fuzzy watershed boundary of q as the fuzzy watershed boundary of S.

Lemma 6.4.18 *Let S be an imprecise minimum on a regular terrain, and let x be any proxy of S. Then $\bigcap_{\emptyset \subset S' \subseteq S} \mathcal{W}_{\cap}(S') = \mathcal{W}_{\cap}(x)$.*

Proof: Let C denote $\bigcap_{\emptyset \subset S' \subseteq S} \mathcal{W}_{\cap}(S')$, the intersection of the persistent watersheds of all non-empty subsets of S. Consider the complement of this set:

$$(C)^c := \left(\bigcap_{\emptyset \subset S' \subseteq S} \mathcal{W}_{\cap}(S') \right)^c = \bigcup_{\emptyset \subset S' \subseteq S} (\mathcal{W}_{\cap}(S'))^c = \bigcup_{\emptyset \subset S' \subseteq S} \mathcal{W}_{\cup}^{\setminus S'} \left((\mathcal{W}_{\cup}(S'))^c \right).$$

By Lemma 6.4.3 we have $\mathcal{W}_{\cup}(S') = \mathcal{W}_{\cup}(x) = \mathcal{W}_{\cup}(S)$ for any non-empty set $S' \subseteq S$, so we have:

$$(C)^c = \bigcup_{\emptyset \subset S' \subseteq S} \mathcal{W}_{\cup}^{\setminus S'} \left((\mathcal{W}_{\cup}(S))^c \right).$$

Now, by Definition 6.4.4, it is impossible for water that reaches x to continue to flow to a node outside of $\mathcal{W}_{\cup}(S)$. Therefore, any flow path to a node outside $\mathcal{W}_{\cup}(S)$ that avoids a non-empty set $S' \subseteq S$, also avoids x, and we have:

$$\mathcal{W}_{\cup}^{\setminus S'} \left((\mathcal{W}_{\cup}(S))^c \right) \subseteq \mathcal{W}_{\cup}^{\setminus x} \left((\mathcal{W}_{\cup}(S))^c \right) = \mathcal{W}_{\cup}^{\setminus x} \left((\mathcal{W}_{\cup}(x))^c \right).$$

Thus we get:

$$(C)^c = \bigcup_{\emptyset \subset S' \subseteq S} \mathcal{W}_{\cup}^{\setminus S'} \left((\mathcal{W}_{\cup}(S))^c \right) = \mathcal{W}_{\cup}^{\setminus x} \left((\mathcal{W}_{\cup}(x))^c \right).$$

By the definition of persistent watersheds, we now have $C = \mathcal{W}_{\cap}(x)$, which completes the proof.

□

We can now further characterize the potential ridge for regular terrains. The following lemma implies that on a regular terrain, the potential ridge is equal to the union of the uncertainty areas of the fuzzy watershed boundaries of any representative set of proxies of the imprecise minima, see Corollary 6.4.20.

Lemma 6.4.19 *Let $S_1, \ldots, S_k$ be the imprecise minima of a regular imprecise terrain and let $q_1, \ldots, q_k$ be associated proxies. For any $1 \leq i \leq k$, we have that $\mathcal{W}_{\cap}(q_i) = \left(\bigcup_{j \neq i} \mathcal{W}_{\cup}(q_j) \right)^c$.*

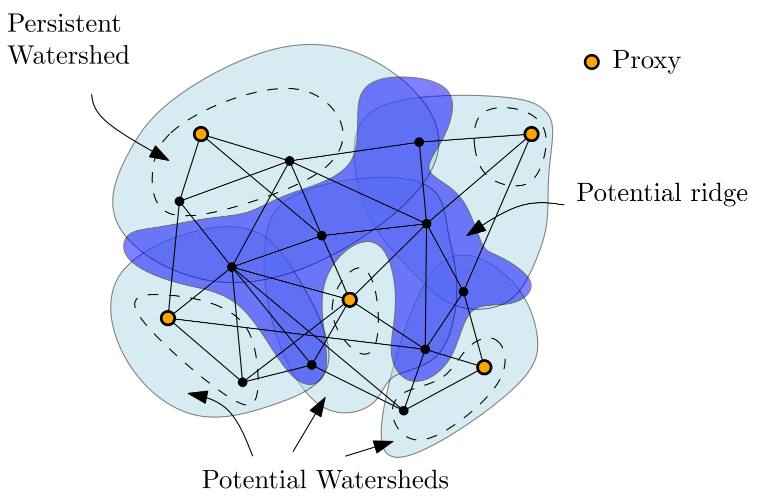


Figure 6.13: Illustration of the potential ridge on a regular terrain.

Proof: Since q_i is a proxy, we have that,

$$(\mathcal{W}_{\Cap}(q_i))^c = \mathcal{W}_{\cup}^{\setminus q_i}\left((\mathcal{W}_{\cup}(q_i))^c\right) = \mathcal{W}_{\cup}\left((\mathcal{W}_{\cup}(q_i))^c\right) = \bigcup_{p\in(\mathcal{W}_{\cup}(q_i))^c} \mathcal{W}_{\cup}(p)\,.$$

Now, for a node $p \in (\mathcal{W}_{\cup}(q_i))^c$, consider a minimum S that is reached by a flow path from p in R^-. By Definition 6.4.2, we have that S is an imprecise minimum, and since $p \in (\mathcal{W}_{\cup}(q_i))^c = (\mathcal{W}_{\cup}(S_i))^c$, we have $S \neq S_i$. As such, S must be equal to some S_j for $j \neq i$. Furthermore, by Lemma 6.4.3 we have $\mathcal{W}_{R^-}(S_j) = \mathcal{W}_{R^-}(q_j)$, therefore $p \in \mathcal{W}_{R^-}(q_j)$. Now, Lemma 6.4.10 implies that $\mathcal{W}_{\cup}(p) \subseteq \mathcal{W}_{\cup}(q_j)$. It follows that $(\mathcal{W}_{\Cap}(q_i))^c \subseteq \bigcup_{i\neq j} \mathcal{W}_{\cup}(q_j)\,.$

Since we also have that $q_j \in (\mathcal{W}_{\cup}(q_i))^c$ for any $j \neq i$, we also get $(\mathcal{W}_{\Cap}(q_i))^c \supseteq \bigcup_{i\neq j} \mathcal{W}_{\cup}(q_j)\,$, which implies the equality. □

Corollary 6.4.20 *Lemma 6.4.19 implies that, given $q_1, \ldots, q_k$, a representative set of proxies for the imprecise minima of a regular imprecise terrain, it holds that*

$$\begin{aligned}
\bigcup_i (\mathcal{W}_{\cup}(q_i) \setminus \mathcal{W}_{\Cap}(q_i)) &= \bigcup_i \left(\mathcal{W}_{\cup}(q_i) \setminus \left(\bigcup_{j\neq i} \mathcal{W}_{\cup}(q_j)\right)^c\right) \\
&= \bigcup_i \left(\mathcal{W}_{\cup}(q_i) \cap \bigcup_{j\neq i} \mathcal{W}_{\cup}(q_j)\right) \\
&= \bigcup_i \bigcup_{j\neq i} (\mathcal{W}_{\cup}(q_i) \cap \mathcal{W}_{\cup}(q_j))\,.
\end{aligned}$$

By Lemma 6.4.3, this is equal to the potential ridge of this terrain as defined in Definition 6.4.17. (This relationship is illustrated in Figure 6.13.)

Combining this with Lemma 6.4.6 and Lemma 6.4.16 we obtain:

Theorem 6.4.21 *We can compute the potential ridge of a regular imprecise terrain in $O(n \log n)$ time, where n is the number of edges of the imprecise terrain.*

Note that Definition 6.4.17 can also be applied to non-regular terrains, since it is solely based on the potential watersheds of the imprecise minima. We can use the algorithm of Section 6.4.2 to compute proxies for these minima, and then use the algorithm of Lemma 6.4.16 to compute a potential ridge between the watersheds of these proxies efficiently for non-regular terrains. However, note that the result may not be exactly the same as the potential ridge according to Definition 6.4.17, because on a non-regular terrain, the potential watersheds of the proxies may be smaller than the potential watersheds of the imprecise minima.

6.4.6 Disconnected persistent watersheds

Lemma 6.4.22 *There exists a regular terrain that contains a persistent watershed that consists of more than one connected component.*

Proof: Refer to Figure 6.14. The figure shows five nodes with their elevation intervals. The edges (a, b), (b, d) and (c, d) have length 1. The edge (d, e) has length 1.6. From a and e, very steep edges lead downwards to nodes not shown in the figure. The potential watershed $\mathcal{W}_{\cup}(e)$ of e is $\{c, d, e\}$. The node d is not in the persistent watershed of e: if d has elevation more than $6\frac{1}{3}$, the flow path from d will lead to b, outside $\mathcal{W}_{\cup}(e)$. In that case c is a local minimum inside $\mathcal{W}_{\cup}(e)$. Whenever c is not a local minimum, the elevation of d must be less than 4, and the flow path from c will lead to d and on to e. Thus c is in the persistent watershed $\mathcal{W}_{\cap}(e)$ of e, but d is not, so we have $\mathcal{W}_{\cap}(e) = \{c, e\}$. □

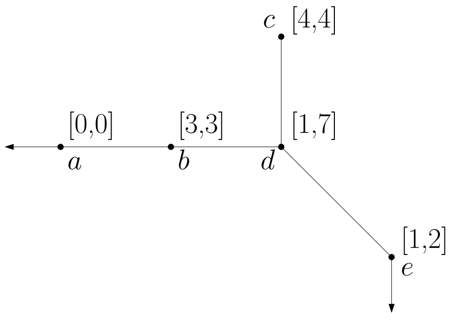


Figure 6.14: Example of disconnected persistent watershed on a regular terrain.

6.5 Concluding remarks

In this chapter we studied flow computations on imprecise terrains under two general models of water flow. For the surface model, where flow paths are traced across the surface of an imprecise polyhedral terrain, we showed NP-hardness for deciding whether water can flow between two points. For the network model, where flow paths are traced along the edges of an imprecise graph, we gave efficient algorithms to compute potential (maximal) and persistent (minimal) watersheds and potential downstream areas. Our algorithms also work for sets of nodes and can therefore be applied to reason about watersheds of areas, such as lakes and river beds.

In order to enable several extensions of these results in the network model, we introduced a certain class of imprecise terrains, which we call regular. We first defined when a set of nodes in an imprecise terrain can be considered a 'stable' imprecise minimum. We then described how to turn a non-regular terrain into a regular terrain using an algorithm by Gray et al. [81] and showed that this regularization algorithm preserves these imprecise minima. Interestingly, this algorithm also minimizes the number of minima of the terrain, while respecting the elevation bounds, as shown in [81].

We showed that persistent watersheds are nested on regular terrains and that these terrains have a fuzzy ridge structure which delineates the persistent watersheds of these stable minima. We gave an algorithm to compute this structure in $O(n \log n)$ time, where n is the number of edges of the terrain. Note that imprecise minima are defined also for non-regular terrains and they directly correspond to the imprecise minima in the regularized terrain. This suggests that the fuzzy watershed decomposition on the regular terrain also allows us to reason about the structure of the watersheds on the original non-regular terrain. We hope that, even though our work is motivated by geographical applications, the results will be useful in other application areas where watersheds are being computed, for instance in image segmentation [125].

Surprisingly, a persistent watershed according to our current definition may consist of multiple connected components: the persistent watershed of a node q may contain a node p such that one cannot walk from p to q without leaving the watershed (see Section 6.4.6). It can be debated whether this is an acceptable and possibly rare consequence of a sensible definition, or if this indicates that our definition needs to be refined or corrected.

There are many open problems for further research of which we want to discuss a few in more detail.

Open Problem 6.1 Geographic terrain data is usually collected at a large scale. For governmental agencies, such as meteorological institutes, it is desirable to work with high-resolution elevation data on a nation-wide scale. Since such large amounts of data usually do not fit into main memory, it would be useful to develop an IO-efficient algorithm for the basic potential watershed computation. The presented algorithm uses a Dijkstra-like approach and it is known that Dijkstra's algorithm is not IO-efficient. However, in our case, unlike in shortest-path computations, the nodes of the terrain are processed in the order of their computed height. Thus, nodes that have approximately the same key in the priority queue lie on approximately the same contour line in the computed realization of the terrain. A recent result by Arge *et al.* provides a data structure for IO-efficient contour-line queries [5]. Adapting these results to imprecise terrains could be a first step towards making the algorithms presented in this chapter IO-efficient.

Open Problem 6.2 Multidirectional flow models have been proposed in the GIS literature, e.g. D-∞, in which the incoming water at a node is distributed among the outgoing descent edges according to a gradient. These models can be seen as modified network models which approximate the steepest-descent direction more truthfully. It would be interesting to extend our definitions also to these models. In order to apply the techniques we developed for watershed computation, we first need to formalize to which extent a node is part of a watershed in these models.

Open Problem 6.3 A natural extension of the elevation range for a node of an imprecise terrain is a probability distribution over the possible elevations that this node may have. In this case, one may ask the question for the probability of a node to be in the watershed of another node. It would be interesting to compute a watershed that has high probability under this model. We are still far away from computing such probabilistic watersheds efficiently. Our work may be a first step towards this ultimate goal.

Open Problem 6.4 It is an obvious question if realistic input assumptions help in the complexity analysis of drainage networks on polyhedral terrains. A result by de Berg *et al.* [52] shows that if all triangles of the terrain are α-fat, the river network can have complexity at most $O(n^2/\alpha^2)$. At the same time they show that if the terrain surface is defined by a Delaunay triangulation, which is often used in practice, the same structure can have a complexity in $\Theta(n^3)$. It is conceivable that fatness helps in the analysis of flow on imprecise terrains, however there might be other assumptions that are more suitable. One can also imagine a combination of fatness with other assumption, for example assuming a low highway dimension on the river network, see Open Problem 3.2.

CHAPTER 7

The expected complexity of Voronoi diagrams on terrains

In this chapter we study the expected complexity of geodesic Voronoi diagrams on terrains, given that the sites are sampled uniformly at random from the domain of the terrain. In Section 7.1 we motivate the problem and discuss related literature. Section 7.3 contains a proof of an upper bound on the complexity of the Voronoi diagram. The constants in the asymptotic analysis depend on how well-behaved the terrain is, which is formalized using the realistic input models described in Section 7.2. In Section 7.4 we analyze the expected complexity if these assumptions on the shape of the terrain are dropped and give some lower bounds. We conclude with a discussion of some open problems in Section 7.5.

7.1 Introduction

Voronoi diagrams on terrains are a basic geometric structure that have applications in many areas: geographic information science (GIS) [8, 62, 126, 42], robot motion planning [142], mesh generation [104] and image analysis [141, 153] to name a few. The geodesic Voronoi diagram of point sites on a polyhedral terrain is a subdivision of the surface into cells, corresponding to the set of sites, such that every cell contains exactly the surface points which are closest to the site that is associated with the cell. Here, the distance is measured by the length of the shortest path on the terrain. It is tempting to believe that in practice – that is, given that the terrain is well-behaved – the complexity of such a geodesic Voronoi diagram should be linear, because of its similarity to the Euclidean Voronoi diagram of point sites in the plane.

However, in the worst case, this complexity can be much higher, even if one makes certain realistic assumptions on the shape of the terrain. Indeed, Aronov *et al.* [17] show that the worst-case complexity is $\Theta(n+m\sqrt{n})$ for a certain class of well-behaved terrains, where n is the number triangles that define the terrain and m is the number of Voronoi sites. This shows that assuming realistic input indeed brings the complexity down, since in general the complexity can be quadratic, however it is still far from being linear. They conjecture that, in order to prove a linear bound, one needs to make further assumptions on how the sites are distributed. Our purpose in this paper is to study the complexity of a geodesic Voronoi diagram if we assume that the sites are being chosen randomly from the terrain.

Previous work. Analyzing the expected complexity of geometric structures for random inputs has a long history in computational geometry. See for instance the work of Rényi and Sulanke [128] and Raynaud [127] on the complexity of convex hulls of random points. Weil and Wieacker give an overview of related results in [155]. The counterparts of Voronoi diagrams are the Delaunay triangulations [21]. Naturally they have been analyzed probabilistically as they are a fundamental data structure, used in fields such as mesh generation [129], surface reconstruction [61], molecular biology, and many others. It is well-known that the complexity of the Voronoi diagram of point sites in $\mathbb{R}^3$ is quadratic in the worst-case, however it is near-linear in most practical situations. To address this dichotomy, people have investigated the complexity when the point sites are: (i) generated by a random processes, (ii) well spaced, (iii) have bounded spread, or (iv) were sampled from surfaces according to curvature. See [60] and references therein for more information on such work.

In particular, there is a vast amount of work on *Poisson Voronoi Diagrams* (*PVD*). Here, the domain has a density associated with it (say, the area). The probability of n points to appear in an area of measure μ has, as the name suggests, a Poisson distribution parameterized by the area. Similarly, the distribution of points selected into disjoint areas is independent. Poisson Voronoi diagrams are used in many areas, such as physics, biology, animal ecology, and others. See [123, 95] and references therein. However, this work does not seem to have considered geodesics at all.

In this chapter, we are interested in the complexity of geodesic Voronoi diagrams on polyhedral terrains. Moet *et al.* [119] were the first to study this complexity using a set of parameterized assumptions that describe realistic terrains. In this approach, one assumes that a certain property, for example, the maximum slope of the terrain, can be bounded by a constant independent of the input size. This allows one to avoid certain worst-case configurations which are highly unlikely to occur in practice. Instead, the analysis is confined to classes of well-behaved inputs and consequently this method is described as using realistic input models, see Section 1.2.4. Moet also did an experimental validation of the used parameters [118] and confirmed that the parameters indeed behave like constants on realistic terrains.

The realistic input models introduced in this work have also been adopted by subsequent papers. As such, Aronov *et al.* [17] improved the bounds given by Moet *et al.* and showed that (i) the bisector between two sites has worst-case complexity $\Theta(n)$, (where n denotes the number of triangles of the terrain) if the triangulation is low-density and the lifted triangles have bounded slope; and (ii) that the worst-case combinatorial complexity of the Voronoi diagram is $\Theta(n + m\sqrt{n})$, (where m denotes the number of sites) if in addition the triangles are of similar size and the aspect ratio of the domain is bounded. The realistic assumptions made in these papers are described in more detail in the next section.

Finally, note that Schreiber and Sharir [131] showed how to compute an implicit representation of the geodesic Voronoi diagram on the surface of a convex polyhedron, in time $O((n+m)\log(n+m))$, so that the site closest to a query point can be reported in time $O(\log(n+m))$. Schreiber [130] also extended their method for single-source shortest paths to the case of non-convex polyhedra using several realistic input models. Naturally, these analyses do not inform about the complexity of the explicit Voronoi diagram.

7.2 Preliminaries

7.2.1 Voronoi diagrams on terrains

A polyhedral terrain $\mathcal{T}$ is defined by a triangulation Δ of a set V of n vertices in $\mathbb{R}^2$, a convex domain $D \subseteq \mathbb{R}^2$ which is equal to the convex hull of V, and a height function on these vertices. The ***surface*** of the terrain is defined by the triangles of Δ lifted according to this height function. We refer to $\mathcal{T}$ simply as a ***terrain*** and we denote the set of edges of the triangulation with E. For simplicity of exposition we restrict our discussion to the case where D is the unit square, however, our results can be easily extended to the more general case of convex regions with bounded aspect ratio. For two points $\mathsf{q}, \mathsf{r} \in D$, we denote their Euclidean distance in the (x, y)-plane with $\|\mathsf{q} - \mathsf{r}\|$. When q and r are lifted to the surface of $\mathcal{T}$, we define their geodesic distance to be the length of the shortest path connecting them that is constrained to lie in the surface of $\mathcal{T}$, and we denote this value by $\mathsf{d}_{\mathcal{T}}(\mathsf{q}, \mathsf{r})$.

The (geodesic) ***Voronoi diagram*** of a set of m points on $\mathcal{T}$ (which are called ***sites***) is a subdivision of the surface of $\mathcal{T}$, where every cell of the subdivision is associated with exactly one site, and such that for any point in the cell the associated site is the closest site, where the distances are measured using the geodesic distance. We denote the Voronoi diagram with $\mathcal{V}or(\mathsf{P})$, where P denotes the set of sites, and we call a cell of the subdivision a ***Voronoi cell***. The ***bisector*** between two sites q and r on the surface of $\mathcal{T}$ is defined as the set of points p, such that p has the same distance to q and r. The Voronoi diagram can be represented as the structured set of curves and straight lines which delineate the Voronoi cells and which are subsets of the bisectors between these points. We call a point which is incident to at least three cells a ***Voronoi vertex*** and we call each maximally connected subset of the bisector incident to two Voronoi cells a ***Voronoi edge*** (note that two cells can have multiple edges between them). Usually one assumes general position of the sites so that no two sites are equidistant from a terrain vertex, which ensures that bisectors are one-dimensional and that the Voronoi cells subdivide the terrain surface without overlap, see also [17]. In our case the sites are randomly sampled and such degenerate configurations are negligible.

Since a terrain $\mathcal{T}$ is defined by a height function over a domain D, there is a natural bijection between points of $\mathcal{T}$ and points of D. Hence, the various objects defined in the previous paragraph can be viewed either in $\mathcal{T}$ or in D. Generally in the paper we shall refer to these objects by their projection in D, unless otherwise stated.

7.2.2 The input model

The main idea of realistic input models is to parametrize certain properties of the input, which are suspected to capture contrived configurations leading to high complexities or running times. In cases where there exists a high discrepancy between the theoretical bounds and the complexities observed in practice, it is often useful to analyze the complexities not only as a function of the input size, but also with respect to these parameters. This sometimes leads to more informative asymptotic bounds. As such, the realistic input assumptions do not only distinguish between "good" and "bad" input, instead they enable a more differentiated view on which inputs are 'better' or 'worse'.

In this chapter, we use the following realistic input model. A set of line segments is ***λ-low-density*** if and only if the number of edges that intersect an arbitrary ball, which are longer than the radius of the ball, is smaller than λ. Low density has been used in the analysis of many different geometric problems, see also the discussion in Section 1.2.4.

To model a ***realistic terrain*** we adopt the realistic assumptions made in [17]. According to these assumptions, there exist constants λ and ξ independent of n, such that

(i) the set of edges of the triangulation is a λ-*low-density* set, and
(ii) any line segment embedded in the lifted triangulation has slope at most ξ.

We now state some useful facts that follow from these assumptions. For notational ease in the rest of the paper we define the constant $\beta = \sqrt{1+\xi^2}$.

First, the number of pairs of objects from two low-density sets that intersect each other is linear in the total number of objects in these sets. This can also be easily verified independently, the idea is to charge each intersecting pair to the smaller of the two objects.

Fact 7.2.1 (Aronov *et al.* [17]) *Let A be a set of n objects with λ-low-density and let B be a set of m objects with ϕ-low-density, then the number of pairs of objects $(u, v) \in A \times B$, such that u intersects v is $O(\lambda m + \phi n)$.*

Second, since the slope is bounded by a constant, the geodesic distance is the same as the Euclidean distance up to a constant factor, and similarly, geodesic disks have an area that is approximately the same as that of planar disks in this case.

Fact 7.2.2 (Aronov *et al.* [17]) *For any two points $\mathsf{q}, \mathsf{r} \in D$, we have that $\|\mathsf{q} - \mathsf{r}\| \leq \mathsf{d}_{\mathcal{T}}(\mathsf{q}, \mathsf{r}) \leq \beta \|\mathsf{q} - \mathsf{r}\|$.*

Lemma 7.2.3 *Let D be a geodesic disk of radius r on the surface of a terrain with bounded slope ξ and let A denote its area. We have that $\pi(r/\beta)^2 \leq A \leq \pi\beta r^2$.*

Proof: Let c be the center of D and let c_p be its projection. Let $D_{r/\beta}$ and D_r be the planar disks with center c_p and radius r/β and r, respectively. Let $T_{r/\beta}$ and T_r denote the portion of the terrain that lies directly above $D_{r/\beta}$ and D_r, respectively.

Clearly the projection of D is contained in D_r. Thus, if we can bound the area of T_r then this will bound the area of D. Observe that D_r consists of a set of triangles from Δ, which have been clipped at the boundary of D_r. If we lift any such (clipped) triangle up to the terrain then its area can increase by at most a factor of β. Therefore the total area of T_r is at most $\pi\beta r^2$.

Now consider the disk $D_{r/\beta}$. First observe that $T_{r/\beta}$ must have area at least as large as that of $D_{r/\beta}$. Second, note that D must contain $T_{r/\beta}$ (since by Fact 7.2.2 the distance between any point $T_{r/\beta}$ and c is at most $\beta(r/\beta) = r$). Therefore the area of D is at least $\pi(r/\beta)^2$. $\square$

7.2.3 The complexity of the Voronoi diagram

The complexity of the Voronoi diagram is measured by the complexity of the set of curves and line segments that delineate the Voronoi cells. This set consists of pieces of bisectors and it can be characterized as follows. Again, we adopt the definitions used in [17].

For most of the points on a bisector, the shortest path to either site will be unique. If the shortest path is not unique, we call p a ***breakpoint***. The breakpoints partition the bisector into a set of curved pieces which we call ***bisector pieces***. The ***combinatorial complexity*** of the Voronoi diagram is now defined as the sum of (i) the number of Voronoi vertices, (ii) the number of breakpoints of Voronoi edges, and (iii) the number of intersections of the bisector pieces of Voronoi edges with the triangulation of the terrain.

We continue with some useful facts and lemmas used in the analysis of the complexity. First, it was observed by Moet *et al.* that the number of breakpoints of the Voronoi diagram is bounded by n, since each of them can be attributed to a terrain vertex. To see why this is true, imagine walking along a bisector while sweeping the shortest paths to either site from the current position. Intuitively, a breakpoint on the bisector corresponds to the event that the path "jumps" across a small mountain or valley and this event can be charged to a terrain vertex that is "skipped" by the otherwise continuous sweep.

Fact 7.2.4 (Moet *et al.* [119]) *Given a terrain $\mathcal{T}$ which is defined by a triangulation with n vertices, the number of breakpoints of any Voronoi diagram on $\mathcal{T}$ is smaller than or equal to n.*

Furthermore, we will use the following result by Aronov *et al.* [17].

Fact 7.2.5 (Aronov *et al.* [17]) *Given two points q and r, the set of bisector pieces that form the bisector of q and r on $\mathcal{T}$ (projected to the (x, y)-plane) is $O(\xi)$-low-density.*

We remark that Aronov *et al.* use this result to show that the bisector has linear complexity. However, note that this result does not imply that the overall set of bisector pieces of the Voronoi diagram is low-density, which would imply a linear complexity for the whole diagram. Consider for example the situation, where all the sites lie close to each other on a straight line, and all the triangles of the terrain surface are coplanar. In this example, the bisectors are pairwise parallel lines, which extend from one side of the domain to the other and could therefore lead to a quadratic complexity Voronoi diagram by intersecting many triangles of the terrain.

Finally, we observe that the number of Voronoi vertices and edges is linear in the number of sites, as the following lemma and corollary testify. This fact was also observed by Aronov *et al.*, we include an independent proof which also shows Corollary 7.2.7 below.

Lemma 7.2.6 *Let $\mathcal{T}$ be a terrain and let P be a set of m points. Then the number of Voronoi edges and Voronoi vertices of $\mathcal{V}or(\mathsf{P})$ is $O(m)$.*

Proof: For $m \leq 2$ the claim is clearly true, since we have at most one bisector, which contributes exactly one Voronoi edge. For $m > 2$, we argue as follows.

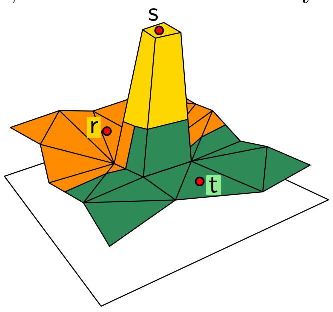
First, observe that the cells in this Voronoi diagram are connected. Indeed, consider a point p that belongs to the interior of the cell of $\mathsf{s} \in \mathsf{P}$. Consider the shortest path π from p to s, and consider any point $\mathsf{q} \in \pi$. If q is closer to some other site t than to s, then we have that

$$\mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{s}) = \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{q}) + \mathsf{d}_{\mathcal{T}}(\mathsf{q},\mathsf{s}) \geq \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{q}) + \mathsf{d}_{\mathcal{T}}(\mathsf{q},\mathsf{t}) \geq \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{t})\,,$$

but this is a contradiction to p being in the interior of the cell of s.

Now, consider the dual graph G of the graph formed by the Voronoi vertices and Voronoi edges. In this graph, every vertex corresponds to a Voronoi cell and every face corresponds to a Voronoi vertex. Note that we can derive a geometric embedding of this graph by using the sites as vertices and picking an arbitrary point on each Voronoi edge and connecting it by its shortest path to either site to form an edge between two vertices.

It is well-known that a cell in the Voronoi diagram might not be simply connected. Indeed, consider a mountain surrounded by plains. If we place a site s on the top of the mountain, and a site t at the bottom of the mountain (and the mountain slope is large enough) then the Voronoi cell of s would be completely surrounded by the cell of t and thus would create a 'hole' in this cell. In G, the two vertices that correspond to the cells of s and t would be connected by an edge, which is incident to only one face in G.



Furthermore, it is known that the dual graph can have multiple edges between two sites. To see this, again, place s on the top of the mountain and place two sites t and r at the bottom, such that the bisector between t and r intersects the mountain. The boundary between the cells of t and r would contain two Voronoi edges from the same bisector.

However, the dual graph G is planar and connected and as such its Euler characteristic is 2. Hence $v - e + f = 2$, where e denotes the number of edges, f the number of faces and v the number of vertices of G.

Now, by definition, every Voronoi vertex is incident to at least three Voronoi cells. Therefore, every face of G is incident to at least three edges of G. Since G is planar, every edge of G is incident to at most two faces of G. Hence, we have that $3f \leq 2e$, which implies that $3f \leq 2(v + f - 2)$, and therefore it holds that $f \leq 2v - 2 = 2m - 2$. It follows that the number of Voronoi vertices is in $O(m)$. Applying Euler's formula again, we obtain that also the number of Voronoi edges is in $O(m)$. □

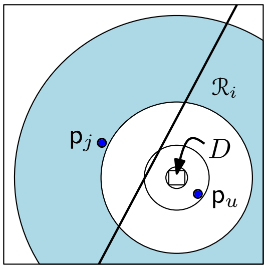
Corollary 7.2.7 *Let $\mathcal{T}$ be a terrain and let P be a set of m points. Let D be a connected subset of the unit square which intersects k Voronoi cells in their projection. The number of Voronoi edges of $\mathcal{V}or(\mathsf{P})$ which intersect D in their projection is in $O(k)$.*

7.3 Upper bound

We prove the following lemma first in the planar case and then extend it to terrains with bounded slope. The bounded expected complexity then follows by examining the number of intersections of the bisector pieces with the terrain triangulation in an $\sqrt{m} \times \sqrt{m}$ grid.

Lemma 7.3.1 *Let P be a set of m points, sampled uniformly at random from a unit square, and let D be a square of side length $1/\sqrt{m}$ contained in the unit square. Then the expected number of points in P that contribute to $\mathcal{V}or(\mathsf{P}) \cap D$ is in $O(1)$.*

Proof: We place a sequence of exponentially growing disks centered at the center point of D. Let $r_i = \frac{1}{\sqrt{2m}}2^i$, for $i = 0, \ldots, k = \lceil \lg \sqrt{2m} \rceil$ (i.e. r_0 is the radius of the circumscribed circle of D). Let d_i be the disk of radius r_i, which is clipped to the unit square and let $\mathcal{R}_i = \mathsf{d}_i \setminus \mathsf{d}_{i-1}$, for $i = 1, \ldots, k$.



Let the points in P be labeled $\mathsf{p}_1, \ldots, \mathsf{p}_m$. Observe that the expected number of points from P that fall into d_2 is $(\pi r_2^2)m = \frac{16\pi m}{2m} = 8\pi = O(1)$. Hence we do not need to worry about their contribution to $\mathcal{V}or(\mathsf{P}) \cap D$. Otherwise, we claim that a point p_j which falls into $\mathcal{R}_i$ for $i > 2$, can only contribute to $\mathcal{V}or(\mathsf{P}) \cap D$ if d_{i-2} contains no points of P. Assume for the sake of contradiction that the Voronoi cell of p_j intersects D, and there exists some point p_u in P that lies in d_{i-2}. By construction, we know p_j has distance greater than $(r_{i-1} - r_0)$ to any point in d_0 and p_u has distance at most $(r_{i-2} + r_0)$ to any point in d_0. Hence we have that $\mathsf{d}(\mathsf{p}_j, \mathsf{d}_0) > r_{i-1} - r_0 = 2r_{i-2} - r_0 \geq r_{i-2} + r_0 \geq \mathsf{d}(\mathsf{p}_u, \mathsf{d}_1)$ for $i > 2$ and as such every point in d_0 is strictly closer to p_u than p_j, where $\mathsf{d}(\mathsf{p}, X) = \min_{\mathsf{q} \in X} \|\mathsf{p} - \mathsf{q}\|$.

Hence it is sufficient to bound the expected number of points p_j which fall into an annulus $\mathcal{R}_i$ such that d_{i-2} is empty. For p_j we define the indicator variable X_i^j which is equal to 1 if and only if $\mathsf{p}_j \in \mathcal{R}_i$, and the indicator variable Y_i^j which is equal to 1 if and only if no other point falls into d_{i-2}. Hence a given point p_j can contribute to $\mathcal{V}or(\mathsf{P}) \cap D$ if and only if $X_i^j Y_i^j = 1$ for some value of i. Now, we know that

$$\mathbf{Pr}\left[X_i^j = 1\right] \leq \pi r_i^2 - \pi r_{i-1}^2 = \frac{\pi}{2m}(2^{2i} - 2^{2(i-1)}) = \frac{3\pi}{2m}4^{i-1},$$

and

$$\begin{aligned}\mathbf{Pr}\left[Y_i^j = 1\right] &\leq \left(1 - \frac{1}{4}\pi r_{i-2}^2\right)^{m-1} \\ &\leq \exp\left(-\frac{\pi}{4} r_{i-2}^2 (m-1)\right) = \exp\left(-\frac{\pi 4^{i-3}(m-1)}{2m}\right) \leq \exp(-4^{i-3}),\end{aligned}$$

where a factor of 1/4 was added in the bound for Y_i^j due to boundary effects that might arise from the position of D in the unit square. Hence the number of points that can affect $\mathcal{V}or(\mathsf{P}) \cap D$ is bounded by $\sum_j \sum_{i>2} X_i^j Y_i^j$, for which in expectation we have,

$$\begin{aligned}\mathbf{E}\left[\sum_j \sum_{i>2} X_i^j Y_i^j\right] &= \sum_j \sum_{i>2} \mathbf{E}\left[X_i^j\right] \mathbf{E}\left[Y_i^j\right] \\ &\le \sum_j \sum_{i>2} \frac{3\pi}{2m} 4^{i-1} e^{-4^{i-3}} = \frac{3\pi}{2} \sum_{i>2} 4^{i-1} e^{-4^{i-3}} = O(1),\end{aligned}$$

by linearity of expectation and the independence of X_i^j and Y_i^j for all i and j. □

Lemma 7.3.2 *Let $\mathcal{T}$ be a terrain with bounded slope ξ. Let P be a random sample of m points, either sampled uniformly from $\mathcal{T}$, or uniformly from the unit square and then lifted vertically up to $\mathcal{T}$. Let D be a square of side length $1/\sqrt{m}$ contained in the unit square. Then the expected number of points in P that contribute to the portion of the Voronoi diagram of P on $\mathcal{T}$ that lies above D is $O(\beta^5)$.*

Proof: Follows by a careful adaptation of the proof of Lemma 7.3.1. To accommodate for the larger distances on the surface, we increase the radii of the disks slightly.

So, let $r_i = \frac{1}{\sqrt{2m}}(2\beta+1)^i$, for $i = 1, \ldots, k$. Let d_i be the disk (in the plane) of radius r_i centered at the center of D. And let $\mathcal{R}_i = \mathsf{d}_i \setminus \mathsf{d}_{i-1}$, for $i = 1, \ldots, k$, as defined above. For points $\mathsf{p}, \mathsf{q} \in \mathsf{d}_{i-2}$ and $\mathsf{r} \in \mathcal{R}_i = \mathsf{d}_i \setminus \mathsf{d}_{i-1}$, we have that

$$\mathsf{d}_{\mathcal{T}}(\mathsf{p}, \mathsf{q}) \le \beta \left\| \mathsf{p} - \mathsf{q} \right\| \le \beta 2 r_{i-2} \le r_{i-1} - r_{i-2} < \left\| \mathsf{q} - \mathsf{r} \right\| \le \mathsf{d}_{\mathcal{T}}(\mathsf{q}, \mathsf{r}) .$$

As such, as before, for a point in $\mathsf{P} \cap \mathcal{R}_i$ to affect the Voronoi diagram on D requires that d_{i-2} is empty of any points of P.

Now, consider the case that the points are sampled from the terrain. Define X_i^j and Y_i^j as in Lemma 7.3.1. Note that the area of the terrain can only increase from one by lifting the individual triangles. By Lemma 7.2.3, we have that

$$\begin{aligned}\mathbf{Pr}\left[X_i^j = 1\right] &\le \frac{\text{area of } \mathcal{R}_i \text{ on terrain}}{\text{area of terrain}} \\ &\le \beta\left(\pi r_i^2 - \pi r_{i-1}^2\right) \le \frac{\pi\beta}{2m}\left((2\beta+1)^{2i} - (2\beta+1)^{2i-2}\right) \le \frac{\pi\beta}{2m}(2\beta+1)^{2i}.\end{aligned}$$

Similarly, since d_{i-2} intersects the terrain with at least a quarter disk, the size of this area is bounded from below by $\pi r_{i-2}^2/(4\beta)$. Plugging this into the analysis of Lemma 7.3.1, we have

$$\mathbf{Pr}\left[Y_i^j = 1\right] \le \left(1 - \frac{\pi r_{i-2}^2}{4\beta^2}\right)^{m-1} \le \exp\left(-\frac{\pi r_{i-2}^2}{4\beta^2}(m-1)\right) \le \exp\left(-(2\beta+1)^{2i-5}\right).$$

As before, we thus have

$$\mathbf{E}\left[\sum_j \sum_{i>2} X_i^j Y_i^j\right] \le \sum_{j=1}^{m} \sum_{i>2} \frac{\pi\beta}{2m}(2\beta+1)^{2i} \cdot \exp\left(-(2\beta+1)^{2i-5}\right) = O(1).$$

The only missing component is bounding the expected number of points of $\mathsf{P} \cap \mathsf{d}_2$, as they can affect the Voronoi diagram in D. Arguing as above, this quantity is bounded by $m\beta \cdot (\text{area of } \mathsf{d}_2) \leq m\beta\pi r_2^2 \leq \frac{\beta}{2}(2\beta+1)^4 = O\big(\beta^5\big)$.

Note that if we drop the factors of β in the bounds for X_i^j, Y_i^j, and the area of d_2, then the above becomes a proof for the case when we sample from the unit square $\square$

Theorem 7.3.3 *Let $\mathcal{T}$ be a terrain. Let P be a random sample of m points, either sampled uniformly from the surface of $\mathcal{T}$, or uniformly from the domain and then lifted vertically up to the surface. The expected combinatorial complexity of $\mathcal{V}or(\mathsf{P})$ is in $O\big(\xi\beta^5\lambda(n+m)\big)$.*

Proof: As described in Section 7.2.3, the combinatorial complexity of the Voronoi diagram is the sum of the number of breakpoints of Voronoi edges, the number of Voronoi vertices and the number of intersections of triangulation edges with bisector pieces of Voronoi edges. By Fact 7.2.4 the number of breakpoints is bounded by $O(n)$, and by Lemma 7.2.6 the number of Voronoi vertices is bounded by $O(m)$.

It remains to bound the number of intersections of the set of bisector pieces with the triangulation. To this end, we place a grid on the domain of the terrain, such that the side length of each grid cell is $l = 1/\sqrt{m}$ and we obtain $M = O(m)$ grid cells which together cover the domain of the terrain. Now, let $\mathsf{C}_1, \ldots, \mathsf{C}_M$ denote these grid cells. Consider the grid cell C_i and the set of bisector pieces of the Voronoi diagram which intersect this grid cell, let this set be B_i. Similarly, let E_i denote the subset of edges of the triangulation, which intersect C_i. Since we assumed that the triangulation is λ-low-density, also E_i is a λ-low density set. By Fact 7.2.5 we have that the set of bisector pieces, which originate from the same Voronoi edge (and therefore from the same bisector) form an $O(\xi)$-low-density set. Let k_i denote the number of Voronoi edges that contribute bisector pieces to B_i. We have that B_i is a $O(\xi k_i)$-low density set. By Fact 7.2.1, the number of intersections between objects of E_i and objects of B_i is in $O(\xi k_i|\mathsf{E}_i| + \lambda|\mathsf{B}_i|)$.

Now, in order to bound the overall number of intersections, let $\mathsf{B}_i^{>l}$ denote the subset of bisector pieces which are longer than l, similarly, let $\mathsf{B}_i^{\leq l}$ denote the bisector pieces in B_i which have length smaller or equal to l and let $\mathsf{E}_i^{\leq l}$ and $\mathsf{E}_i^{>l}$ be defined analogously. By the above analysis, we have that there exists some constant c_1, such that it holds for the overall number of intersections χ,

$$\chi \leq c_1 \sum_{i\geq 1}^{M} (\xi k_i|\mathsf{E}_i| + \lambda|\mathsf{B}_i|) = c_1 \sum_{i\geq 1}^{M} \Big(\xi k_i\Big(|\mathsf{E}_i^{>l}| + |\mathsf{E}_i^{\leq l}|\Big) + \lambda\Big(|\mathsf{B}_i^{>l}| + |\mathsf{B}_i^{\leq l}|\Big)\Big).$$

By the definition of low-density sets, we have that $|\mathsf{E}_i^{>l}| = O(\lambda)$ and $|\mathsf{B}_i^{>l}| = O(\xi k_i)$, since they intersect the bounding ball of the grid cell C_i, which has radius $O(l)$. Therefore, it must be that there exists a constant c_2 such that,

$$\chi \leq c_1 \sum_{i\geq 1}^{M} \Big(\xi k_i|\mathsf{E}_i^{\leq l}| + \lambda|\mathsf{B}_i^{\leq l}| + c_2\lambda\xi k_i\Big) \leq c_1 \sum_{i\geq 1}^{M} \Big(\xi k_i|\mathsf{E}_i^{\leq l}| + c_2\lambda\xi k_i\Big) + c_1\lambda 4|\mathsf{B}|,$$

where the last inequality follows from the fact that any bisector piece in $\mathsf{B}_i^{\leq l}$ can intersect at most four grid cells, since the grid cells have side length equal to l (similarly any edge in $\mathsf{E}_i^{\leq l}$ can intersect at most three grid cells).

Finally, note that the number of Voronoi cells that are expected to intersect a grid cell in their projection is bounded by $O(\beta^5)$ by Lemma 7.3.2. Thus by Corollary 7.2.7 we have that $\mathbf{E}[k_i] = O(\beta^5)$. Therefore in expectation,

$$\begin{aligned}\mathbf{E}[\chi] &\leq c_1\,\mathbf{E}\left[\sum_{i\geq 1}^{M}\Big(\xi k_i|\mathsf{E}_i^{\leq l}| + c_2\lambda\xi k_i\Big)\right] + c_1\lambda 4|\mathsf{B}| \\ &= c_1\sum_{i\geq 1}^{M}\Big(\xi\,\mathbf{E}[k_i]\,|\mathsf{E}_i^{\leq l}| + c_2\lambda\xi\,\mathbf{E}[k_i]\Big) + c_1\lambda 4|\mathsf{B}| \\ &\leq c_3\xi\beta^5\lambda\left(\sum_{i\geq 1}^{M}\Big(|\mathsf{E}_i^{\leq l}| + 1\Big) + 4|\mathsf{B}|\right) \\ &\leq c_3\xi\beta^5\lambda(3|\mathsf{E}| + M + 4|\mathsf{B}|)\end{aligned}$$

for some constant c_3 (note that we used the fact that $|\mathsf{E}_i^{\leq l}|$ is independent of the random sampling). Furthermore, observe that by Lemma 7.2.6 the overall number of Voronoi edges is $O(m)$. Recall that every Voronoi edge is broken up by breakpoints into bisector pieces. Every breakpoint increases the number of bisector pieces by one. Using Fact 7.2.4, it follows that the overall number of bisector pieces $|\mathsf{B}|$ is in $O(n+m)$. Since the number of edges of the triangulation $|\mathsf{E}|$ is $O(n)$, we conclude that $\mathbf{E}[\chi] = O(\xi\beta^5\lambda(n+m))$. $\square$

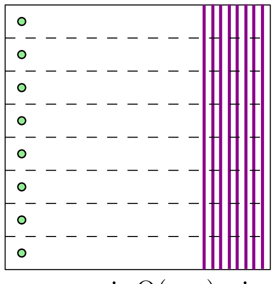
7.4 Lower bound

In this section we show that if we drop the assumptions on the terrain, then the expected worst-case complexity of the resulting geodesic Voronoi diagram can be $\Omega\big(nm^{2/3}\big)$ if the sites are sampled uniformly at random from the unit square.

In the following we will refer to the walls of the unit square defined by $x = 0$, $x = 1$, $y = 0$, and $y = 1$ as the ***west, east, south, and north walls***, respectively.

7.4.1 A simple example

We start with a simple example of a Voronoi diagram of points in the plane overlayed with a planar map, such that the overlay has a high complexity. The later constructions use this as their starting point. There, the triangulation of the terrain surface will take the place of the planar map.



First place m points in a column near and parallel to the west wall of the unit square such that the spacing between each adjacent pair of points is $\Theta(1/m)$. The planar map consists of n vertical lines near the east wall of the unit square that extend from the north wall to the south wall. Now, the boundaries of the Voronoi cells of these points extend from the west to the east wall and are parallel to the north and south walls, and hence the complexity of the overlay of the Voronoi diagram with the planar map is $\Theta(nm)$, since it is an $n \times m$ grid.

7.4.2 Farming – an $\Omega(n\sqrt{m})$ example

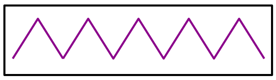
7.4.2.1 Construction

The height function used in the following construction of a terrain has (essentially) only two values, zero and h. The areas between a part of the terrain that is of height zero and of height h consist of very narrow and steep boundary regions. In our construction this intermediate boundary has a very small measure in the projection, and the reader can think of it as having measure zero. Moreover, h is chosen to be sufficiently large so that no point at height zero can affect the Voronoi diagram at height h.

One can therefore view the following terrain construction as a flat unit square, where we have cut out or "forbidden" areas (that have height 0). Therefore, for the sake of simplicity of exposition, an area being constructed is flat, at height h, and the adjacent forbidden area is at height zero.

Our main building blocks will be farms. We define a ***farm*** to be a square of side length $1/(c\sqrt{m})$. Intuitively, farms are part of the terrain which with constant probability (the constant will depend on c) will receive at least one point from the random sample (i.e. a farm takes the place of a point from the example in Section 7.4.1).

We define the ***diameter*** of a farm to be the quantity $\delta = \sqrt{2}/(c\sqrt{m})$ (that is, the distance of the furthest two points in a farm).



We now define a sequence of ridges to take the place of the planar map from Section 7.4.1 (i.e. in expectation we would like the Voronoi diagram to look like a grid over the ridges).

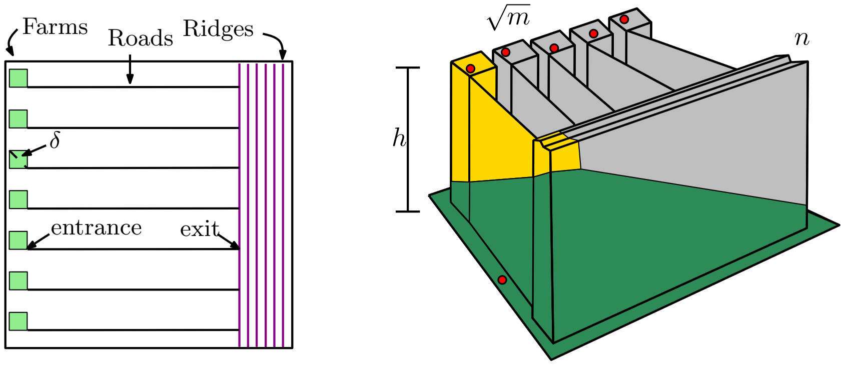
Formally, let a ***sequence of ridges*** of length n be a sequence of $2n$ rectangles, $r_1, \ldots, r_{2m}$ such that the right edge of r_i is the same as the west edge of r_{i+1} for $i = 1, \ldots, 2n-1$, r_i has a slope of 45° for odd i and $-45°$ for even i, all rectangles extend from the north-to-south walls of the unit square, and the geodesic distance from the left edge of r_1 to the right edge of r_{2n} is $1/(c2^n)$ (which is $O(1/2^m)$ since we assumed $m = O(n)$). Refer to the figure above to the right for a side view of the ridges.

The construction of the $\Omega(n\sqrt{m})$ example is as follows. The layout is illustrated in Box 7.1. Place $\Theta(\sqrt{m})$ farms from north-to-south along the west wall of the unit square, with $2/(c\sqrt{m})$ spacing in between each adjacent pair. Next build a sequence of $\Theta(n)$ ridges near (and parallel to) the east wall of the unit square. Then connect each farm directly to the leftmost ridge by creating a line parallel to the north and south walls connecting the south-east corner of the farm to the first ridge. See figure on the right. We refer to such a line as ***road***. The roads stay at height h and to the left and right of a road, the height drops to zero as described earlier.

7.4.2.2 Analysis

Definition 7.4.1 The point at which a farm connects to its road is its ***entrance*** (i.e. the south-east corner of the farm), and the point at which the road connects to the leftmost ridge is its ***exit***. We say that the point (from the random sample of m points) that is closest to the entrance for a farm, is the farm's ***dominating point.***

Box 7.1 Basic farming layout



Layout of the terrain construction described in Section 7.4.2.1. A topview is shown on the left, while a perspective view is shown on the right. On the right, a random sample of point sites and two cells of the Voronoi diagram are indicated. The height parameter h is chosen such that the point sites at the bottom do not influence the Voronoi diagram at the top level.

Lemma 7.4.2 *For the construction of the terrain described above, if one picks uniform at random m points in the unit square, their induced geodesic Voronoi diagram on this terrain has complexity $\Omega(n\sqrt{m})$.*

Proof: The area of each farm is $\Theta(1/m)$ and hence a sample of m points picked uniformly at random from the unit square, will have at least one point with constant probability in each farm. Moreover, since we constructed $\Theta(\sqrt{m})$ farms, this implies that in expectation $\Theta(\sqrt{m})$ farms will receive at least one point. Furthermore, the width of the sequence of ridges and roads was chosen such that the probability that either receives a point is exponentially small (and hence in the following we assume they do not receive any point).

Now consider a farm which received at least one point, and let p be its dominating point. Observe that the Voronoi cell of p contains the entire road connecting this farm to the ridges, and its Voronoi cell extends all the way to the rightmost edge of the sequence of ridges, and hence will be of complexity $\Omega(n)$. Indeed, by our construction, only a point from another farm can prevent the Voronoi cell of p from reaching the rightmost ridge. However, the spacing of the farms was chosen to prevent this. In the worst case p is in the north-west corner of its farm, and an adjacent farm has a point q at the south-east corner. Let l be the length of a road. Let $\mathsf{z_p}$ (resp. $\mathsf{z_q}$) be the exit of the farm containing p (resp q). Now consider the geodesic shortest path connecting $\mathsf{z_p}$ to the rightmost ridge. Every point on this segment is in distance at most $\delta + l + 1/(c2^n)$ from p. However, the closest point on this segment from q is at a distance of at least $l + 2/(c\sqrt{m}) \geq \delta + l + 1/(c2^n)$, and so q cannot prevent the Voronoi cell of p from reaching all the way to the rightmost ridge.

Therefore, in expectation, we have that $\Theta(\sqrt{m})$ farms have a point whose Voronoi cell extends all the way across the sequence of ridge, which gives a Voronoi diagram that in expectation has complexity $\Omega(n\sqrt{m})$. □

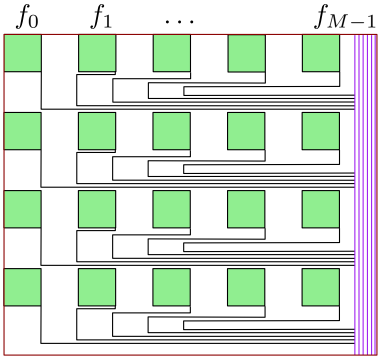
7.4.3 Industrial farming – an $\Omega\big(nm^{2/3}\big)$ example

The challenge in improving the example above is that the distance of a dominating point to the exit of a farm has too much variance (i.e., $\sqrt{1/m}$). Since there does not seem to be a way to decrease the variance directly, we instead connect all the farms to the ridges, and carefully argue about the expected complexity of the generated Voronoi diagram.

7.4.3.1 Construction

We set the side length of each square farm to be $1/\sqrt{m}$ and construct a sequence of $\Theta(n)$ ridges near the east wall of the unit square. We will place an $M \times M$ grid of farms inside the unit square, where $M = \lfloor \sqrt{m}/5 \rfloor$. Specifically, the spacing between columns (which extend from north-to-south) will be $1/\sqrt{m}$ and the spacing between rows (which extend from west to east) will be $2/\sqrt{m}$. The grid starts in the north-west corner of the unit square.

We now describe the connecting roads from the farms to the ridges. The construction will ensure that the length of each road is the same and that the distance between adjacent exits along the ridges is at least $1/m$. These two properties will be sufficient for the analysis in the next section to go through.



Consider a given row of farms. Number the farms in this row $f_0, \dots, f_{M-1}$ in increasing order of their distance to the west wall. Every farm has dimensions $1/\sqrt{m} \times 1/\sqrt{m}$, and the spacing between two consecutive farms in a row is $1/\sqrt{m}$. As such, the x coordinate of the entrance of the ith farm is $x_i = (2i+1)/\sqrt{m}$ (as before, the entrance to each road will be at the south-east corner of the farm). The directions the ith farm's road goes from entrance to exit is described as follows:

(a) south for a distance of $\alpha_{i,a} = i/m$,
(b) west for a distance of $\alpha_{i,b} = (x_i + \alpha_{i,a})/2$,
(c) south for a distance of $\alpha_{i,c} = 1/\sqrt{m} - 2\alpha_{i,a}$, and
(d) east for a distance of $\alpha_{i,d} = w - (x_i - \alpha_{i,b})$ all the way to the first ridge, where w is the distance from the west wall to the first ridge.

This layout is sketched in the figure above to the right. Note that the spacing in this figure only approximately matches the description.

Sanity checks. The road of the ith farm starts at x coordinate x_i, goes west for a distance of $\alpha_{i,b}$ and east for a distance of $\alpha_{i,d}$. Observe that the x coordinate of the exit of this road is $x_i - \alpha_{i,b} + \alpha_{i,d} = x_i - \alpha_{i,b} + w - (x_i - \alpha_{i,b}) = w$.

Observe that the ith farm will connect to the ridges in north-to-south distance $\alpha_{i,a} + \alpha_{i,c} = 1/\sqrt{m} - i/m$ from the southern boundary of the row of farms. That is, adjacent farms in the row have exits in distance $1/m$ apart along the first ridge (exits are $\Theta(1/\sqrt{m})$ apart between rows). Furthermore, each road is of the same length.

Indeed, let r_i be the length of the road for the ith farm to its exit. We have that

$$r_i = \alpha_{i,a} + \underbrace{\frac{x_i + \alpha_{i,a}}{2}}_{\alpha_{i,b}} + \underbrace{\frac{1}{\sqrt{m}} - 2\alpha_{i,a}}_{\alpha_{i,c}} + \underbrace{w - \left(x_i - \frac{x_i + \alpha_{i,a}}{2} \right)}_{\alpha_{i,d}} = \frac{1}{\sqrt{m}} + w.$$

That is, all the roads have exactly the same length.

Each farm contributes at most 16 vertices to the complexity of the terrain (for building itself and the connecting road). The construction uses $M^2 = m/25$ farms, thus leaving a contingent of at least $n - \frac{16}{25}m$ vertices for building the ridges. If $m \leq n$, then this is in $\Theta(n)$ as desired.

7.4.3.2 Competing farms

We now prove that in expectation $\Theta\bigl(m^{2/3}\bigr)$ dominating points will have their Voronoi cells reach all the way to the east wall across the sequence of ridges.

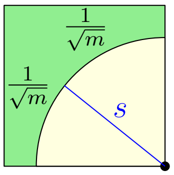
Definition 7.4.3 Let p and z be the dominating point and exit, respectively, of some farm. We say that another point q from the random sample that is contained in another farm and such that $\mathsf{d}_{\mathcal{T}}(\mathsf{q},\mathsf{z}) < \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{z})$, ***eliminates*** (the Voronoi cell of) p, where $\mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{z})$ denotes the shortest path on the terrain from p to z. If there are no points which eliminate a given dominating point, then the dominating point is ***alive.***

Observation 7.4.4 Let p and $\mathsf{z_p}$ be the dominating point and exit, respectively, of some farm f. Let q be a point which is in some other farm f' with exit $\mathsf{z_q}$. If f' is i farms away (in the north to south order of the exits of the farms along the first ridge) then $\|\mathsf{z_p} - \mathsf{z_q}\| \geq i/m$ and hence if q eliminates p then it must be that $\mathsf{d}_{\mathcal{T}}(\mathsf{q},\mathsf{z_p}) = \mathsf{d}_{\mathcal{T}}(\mathsf{q},\mathsf{z_q}) + i/m < \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{z_p})$.

Next, we prove that each dominating point is alive with probability $\Omega\bigl(1/m^{1/3}\bigr)$, using the following lemmas.

Lemma 7.4.5 *Let f be a farm, and let R be the random variable that is the distance of the closest site (that falls into this farm) from the farm entrance (if there is no site in this farm, we set $\mathrm{R} = \sqrt{2/m}$). Then, for any distance s, we have* $\mathbf{Pr}\Bigl[\mathrm{R} \leq s\Bigr] \leq ms^2\pi/4$, *where equality holds for* $0 \leq s \leq 1/\sqrt{m}$.

Proof: Recall that the entrance of a farm is at the south-east corner. Hence a point in the farm which is in distance at most s from the entrance must fall into the intersection of the farm with a circle of radius s whose center is at the entrance of the farm, see the figure to the right.



Therefore, if the radius of the circle is less than the side length of the square, i.e. $s \leq 1/\sqrt{m}$, then the intersection is a quarter disk and so the area is exactly $\pi s^2/4$. Otherwise, the top and left portions of the quarter disk needs to be clipped to the farm and so the area is at most $\pi s^2/4$. Now, the probability of the ith site to fall into this disk is at most $\pi s^2/4$, and since we sample m sites (independently), by the union bound the claim follows. $\square$

Observation 7.4.6 The above raises the following natural question: Given that a point p is picked at random falls into a farm f, what is the probability that it is in distance at most r from the entrance e of f. For $r \leq 1/\sqrt{m}$, the above argument implies that

$$\mathbf{Pr}\Big[\mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{e}) \leq r \;\Big|\; \mathsf{p} \in f\Big] = \frac{r^2\pi/4}{\text{area}(f)} = \frac{mr^2\pi}{4}, \tag{7.1}$$

as $\mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{e}) = \|\mathsf{p} - \mathsf{e}\|$ since the surface is horizontal on top of a farm.

Lemma 7.4.7 *Let* p *and* z *be the dominating point and exit, respectively, of a farm* f. *Let* $r = \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{z})$. *Let* f_i *be a farm which is* i *farms away from* f *(either in north or in south direction), and let* X_i *be the number of points which fell into* f_i. *Let* α_{X_i} *denote the probability that no point from* f_i *eliminates* p *(see Definition 7.4.3). Then* $\alpha_{X_i} \geq \exp\big(-m(r - i/m)^2\pi X_i/2\big)$.

Proof: Let $\mathsf{q}_1, \ldots, \mathsf{q}_{X_i}$ be the X_i points that fall into farm f_i. Let z_i be the exit of f_i, and let $d_j = \mathsf{d}_{\mathcal{T}}(\mathsf{q}_j, \mathsf{z}_i)$, for $j = 1, \ldots, X_i$. Arguing as in Lemma 7.4.5, we have that $\mathbf{Pr}[d_j \leq s] \leq s^2\pi/4$ for all j. By Observation 7.4.4, a point q_j eliminates p if and only if $d_j < r - i/m$. For $j \neq l$, whether or not q_j or q_l eliminate p are independent events and hence

$$\alpha_{X_i} = \prod_{j=1}^{X_i} \mathbf{Pr}\Big[d_j \geq r - i/m\Big] \geq \left(1 - (r - i/m)^2\frac{\pi}{4}\right)^{X_i} \geq \exp\left(-m(r - i/m)^2\frac{\pi X_i}{2}\right).$$

□

Lemma 7.4.8 *Let* p *and* τ *be the dominating point and exit, respectively, of some farm* f. *Let* $r = \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{z})$. *Let* X_i *(resp.* Y_i*), for* $i = 1, \ldots, \lfloor rm \rfloor$, *denote the number of points which fall into the farm which is* i *farms to the north (resp. south), from* f *in the order of the exits along the first ridge. Let* $\mathcal{X} = \big\langle X_1, \ldots, X_{\lfloor rm \rfloor}, Y_1, \ldots, Y_{\lfloor rm \rfloor}\big\rangle$. *Then the probability that* p *is not eliminated given* $\mathcal{X}$ *is*

$$\mathbf{Pr}\Big[\mathsf{p} \textit{ is alive} \;\Big|\; \mathcal{X}\Big] \geq e^{-T}, \quad \textit{where} \quad T = \pi m \sum_{i=-\lfloor rm \rfloor}^{\lfloor rm \rfloor} (r - i/m)^2 (X_i + Y_i)/2.$$

Proof: First note that for $i' > \lfloor rm \rfloor$, we have that $i'/m \geq (\lfloor rm \rfloor + 1)/m > r$. Namely no point from a farm i' farms away can eliminate p, and hence we can ignore such farms.

Given the value X_i and Y_i for all i, whether a farm contains a point which eliminates p is independent from whether any other farm contains a point which eliminates p. Therefore, by Lemma 7.4.7,

$$\begin{aligned}
\mathbf{Pr}\Big[\mathsf{p} \text{ is alive} \,\Big|\, \mathcal{X}\Big] &= \left(\prod_{i=1}^{\lfloor rm\rfloor} \alpha_{X_i}\right)\left(\prod_{i=1}^{\lfloor rm\rfloor} \alpha_{Y_i}\right) \\
&\geq \left(\prod_{i=1}^{\lfloor rm\rfloor} \exp\big(-m(r-i/m)^2\pi X_i/2\big)\right)\left(\prod_{i=1}^{\lfloor rm\rfloor} \exp\big(-m(r-i/m)^2\pi Y_i/2\big)\right) \\
&= \exp\left(-m\sum_{i=1}^{\lfloor rm\rfloor}(r-i/m)^2\pi(X_i+Y_i)/2\right)
\end{aligned}$$

□

Lemma 7.4.9 *Let X be a positive random variable with expected value μ. We then have that* $\mathbf{E}\big[e^{-X}\big] \geq e^{-2\mu}/2$

Proof: Markov's inequality implies that $\mathbf{Pr}[X < 2\mu] = 1 - \mathbf{Pr}[X \geq 2\mu] \geq 1 - \mu/2\mu = 1/2$. Therefore, by the definition of expectation, we get $\mathbf{E}\big[e^{-X}\big] \geq \mathbf{Pr}[X \geq 2\mu]\cdot 0 + \mathbf{Pr}[X < 2\mu]\cdot e^{-2\mu} \geq e^{-2\mu}/2$. □

Lemma 7.4.10 *Let p and z be the dominating point and exit, respectively, of some farm f, and let $r = \mathsf{d}_{\mathcal{T}}(\mathsf{p},\mathsf{z})$. Then the probability that p is alive is at least $\frac{1}{2}\exp\big(-2r^3m^2\big)$.*

Proof: Let X_i and Y_i, for $i = 1,\ldots,\lfloor rm\rfloor$, be random variables equal to the number of points which fall into the farm which is i farms to the north or south, respectively, from f in the order of the exits along the first ridge (note that if there is no farm i farms to the north or south, then $X_i = 0$ or $Y_i = 0$, respectively).

For the quantity T from Lemma 7.4.8 we have

$$\begin{aligned}
\mathbf{E}[T] = \mathbf{E}\left[\pi m\sum_{i=1}^{\lfloor rm\rfloor}(r-i/m)^2\frac{X_i+Y_i}{2}\right] &= \pi m\sum_{i=1}^{\lfloor rm\rfloor}(r-i/m)^2\frac{\mathbf{E}[X_i]+\mathbf{E}[Y_i]}{2} \\
&\geq \pi m\sum_{i=1}^{\lfloor rm\rfloor}(r-i/m)^2 \\
&\geq \pi m\cdot rm\cdot r^2 \\
&= \pi r^3m^2,
\end{aligned}$$

by linearity of expectation and since $\mathbf{E}[X_i] \leq 1$ and $\mathbf{E}[Y_i] \leq 1$ for $i = 1,\ldots,\lfloor rm\rfloor$.

For the probability that p is alive, we have

$$\begin{aligned}
\mathbf{Pr}\Big[\mathsf{p} \text{ is alive}\Big] &= \sum_X \mathbf{Pr}\Big[(\mathsf{p} \text{ is alive}) \cap (\mathcal{X} = X)\Big] \\
&= \sum_X \mathbf{Pr}\Big[\mathsf{p} \text{ is alive} \,\Big|\, \mathcal{X} = X\Big]\,\mathbf{Pr}[\mathcal{X} = X] \\
&= \mathbf{E}\Big[\mathbf{Pr}\Big[\mathsf{p} \text{ is alive} \,\Big|\, \mathcal{X}\Big]\Big],
\end{aligned}$$

where $\mathcal{X} = \left\langle X_1, \ldots, X_{\lfloor rm \rfloor}, Y_1, \ldots, Y_{\lfloor rm \rfloor} \right\rangle$, as before.

By Lemma 7.4.8 and Lemma 7.4.9, we have

$$\mathbf{E}\left[\mathbf{Pr}\left[\mathsf{p} \text{ is alive} \;\middle|\; \mathcal{X}\right]\right] \geq \mathbf{E}\left[e^{-T}\right] \geq \frac{1}{2}\exp\Bigl(-2\,\mathbf{E}[T]\Bigr)$$

Putting everything together, we conclude that $\mathbf{Pr}\left[\mathsf{p} \text{ is alive}\right] \geq \frac{1}{2}\exp\bigl(-2\pi r^3 m^2\bigr)$. $\square$

Observation 7.4.11 Lemma 7.4.10 implies that if $r \leq 1/m^{2/3}$ then

$$\mathbf{Pr}[\mathsf{p} \text{ is alive}] \geq \frac{1}{2}\exp\bigl(-2\pi r^3 m^2\bigr) \geq \frac{1}{2e^{2\pi}}.$$

Lemma 7.4.12 *The probability that a farm f gives rise to a Voronoi cell that is not eliminated is at least $\pi/(8e^{2\pi}m^{1/3})$.*

Proof: Let p be the dominating point of f, and let R be the distance of p from the entrance e of f. Also, let q be a random point selected uniformly from f. By Observation 7.4.11 and Lemma 7.4.5, we have

$$\begin{aligned}
\mathbf{Pr}\left[\mathsf{p} \text{ is alive}\right] &\geq \mathbf{Pr}\left[(\mathsf{p} \text{ is alive}) \cap \left(\mathrm{R} \leq m^{-2/3}\right)\right] \\
&= \mathbf{Pr}\left[\, \mathsf{p} \text{ is alive} \;\middle|\; \mathrm{R} \leq m^{-2/3}\right] \mathbf{Pr}\left[\mathrm{R} \leq m^{-2/3}\right] \\
&\geq \frac{1}{2e^{2\pi}}\,\mathbf{Pr}\left[\mathrm{R} \leq m^{-2/3}\right] \\
&\geq \frac{1}{2e^{2\pi}}\,\mathbf{Pr}\left[\, \|\mathsf{q} - \mathsf{e}\| \leq m^{-2/3}\right] \\
&= \frac{1}{2e^{2\pi}} \frac{m\left(m^{-2/3}\right)^2 \pi}{4} \\
&= \frac{\pi}{8e^{2\pi}m^{1/3}},
\end{aligned}$$

by Eq. (7.1). $\square$

Theorem 7.4.13 *For any n, m with $m \leq n$ there exists a terrain of complexity n such that in expectation the Voronoi diagram of m point sites sampled uniformly in the domain will be of complexity $\Omega\bigl(nm^{2/3}\bigr)$.*

Proof: Every farm which receives a point from the random sample has a dominating point. Since $\Theta(m)$ farms were built, and each farm receives one point in expectation, the expected number of dominating points is $\Theta(m)$. Therefore, by Lemma 7.4.12, the expected number of alive dominating points is $\Omega\bigl(m \cdot (\pi/(8e^{2\pi}m^{1/3}))\bigr) = \Omega\bigl(m^{2/3}\bigr)$.

Given that a dominating point is alive, the probability that its Voronoi cell does not reach the rightmost ridge is negligible. Therefore, in expectation, if there are $\Omega\bigl(m^{2/3}\bigr)$ alive dominating points and $\Theta(n)$ ridges then the complexity of the Voronoi diagram will be $\Omega\bigl(nm^{2/3}\bigr)$. $\square$

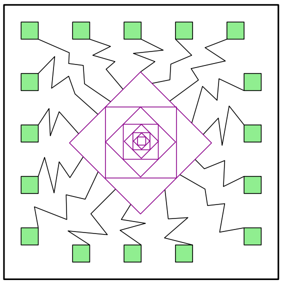
7.5 Concluding remarks

We investigated the expected combinatorial complexity of geodesic Voronoi diagrams on polyhedral terrains in two settings where the sites are being picked randomly. Usually, such random settings are the great simplifier – for example, the expected complexity of the convex hull of n points picked uniformly in the unit square is $O(\log n)$ – but in our case the situation is considerably more subtle.

We proved that the expected complexity is linear if one assumes low density and bounded slope and if the domain of the terrain is a unit square. On the other hand, we described a worst-case construction of a terrain which implies a super-linear lower bound on the expected complexity if these assumptions are dropped. This implies that our probabilistic analysis alone does not yield a linear complexity.

There are still many interesting open questions for further research. We conclude by discussing some of them in more detail.

Open Problem 7.1 If the realistic input assumptions are partially relaxed, is the expected complexity still linear or can one show other lower bounds? One could, for instance allow the terrain to have a constant number of triangles, where the slope is unbounded, or drop the steepness assumption completely.



Consider the farming layout depicted to the right. The layout tries to emulate the lower bound examples we saw earlier on a low-density terrain. We still assume that the slope can be unbounded, but the so-called *sequence of ridges* has been replaced by a recursive low-density construction. One problem with this construction is that the low-density assumption requires the ridge construction in the center to have a non-negligible area, which in turn could catch points from the random sample and therefore break the analysis.

Open Problem 7.2 Can the analysis of the upper bound be extended to polyhedral surfaces which are not necessarily terrains? For this, one needs to make a different set of assumptions on the surface. A natural assumption would be to bound the doubling dimension[1]. One could require the surface to be formed by fat triangles only and bound the vertex degree of the triangulation. To extend the result on the upper bound, one needs to analyze the bisector complexity on this surface and hence reprove Fact 7.2.5 in this setting. Since the previous argument was carried out solely in the projection, a different notion of low-density would be necessary now, i.e., one that uses geodesic discs.

[1]The doubling dimension of a metric space X is the smallest positive integer k such that every ball of X can be covered by $2k$ balls of half the radius.

CHAPTER 8

Conclusions

8.1 Summary

In this thesis we saw realistic analysis being applied to a variety of research problems. We saw that it can help to solve long-standing open problems, such as approximating the Fréchet distance in subquadratic time (Chapter 3). We used it to devise a simple approximation algorithm for a sophisticated variant of the Fréchet distance, the shortcut Fréchet distance (Chapter 5). Even though computing this variant of the Fréchet distance is NP-hard in the general case, the presented approximation algorithm runs in near-linear time if the input curves are c-packed and the shortcuts are sufficiently restricted. Along the way, we developed data structures to support Fréchet-distance queries, which are among the first of their kind (Chapter 4).

We went on to investigate hydrological computations on digital terrain models (Chapter 6). We observed that the steepest descent paths which carry the water are extremely sensitive to elevation error, which is inevitable in real data. To cater for this, we allowed the elevations to be imprecise. It turns out that in this case even deciding if water can flow between two points on a polyhedral terrain is NP-hard. However, we were able to give algorithms to compute watersheds in near-linear time under a discrete model of the terrain surface.

Finally, we showed that realistic analysis can be used to close the gap between complexities observed in practice and theoretical worst-case bounds. We proved that geodesic Voronoi diagrams can be expected to have linear complexity on well-behaved terrains (Chapter 7), even though their complexity is quadratic in the worst case.

These results demonstrate that realistic analysis, and in particular the realistic input models packedness and low-density, are a powerful tool in the study of algorithmic problems for real data.

8.2 Outlook

A number of open problems have been discussed already in the concluding remarks of the individual chapters of this thesis, including ideas on how to approach them. In this final section, we want to put the contributions of this thesis into a wider context by discussing two particular research areas where our results may become relevant. We first discuss a fundamental problem in structural biology where the shape matching techniques of Chapters 3 and 5 open a new perspective. Secondly, we revisit the problem studied in Chapter 6 in a critical literature review across several scientific disciplines.

8.2.1 Shape-matching protein structures

Aligning two protein structures with respect to similarity is a key problem in structural biology. The three-dimensional shapes of the protein structures are known to preserve information about distant evolutionary relations, unlike their one-dimensional counterparts, the amino-acid sequences. Much of this information is encoded in substructures called *domains*, which are not easy to identify. Refer to Figure 8.1 for a schematic drawing of a three-dimensional protein structure.

A commonly used representation of a protein structure consists of a polygonal curve that spans an ordered succession of residue centers (C-α atoms). Their linear order along this curve is considered a key aspect in shape comparison [94]. The standard method to align two such curves is to minimize their *root-mean-square deviation (RMSD)* under rigid motions using the algorithm by Kabsch or equivalent algorithms [45]. This method has two major drawbacks:
(A) a one-to-one correspondence of the C-α atoms has to be established in advance,
(B) the portions of the curves which are dissimilar dominate the global distance value.
Thus, curves which have similar domains, but are not entirely similar, cannot be properly aligned. A standard distance measure to compare the string representations of amino-acid sequences is the well-known *edit distance*, which does not have the issues described above. However, the edit distance cannot be used to compare three-dimensional shapes.

Recently there has been some interest in using the *Fréchet distance* in place of the RMSD [161, 96, 159]. One can argue that the Fréchet distance captures geometric shape similarity while being to a large extent independent of the sampling of the curve. Thus, it should be superior to the RMSD with respect to issue (A) described above. However, it has the same disadvantages as described under (B) above. A variant of the Fréchet distance which would also be superior with respect to (B), is the shortcut Fréchet distance, which is discussed in Chapter 5. It can be interpreted as an edit distance for polygonal curves. Here, introducing a shortcut corresponds to performing one edit to the curve. It is most likely suitable for alignment with respect to similar domains that are unknown in advance. One could imagine modifying the price of a tunnel in the free-space diagram to introduce a penality for performing a shortcut. If the chosen penalty preserves the monotonicity of the tunnel prices, then this does not affect the analysis of the presented algorithms.

The results presented in this thesis are an important step towards conducting the research outlined above. The next step would be to develop an approximation

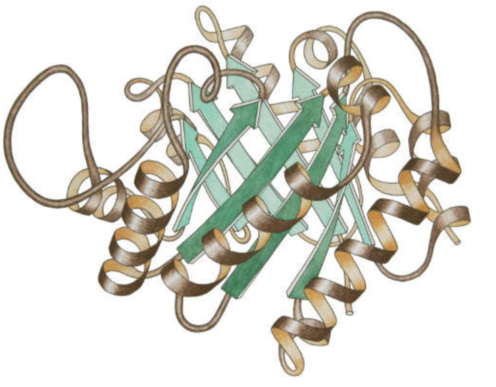


Figure 8.1: Schematic drawing of a 3D-protein structure by Jane Richardson.

algorithm for the Fréchet distance under transformations (Open Problem 3.3). Subsequently, these results could be extended to the shortcut Fréchet distance. Here, one of the main open problems is to analyze the tunnel events which may happen if tunnels can start and end inside free-space cells (Open Problem 5.2). To circumvent the NP-hardness of the problem, one could aim to design a fixed-parameter tractable algorithm with respect to the number of shortcuts used. Furthermore, realistic input models for protein structures as suggested by Aronov *et al.* [20] could be used in the analysis.

8.2.2 Towards robust hydrological computations

Simulating the flow of water on a terrain has a long history in hydrology. Applications range from regular weather forecasting over inundation modelling (for heavy rainfall and flash floods) to climate change predictions. Beven distinguishes the simulation problem from the forecasting problem [26]. In the former, hydrologists aim to understand the complex dynamics of catchment areas by developing holistic hydrological models, while in the forecasting problem a particular response of the catchment should be predicted within a fixed lead time. The former is for research purposes while the latter is purely practical. However, the former models are also used for the latter purposes. Over the years different models have been proposed, among the most popular are *Topmodel* (Beven and Kirkby [27]) and *SWAT* [16]. A recent adaptation of topmpodel is the *Isba-Topmodel* used by Girard *et al.* [79], for example. Both models use the size of the upstream area (also called contributing area or watershed) of a point on the terrain as a local parameter to determine the response of the catchment to rainfall. In the hydrology literature this is called a *hydrologic index*.

In order to automatically determine hydrologic indices from a digital elevation model, one needs to define flow directions on the terrain surface. Since water flow is predominantly affected by gravity, these directions are modelled after the gradient or steepest descent. The literature has focussed mostly on grid terrains, and a model which is commonly used is the Deterministic-8 (D-8) model, where water can flow to one of the eight neighbors of a grid cell. According to Wechsler [154] the hydrological community has not reached a consensus on appropriate algorithms for computing flow direction and this is also reflected by the frequency of algorithms being published (see for example [132, 124, 144, 146]).

Elevation data usually contains what hydrologists call *sinks* and *flats*, which pose a particular challenge to the automated derivation of fully connected drainage networks [108]. A sink is a local minimum and a flat is an area where the slope is zero. Both could be either naturally present in the real terrain (a sink could be due to a bridge, a flat could be the surface of a glacial lake or a flood plain) or introduced due to measurement errors. The challenge lies in distinguishing spurious from real features. To handle them, one can either modify the elevation data, or define altered flow directions based on additional topographic indices. For both approaches a plethora of methods exists, for an overview see Kenny *et al.* [99]. Furthermore, it is common practice to alter the data either manually or automatically by "carving" streams into the surface in order to force the "correct" outcome of flow computations. Consequently, these surface modification procedures are also referred to as *hydrological correction*. Wechsler notes [154] that the impact of these procedures on derived hydrologic indices has not been addressed in the literature.

Recently, the influence of elevation error on hydrological computations was brought up by Lindsay and Evans [109]. A study of three basins in the UK showed that several hydrologic indices can have multimodal frequency distributions under moderate elevation error (averaged 1.8m), i.e. two or more estimated values were highly likely. This is not surprising since there might be more than one adjacent basin that water could spill into by a minor modification of the elevations near the spill-points. In fact, this is a property which we exploited in the NP-hardness reduction described in Section 6.2.

Independently, similar results were found by Hebeler and Purves [91]. They studied three mountainous regions (Alps, Pyrenees and Turkey) and showed that global statistics for a range of topographic indices can be made robust to the introduction of uncertainty. However, the derivation of hydrologic indices was shown to vary significantly as a result of the introduced uncertainty. Figure 8.2 is a reproduction of a figure from their paper which illustrates this variation by focussing on two particular watersheds.

Wechsler raises similar issues regarding uncertainty in a survey [154], where he also discusses how stochastic methods can help in estimating the uncertainty in a hydrological model. In fact there is an ongoing discussion about uncertainty in the hydrological literature [136, 120, 158] which was initiated by Beven [25]. Note that Beven raised the fundamental issues involved in deriving parameters from elevation data, which are discussed above, already in the late nineties [24]. Buytaert *et al.* [40] observe that low prediction rates[1] are considered acceptable among hydrologists. According to Wagener and Gupta it is being broadly recognized that proper consideration of uncertainty in hydrologic predictions is essential for purposes of both research and operational modeling [151].

One way to improve the accuracy of the model is to incorporate historical data about the response of the catchment to rainfall; that is, time series data which contain information about observed rainfall and runoff of the specific catchment. To estimate the runoff, typically water levels are measured at several points in the river

[1]Measured using the standard Nash-Sutcliffe model efficiency coefficient.

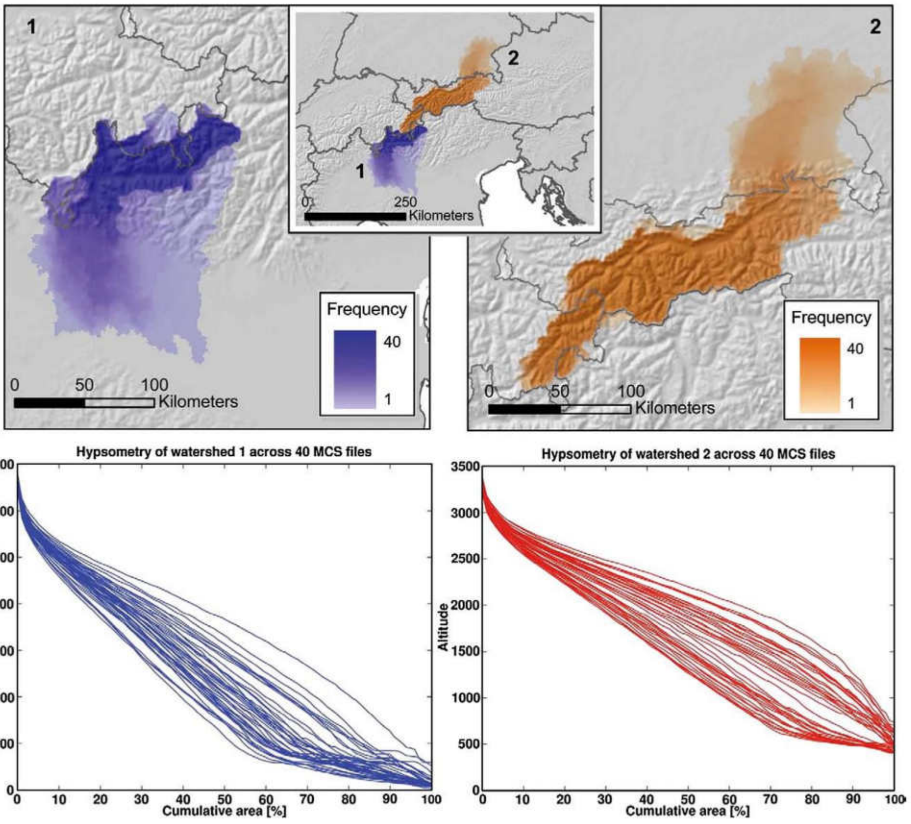


Figure 8.2: Reproduction of a figure by Hebeler and Purves [91]. It shows the frequency of cells belonging to two selected watersheds W1 (left) and W2 (right) across all sampled realizations of the terrain. Most variation in watershed size is detectable in the lower regions of the catchment area, but some variation is also evident in the high mountain regions of both watersheds. Hypsometric curve (bottom) across all samples for the two selected watersheds, showing a considerable amount of variation in form. A hypsometric curve is an empirical cumulative distribution function of elevations in a catchment. It provides hydrologist and geomorphologist with a way to assess the similarity of watersheds.

which receives the water. Gupta *et al.* [150, 110] propose so-called *data assimilation* techniques for this purpose, which have previously been used in the atmospheric and oceanic sciences.

In the hydrological literature, the optimization of the output of a forecasting model with respect to historical data is called *model calibration.* Basic machine learning techniques for calibration such as regression analysis and Kalman filters have always been in the standard toolkit of hydrologists [97]. In his book [26], Beven gives an overview of state-of-the-art hydrological models for calibration and discusses how to estimate their uncertainty. Theoretically, these algorithms do not need any explicit elevation information as long as they have enough historical data. Beven also argues that, contrary to common belief, decision makers can deal with explicit uncertainty in the output of a GIS. He also raises philosophical questions and questions of accountability in this context.

Previous research in computational geometry has focussed mostly on the efficiency of drainage network computations assuming elevation data is exact. We can identify two main directions of research here. Firstly, an intriguing aspect of this work is that it enables the computation of drainage networks according to the true steepest descent along a polyhedral surface. In this context it should be mentioned that the D-8 model has been criticized in the GIS-literature for allowing only a constant number of stream directions, and as a result creating many artifactual streams which run in parallel. On the other hand, when tracing rivers along the true gradients of a polyhedral surface, the river network can have very high complexity. Even if the surface is defined by a Delaunay triangulation, which is considered well-behaved in practice, it can have a complexity which is cubic in the complexity of the terrain [52]. In a recent result, de Berg *et al.* trace flow paths implicitly along the surface resulting in a quadratic-time algorithm for computing the area of a watershed [54]. Algorithms for watershed computations on polyhedral terrains have been implemented in an experimental setting by de Berg and Tsirogiannis, see [58]. Note that there is also work on hydrological correction for polyhedral terrains [59, 149, 135, 81].

Another line of research in this context is the study of I/O-efficiency on discrete terrains. The resulting algorithms are efficient even on normal desktop computers for high-resolution terrain data, as accurate as one elevation point per square meter, For comparison, at the time of the publication of Beven's critique [24] in 1997, the common resolution of digital terrain data was ranging from 10 m to 1 km or more. The algorithms have been implemented and are commercially available [49]. Arge *et al.* [15] also note that the contributing area is not a static index. Basins may fill up during rainfall and start flowing into adjacent basins thereby forming bigger catchments. Therefore, they model and compute the flow of water over time. Additional references can be found in a recent survey by Haverkort and Toma [89].

To summarize, while the problem of uncertainty in flow prediction is acknowledged as a major problem by hydrologists and elevation error is one of the main sources of this uncertainty, none[2] of the research in computational geometry, neither on polyhedral surfaces nor on grid terrains, take elevation error into account so far.

In Chapter 6 we studied water-flow under a non-probabilistic model of imprecision. In this model, the exact elevation of each surface point is replaced by an imprecision interval. The outcome of the most optimistic and pessimistic scenarios are then

[2]Apart from [82] and [81], which is for hydrological correction only.

computed exactly. This enables efficient algorithms to compute conservative upper and lower bounds on hydrologic indices. It would be interesting to improve the I/O-efficiency (Open Problem 6.1). However, the next step would be to start a discussion with domain experts to evaluate the practical relevance of the presented flow model (see also Problems 6.2 and 6.3). There seems to be little communication between the disciplines of computational geometry and hydrology about these basic concepts, at least as far as it is documented in the literature. One may conjecture that great potential in interdisciplinary research lies here.

References

[1] M. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. *Discrete & Computational Geometry*, 43:497–515, 2010.

[2] I. Abraham, A. Fiat, A. V. Goldberg, and R. F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms*, pages 782–793, 2010.

[3] P. K. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms*, pages 156–167, 2013.

[4] P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification in two and three dimensions. *Algorithmica*, 42:203–219, 2005.

[5] P. K. Agarwal, T. Mølhave, and B. Sadri. I/O-efficient contour queries on terrains. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 268–284, 2011.

[6] C. Aggarwal and S. Yu. An effective and efficient algorithm for high-dimensional outlier detection. *International Journal on Very Large Data Bases*, 14:211–221, April 2005.

[7] H.-K. Ahn, C. Knauer, M. Scherfenberg, L. Schlipf, and A. Vigneron. Computing the discrete Freéchet distance with imprecise input. *International Journal of Computational Geometry & Applications*, 22(1):27–44, 2012.

[8] H. Alani, C. B. Jones, and D. Tudhope. Voronoi-based region approximation for geographical information retrieval with gazetteers. *International Journal of Geographical Information Science*, 15(4):287–306, 2001.

[9] H. Alt. The computational geometry of comparing shapes. In *Efficient Algorithms: Essays Dedicated to Kurt Mehlhorn on the Occasion of His 60th Birthday*, pages 235–248. Springer-Verlag, 2009.

[10] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49:262–283, 2003.

[11] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Nher, S. Schirra, and C. Uhrig. Approximate motion planning and the complexity of the boundary of the union of simple geometric figures. *Algorithmica*, 8:391–406, 1992.

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

[13] H. Alt, C. Knauer, and C. Wenk. Matching polygonal curves with respect to the Fréchet distance. In *Proceedings of the 18th Symposium on Theoretical Aspects of Computer Science*, volume 2010 of *Lecture Notes in Computer Science*, pages 63–74. 2001.

[14] H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.

[15] L. Arge, M. Revsbæk, and N. Zeh. I/O-efficient computation of water flow across a terrain. In *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 403–412, 2010.

[16] J. G. Arnold, P. M. Allen, and G. Bernhardt. A comprehensive surface-groundwater flow model. *Journal of Hydrology*, 142(1):47–69, 1993.

[17] B. Aronov, M. de Berg, and S. Thite. The complexity of bisectors and Voronoi diagrams on realistic terrains. In *Proceedings of the 16th Annual European Symposium on Algorithms*, pages 100–111, 2008.

[18] B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories on non-monotone criteria. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms*, pages 1897–1911, 2013.

[19] B. Aronov and S. Har-Peled. On approximating the depth and related problems. *SIAM Journal of Computing*, 38(3):899–921, 2008.

[20] B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet distance for curves, Revisited. In *Proceedings of the 14th Annual European Symposium on Algorithms*, pages 52–63, 2006.

[21] F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

[22] R. Beecham, J. Wood, and A. Bowerman. A visual analytics approach to understanding cycling behaviour. In *Proceedings of the 2012 IEEE Conference on Visual Analytics Science and Technology*, pages 207–208.

[23] S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proceedings of the 8th Latin American Conference on Theoretical Informatics*, pages 630–641, 2008.

[24] K. Beven. Topmodel: a critique. *Hydrological Processes*, 11(9):1069–1085, 1997.

[25] K. Beven. On undermining the science? *Hydrological Processes*, 20(14):3141–3146, 2006.

[26] K. Beven. *Environmental modelling: An uncertain future?* Routledge, 2009.

[27] K. Beven and M. Kirkby. A physically based, variable contributing area model of basin hydrology/un modèle à base physique de zone d'appel variable de l'hydrologie du bassin versant. *Hydrological Sciences Journal*, 24(1):43–69, 1979.

[28] M. Borga, E. Gaume, J. Creutin, and L. Marchi. Surveying flash floods: gauging the ungauged extremes. *Hydrological Processes*, 22:3883–3885, 2008.

[29] P. Bose, O. Cheong, and V. Dujmović. A note on the perimeter of fat objects. *Computational Geometry: Theory and Applications*, 44(1):1–8, 2011.

[30] 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, 2005.

[31] 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*, pages 288–297, 2008.

[32] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, pages 644–655, 2008.

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

[34] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog—with an application to Alt's conjecture. *Computing Research Repository (arXiv)*, abs/1209.4403, 2012.

[35] K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 645–654, 2009.

[36] K. Buchin, M. Buchin, and C. Wenk. Computing the Fréchet distance between simple polygons. *Computational Geometry: Theory and Applications*, 41(1-2):2–20, 2008.

[37] M. Buchin, A. Driemel, and B. Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Abstracts of the 29th European Workshop on Computational Geometry*, pages 43–46, 2013. Full version available at `http://arXiv.org/abs/1307.2097`.

[38] M. Buchin, A. Driemel, M. van Kreveld, and V. Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3(1):33–63, 2011.

[39] D. Butler. Virtual globes: The web-wide world. *Nature*, 439(7078):776–778, 2006.

[40] W. Buytaert, D. Reusser, S. Krause, and J. Renaud. Why can't we do better than Topmodel? *Hydrological Processes*, 22:4175–4179, 2008.

[41] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the Association for Computing Machinery*, 42:67–90, 1995.

[42] R. Cavalli and S. Grigolato. Influence of characteristics and extension of a forest road network on the supply cost of forest woodchips. *Journal of Forest Research*, 15:202–209, 2010.

[43] D. Chen, A. Driemel, L. Guibas, A. Nguyen, and C. Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the 13th Workshop on Algorithm Engineering & Experiments*, pages 75–83, 2011.

[44] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, third edition, 2009.

[45] E. A. Coutsias, C. Seok, and K. A. Dill. Using quaternions to calculate RMSD. *Journal of Computational Chemistry*, 25(15):1849–57, 2004.

[46] W. Craddock, E. Kirby, N. Harkins, H. Zhang, X. Shi, and J. Liu. Rapid fluvial incision along the Yellow River during headward basin integration. *Nature Geoscience*, 3:209–213, 2010.

[47] I. F. Cruz, D. Agrawal, C. S. Jensen, E. Ofek, and E. Tanin, editors. *Proceedings of the 19th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*, 2011.

[48] I. F. Cruz, C. Knoblock, P. Kröger, E. Tanin, and P. Widmayer, editors. *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2012.

[49] A. Danner, T. Mølhave, K. Yi, P. K. Agarwal, L. Arge, and H. Mitásová. TerraStream: from elevation data to watershed hierarchies. In *Proceedings of the 15th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 212–219, 2007.

[50] M. de Berg. Improved bounds on the union complexity of fat objects. *Discrete & Computational Geometry*, 40(1):127–140, 2008.

[51] M. de Berg, P. Bose, K. Dobrint, M. J. van Kreveld, M. H. Overmars, M. de Groot, T. Roos, J. Snoeyink, and S. Yu. The complexity of rivers in triangulated terrains. In *Proceedings of the 8th Canadian Conference on Computational Geometry*, pages 325–330, 1996.

[52] M. de Berg, O. Cheong, H. Haverkort, J.-G. Lim, and L. Toma. The complexity of flow on fat terrains and its I/O-efficient computation. *Computational Geometry: Theory and Applications*, 43(4):331–356, 2010.

[53] M. de Berg, A. F. Cook IV, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry: Theory and Applications*, 46(6):747 – 755, 2013.

[54] M. de Berg, H. Haverkort, and C. Tsirogiannis. Implicit flow routing on terrains with applications to surface networks and drainage structures. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms*, pages 285–296, 2011.

[55] M. de Berg, H. Haverkort, and C. P. Tsirogiannis. Visibility maps of realistic terrains have linear smoothed complexity. *Journal of Computational Geometry*, 1(1):57–71, 2010.

[56] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. *Algorithmica*, 34:81–97, 2002.

[57] M. de Berg and M. Streppel. Approximate range searching using binary space partitions. *Computational Geometry: Theory and Applications*, 33(3):139 – 151, 2006.

[58] M. de Berg and C. P. Tsirogiannis. Exact and approximate computations of watersheds on triangulated terrains. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 74–83, 2011.

[59] T. de Kok, M. van Kreveld, and M. Löffler. Generating realistic terrains with higher-order Delaunay triangulations. *Computational Geometry: Theory and Applications*, 36(1):52–65, 2007.

[60] O. Devillers, J. Erickson, and X. Goaoc. Empty-ellipse graphs. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms*, pages 1249–1257, 2008.

[61] T. Dey. *Curve and Surface Reconstruction: Algorithms with Mathematical Analysis*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2011.

[62] M. T. Dickerson and M. T. Goodrich. Two-site Voronoi diagrams in geographic networks. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 59:1–59:4, 2008.

[63] E. W. Dijkstra. On a methodology of design (ewd-317). Circulated privately. Available at `http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD317.PDF`.

[64] E. W. Dijkstra. On the cruelty of really teaching computing science (ewd-1036). Circulated privately, 1988. Available at `http://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF`.

[65] E. W. Dijkstra. Why American Computing Science seems incurable (ewd-1209). Circulated privately, 1995. Available at `http://www.cs.utexas.edu/users/EWD/ewd12xx/EWD1209.PDF`.

[66] E. W. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, 1982.

[67] A. Driemel and S. Har-Peled. Jaywalking your dog – computing the Fréchet distance with shortcuts. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 318–337, 2011.

[68] A. Driemel and S. Har-Peled. Jaywalking your dog – computing the Fréchet distance with shortcuts. *SIAM Journal of Computing*, 2013. To appear.

[69] A. Driemel, S. Har-Peled, and B. Raichel. On the expected complexity of Voronoi diagrams on terrains. In *Proceedings of the 28th ACM Symposium on Computational Geometry*, pages 101–110, 2012.

[70] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near-linear time. In *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 365–374, 2010.

[71] 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.

[72] A. Driemel, H. Haverkort, M. Löffler, and R. Silveira. Flow computations on imprecise terrains. In *Proceedings of the 12th Algorithms and Data Structures Symposium - WADS (formerly Workshop on Algorithms and Data Structures)*, pages 350–361, 2011.

[73] A. Driemel, H. Haverkort, M. Löffler, and R. Silveira. Flow computations on imprecise terrains. *Journal of Computational Geometry*, 4(1):38–78, 2012.

[74] A. Efrat. The complexity of the union of (α, β)-covered objects. *SIAM Journal of Computing*, 34(4):775–787, 2005.

[75] A. Efrat, P. Indyk, and S. Venkatasubramanian. Pattern matching for sets of segments. In *Proceedings of the 12th ACM-SIAM Symposium on Discrete Algorithms*, pages 295–304, 2001.

[76] J. Erickson. On the relative complexities of some geometric problems. In *Proceedings of the 7th Canadian Conference on Computational Geometry*, pages 85–90, 1995.

[77] P. F. Fisher and N. J. Tate. Causes and consequences of error in digital elevation models. *Progress in Physical Geography*, 30(4):467–489, 2006.

[78] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: sorted matrices. *SIAM Journal of Computing*, 13:14–30, 1984.

[79] C. Girard, T. Godfroy, M. Erlich, E. David, C. Sorbet, V. Pourret, M. Veysseire, and B. Vincendon. 2D hydraulic model integration to real-time flash flood forecasting chain. In *Comprehensive Flood Risk Management*, pages 382–383. CRC Press, 2013.

[80] M. F. Goodchild. Geographical information science. *International Journal of Geographical Information Science*, 6(1):31–45, 1992.

[81] C. Gray, F. Kammer, M. Löffler, and R. I. Silveira. Removing local extrema from imprecise terrains. *Computational Geometry: Theory and Applications*, 45:334–349, 2012.

[82] C. Gray, M. Löffler, and R. I. Silveira. Smoothing imprecise 1.5D terrains. *International Journal of Computational Geometry & Applications*, 20(4):381–414, 2010.

[83] J. Gudmundsson and M. Smid. Fréchet queries in geometric trees. In *Proceedings of the 21st Annual European Symposium on Algorithms*, 2013. To appear.

[84] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. Snoeyink. Approximating polygons and subdivisions with minimum link paths. In *Proceedings of the 2nd International Symposium on Algorithms*, pages 151–162, 1991.

[85] E. Gurarie, R. D. Andrews, and K. L. Laidre. A novel method for identifying behavioural changes in animal movement data. *Ecology Letters*, 12(5):395–408, 2009.

[86] S. Har-Peled. *Geometric Approximation Algorithms.* American Mathematical Society, Boston, MA, USA, 2011.

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

[88] S. Har-Peled and B. Raichel. The Fréchet distance revisited and extended. *ACM Transactions on Algorithms*, 2013. To appear.

[89] H. Haverkort and L. Toma. Terrain modeling for the geoscience. In T. Gonzales, editor, *Computing Handbook Set – Computer science*, volume 1. CRC Press. To appear.

[90] H. Haverkort and C. Tsirogiannis. Flow on noisy terrains: An experimental evaluation. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 84–91, 2011.

[91] F. Hebeler and R. Purves. The influence of elevation uncertainty on derivation of topographic indices. *Geomorphology*, 111(1-2):4–16, 2009.

[92] 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.

[93] D. I. Heywood, S. Cornelius, and S. Carver. *An Introduction to Geographical Information Systems.* Prentice Hall, third edition, 2006.

[94] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273(5275):595–603, 1996.

[95] F. Jarai-Szabo and Z. Neda. On the size-distribution of Poisson Voronoi cells. *Computing Research Repository (arXiv)*, cond-mat/0406116, June 2004.

[96] M. Jiang, Y. Xu, and B. Zhu. Protein structure-structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 06(01):51–64, 2008.

[97] M. Karlsson and S. Yakowitz. Rainfall-runoff forecasting methods, old and new. *Stochastic Hydrology and Hydraulics*, 1(4):303–318, 1987.

[98] J. Kay, P. Lukowicz, H. Tokuda, P. Olivier, and A. Krüger, editors. *Proceedings of the 10th International Conference on Pervasive Computing*, volume 7319 of *Lecture Notes in Computer Science*, 2012.

[99] F. Kenny, B. Matthews, and K. Todd. Routing overland flow through sinks and flats in interpolated raster terrain surfaces. *Computers & Geosciences*, 34(11):1417–1430, 2008.

[100] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping to massive dataset. In *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, pages 1–11, 1999.

[101] Y. Kholondyrev and W. Evans. Optimistic and pessimistic shortest paths on uncertain terrains. In *Proceedings of the 19th Canadian Conference on Computational Geometry*, pages 197–200, 2007.

[102] M. Kim, S. Kim, and M. Shin. Optimization of subsequence matching under time warping in time-series databases. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 581–586.

[103] R. D. Koster, S. P. P. Mahanama, B. Livneh, D. P. Lettenmaier, and R. H. Reichle. Skill in streamflow forecasts derived from large-scale estimates of soil moisture and snow. *Nature Geoscience*, 3(9):613–616, 2010.

[104] R. Kunze, F.-E. Wolter, and T. Rausch. Geodesic Voronoi diagrams on parametric surfaces. In *Proceedings of the 1997 IEEE Conference on Computer Graphics International*, pages 230–237.

[105] S. Kwong, Q. H. He, K. F. Man, K. S. Tang, and C. W. Chau. Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(5):573–594, August 1998.

[106] J. K. Laurila, D. Gatica-Perez, I. Aad, J. Blom, O. Bornet, T.-M.-T. Do, O. Dousse, J. Eberle, and M. Miettinen. The mobile data challenge: Big data for mobile computing research. In *Proceedings of the 2012 Workshop on the Nokia Mobile Data Challenge*, pages 1–8.

[107] X.-Y. Li, P.-J. Wan, and O. Frieder. Coverage in wireless ad-hoc sensor networks. *IEEE Transactions on Computers*, 52(6):753–763, 2003.

[108] C. Liang and D. S. Mackay. A general model of watershed extraction and representation using globally optimal flow paths and up-slope contributing areas. *International Journal of Geographical Information Science*, 14(4):337–358, 2000.

[109] J. Lindsay and M. Evans. The influence of elevation error on the morphometrics of channel networks extracted from DEMs and the implications for hydrological modelling. *Hydrological Processes*, 22(11):1588–1603, 2008.

[110] Y. Liu and H. V. Gupta. Uncertainty in hydrologic modeling: Toward an integrated data assimilation framework. *Water Resources Research*, 43(7), 2007.

[111] Y. Liu and J. Snoeyink. Flooding triangulated terrain. In *Proceedings of the 11th International Symposium on Spatial Data Handling*, pages 137–148, 2005.

[112] M. Löffler. *Data Imprecision in Computational Geometry.* PhD thesis, Utrecht University, 2009.

[113] R. Maronna, D. Martin, and V. Yohai. *Robust Statistics: Theory and Methods.* Wiley, 2006.

[114] A. Mascret, T. Devogele, I. L. Berre, and A. Hénaff. Coastline matching process based on the discrete Fréchet distance. In *Proceedings of the 12th International Symposium on Spatial Data Handling*, pages 383–400, 2006.

[115] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl. Fat triangles determine linearly many holes. *SIAM Journal of Computing*, 23(1):154–169, 1994.

[116] M. McAllister and J. Snoeyink. Extracting consistent watersheds from digital river and elevation data. In *Proceedings of the ASPRS/ACSM Annual Conference*, 1999.

[117] F. Mémoli and G. Sapiro. Distance functions and geodesics on submanifolds of $\mathbb{R}^d$ and point clouds. *SIAM Journal on Applied Mathematics*, 65(4):1227–1260, 2005.

[118] E. Moet. *Computation and Complexity of Visibility in Geometric Environments.* PhD thesis, Utrecht University, 2008.

[119] E. Moet, M. J. van Kreveld, and A. F. van der Stappen. On realistic terrains. *Computational Geometry: Theory and Applications*, 41(1-2):48–67, 2008.

[120] A. Montanari. What do we mean by 'uncertainty'? The need for a consistent wording about uncertainty assessment in hydrology. *Hydrological Processes*, 21:841–845, 2006.

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

[122] J. O'Callaghan and D. Mark. The extraction of drainage networks from digital elevation data. *Computer vision, graphics, and image processing*, 28(3):323–344, 1984.

[123] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams.* Wiley, 2nd edition, 2000.

[124] S. Orlandini and G. Moretti. Determination of surface flow paths from gridded elevation data. *Water Resources Research*, 45(3), 2009.

[125] D. L. Pham, C. Xu, and J. L. Prince. Current methods in medical image segmentation. *Biomedical Engineering*, 2:315–337, 2000.

[126] J. Portela and M. Alencar. Cellular network as a multiplicatively weighted Voronoi diagram. In *Proceedings of the 3rd IEEE Consumer Communications and Networking Conference*, volume 2, pages 913—917, 2006.

[127] H. Raynaud. Sur l'enveloppe convex des nuages de points aleatoires dans $\mathbb{R}^n$. *Journal of Applied Probability*, 7:35–48, 1970.

[128] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gewählten Punkten I. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2:75–84, 1963.

[129] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *Journal of Algorithms*, 18(3):548–585, 1995.

[130] Y. Schreiber. An optimal-time algorithm for shortest paths on realistic polyhedra. *Discrete & Computational Geometry*, 43:21–53, 2010.

[131] Y. Schreiber and M. Sharir. An optimal-time algorithm for shortest paths on a convex polytope in three dimensions. *Discrete & Computational Geometry*, 39:500–579, 2008.

[132] J. Seibert and B. L. McGlynn. A new triangular multiple flow direction algorithm for computing upslope areas from gridded digital elevation models. *Water Resources Research*, 43(4):W04501, 2007.

[133] J. Serrà, E. Gómez, P. Herrera, and X. Serra. Chroma binary similarity and local alignment applied to cover song identification. *IEEE Transactions on Audio, Speech & Language Processing*, 16(6):1138–1151, 2008.

[134] T. Sikora. The MPEG-7 visual standard for content description-an overview. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):696–702, 2001.

[135] R. I. Silveira and R. van Oostrum. Flooding countries and destroying dams. *International Journal of Computational Geometry & Applications*, 20(3):361–380, 2010.

[136] B. Sivakumar. The more things change, the more they stay the same: the state of hydrologic modelling. *Hydrological Processes*, 22:4333–4337, 2008.

[137] D. Spielman and S.-H. Teng. Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, pages 296–305, 2001.

[138] E. 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*, pages 461–465, 2007.

[139] I. Stojmenovic, A. P. Ruhil, and D. Lobiyal. Voronoi diagram and convex hull based geocasting and routing in wireless networks. *Wireless Communications and Mobile Computing*, 6(2):247–258, 2006.

[140] G. Stylianou and G. Farin. Crest lines for surface segmentation and flattening. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):536–544, 2004.

[141] G. Stylianou and G. Farin. Crest lines for surface segmentation and flattening. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):536–544, 2004.

[142] O. Takahashi and R. Schilling. Motion planning in a plane using generalized Voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143–150, 1989.

[143] D. Tarboton. A new method for the determination of flow directions and upslope areas in grid digitial elevation models. *Water Resources Research*, 33(2):309–319, 1997.

[144] D. Tarboton, K. Schreuders, D. Watson, and M. Baker. Generalized terrain-based flow analysis of digital elevation models. In *Proceedings of the 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation*, pages 2000–2006, 2009.

[145] D. Tetzlaff, J. McDonnell, S. Uhlenbrook, K. McGuire, P. Bogaart, F. Naef, A. Baird, S. Dunn, and C. Soulsby. Conceptualizing catchment processes: simply too complex? *Hydrological Processes*, 22:1727–1730, 2008.

[146] N. Thommeret, J. Bailly, C. Puech, et al. Extraction of thalweg networks from DTMs: application to badlands. *Hydrology and Earth System Sciences*, 14(8):1527–1536, 2010.

[147] A. F. van der Stappen. Realistic environment models and their impact on the exact solution of the motion planning problem. In *Sensor Based Intelligent Robots*, volume 1724 of *Lecture Notes in Computer Science*, pages 180–199, 1998.

[148] A. F. van der Stappen, M. H. Overmars, M. de Berg, and J. Vleugels. Motion planning in environments with low obstacle density. *Discrete & Computational Geometry*, 20:561–587, 1998.

[149] M. J. van Kreveld and R. I. Silveira. Embedding rivers in polyhedral terrains. In *Proceedings of the 25th ACM Symposium on Computational Geometry*, pages 169–178, 2009.

[150] J. A. Vrugt, C. G. H. Diks, H. V. Gupta, W. Bouten, and J. M. Verstraten. Improved treatment of uncertainty in hydrologic modeling: Combining the strengths of global optimization and data assimilation. *Water Resources Research*, 41, 2005.

[151] T. Wagener and H. V. Gupta. Model identification for hydrological forecasting under uncertainty. *Stochastic Environmental Research and Risk Assessment*, 19(6):378–387, 2005.

[152] Z. Wang, H. Guo, B. Yu, and X. Yuan. Interactive visualization of 160 years global hurricane trajectory data. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 37–38, 2011.

[153] O. Weber, Y. S. Devir, A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Transactions on Graphics*, 27(4):104:1–104:16, 2008.

[154] S. P. Wechsler. Uncertainties associated with digital elevation models for hydrologic applications: a review. *Hydrology and Earth System Sciences*, 11(4):1481–1500, 2007.

[155] W. Weil and J. A. Wieacker. Stochastic geometry. In P. M. Gruber and J. M. Wills, editors, *Handbook of Convex Geometry*, volume B, chapter 5.2, pages 1393–1438. North-Holland, 1993.

[156] C. Wenk. *Shape Matching in Higher Dimensions.* PhD thesis, Free University of Berlin, 2002.

[157] C. Wenk, R. Salas, and D. Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management*, pages 879–888, 2006.

[158] T. Willis, P. Sleigh, and N. Wright. Analysis of uncertainty associated with numerical schemes of inundation models. In *Comprehensive Flood Risk Management*, pages 129–130. CRC Press, 2013.

[159] T. Wylie, J. Luo, and B. Zhu. A practical solution for aligning and simplifying pairs of protein backbones under the discrete Fréchet distance. In *Computational Science and Its Applications*, volume 6784 of *Lecture Notes in Computer Science*, pages 74–83. 2011.

[160] J. Yuan, Y. Zheng, X. Xie, and G. Sun. T-drive: Enhancing driving directions with taxi drivers' intelligence. *IEEE Transactions on Knowledge and Data Engineering*, 25(1):220–232, 2013.

[161] B. Zhu. Protein local structure alignment under the discrete Fréchet distance. *Journal of Computational Biology*, 14(10):1343–1351, 2007.

Publications

The main chapters of this thesis are based on the following publications.

Chapter 3

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. (Also appeared in: *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 365–374, 2010.)

Chapters 4 and 5

A. Driemel and S. Har-Peled. Jaywalking your dog – computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 2013. To appear. (Also appeared in *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms*, pages 318–337, 2011.)

M. Buchin, A. Driemel, and B. Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. *Computing Research Repository (arXiv)*, abs/1307.2097, 2013. (Also appeared in *Abstracts of the 29th European Workshop on Computational Geometry*, pages 43–46, 2013.)

Chapter 6

A. Driemel, H. Haverkort, M. Löffler, and R. Silveira. Flow computations on imprecise terrains. *Journal of Computational Geometry*, 4(1):38–78, 2012. (Also appeared in *Proceedings of the 12th Algorithms and Data Structures Symposium - WADS (formerly Workshop on Algorithms and Data Structures)*, pages 350–361, 2011.)

Chapter 7

A. Driemel, S. Har-Peled, and B. Raichel. On the expected complexity of Voronoi diagrams on terrains. In *Proceedings of the 28th ACM Symposium on Computational Geometry*, pages 101–110, 2012.

The following are additional publications of the author.

M. Buchin, A. Driemel, M. van Kreveld, and V. Sacristan. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3(1):33–63, 2011. (Also appeared in *Proceedings of the 18th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems*, pages 202–211, 2010.)

B. Aronov, A. Driemel, M. van Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories on non-monotone criteria. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms*, page 1897–1911, 2013.

A. F. Cook, A. Driemel, S. Har-Peled, J. Sherette, C. Wenk. Computing the Fréchet distance between folded polygons. In *Proceedings of the 12th Algorithms and Data Structures Symposium - WADS (formerly Workshop on Algorithms and Data Structures)*, pages 267-278, 2011.

D. Chen, A. Driemel, L. Guibas, A. Nguyen, and C. Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the 13th Workshop on Algorithm Engineering & Experiments*, pages 75-83, 2011.

Samenvatting

Dit proefschrift gaat over de werkelijkheidsgetrouwe analyse van verschillende algoritmische problemen die betrekking hebben op geografische gegevens. Hieronder worden eerst de algemene methodiek en de aard van de gegevens beschreven. Daarna volgt een samenvatting van de specifieke vraagstellingen en resultaten in dit proefschrift.

Geografische gegevens. Geografische gegevens worden op verschillende manieren met computers verwerkt. Twee van de meest voorkomende soorten geografische gegevens zijn enerzijds gegevens van verplaatsingen en anderzijds beschrijvingen van terreinen. Een verplaatsing wordt beschreven door middel van een reeks plaatsaanduidingen met tijdstempels waaruit blijkt welke weg een bewegend object aflegt. Voor een terrein worden van een stuk van het aardoppervlak hoogtemetingen verzameld. Uit de hoogtemetingen wordt in de computer een meetkundig oppervlak geconstrueerd, dat het opgemeten stuk aardoppervlak voorstelt. Bij een getrianguleerd terrein worden naburige meetpunten met elkaar verbonden zodat een driehoeksraster ontstaat. De driehoekige facetten van het raster vormen dan een oppervlak. Dit noemt men een digitaal terreinmodel.

Werkelijkheidsgetrouwe analyse. Voor het verwerken van de gegevens zijn algoritmen nodig die bepalen hoe de verwerking verloopt. De theorie onderscheidt problemen die moeilijk efficiënt kunnen worden opgelost, en problemen die met behulp van algoritmen efficiënt oplosbaar zijn. Het onderscheid wordt normaliter gemaakt op grond van het moeilijkste geval. Dat wil zeggen dat een probleem als efficiënt oplosbaar geldt, als dit ook geldt voor de moeilijkst denkbare invoer van te verwerken gegevens. Hierbij wordt de efficiëntie van een algoritme bepaald door de benodigde looptijd en de benodigde hoeveelheid werkgeheugen. Interessant zijn daarbij niet de absolute hoeveelheden tijd en geheugen die nodig zijn, maar hoe deze hoeveelheden zich asymptotisch verhouden tot de hoeveelheid te verwerken gegevens, ofwel het aantal meetpunten die het terrein (of de verplaatsingen) representeren.

Bij ruimtelijke gegevens is het geval dat in theorie het moeilijkst is, vaak een gekunstelde configuratie die in de praktijk nooit zal voorkomen. Daardoor boet de theoretische analyse hier aan zeggingskracht in. Om dat tegen te gaan, zijn er diverse methoden ontwikkeld die een werkelijkheidsgetrouwe analyse mogelijk maken en toch wiskundig onderbouwd zijn. Wijdverbreid is de analyse naar verwachting, die op

toepassing van de waarschijnlijkheidsleer berust. Hier wordt een probleem efficiënt oplosbaar bevonden als de verwachte benodigde looptijd kort is.

Verder kan de ruimtelijke verdeling en vorm van de invoer worden beschreven aan de hand van invoermodellen, die in de looptijdanalyse mee worden genomen. Dit maakt een specifieke analyse mogelijk, die de moeilijkheid van de specifieke invoer in aanmerking neemt en tegelijk ook voor minder moeilijke gevallen zeggingskracht heeft. In dit proefschrift wordt een nieuw invoermodel voor verplaatsingen voorgesteld, dat vooral nuttig blijkt bij de berekening van de mate van gelijkenis tussen verplaatsingen.

Gelijkenis berekenen. Hoe men efficiënt kan berekenen hoe sterk verplaatsingen op elkaar lijken, is een vraagstuk waarvan het belang tot ver buiten de geografische informatiekunde strekt. Voor de berekening moet het begrip gelijkenis eerst exact worden gedefinieerd. Zo'n definitie wordt geboden door de Fréchet-afstand. Stel, een man loopt met zijn hond aan de lijn. De man volgt de weg die door één reeks verplaatsingen wordt gegeven; de hond volgt de weg die door een tweede reeks verplaatsingen wordt gegeven. Beiden kunnen hun snelheid zelf bepalen, maar nooit teruglopen langs de reeds afgelegde weg. De lengte van de kortste lijn die het voltooien van zo'n wandeling met hangen en wurgen mogelijk maakt, is de Fréchet-afstand tussen de beide reeksen verplaatsingen. Deze lengte is omgekeerd evenredig met de mate van gelijkenis tussen beide reeksen verplaatsingen.

Het eerste deel van dit proefschrift gaat over het efficiënt berekenen van de Fréchet-afstand. Dit vraagstuk wordt al meer dan twintig jaar onderzocht. Tot nu toe hebben de beste algoritmen een looptijd die ongeveer kwadratisch toeneemt met de grootte van de invoer, en als zodanig zijn ze weinig efficiënt. Wij beschrijven een nieuw algoritme dat op basis van een nieuw meetkundig invoermodel in weinig meer dan lineaire looptijd een benadering van de Fréchet-afstand uitrekent. Het invoermodel beperkt de lengte van de afgelegde weg in verhouding tot de doorsnede van het bezochte gebied.

Een nadeel van de Fréchet-afstand is zijn gevoeligheid voor foutieve gegevens. Een enkele fout in de plaatsbepalingen kan de berekende afstand sterk vertekenen, ook als de overige plaatsbepalingen juist zijn en de afgelegde wegen verder sterk op elkaar lijken. Wij stellen daarom een nieuwe variant van de definitie van de Fréchet-afstand voor. In deze variant mag de man of de hond tijdens de wandeling stukken afsnijden, in de hoop dat de foutieve plaatsbepalingen zo vanzelf worden overgeslagen en daardoor niet in de berekende afstand worden meegenomen.

Ons onderzoek bracht aan de ene kant aan het licht, dat het in het algemeen moeilijk is om deze variant van de afstandsmaat efficiënt te berekenen. Aan de andere kant beschrijven we een nieuw algoritme dat, onder bepaalde aannamen, de afstand in weinig meer dan lineaire tijd bij benadering uitrekent. Daarvoor maken we opnieuw gebruik van de nieuwe methoden die in het eerste deel van dit proefschrift werden voorgesteld voor de Fréchet-afstand zoals oorspronkelijk gedefinieerd.

De moeilijkheid van het probleem wordt aangetoond met een zogenaamde reductie. We laten zien hoe een verondersteld algoritme dat deze nieuwe variant van de Fréchet-afstand exact uitrekent, zou kunnen worden gebruikt om een tweede probleem op te lossen. Dat tweede probleem is te bepalen of een gegeven getal kan worden uitgedrukt als de som van een aantal andere getallen die uit een gegeven verzameling moeten worden gekozen. Daar dit probleem als moeilijk erkend wordt, volgt dat dit ook geldt voor de berekening van de afstandsmaat.

Nabootsing van waterstromen. Digitale terreinmodellen worden gebruikt om na te bootsen hoe regenwater over het aardoppervlak afstroomt en zich verzamelt. Dit kan helpen om bij sterke regenval de stijging van het waterpeil te voorspellen en eventueel de bevolking te waarschuwen. Het uitgangspunt van de simulatie is de aanname dat water, onder invloed van de zwaartekracht, altijd naar beneden stroomt in de richting van de steilste daling. De hoogtemetingen van het terreinoppervlak zijn in het algemeen echter niet precies genoeg om de steilste richting nauwkeurig te kunnen bepalen. Dat is een ernstig probleem, want de berekende hoeveelheid water die naar een bepaald punt in het dal stroomt kan zeer sterk wisselen afhankelijk van de stroomrichtingen die voor het water in de bergen worden berekend.

Wij stellen een nieuw model voor dat de stroming van water op een stabiele manier nabootst, ook wanneer de hoogtemetingen onnauwkeurig zijn. We laten zien dat het moeilijk is om deze simulatie over getrianguleerde terreinen efficiënt te berekenen als de waterstromen dwars over de driehoekige facetten mogen lopen. Als we de stroomrichtingen echter beperken tot het volgen van de randen van de driehoeken, dan kunnen we de minimale en maximale stijging van het waterpeil efficiënt berekenen.

De moeilijkheid van het probleem wordt ook hier aangetoond met een reductie. Stel dat we een algoritme zouden hebben dat kan bepalen of water tussen twee gegeven punten op een getrianguleerd terreinoppervlak zou kunnen stromen. Wij tonen aan dat zo'n algoritme dan zou kunnen worden gebruikt om een tweede probleem op te lossen. Het tweede probleem is deze keer te bepalen of een logische formule van een bepaalde vorm oplosbaar is. Ook voor dit probleem geldt, dat de moeilijkheid ervan erkend wordt, en daaruit volgt dat het ook moeilijk is om te bepalen welke waterstromen mogelijk zijn.

Voronoi-diagrammen. De afstand tussen twee punten op het oppervlak van een getrianguleerd terrein is goed gedefinieerd als de lengte van de kortste weg, die ze over het oppervlak met elkaar verbindt. Het Voronoi-diagram is een opdeling van het oppervlak in gebieden die, met betrekking tot een vaste verzameling van standplaatsen, voor elk punt op het oppervlak aangeven welke standplaats het dichtst bij is. Om vragen naar de dichtstbijzijnde standplaats voor elk punt efficiënt te kunnen beantwoorden, wordt het Voronoi-diagram in een computer opgeslagen. De efficiëntie hangt onder andere af van de hoeveelheid opslagruimte die het diagram inneemt. Volgens de klassieke analyse neemt de opslagruimte kwadratisch toe met de grootte van het terreinmodel. Het kwadratische geval komt in de praktijk echter nooit voor.

In het laatste deel van dit proefschrift analyseren we de grootte van het Voronoi-diagram onder de aanname dat de standplaatsen gelijkmatig over het oppervlak verdeeld zijn. Onder deze aanname bevestigt ons resultaat een vermoeden van Aronov *et al.* uit 2008: de grootte van het diagram is additief lineair in de grootte van het terrein en het aantal standplaatsen, als de vorm van het getrianguleerde terrein werkelijkheidsgetrouw is, en bovendien aannamen worden gedaan over de ruimtelijke spreiding van de standplaatsen.

Zusammenfassung

In dieser Arbeit geht es um die wirklichkeitsgetreue Analyse von verschiedenen algorithmischen Problemen, welche sich mit geografischen Daten befassen. Im folgenden wird zunächst die allgemeine Methodik und die Art der Daten beschrieben. Danach werden die spezifischen Fragestellungen und Ergebnisse der Arbeit zusammengefasst.

Geografische Daten. Geografische Daten werden auf verschiedene Arten im Computer verarbeitet. Zwei der häufigsten Formen, die sie dabei annehmen, sind einerseits Trajektorien und andererseits Terrains. Eine Trajektorie ist eine Folge von Positionsangaben mit Zeitstempel und protokolliert den Pfad eines sich bewegenden Objekts. Bei einem Terrain wird eine Menge von Höhenmessungen auf einem Bereich der Erdoberfläche vorgenommen. Aus den Höhenangaben wird im Computer eine geometrische Oberfläche erzeugt, welche die ausgemessene geografische Fläche repräsentiert. Bei einem triangulierten Terrain werden benachbarte Messpunkte zu einem Dreiecksnetz verbunden. Die Dreiecksflächen bilden dann zusammen die Oberfläche. Man spricht von einem digitalen Terrainmodell.

Wirklichkeitsgetreue Analyse. Für die Verarbeitung der Daten werden Algorithmen benötigt, die den Ablauf bestimmen. Die Theorie unterscheidet zwischen schwer effizient lösbaren Problemen, und solchen die mithilfe eines Algorithmus effizient lösbar sind. Die Bewertung orientiert sich standardmäßig am schwersten Fall. Das heißt, ein Problem ist effizient lösbar, wenn dies auch für die schwerst-mögliche Eingabe gilt. Hierbei wird die Effizienz eines Algorithmus anhand der Länge der Laufzeit und des Speicherplatzverbrauchs bewertet. Interessant sind dabei nicht die absoluten Größen, sondern wie sich diese asymptotisch mit Bezug auf die Größe der Eingabe, also die Anzahl der Messpunkte welche die Trajektorie oder das Terrain definieren, verhalten.

Bei räumlichen Daten ist der theoretisch schwerste Fall oft eine konstruierte Konfiguration, die in der Praxis nie vorkommen wird. Dadurch büßt die theoretische Analyse hier an Aussagekraft ein. Um dem entgegenzuwirken, sind eine Reihe von Methoden entwickelt worden, die eine wirklichkeitsgetreue Analyse ermöglichen und trotzdem mathematisch fundiert sind. Weit verbreitet ist die Analyse im erwarteten Fall, welche sich der Wahrscheinlichkeitstheorie bedient. Hier wird ein Problem für effizient lösbar befunden, wenn dies im Erwartungswert gilt. Weiter kann die geometrische Verteilung und Form der Eingabe mithilfe von Eingabemodellen beschrieben

werden, die in die Laufzeitanalyse miteingehen. Dies erlaubt eine spezifische Analyse, welche die Schwerheit der Eingabe-Instanz miteinbezieht und somit auch in weniger schweren Fällen aussagekräftig ist.

In dieser Arbeit wird ein neues Eingabemodell für Trajektorien vorgestellt, welches sich speziell für die Berechnung der Ähnlichkeit von Trajektorien als hilfreich erweist.

Ähnlichkeitsberechnung. Wie die Ähnlichkeit von Trajektorien effizient berechnet werden kann ist eine wichtige Fragestellung, die weit über den geografischen Bereich hinaus relevant ist. Für die Berechnung muss die Ähnlichkeit zunächst exakt definiert werden. Eine solche Definition bietet der Fréchet-Abstand. Stellen wir uns vor, ein Mann läuft mit seinem Hund an der Leine, beide jeweils auf einem der beiden Pfade, die durch die Trajektorien gegeben sind. Der Mann und der Hund können ihre Geschwindigkeit selbst bestimmen, aber nicht rückwärts auf dem Pfad laufen. Die Länge einer Leine, welche die Ausführung eines solchen Spaziergangs mit Müh und Not erlaubt, stellt den Fréchet-Abstand der beiden Trajektorien dar. Diese Länge ist umgekehrt proportional zu der Ähnlichkeit der Trajektorien.

Im ersten Teil der Arbeit geht es um die effiziente Berechnung des Fréchet-Abstands. Diese Fragestellung wird seit mehr als 20 Jahren untersucht. Bis heute haben die besten Algorithmen eine Laufzeit, die ungefähr quadratisch mit der Eingabe wächst, und sind somit mäßig effizient.

Wir beschreiben einen neuen Algorithmus, welcher den Fréchet-Abstand unter einem neuen geometrischen Eingabemodell mit fast linearer Laufzeit approximiert. Das Eingabemodell beschränkt die Länge eines Pfades, der von der Trajektorie beschritten wird, relativ zu dem Durchmesser des besuchten Bereiches.

Ein Nachteil des Fréchet-Abstands ist seine Anfälligkeit gegenüber fehlerhaften Daten. Ein einzelner Fehler in der Lokalisierung der Trajektorie kann das Maß verfälschen, auch wenn die restlichen Positionsdaten korrekt, und die Trajektorien sonst sehr ähnlich sind. Wir stellen eine neue Variante vor, in welcher der Mann oder der Hund bei dem Spaziergang Abkürzungen nehmen dürfen, in der Hoffnung dass die fehlerhaften Werte automatisch übersprungen werden und somit nicht in das Maß miteingehen.

Unsere Untersuchung zeigt einerseits, dass diese Variante des Abstandsmaßes im Allgemeinen schwer effizient zu berechnen ist. Andererseits beschreiben wir einen neuen Algorithmus, um den Abstand unter bestimmten Annahmen in fast linearer Laufzeit zu approximieren. Dabei greifen wir auf neue Methoden zurück, die im ersten Teil der Arbeit für den originalen Fréchet-Abstand eingeführt wurden.

Die Schwerheit des Problems wird mittels einer sogenannten Reduktion gezeigt. Wir zeigen, wie ein hypothetischer Algorithmus, der diese neue Variante des Fréchet-Abstands exakt berechnet, für die Lösung eines zweiten Problems angewandt werden kann. Das zweite Problem ist, zu entscheiden, ob für einen Eingabewert eine Zerlegung in eine Summe von Werten existiert, welche aus einer Menge von Eingabewerten gewählt werden müssen. Da dieses Problem als schwer anerkannt ist, folgt dies auch für die Berechnung des Abstandswerts.

Flusssimulation. Digitale Terrainmodelle werden benutzt, um zu simulieren, wie Regenwasser auf der Oberfläche abfließt und sich sammelt. Dies kann dabei helfen, bei starken Regenfällen den Anstieg des Wasserpegels vorherzusagen und gegebenenfalls die Bevölkerung zu alarmieren. Die Grundlage dieser Simulationen ist die Annahme, dass das Wasser durch den Einfluss der Erdanziehungskraft in die steilsten Richtung fließt. Allerdings sind die gemessenen Höhenwerte der Terrainoberfläche im Allgemeinen nicht genau genug, um diese Richtung akkurat berechnen zu können. Dies ist ein schwerwiegendes Problem, denn die berechnete Wassermenge, die sich an einem bestimmten Punkt im Tal sammelt, kann sehr stark schwanken, je nachdem welche Flussrichtungen weiter oben am Berg berechnet wurden.

Wir stellen ein neues Modell vor, welches den Wasserfluss robust simuliert, auch wenn die Höhenangaben ungenau sind. Wir zeigen, dass die Simulation auf triangulierten Oberflächen schwer effizient zu berechnen ist, wenn die Flussrichtungen das Innere der Dreiecksflächen kreuzen dürfen. Wenn die Flussrichtungen allerdings auf die Ränder der Dreiecke beschränkt sind, können wir den Anstieg des Wasserpegels effizient abschätzen.

Hier wird die Schwerheit des Problems wieder durch eine Reduktion gezeigt. Angenommen wir hätten einen Algorithmus, der bestimmt, ob es einen Flusspfad zwischen zwei gegebenen Oberflächenpunkten geben kann. Wir zeigen dass wir mithilfe dieses Algorithmus entscheiden könnten, ob eine logische Formel einer bestimmten Form lösbar ist. Auch für dieses Problem gilt, dass die Schwerheit anerkannt ist, und daher folgt die Schwerheit für die Bestimmung des Flusspfads.

Voronoi-Diagramme. Auf der Oberfläche eines triangulierten Terrains ist der Abstand zwischen zwei Punkten durch die Länge des kürzesten Weges, der sie entlang der Oberfläche verbindet, wohldefiniert. Das Voronoi-Diagramm ist eine Unterteilung der Oberfläche in Bereiche, die, in Bezug auf eine feste Menge von Standorten, für jeden Punkt auf der Oberfläche angeben, welchem Standort er am nächsten ist. Um Anfragen nach dem nächstgelegenen Standort effizient zu verarbeiten, wird das Voronoi-Diagramm im Computer gespeichert. Die Effizienz hängt unter anderem davon ab, wieviel Speicherplatz das Diagramm einnimmt. Der klassischen Analyse zufolge wächst der Speicherplatz quadratisch in der Größe des Terrainmodells. Der quadratische Fall kommt allerdings in der Praxis nie vor.

Im letzten Teil der Arbeit analysieren wir die Größe des Voronoi-Diagramms unter der Annahme, dass die Standpunkte gleichverteilt auf der Oberfläche sind. Unser Ergebnis bestätigt eine Vermutung von Aronov *et al.* von 2008: die Komplexität des Diagramms ist additiv linear in der Größe des Terrains und der Anzahl der Standpunkte, wenn die Form des triangulierten Terrains wirklichkeitsgetreu ist und zusätzlich Annahmen über die Verteilung der Standpunkte gemacht werden.

Summary

This thesis is about the realistic analysis of different algorithmic problems that deal with geographical data. In the following we first describe the general methodology and the type of data. Then, the specific questions and results of this thesis are summarized.

Geographical data. Geographical data is processed in different ways in the computer. The two most common types of geographical data being processed are on the one hand trajectories and on the other hand terrains. A trajectory is a series of time-stamped positions which records the path of a moving object. For a terrain, a number of height measurements are taken within an area of the surface of the earth. From the height values, the computer generates a geometric surface which represents the geographic region that was measured. For a triangulated terrain, neighboring points are connected to a net of triangles. The faces of the triangles from the surface. The result is referred to as a digital terrain model.

Realistic analysis. To process the data we need algorithms that determine the procedure. Computer science theory distinguishes between problems that are hard to solve efficiently and problems that can be solved efficiently by using an algorithm. By default, the efficiency is evaluated based on the worst case. That is, a problem is called efficiently solvable if this is true for the worst possible input. Here, the efficiency of an algorithm is measured by its running time and space requirements. We are interested not in the absolute quantities, but their asymptotic behaviour with respect to the size of the input, which is defined by the number of measurement points that the trajectory (or the terrain) is represented by.

For spatial data, the theoretically worst case is often a contrived configuration which would never occur in practice. Therefore, the theoretical analysis sometimes fails to describe the actual behaviour of the algorithm. To remedy this, a collection of techniques has been developed, which enable a realistic analysis with mathematically provable bounds. In this sense, it is common to analyze the expected behaviour by using probability theory. Here we call a problem efficiently solvable if this is true in the expected case. Furthermore, we can constrain the distribution and shape of the input by using input models which influence the running time analysis. This allows for a specific analysis that takes the hardness of the input instance into account and

is therefore also meaningful for cases that are less hard. In this thesis we introduce a new input model for trajectories that has proven to be useful for the problem of computing the similarity of trajectories.

Computing similarity. The question how to compute the similarity of trajectories is relevant far beyond geographical applications. To begin with, this similarity has to be defined exactly in order to be able to compute it. One such definition is provided by the Fréchet distance. Imagine a man walking a dog on a leash. They are each walking on one of the paths defined by the two trajectories. They can adjust their speeds, but not walk backwards along the path. The length of a leash that only just admits such a walk corresponds to the Fréchet distance of the two trajectories. This length is inversely proportional to the similarity of the trajectories.

In the first part of the thesis we are concerned with the efficient computation of the Fréchet distance. This question has been studied for more than 20 years. The running time of the most efficient algorithms to date grows roughly quadratically with the size of the input. Thus, the best known algorithms are not particularly efficient.

We describe a new algorithm that approximates the Fréchet distance in near-linear time under a new geometric input model. The input model constrains the length of a path that is traced out by the trajectory relative to the diameter of the visited region.

A drawback of the Fréchet distance is its sensitivity to noise. A single error in the localization of the trajectory can distort the measure, even if the remaining positioning data is correct and the trajectories are otherwise very similar. We propose a new variant of the Fréchet distance in which the man or the dog can take shortcuts, in the hope that erroneous points will be omitted automatically and will therefore not influence the measure.

Our results show that this new variant of the distance measure is hard to compute efficiently. On the other hand, we describe a new algorithm to approximate the distance measure under certain assumptions in near-linear time. For this we use some of the new methods which we introduced in the first part of the thesis for the original Fréchet distance.

The hardness of the problem is shown with the help of a so-called reduction. We show that a hypothetical algorithm that computes this new variant of the Fréchet distance exactly, can be used to solve yet another problem. The second problem is to decide for a given input value whether there exists a decomposition into a sum, where the sum values have to be chosen from a given input set. Since this problem is recognized to be hard, the hardness follows also for the computation of the distance measure.

Flow simulation. Digital terrain models are used to simulate how water from precipitation runs off and collects on the surface. During heavy rainfall this can help to predict the increase of water levels and to notify the population if necessary. The basis of this simulation is the assumption that, under the influence of gravity, water flows in the steepest direction. However, the elevation data is generally not exact enough to predict this flow direction accurately. This is a serious problem since the computed amount of water arriving at a certain point in the valley can vary a lot depending on the flow directions higher up on the slope of the mountain.

We propose a new model that simulates water flow robustly even if the elevation data is imprecise. We show that the simulation on triangulated surfaces is hard to compute efficiently if flow directions may cross the inside of the triangles. However, if flow directions are confined to the edges of the triangles, then we can compute upper and lower bounds on the increase in water levels efficiently.

Here the hardness is shown again by using a reduction. Assume that we had an algorithm that decides if there can be a flow path between two points on the surface. We show how this algorithm could be used to solve a logical formula of certain form. Also here, the second problem is recognized to be hard and therefore the hardness follows for the computation of the flow path.

Voronoi diagrams. Distances between points on the surface of a triangulated terrain are well-defined by the length of a shortest path connecting the points along the surface. The Voronoi diagram is a subdivision of the surface into regions with respect to a fixed set of sites on the surface. It tells us for any point on the surface which site is closest to it. To answer queries for the closest site efficiently, the Voronoi diagram is stored in the computer. The efficiency depends among other things on the amount of space taken up by the diagram. According to the classical worst-case analysis the space grows quadratically with the size of the terrain. However, the quadratic case never occurs in practice.

In the last part of the thesis we analyze the size of the Voronoi diagram under the assumption that the sites are distributed uniformly likely on the surface. Our results confirm a conjecture by Aronov *et al.* from 2008: the complexity of the diagram is additively linear in the size of the terrain and the number of sites, if the shape of the terrain is realistic, and additional assumptions are made on the distribution of the sites.

Acknowledgements

I would like to take this opportunity to thank all my co-authors for teaching me how to read and write academically. These are Boris Aronov, Maike Buchin, Daniel Chen, Atlas F. Cook IV, Leonidas J. Guibas, Sariel Har-Peled, Herman Haverkort, Marc van Kreveld, Maarten Löffler, Andy Nguyen, Benjamin Raichel, Vera Sacristán, Jessica Sherette, Rodrigo Silveira, Bettina Speckmann, Frank Staals, and Carola Wenk.

I also want to thank all anonymous reviewers of our papers for reading them and for giving useful comments.

I thank my PhD-advisors Mark de Berg and Marc van Kreveld for giving me the opportunity to do research. In particular, I thank Marc for inviting me to his workshop in the first month of my PhD studies, October 2009. It is safe to say that this kick-started my career. I thank Sariel Har-Peled for inviting me to UIUC in the summer of 2010, for visiting me in Eindhoven in the summer of 2011, and for sharing countless ideas that became part of this thesis.

I also thank Jack Snoeyink and Carola Wenk for inviting me to short research stays at UNC and Tulane University. I really enjoyed visiting these places and meeting other graduate students in their natural habitat.

I thank Carola for many personal discussions and for being an authentic role-model. I thank the organizers of the "Women in Theory"-Workshop for a great workshop and in particular I thank Tal Rabin. Her advice on balancing work and life stayed with me for the past three years and will continue to be helpful.

Most of all, I thank Marc and Mark for advising me on every aspect of work in academia, for their kindness, their dedication and their flexibility.

Furthermore, I thank Sariel Har-Peled, Günter Rote, Bettina Speckmann, and Remco Veltkamp for agreeing to be part of the reading committee and thereby enduring these 200 pages.

I thank my sister, Imke Driemel, for proof-reading parts of this thesis and for her expert advice on shoes. I thank Imke and Maarten for agreeing to support me during the defence ceremony as paranymphs.

I thank my grandparents Alma and Eberhard Krüger, Anneliese and Rudi Driemel, as well as my parents, Sibylle and Holger Driemel, for supporting me in whatever I want to do.

Thank you, Herman, for meticulously translating my German summary into Dutch and for supporting me and our family with the same passion while I was writing this thesis. Your everlasting enthusiasm is a real blessing.

Curriculum Vitae

Anne Driemel was born on May 19, 1983, in Rüdersdorf bei Berlin, Germany. She did not get her own computer, but was allowed to use the computer of the family. After she realized that her male friends did not want to talk to her about computers, she decided to study computer science (instead of mathematics) at the Free University of Berlin. At the time she thought that computer science was about computers. The first year lecture on algorithms exposed this common misconception and in fact captivated her. During her studies she also spent time at the University of Pennsylvania in Philadelphia, USA, and the University Denis Diderot in Paris, France. Her Master's studies on curve similarity were supervised by Helmut Alt at FU Berlin and Jean Gallier at UPenn and resulted in a Master's degree from FU Berlin in 2009. Since October 2009 she has been working towards this thesis under the supervision of Marc van Kreveld and Mark de Berg at Utrecht University and at the TU Eindhoven. During this time she also gave birth to her first daughter, Emma. In October 2013 she will join the research group of Christian Sohler at the TU Dortmund, Germany.

Colophon

This thesis was typeset in LaTeX. All figures were created by Anne Driemel and authors of the respective publications listed on page 199 using the IPE extensible drawing editor except for the following:

Figure 1.1 is by the Archipelagos Institute of Marine Conservation
Source: `http://www.archipelago.gr/`

Figure 8.1 is by Jane Richardson
License: Creative Commons Attribution 3.0 Unported
Source: Wikimedia Commons

Figure 8.2 is by Felix Hebeler and Ross Purves
Source: [91]

Original picture of maple leaf used for cover art is by User:Fcb981 (English Wikipedia)
License: Creative Commons Attribution-Share Alike 3.0 Unported
Source: Wikimedia Commons

Cover designed by Anne Driemel using IPE