Fréchet Distance for Simple Polygons

Evan Cordell

April 29, 2013

Abstract

Shape matching has applications in such diverse fields such as archaeology, architecture, medical imaging, image recognition, and vehicle tracking. The Fréchet distance is one way to measure the similarity of shapes, and several algorithms have been developed to compute the Fréchet distance for certain types of curves and surfaces. The Fréchet distance between simple polygons, for example, can be computed in polynomial time.

The primary result of this thesis is an improvement of $O(n)$ to the runtime of the algorithm to compute Fréchet distance for simple polygons. We model the problem in a novel way that builds on previous work, which results in the runtime improvement and the ability to recover the map or even maps that realize the Fréchet distance. The additional information afforded by the capacity for recovering the original maps lends itself to the investigation of the Fréchet distance as it applies to morphing and other problems.

Acknowledgments

There is a shortage of university professors that care passionately both about their fields and about their students, and Carola Wenk is one of those few; it was a pleasure and privilege to work with her. I would also like to thank Lev Kaplan and Tái Há for their encouragement, support, and feedback. I am indebted to the countless others at Tulane who have helped me beyond the normal call of duty this past year and earlier, especially Elio Brancaforte, Agnieszka Nance, Dietmar Felber, and Michael Mislove.

I would also like to thank: my parents, for their continuing support in my academic endeavors; Emily Sherman, for putting up with my company despite my sleep deprivation; and Tardar Sauce the Grumpy Cat, for bringing a smile to my face in otherwise stressful times.

Contents

List of Tables

List of Figures

1

Introduction

Comparing the similarity of curves and surfaces is a difficult task that arises in numerous areas and has plenty of real-world applications. There have been many approaches to the various manifestations of this problem, and while many of them are highly specific to their applications, others find a broader appeal. Shape matching has diverse applications in fields such as archaeology, architecture, medical imaging, image recognition, and vehicle tracking.

The Fréchet distance is a measure of similarity that finds utility in its own small circle of applications. It has been used in matching time series in databases [KKS05], time warping [KP99], speech recognition [KHM+98], signature verification [MP99], handwriting recognition [SKB07], aligning protein structures [JXZ07], computer vision, map matching [AW12, CDG+11, WS06], moving object analysis [BBG08a, BBG+08b], matching coastlines over time [MDH06], and music information retrieval [SGHS08], to name a few. The Fréchet distance is usefel in any situation where not just the nearness of a set of points, but also the order of those points, is important to the application. For a tangible, everyday example, consider how a map on the GPS system in your car must decide on which roads to show your path — at first this might be seem trivial, but consider how a simple "closest road" metric would break down at intersections or on dense city grids, or even worse, on the high-flying

spaghetti junctions of larger cities. Couple that with an international restriction on consumer GPS accuracy, and the need for smarter distance measures is clear.

With the broad applicability of the Fréchet distance comes a strong interest in its computation, and much work has been done to determine the most efficient ways of computing and applying the Fréchet distance in its various manifestations.

Alt and Godau describe a method for computing the Fréchet distance between polygonal curves in $O(pq \log(pq))$ for P and Q with p and q edges, respectively [AG95]. In the same paper, they also develop an algorithm for the case of closed polygonal curves which runs in $O(pq \log^2(pq))$, and an algorithm for partial curve matching with the same runtime. A variant of the Fréchet distance known as the *discrete Fréchet distance* can be computed in $O(pq)$ time [EEMM94].

For two-dimensional surfaces, the computation of the Fréchet distance is surprisingly hard. Computing the Fréchet distance between piecewise linear surfaces, even if one surface is a triangle, is NP-hard [God98], as is the case when both surfaces are polygons with holes or terrains [BBS10]. In general, the Fréchet distance is upper-semi-computable, and it is unknown if it is computable [AB10]. However, when more restrictive cases are considered, the problems become more tractable: the Fréchet distance for simple polygons can be computed in polynomial time [BBW07], the partial Fréchet distance for simple polygons can be decided in polynomial time [SW12], and the Fréchet distance for *folded polygons* (polygons folded in specific, nice ways) can be approximated in polynomial time [CDHP$^+$11].

The primary result of this thesis is an improvement of $O(n)$ to the runtime of the Fréchet distance for simple polygons. We model the problem in a slightly different way than before [BBW07], which results in the runtime improvement and the ability to recover the map or even maps that realize the Fréchet distance. The additional information afforded by the capacity for recovering the original maps lends itself to the investigation of the Fréchet distance as it applies to morphing and other problems.

We begin by defining notation and terms that will be used in later chapters. We assume a basic familiarity with mathematics and computation in general, but not with the specifics of this topic. We then consider the Fréchet distance for curves, including an overview of the Hausdorff distance (a related metric), focusing in particular on the computation of the Fréchet distance for polygonal curves, but also giving due time to the Fréchet distance defined for piecewise smooth curves. The important distinction between a decision problem and an optimization problem are discussed, as well as several methods of converting between the two that are particularly beneficial to this specific application.

We continue to define the Fréchet distance for simple polygons, situating it within the larger context of the computability of various incarnations of the Fréchet distance for surfaces. The approach for the computation of the Fréchet distance for surfaces will be discussed at length, followed by an equally involved discussion of the reconceptualization of that approach and corresponding differences in runtime and computation. We finish by discussing some of the applications to which this new approach is particularly well suited, including our own investigation into the problem of morphing.

2

Notation and Terminology

Below is an overview of the notation and terminology that will be used in this thesis. This should serve as both a review of simple concepts, an overview of the work to come, and a reference for later use.

2.1 Geometric Fundamentals

Definition 2.1. (Vector Space) A *vector space* over a field F is a set V together with two binary operations such that the operation (addition) over elements in V (vectors) is associative and commutative, has an identity element, and has inverse elements. The operation (scalar multiplication) of elements in F (scalars) with elements in V distributes over both F and V, is compatible with the field multiplication, and also has an identity element.

Definition 2.2. (Line Segment) A *line segment* $L = \{\vec{u} + t\vec{v} \mid t \in [0, 1]\}$, where $\vec{u}, \vec{v} \in V$ and V is a vector space over $\mathbb{R}$ or $\mathbb{C}$.

Definition 2.3. (Polygonal Line) A *polygonal line*, *polygonal curve*, or *piecewise linear curve* P is a curve defined by a series of points $(A_1, A_2, \ldots, A_n)$ called *vertices* such that the curve consists of line segments connecting adjacent vertices.

Definition 2.4. (Polygon) A *polygon* is a closed polygonal line; i.e., a polygonal line such that the start and end points are the same.

Definition 2.5. (Simple) A polygon or polygonal curve is *simple* if no two segments intersect.

Definition 2.6. (Parametric Curve) A *parametric curve* or *parameterized curve* is a representation of a curve as a function of a variable called the *parameter*.

Definition 2.7. (Convex) A shape is *convex* if for every pair of points within the object, every line segment that joins them lies entirely within the object. A *convex polygon* is a polygon that is convex.

Definition 2.8. (Convex Decomposition) A *convex decomposition* is the division of an object into convex pieces by the addition of line segments between vertices.

Definition 2.9. (Shortest Path) A *shortest path* is a path (curve) between two points with the shortest total length. Shortest paths within polygonal surfaces, even those with holes, are always polygonal lines under the Euclidean norm.

Definition 2.10. (Piecewise Linear Surface) A *piecewise linear surface* is a surface in which the shortest path between any two points is a piecewise linear curve.

2.2 Metrics and Distance

Definition 2.11. (Metric) A *metric* or *distance function* is a function which defines a distance between elements of a set. A metric d over a set X is a function $X \times X \rightarrow [0, \infty)$ such that the following conditions hold:

- $d(x, y) = 0$ if and only if $x = y$
- $d(x, y) = d(y, x)$

- $d(x, z) \leq d(x, y) + d(y, z)$

Definition 2.12. (Metric Space) A set equipped with a metric is a *metric space*.

Definition 2.13. (Homeomorphism) A *homeomorphism* is a function $f : X \to Y$ between two topological spaces such that the following properties are true:

- f is a bijection
- f is continuous
- The inverse, f^{-1}, is continuous.

Two spaces are *homeomorphic* if there exists a homeomorphism between them.

Definition 2.14. (Parameter Space) The set of all possible combinations of values of parameters.

Definition 2.15. (Hausdorff Distance) The *Hausdorff distance* d_H is a function such that

$$d_H(A, B) = \max(\vec{d}_H(A, B), \vec{d}_H(B, A))$$

where,

$$\vec{d}_H(A, B) = \max_{a \in A} \min_{b \in B} d(a, b)$$

and A and B are subsets of a metric space, and d is the associated metric. We will be working in Euclidean space where $d(a, b) = \|a - b\|$.

Definition 2.16. (Fréchet Distance for Curves) Let V be an arbitrary Euclidean vector space, and let $f : [a, a'] \to V$ and $g : [b, b'] \to V$ be parametric curves. The Fréchet distance δ_F for curves is defined as

$$\delta_F(f, g) = \inf_{\substack{\alpha[0,1] \to [a,a'] \\ \beta[0,1] \to [b,b']}} \max_{t \in [0,1]} \|f(\alpha(t)) - g(\beta(t))\|$$

where α and β denote continuous, monotonically increasing reparameterizations of f and g, respectively.

Definition 2.17. (Fréchet Distance for Surfaces) Let $f : A \to \mathbb{R}^d$ and $g : B \to \mathbb{R}^d$ for some $A, B \subseteq \mathbb{R}^k$ where $1 \leq k \leq d$ be two surfaces with homeomorphic parameter spaces A and B. The Fréchet distance between those surfaces is defined as.

$$\delta_F(f, g) \;=\; \inf_{\sigma\colon A \to B} \sup_{x \in A} \|f(x) - g(\sigma(x))\|$$

Definition 2.18. (Free Space) Given curves P and Q, the *free space* F_ε is a subset of the parameter space of P and Q such that

$$F_\varepsilon = \{(s, t) \in [0, 1]^2 \mid d(P(s), Q(t)) \leq \varepsilon\}$$

Definition 2.19. (Ambient Isotopy) An *ambient isotopy* is a continuous distortion of a manifold that takes a submanifold to another submanifold. Let N and M be manifolds and g and h be embeddings of N in M. A continuous map

$$F : M \times [0, 1] \to M$$

is defined to be the ambient isotopy taking g to h if F_0 is the identity map, each map F_t is a homeomorphism from M to itself, and $F_1 \circ g = h$.

Definition 2.20. (Isotopic Fréchet Distance) Let M be a metric space with $A, B \subset M$ and with A, B, and X homeomorphic. The *isotopic Fréchet distance* between them is

$$\mathcal{I}(A, B) = \inf_{h} \max_{x \in X} \operatorname{len} h(x, \cdot)$$

where $h : M \times I \to M$, $h(\cdot, t)$ homeomorphism, $h(x, 0) = x \;\forall\; x \in X$, and $h(A, 1) = B$

3

Fréchet Distance for Curves

There are innumerable ways to define distance outside the standard notion from Euclidean geometry, and many such metrics have useful applications in the real world. In a dense city grid, for example, the shortest path between two points follows the zig-zagging path of the streets rather than simply a straight line (this is known as the Manhattan distance). Another metric, sometimes called the Post Office metric or the British Rail metric, measures distance by enforcing that any path from one point to another must pass through some other predefined point (the "post office"). Everything that follows will be defined for the Euclidean norm and Euclidean metric, but many of the theorems and definitions below are equally applicable in any metric space or vector space.

There are several ways that one could generalize the notion of distance between points to distance between sets or curves. The Hausdorff distance is one of the most straightforward generalizations, so we begin there. The Hausdorff distance is used in numerous applications, but its utility is limited when considering curves as continuous collections of points rather than simply sets of points. We then move to the Fréchet distance, which is far better suited to measuring the distance between continuous objects such as curves or surfaces. The Fréchet distance for curves will provide a useful starting point for a later discussion about distances between surfaces, and

simple polygons in particular.

3.1 Hausdorff Distance

Let M be a metric space with an associated metric d, so that $d(x, y)$ is the distance between two points in M. Suppose that we wish to define the distance between a point and a set. The intuitive definition would be

$$d(x, Y) = \inf\{d(x, y) \mid y \in Y\}$$

where x is a point and Y is a set. If Y is continuous, we can think of this as being the normal distance from Y to x. Suppose then that we wish to define a distance between two sets. A further straightforward extension would be

$$d(X, Y) = \sup\{d(x, Y) \mid x \in X\}$$

where X and Y are both sets.

This is exactly the definition of the (directed) Hausdorff distance, and we can rewrite it as

$$\vec{d}_H(X, Y) = \sup \inf\{d(x, y) \mid x \in X \text{and } y \in Y\}$$

It is important to note that $\vec{d}_H(X, Y) \neq \vec{d}_H(Y, X)$ in general and that $\vec{d}_H(X, Y) = 0$ does not always imply that $X = Y$, therefore $\vec{d}_H$ for sets is not a metric. For this reason we define the undirected Hausdorff distance as below:

$$d_H(X, Y) = \max\{\vec{d}_H(X, Y), \vec{d}_H(Y, X)\}$$

which is in fact a metric on the set of compact subsets of M.

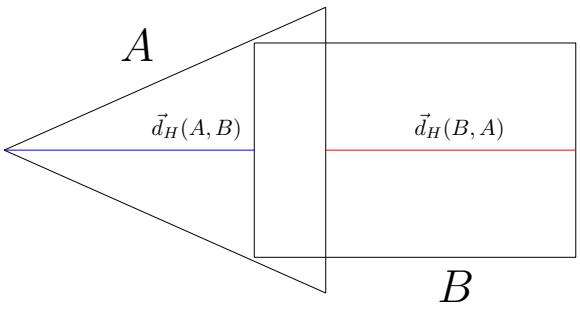

Figure 3.1: An example showing $\vec{d}_H(B,A)$ and $\vec{d}_H(B,A)$

We now have a basic way to talk about the distance between sets of points. Figure 3.1 shows the two directed distances between A and B, considered as sets of points. The undirected distance is, then, the larger of the two.

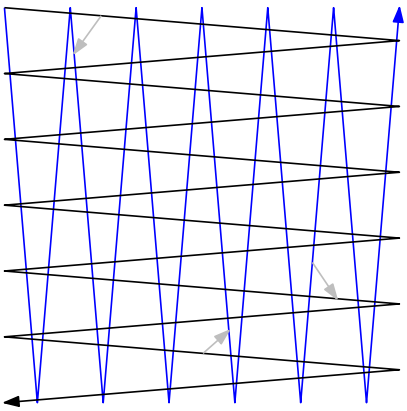

Figure 3.2: An example where the Hausdorff distance fails

Before we get too excited, however, let us consider a case where the Hausdorff distance gives undesirable results. Figure 3.2 shows two polygonal curves. Note that they have a very small Hausdorff distance, because for every point in A there is a point in B that is very close. If we think about them as curves, however, they seem about as dissimilar as could be! The Hausdorff distance is clearly inadequate for use in comparing the similarity of curves.

3.1.1 Computability

We briefly mention the computability of the Hausdorff distance for completeness, but we will not explore the details of different algorithms or implementations.

The Hausdorff distance $d_H(A, B)$ in two dimensions can be computed efficiently if A and B are sets of line segments or, in a higher fixed dimension d, if A and B are sets of k-dimensional simplices for $k < d$ [AB10]. The Hausdorff distance between two polygons can be determined in time $O(n \log n)$ [ABB95]. If P and Q are sets of n and m points respectively, then the Hausdorff distance can be computed in $O((m + n) \log(m + n))$ by first constructing the Voronoi diagram and then using a sweep algorithm [ABG+03]. Generalizing further, given two sets $P, Q \subseteq \mathbb{R}^d$ of n and m k-dimensional simplices, respectively, then $\vec{d}_H(P, Q)$ can be computed in $O(nm^{k+2})$ time [ABG+03]. Faster algorithms for specific special cases of the general problem have been developed, see [ABB95, AB10, AG99, ABG+03].

3.2 Fréchet Distance

We saw in Figure 3.2 why the Hausdorff distance does not work well for curves and surfaces. It is then necessary to define a new metric that accounts for the continuous, connected nature of curves.

If we have two curves f and g, then they have a defined start and endpoint. We might consider taking parameterizations of f and g, and comparing the distance between the two curves at each point along those parameterizations. In this sense, we are measuring a notion of distance that takes the continuity into account, and this is perhaps the most straightforward way to do so. There are two problems with this, though, which is that there are an infinite number of parameterizations (which one do we choose?) of f and g, and that this approach gives us a set of distances rather than a single value, so it has no utility as a distance metric.

There are different ways to solve these two problems, and they result in different variants of the Fréchet distance. The standard Fréchet distance (Definition 2.16) considers the infimum of the maximum point-to-point distance for every monotonically increasing parameterization of f and g. If we allow the parameterizations to backtrack (remove the monotonicity requirement), we have what is known as that weak Fréchet distance. If we want to consider the average point-to-point distance instead of the maximum, then we have the integral Fréchet distance. We will focus on the standard Fréchet distance.

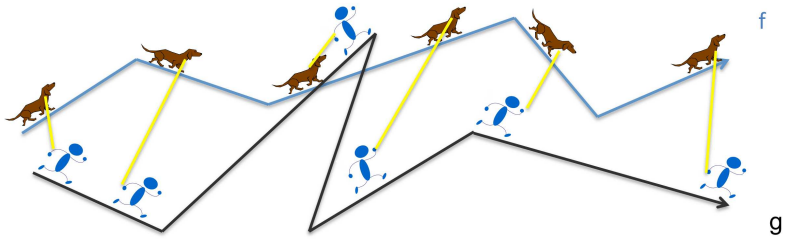

Figure 3.3: The Fréchet distance can be viewed as the shortest leash length that allows a man on f to walk his dog on g.

There is a stock illustration for the Fréchet distance, as shown in Figure 3.3. Imagine that you have two curves, f and g, and there is a man walking continuously on f and his dog is walking continuously on g. They are both only allowed to go forward or stop. The Fréchet distance is then defined as the shortest leash length between them that will still allow them to make their walk.

Although in general $d_H(f, g) \leq \delta_F(f, g)$, it is worth noting that if f and g are *convex closed* curves, then $d_H(f, g) = \delta_F(f, g)$ [ABW90].

Most curves and surfaces can be well approximated by polygonal curves and polygons, therefore the majority of this thesis will concern itself with those cases. It is worth noting, however, that work has been done for the Fréchet distance between piecewise smooth curves rather than simply piecewise linear curves. Günter Rote showed in 2004 that the Fréchet distance between piecewise smooth curves (whose segments are algebraic curves bounded by a constant degree) can be computed in

$O(pq \log pq)$ for curves with p and q segments.

3.3 Decision vs. Optimization

At first glance it may seem difficult to compute the Fréchet distance between two curves; after all, a naïve approach might involve inspecting every leash length in every possible parameterization of f and g — an infinite number of possibilities, which would be intractable even by sampling.

This seemingly insurmountable obstacle can be hurdled simply by changing the our perspective. Instead of asking "What is the Fréchet distance between f and g?" we ask "Is the Fréchet distance between f and g less than ε?"

In other words, we turn the original question into a *decision problem* which can then be *optimized*. We must first decide if $\delta_F < \varepsilon$, and then either increase or decrease our estimation of ε. This is a common technique for developing algorithms for otherwise difficult problems.

We will develop an algorithm for the decision problem later. This section focuses on methods of computing the actual value of ε, assuming an algorithm for the decision problem exists (we assume it can be determined in $O(pq)$ time). As one might imagine, there are several ways of optimizing the value of ε, and the choice of which to implement is often a trade-off between simplicity and speed.

3.3.1 Binary Search

The most straightforward method to optimize ε is one familiar to anyone who has ever played a simple number guessing game ("I'm thinking of a number between 0 and 100"). A smart strategy here is to start at the midpoint: 50. If the other player says that is too high, subtract half of the length of the interval (25); if it is too low, add half of the length of the interval. Thus if the number was 65, then the guesses

would be as follows: 50, 75, 63, 69, 66, 65.

In the worst case, this strategy will find the solution in $O(\log n)$ time. This strategy is commonly called a binary search, since each guess leads to two possible better estimations, either greater than or less than the guess. Although the idea is relatively simple, because of overflows and rounding, a correct implementation can be tricky. [Knu98].

The runtime $O(\log n)$ assumes that we can decide if the number is less than, equal to, or greater than the guess in constant time. If instead it takes some amount of time to determine the value of the comparison, the runtime is much higher. For our decision problem of $O(pq)$, finding the optimum value with a binary search would take $O(pq \log n)$. The problem is that, unlike the guessing example, we don't know what n is to begin with.

The range of ε is bounded below by 0, because it comes from a distance metric. We can find an upper bound easily using geometric properties of the Fréchet distance (discussed later) or by repeatedly doubling the guess for the upper bound. Once we have a range, we can begin the binary search to find ε, however, unlike the integers we were working with before, we have a continuous space that can be infinitely divided. Therefore we must pick some precision ρ ahead of time, and say that the solution has been reached if the two previous guesses are within ρ of each other.

If our range is determined to be $[0, \mu]$, then the number of possible solutions for the binary search, in other words n, is $\frac{\mu}{\rho}$. Since ρ must be very small with respect to μ in order to ensure a good solution, we can conclude that n will be quite large for this application. In general for a binary search with a desired precision of b bits runs in $O(\log b)$.

3.3.2 Parametric Search

Parametric search is an approach to optimization that was developed by Meggido in the early 1980s [Meg83] and has plenty of applications in computational geometry [AST92]. It assumes that the decision problem is monotonous (i.e. if $\delta_F(f, g) < \varepsilon$, then for all $\varepsilon' < \varepsilon$, $\delta_F(f, g) < \varepsilon'$), and requires the design of a parallel algorithm. For many applications, it suffices to use a parallel sorting algorithm, of which there are several, but few of which are easy to implement. Cole later improved the runtime of the original technique, but his optimization is complex and difficult to implement [Col87]. In 2002, Oostrum and Veltkamp [OV02] showed that for several applications, including computation of the Fréchet distance, it suffices to use a simple QuickSort as the search algorithm, and that an efficient algorithm can be obtained even without Cole's optimization. Parametric search rarely sees practical use because of how difficult it is to implement, and it therefore usually remains a theoretical tool.

Although an in-depth treatment of parametric searching is outside the scope of this thesis, it should be noted that it is particularly suited to computing the Fréchet distance. When applied to the Fréchet distance problem, Oostrum and Veltkamp showed that their QuickSort-based, randomized algorithm without Cole's trick still resulted in an expected runtime of $O(pq \log pq)$ [OV02]. We will take it for granted in later sections that the Fréchet distance between two polygonal curves with p and q segments can be computed in $O(pq \log pq)$.

3.4 Computing the Fréchet Distance for Polygonal Curves

Alt and Godau developed an algorithm that computes the Fréchet distance between polygonal curves in $O(pq \log pq)$ for curves with p and q nodes, respectively [AG95].

We discuss their approach here, as many of the concepts they develop will be used or extended later. Their algorithm employs parametric search as described above; the meat of their work is an efficient algorithm for the decision problem.

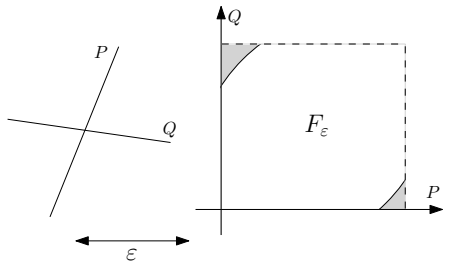

Figure 3.4: F_ε for two line segments P and Q, with ε.

The key idea is their use of the *free space* (see Definition 2.18). Consider the case where $p = q = 1$, in other words, where P and Q are line segments. Then the free space F_ε is the intersection of the unit square with an ellipse. Figure 3.4 shows this case. Note that this defines at most four free intervals, one per side.

We now wish to extend the notion of free space for arbitrary polygonal curves.

Definition 3.1. Given curves P and Q, the *free space* F_ε is a subset of the parameter space of P and Q such that

$$F_\varepsilon = \{(s,t) \in [0,p] \times [0,q] \mid d(P(s), Q(t)) \leq \varepsilon\}$$

This clearly corresponds to the free space for all possible pairs of line segments in polygonal curves P and Q, see Figure 3.5. We define a *free space cell* C_{ij} by $C_{ij} = [i-1, i] \times [j-1, j]$. For each cell, we can see that the free space corresponds to the first definition, and is therefore the intersection of an ellipse with a square. The approach for the free space will be to calculate the intersection of pq ellipses with pq squares. Note that the intersection of a single cell with an ellipse can be computed in $O(1)$ time.

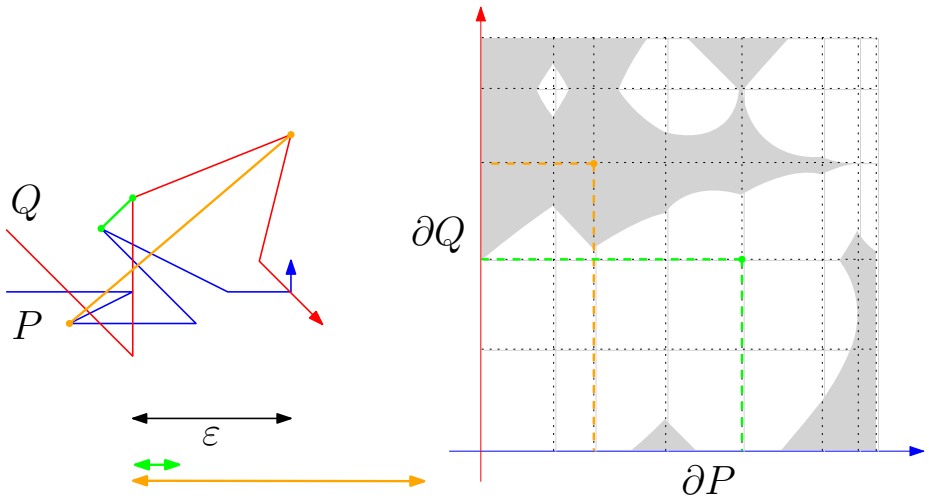

Figure 3.5: The free space for two polygonal curves P and Q showing the relationship between F_ε and the original curves.

Why is the free space useful? Any given point in the free space corresponds to a point on P and a point on Q such that the distance between those points is less than ε (Figure 3.5). Note that a path from $[0,0]$ to $[p,q]$ defines a correspondence between every point in P and every point in Q, in other words, it corresponds to a parameterization of P and Q. It is apparent that a monotone path through the free space would correspond exactly to monotone parameterizations of P and Q, thus, to answer the decision problem we need only decide if there exists a monotone path through the entire free space.

Since the free space itself does not hold information about the monotone paths through it, we define a subset of free space called the *reachable space*.

Definition 3.2. (Reachable Space) The *reachable space* R_ε is a subset of the parameter space of the free space F_ε such that

$$R_\varepsilon = \{(s,t) \in F_\varepsilon \mid \text{there exists a monotone curve within } F_\varepsilon \text{ from } (0,0) \text{ to } (s,t)\}$$

From the reachable space we can develop the following dynamic programming algorithm, which builds R_ε along the bottom and left edges of the free space first,

and then fills in the rest, until finally we know if (p, q) is reachable. If (p, q) is reachable, there exists a monotone path through F_ε, and therefore $\delta_F < \varepsilon$.

Algorithm 3.1 Compute the Fréchet distance between polygonal curves (decision problem)

for all cells $C_{i,j}$ **do**
    compute the free space
end for
for $i = 1 \to p$ **do**
    determine $R_\varepsilon \cap C_{i,1}$
end for
for $j = 1 \to q$ **do**
    determine $R_\varepsilon \cap C_{1,j}$
end for
for $i = 1 \to p$ **do**
    for $j = 1 \to q$ **do**
        determine $R_\varepsilon \cap C_{i,j+1}$ and $R_\varepsilon \cap C_{i+1,j}$ from previously computed cells
        answer yes if $(p, q) \in R_\varepsilon \cap C_i + 1, j$, no otherwise
    end for
end for

The runtime of Algorithm 3.1 is dominated by the nested for loop, which runs in $O(pq)$ time. As mentioned in Section 3.3.2, the decision problem can then be used to find the Fréchet distance between polygonal curves in $O(pq \log pq)$ time.

4

Fréchet Distance for Simple Polygons

Recall that the Fréchet distance can be defined for surfaces rather than curves (see Definition 2.17). In general, computing the Fréchet distance between arbitrary surfaces is NP-hard [God98]. It has been shown that for a general surface, the Fréchet distance is upper-semi-computable, and it is currently unknown if it is computable [AB10]. Computing the Fréchet distance between piecewise linear surfaces, even if one surface is a triangle, is NP-hard [God98], as is the case when both surfaces are polygons with holes or terrains [BBS10]. Much of the previous and current work on the Fréchet distance between surfaces relies on either restricting the surfaces that are considered so that they have some "nice" properties that make them easier to compute, or on approximating the Fréchet distance, or both.

For example, if we restrict the surfaces to simple polygons, the Fréchet distance can be computed in polynomial time [BBW07], as can the partial Fréchet distance [SW12]. The Fréchet distance for folded polygons can be approximated in polynomial time [CDHP$^+$11].

In this chapter we discuss the Fréchet distance for surfaces, with a focus on simple polygons. For the same reason that we focused on polygonal curves in the previous chapter, we focus here on simple polygons, namely, because simple polygons can often be employed as good approximations of other shapes or surfaces. We begin by

discussing previous work by Kevin and Maike Buchin and Carola Wenk [BBW07] on simple polygons and then describe a new approach to their algorithm which improves the runtime by a linear factor.

4.1 Simplifying the Fréchet Distance for Surfaces

The Fréchet distance for surfaces (Definition 2.17) is difficult to work with because it involves taking the infimum over an infinite set of homeomorphisms. In this section we develop the theory needed to simplify the homeomorphisms to a certain, well-behaved class of maps that will allow us to compute the Fréchet distance between simple polygons. First we consider the definition of the Fréchet distance for surfaces restricted to simple polygons.

Definition 4.1. (Fréchet Distance for Simple Polygons) Let P and Q be two simple polygons with p and q vertices, respectively. Then the Fréchet distance for surfaces simplifies to:

$$\delta_F(P, Q) = \inf_{\sigma: P \to Q} \max_{t \in P} \|t - \sigma(t)\|$$

assuming $f : P \to P$ and $g : Q \to Q$ and where σ ranges only over orientation-preserving homeomorphisms.

Note that σ is defined as orientation-preserving, and that this orientation will be clear given an ordering of vertices. It is also worth noting that the theory and algorithm developed below can be extended to orientation-reversing homeomorphisms and, by extension, general homeomorphisms.

Definition 4.2. A map $\sigma : P \to Q$ such that $\max_{t \in P} \|t - \sigma(t)\| \leq \varepsilon$ is an *ε-realizing map*.

In other words, the Fréchet distance for simple polygons is the infimum over all

ε-realizing homeomorphisms. The decision problem then becomes

$$\delta_f(P,Q) \leq \varepsilon \Leftrightarrow \text{ for all } \varepsilon' > \varepsilon \text{ there exists an } \varepsilon'\text{-realizing homeomorphism}$$

Note that an ε-realizing homeomorphism may not exist, but there will be a ε'-realizing homomorphism such that ε' is arbitrarily close to ε. In particular, the infimum is not a minimum whenever the limit of the homeomorphisms is no longer injective.

4.1.1 Fréchet Distance between Boundary Curves

It is a natural and necessary question to ask if the Fréchet distance between two polygons is equal to the Fréchet distance between their boundaries. It can be shown that this is not true in general.

Theorem 4.3. *Let P and Q be simple polygons with boundary curves ∂P and ∂Q. Then $\delta_F(P,Q)$ is not necessarily equal to $\delta_F(\partial P, \partial Q)$*

The proof of the theorem can be found in [BBW07], and relies on the illustration in Figure 4.1 and the shape known as "Fréchet's pants." It can be shown that the Fréchet distance between the two surfaces of the polygons in Figure 4.1 is $h/2$ as w and δ tend to zero, even while the Fréchet distance between their boundaries clearly tends to zero under the same transformation.

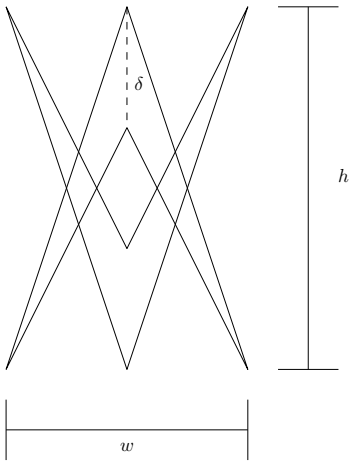

Figure 4.1: The Fréchet distance between the boundaries is different from the Fréchet distance between the surfaces.

Theorem 4.4. *The Fréchet distance between convex polygons equals the Fréchet distance between their boundary curves.*

Theorem 4.4 is a direct corollary of Theorem 4.11. The convex decomposition of a convex polygon is the polygon itself, and therefore a shortest path map is a homeomorphism on the boundary. It can be shown that the theorem holds for convex polytopes in general [BBW07].

Our ultimate goal is to restrict the homeomorphisms that we need to consider so that they can be more easily computed. To that end, we present the following lemmas needed to reach the result presented in Theorem 4.10. They are stated here without proof, but a more in-depth treatment can be found in [BBW07].

First we review the concept of an *hourglass*, presented by Guibas et al. in [GHL$^+$86].

Definition 4.5. (Hourglass) Let A be a simple polygon and let S_1 and S_2 be non-adjacent sides of A with endpoints a_1, a_2 and b_1, b_2 respectively. The *hourglass* of S_1 and S_2 is the set H such that

$$H = \{s \in A \mid s \text{ lies on the shortest path in } A \text{ from } a_i \text{ to } b_i\}$$

where $a_1 \leq a_i \leq a_2$ and $b_1 \leq b_i \leq b_2$.

Note that this defines a subset of the polygon, and that the boundaries of that subset are defined by S_1, S_2 and the shortest path from a_1 to a_2 and b_1 to b_2.

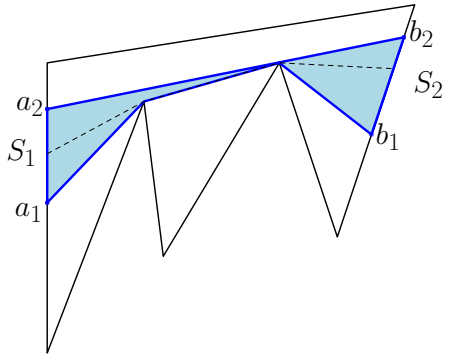

Figure 4.2: The hourglass between S_1 and S_2

Figure 4.2 shows an example of an hourglass. It's also important to note that the concept holds for chains of edges as well, not simply subsets of two single edges.

Lemma 4.6. *Let $f : [0,1] \to \mathbb{R}^d$ be a curve, let $s : [0,1] \to \mathbb{R}^d$ be a parameterized line segment, and let $0 \leq t_1 < t_2 \leq 1$. Define $f' : [0,1] \to \mathbb{R}^d$ to coincide with f on $[0,t_1] \cup [t_2,1]$ and to coincide with a parameterized segment from $f(t_1)$ to $f(t_2)$ on $[t_1,t_2]$. Then $\delta_F(f',s) \leq \delta_F(f,s)$.*

Lemma 4.6 implies that, given a line segment and any arbitrary curve, we can replace a portion of the curve with a line segment (known as "shortcutting") without increasing the Fréchet distance between the two. This will allows us to tame the interior of the homeomorphisms we are considering into nicer maps.

It is easier to see why Lemma 4.6 is true with a picture (see Figure 4.3). The proof hinges on the fact that the maximum distance between a segment and a polygonal line is realized by the endpoints, thus, replacing part of the interior with a line segment does not increase the overall Fréchet distance. In effect, we're removing some of the interior of the free space diagram, without changing $(0,0)$ or (p,q) — the parts that matter.

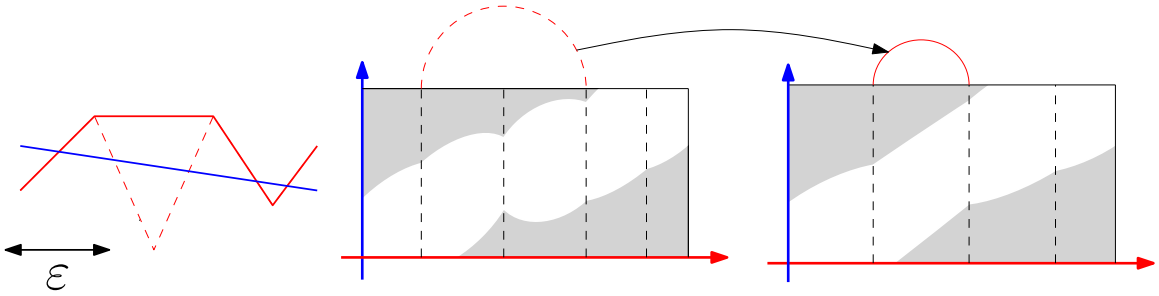

Figure 4.3: An illustration of simplifying a curve. Note that this simply removes a column from the free space diagram.

Lemma 4.7. *Let L be a line segment from a to b, and I_1 and I_2 be two non-intersecting intervals along the boundary of a polygon such that I_1 is in the ε-disk around a and I_2 is in the ε-disk around b. Suppose S is the shortest path connecting $i_1 \in I_1$ and $i_2 \in I_2$, and that $\delta_F(L, S) \leq \varepsilon$. Then for all shortest paths S' connecting $i_i \in I_1$ to $i_j \in I_2$, $\delta_F(L, S') \leq \varepsilon$*

Lemma 4.7 makes a fairly bold statment: Given a line segment and an hourglass, if the Fréchet distance between a single shortest path in the hourglass and the line is less than ε, then the Fréchet distance between the segment and any shortest path in the hourglass is also less than ε.

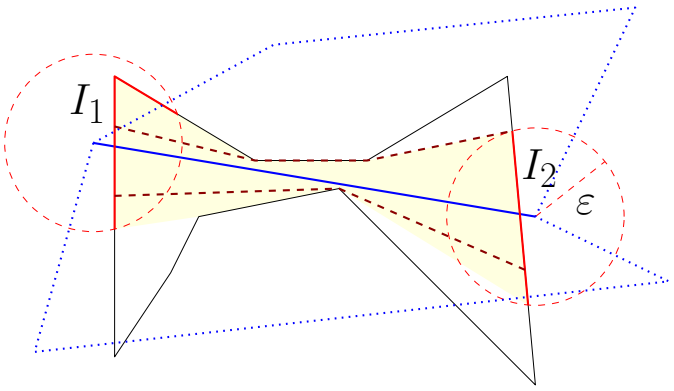

Figure 4.4: Hourglass Lemma. The blue diagonal defines I_1 and I_2, and any shortest path in the hourglass between them has Fréchet distance less than ε to the diagonal.

This, too, is more easily seen with a picture. Because any point in the intervals I_1 and I_2 is within an ε-ball of the endpoints of L, if the Fréchet distance between L and

one shortest path S is less than ε, then the same is true for any shortest path in the hourglass. The hourglass restricts the interior of the path so that all paths behave similarly, and the endpoints are restricted by the ε-ball that defines the interval.

Lemma 4.8. *Given two simple polygons P and Q, a diagonal D of P and a homeomorphism $\sigma : P \to Q$, let $\sigma' : P \to Q$ map the diagonal D homeomorphically to the shortest path between the images of its endpoints under σ. Then*

$$\delta_F(D, \sigma'(D)) \leq \delta_F(D, \sigma(D))$$

Lemma 4.8 simply states that by restricting the homeomorphisms to shortest path maps, we do not increase the Fréchet distance. This relies on Lemma 4.6, and note that any diagonal D is a line segment and that therefore the lemma is applicable. We know that a homeomorphism σ must map the endpoints of D to the boundary of Q, and that therefore D is mapped to some curve in Q. Using Lemma 4.6, we can progressively refine the curve that D maps to, without increasing the Fréchet distance between it and D, until the path is in fact the shortest path between the endpoints. In other words, if $\delta_F(P, Q) \leq \varepsilon$, then for all $\varepsilon' < \varepsilon$, there exists an ε'-realizing shortest path map.

Lemma 4.9. *Given two simple polygons P and Q, a convex decomposition C of P and a shortest path map $\sigma' : C \to Q$, then for all $\delta > 0$ there exists a homeomorphism $\sigma_\delta : P \to Q$ that realizes a Fréchet distance not larger than δ minus the Fréchet distance realized by σ'. In other words,*

$$\max_{t \in P} \|t - \sigma_\delta(t)\| \leq \max_{t \in C} \|t - \sigma'(t)\| + \delta$$

Lemma 4.9 implies that, if for all $\varepsilon' > \varepsilon$ there exists an ε' realizing shortest path

map. Lemma 4.9 and Lemma 4.8 together imply that

$$\delta_F(P,Q) \leq \varepsilon \Leftrightarrow \text{ for all } \varepsilon' > \varepsilon, \text{ there exists an } \varepsilon'\text{-realizing shortest path map.}$$

This states that by restricting the homeomorphisms to shortest path maps, the Fréchet distance does not change.

Theorem 4.10. *Let P and Q be two simple polygons with p and q vertices, respectively. Then the Fréchet distance between them can be defined as:*

$$\delta_F(P,Q) = \inf_{\sigma' : E_C \to Q} \max_{t \in C} \|t - \sigma'(t)\|$$

where C is an arbitrary convex decomposition of P and σ' ranges only over orientatioin-preserving shortest path maps from C to Q.

Furthermore, recall that the Fréchet distance between closed curves is equal to the Hausdorff distance between them. Therefore, the Fréchet distance between two convex polygons is equal to the Hausdorff distance between their boundary curves, which can be computed in $O((p+q)\log(p+q))$

Theorem 4.11. *The Fréchet distance between two simple polygons P and Q such that either P or Q is convex is equal to the Fréchet distance between their boundary curves.*

The convex decomposition of a convex polygon is the polygon itself, and therefore a shortest path map is a homeomorphism on the boundery.

Now that we have restricted the homeomorphisms to shortest path maps, we have something a bit easier to compute, and we are ready to develop the algorithm to do so.

4.2 Algorithm for the Fréchet Distance for Simple Polygons

In this section we develop an algorithm for the decision problem for the Fréchet distance between simple polygons as outlined in [BBW07]. In Section 4.1, we demonstrated that it suffices to restrict the space of homeomorphisms to only those that map diagonals of a convex decomposition of P to shortest paths in Q such that the Fréchet distance between each diagonal and its corresponding shortest path is at most ε.

The approach is similar to that used for calculating the Fréchet distance for polygonal curves. Before we characterize the decision problem in terms of the free space, it is important to note that there are two major differences between the problem for simple polygons and the problem for polygonal curves. The first is that polygonal curves have a defined start and end point, and therefore it is known beforehand where the path through the free space diagram should start and end (for the case of polygonal curves, $(0, 0)$ and (p, q)). The second problem is the issue of the distance between the diagonal and the shortest paths. It is clear that we can change where the endpoints of a diagonal in P map to in Q simply choosing a different path through the free space diagram.

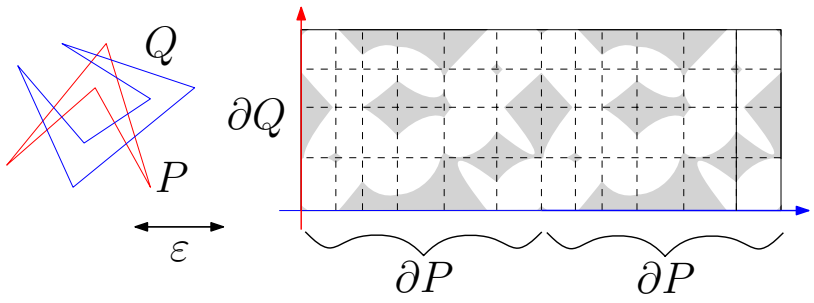

Figure 4.5: The double free space for P and Q.

Alt and Godau proposed a solution to the first problem in [AG95]. Instead of

simply considering the free space diagram with the boundary of P on the x-axis and the boundary of Q on the y-axis, we construct the *double free space diagram* (Figure 4.5) so that the boundary of P is doubled on the bottom edge. This gives us multiple places to start (possibly), which means that instead of finding a path from $(0, 0)$ to (p, q), one must find a path from somewhere in $[0, 2p]$ to $[q, 2p]$ such that the difference in x-values of the start and end points is exactly p.

4.2.1 Reachability Structure

The *reachability structure* proposed by Alt and Godau [AG95] is a partition of the boundary of the double free space diagram into $O(pq)$ intervals. Each interval on the bottom or left of the boundary is labeled *n-type*, *s-type*, or *r-type*, corresponding to *non-reachable*, *see-through*, and *reachable* intervals respectively. These intervals are calculated for each cell, and then recursively merged into the full reachability structure.

Definition 4.12. Given a free space diagram D, let B, T, L, and R be the bottom, top, left, and right boundaries of D, respectively. The reachability structure is a finite partition of the boundary of D into three types of intervals:

- **type n, non-reachable**: a connected subset $I \subseteq L \cup B$ such that from *no* point on I can any point on on $R \cup T$ be reached by a monotone path in F_ε.
- **type r, reachable**: a connected subset $I \subseteq L \cup B$ such that from any two points in I, the same set of points in $R \cup T$ can be *reached*.
- **type s, see-through**: a connected subset $I \subseteq L$ ($I \subseteq B$) such that from any point in I the horizontal (vertical) line segment connecting it with R (T) lies completely within F_ε.

In addition to storing the interval information, we also store a "high" and "low"

pointer for each interval of type r and s on $L \cup B$, pointing to the highest and lowest reachable points on $R \cup T$. See Figure 4.6.

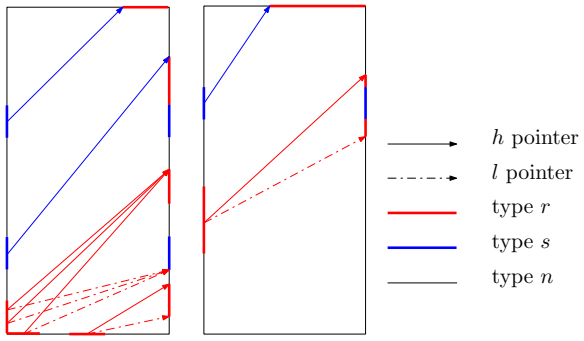

Figure 4.6: Two free space cells with interval types and pointers marked.

If D is the free space diagram between two line segments, it's clear that we can compute the reachability structure in constant time. For a larger diagram we use a divide-and-conquer approach by splitting along the longer side. This requires that we define a method of merging two adjacent cells in the free space diagram.

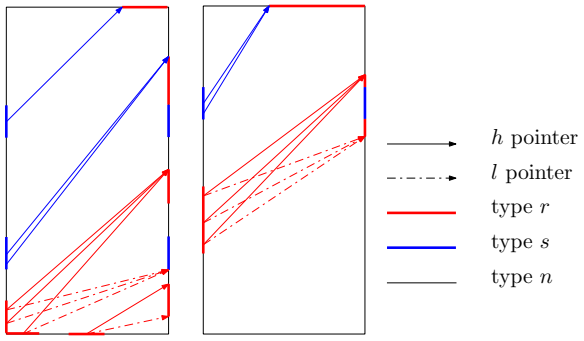

Figure 4.7: The same free space cells with their adjacent boundaries refined.

To merge two smaller reachability graphs, D_1 and D_2 along a vertical into a larger one, D, we first refine the partitions of the shared edge (R_1 and L_2, using our notation), which in turn causes a refinement of the partition along L_1 and R_2. Each new interval acquires the type of the interval of which it is a subset, and the arrows of the superset interval are transferred to the new one. See Figure 4.6 and Figure 4.7 for an example.

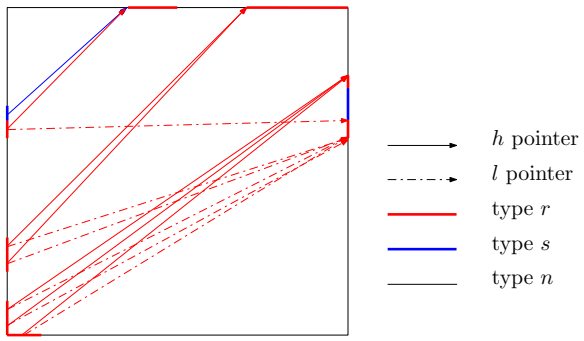

Figure 4.8: Cells fully merged with all boundaries refined.

The details of transferring the refinement of the shared boundary (R_1 and L_2) are omitted for brevity, but for the most part merging is intuitive; i.e., an r-interval on B that points to an r-interval on the newly refined edge, then the r-interval on B gets the high and low pointers that r-interval on the boundary has. See [AG95] for the other cases, and Figure 4.8 for an example of the final, merged reachability graph D.

After all cells of the reachability structure are merged, if the start point lies within an interval of type r or s, and the end point is between the high and low pointers *and* the start point lies within the high and low pointers of the end point with the directions of the graph reversed, then there exists a path through the free space.

Note that the reachability structure is similiar in concept to Algorithm 3.1. The difference is that, instead of an algorithm that runs once and returns the answer "yes" or "no," we have a structure that we can then query for that answer. This is important, because as we noted earlier, there is no defined start or endpoint for closed curves. The reachability structure allows us to check each reachable interval on the bottom to see if a path exists that will realize the Fréchet distance.

4.2.2 Combined Reachability Graph

In [BBW07], the solution to the problem of mapping diagonals to shortest paths is discussed in detail. The idea of the reachability structure is extended to a structure called the *combined reachability graph*. This is defined exactly as the reachability

structure is defined, except that during its construction, we remove intervals that would create an invalid mapping of a diagonal.

There are a couple of key observations that highlight the need for a structure different from the normal reachability graph. The first has been mentioned already, which is simply that whatever method we use to compute the Fréchet distance between P and Q must also ensure that the Fréchet distance between each diagonal and its corresponding shortest path is also less than ε. The second observation is that diagonals have an implicit ordering, and not only that, but this ordering is important and must be accounted for by the algorithm.

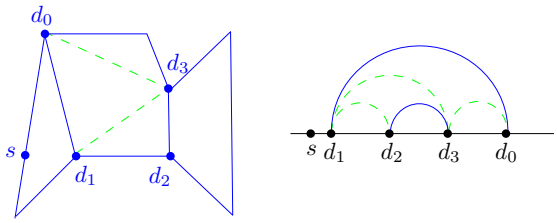

Figure 4.9: The diagonals of the convex decomposition define a nesting structure.

For example, consider the convex decomposition shown in Figure 4.9. Starting at the point s, we walk around the polygon in counter-clockwise order, and note the order of the vertices that we hit. If we then connect the diagonals as defined by our decomposition, we see that a diagonal may be nested inside another diagonal; in other words, we hit the start point of a diagonal, then the start point of a second diagonal, then the endpoint of the second diagonal, and finally the end point of the first diagonal. Figure 4.9 shows this nesting structure along the boundary.

This has implications for any algorithm that computes the Fréchet distance between simple polygons. Consider again the polygon with its decomposition shown in Figure 4.9. Suppose that we find a homeomorphism that maps the boundary of the polygon to another polygon, and maps d_1 to d_0 to points s_1, s_0 such that the Fréchet distance between the diagonal and the shortest path from s_1 to s_0 is less than ε. That information alone does not guarantee that the diagonal from d_2 to d_3 also

maps to points connected by a shortest path within ε.

It is clear, then, that we must first find a mapping for the inner diagonal, and then find a mapping for the outer diagonal that respects the mapping of the inner. (One could, theoretically, go in the other direction, but it would only require more work — consider the case where one diagonal is nested within three larger diagonals; the inner diagonal would need to be checked and possibly modified three times).

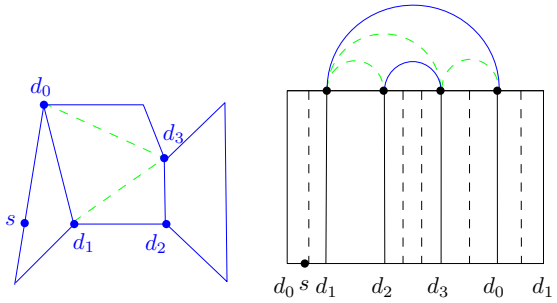

Figure 4.10: The diagonals of the convex decomposition correspond to a set of columns in the free space.

Note that a diagonal in P corresponds to a set of columns in the free space (see Figure 4.10). This gives us a straightforward way to enforce our diagonal requirements for constructing the combined reachability graph. First, we must determine the diagonal nesting structure, then we proceed as normal, except instead of simply merging recursively, we merge according to the diagonal structure, so that inner diagonals are merged first.

Theorem 4.13. *The nesting structure of the diagonals depends on the start point s.*

The proof of Theorem 4.13 can be seen by considering Figure 4.10. This means that, given different start points, we will need to merge columns in a different order. The effects of this can be minimized by memoizing columns that have been merged and reusing them when appropriate.

4.2.3 Algorithm

We are finally able to define an algorithm to compute the Fréchet distance between simple polygons.

Algorithm 4.1 Compute the Fréchet distance between simple polygons (decision problem)

Compute the convex decomposition of P with d diagonals and compute the free space diagram of the boundary curves of P and Q.
for all diagonals in the convex decomposition of P **do**
  for all possible hourglasses for the given diagonal **do**
    Decide if $\delta_F(\text{diagonal}, \text{shortest path}) \leq \varepsilon$ for a shortest path in the hourglass
    Store the result so that we can find valid intervals in $O(1)$.
  end for
end for
for All columns of the reachability graph **do**
  Merge according to the diagonal nesting structure.
  Query for a feasible path
end for
Answer "yes" if a feasible path has been found, else "no"

The runtime for this approach is $O(dT(n))$, where d is the number of diagonals and $O(T(n))$ is the time it takes to multiply two $N \times N$ matrices [BBW07]. The matrix multiplication is used to compute the transitive closure of the reachable intervals when merging columns. The naïve implemention is $T(n) = O(n^3)$, but this can be improved using Strassen's algorithm to $O(n^{\log_2 7}) = O(n^{2.807})$ [Str69]. There is an algorithm that has a time complexity of $O(n^{2.37})$ [Vas11], however, it is difficult to impossible to implement and has huge constant factors.

If we allow that $p = q = d = n$, then the runtime for this algorithm is $O(n^7)$, or,

using Strassen's algorithm, $O(n^{6.614})$.

4.3 Improved Algorithm for the Fréchet Distance for Simple Polgyons

In this section we develop an algorithm that improves the runtime of Algorithm 4.1 by a linear factor. It should be noted that for all of the algorithms mentioned above, it is impossible to actually recover the path through the free space from the reachability graph or the combined reachability graph; the only information available after building those structures is high and low pointers. The motivation for this algorithm came from a desire to recover an original path, which it does, and does so with the added benefit of being faster.

4.3.1 Intervals, Revisited

In Definition 4.12 we defined intervals of type n, r, or s. These intervals were defined such that they could span multiple cells in the free space diagram, so that any given merge step in the algorithm could cause some of the intervals to change or split.

We begin by redefining our interval types so that they better encapsulate local, rather than global, free space information.

Definition 4.14. Given a cell C_{ij} of a free space diagram D, let B, T, L, and R be the bottom, top, left, and right boundaries of C_{ij}, respectively. Then we define the following types of intervals:

- **type n, non-reachable**: a connected subset I of a single edge of the boundary of L or B such that from *no* point on I can any point on on $R \cup T$ be reached by a monotone path in F_ε.

- **type c, combined-reachable**: a connected subset I of a single edge of the boundary of L or B such that for any $i \in I$, there exists a monotone path from i to a point on $R \cup T$ that lies completely within F_ε. Note that this is essentially an $(s \cup r)$-type interval.

We further define the corresponding intervals on R and T analogously, i.e., if a point in a c-type interval can reach a point in $R \cup T$, that point is part of a c-type interval, otherwise it is an n-type interval.

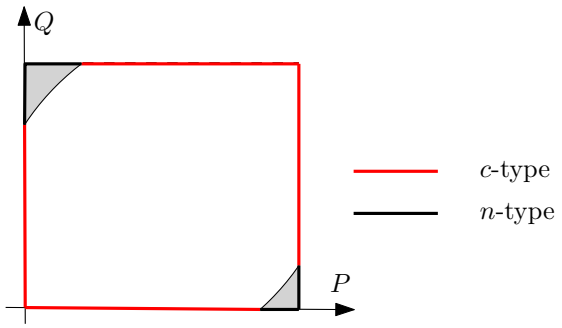

Figure 4.11: The c and n intervals for a single cell on $L \cup B$, and the corresponding intervals on $R \cup T$.

Figure 4.11 demonstrates that for a single cell C_{ij} of the free space, the c-intervals correspond exactly with intervals defined by $F_\varepsilon \cap C_{ij}$. The approach is the same as before; we have seen what intervals look like for a single cell and we wish to find a way to merge them so that we can use a divide-and-conquer approach to finding the c-intervals for the entire free space.

4.3.2 Arrows

The merging step is considerably more involved in this case, so we introduce the notion of an *arrow*. The definition aids with the merging steps but will have several consequences which we explore in this section.

Definition 4.15. Given two c-intervals I_1 and I_2, we define an *arrow* from I_1 to I_2 iff for any point $s \in I_1$ there exists a monotone path to some point $p \in I_2$.

It should be clear from the definition that an arrow can only exist if I_1 lies on $L \cup B$ and I_2 lies on $R \cup T$. Recall that in this case, L, B, R, and T are defined locally, and therefore I_1 and I_2 can be in different cells. For a single cell, the arrows are as shown in Figure 4.12 below.

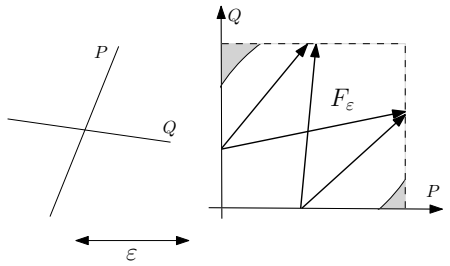

Figure 4.12: A single cell of the free space with arrows added.

Stacking

It's important to note that c-intervals can be *stacked* on top of one another. Suppose we have two adjacent cells in the free space diagram, C_1 and C_2, such that there is a c-interval on L_1, R_1, L_2, and R_2 each, which we will call l_1, r_1, l_2, and r_2, respectively. Further suppose that, ignoring their x-values, $l_1 = r_1$ and $r_2 \subset l_2$. Now consider the arrows that these intervals define:

- There is an arrow A_1 from l_1 to r_1.
- There is an arrow A_2 from l_2 to r_2.
- There is an arrow A from $I' \subset l_1$ to r_2.

Note that I' is a subset of I_1, which is what we mean by *stacked*: two c-intervals can occupy the same space on an interval, but the parts of $R \cup T$ that they can reach are different (more precisely, they are defined with respect to a different R or T). Figure 4.13 illustrates stacking.

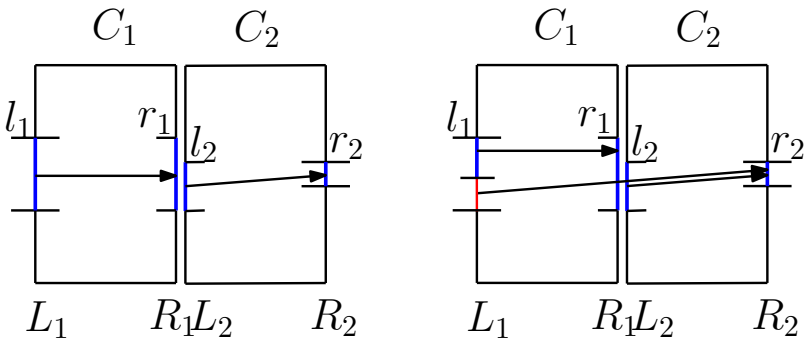

Figure 4.13: On the right, A is drawn in so that I', the red interval, is stacked on top of I_1

We would like to develop a way to determine A from A_1 and A_2 so that we ultimately have a way to merge cells. One of the key points of this approach is after we have A, we no longer need A_1 or A_2. Any monotone path that would have passed first through A_1 and then A_2 is now entirely represented by the arrow A. (It should be noted, however, that this does not mean that stacking does not occur if we forget A_1 and A_2 immediately after merging — what if there was a path from l_1 to t_2, a c-interval on T_2, that passes through r_1?)

Sub-Arrows

A problem arises if we completely forget A_1 and A_2, however, which is that we lose information about the boundary between C_1 and C_2. Without that intermediate interval, we have no way to know what the path between I_b and r_2 looks like. For this reason we define the notion of a *sub-arrow*, which we will discuss in Section 4.3.2.

An arrow could have two sub-arrows, both or either of which have subarrows themselves. This defines an *arrow tree* for any given arrow, such that the leaves of the arrow tree are precisely the arrows that span a single cell in the free space diagram.

Merging

It should be clear now that defining how to merge two cells together is the same as defining how to merge two arrows together: for every pair of arrows in adjacent cells, we merge them if necessary.

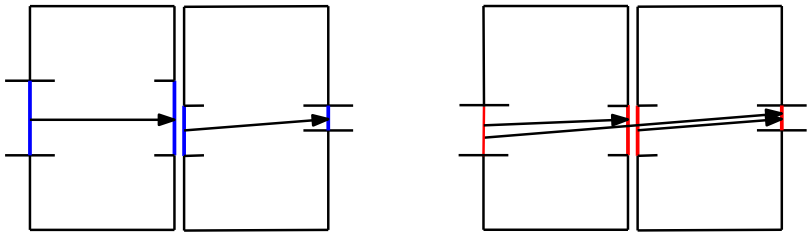

Figure 4.14: The arrows are merged and the new arrow has references to modified copies of the original arrows.

Suppose we have two adjacent cells (either or both of which may have been merged from other cells) in the free space diagram, C_1 and C_2 such that there is an arrow A_1 from l_1 to r_1 and an arrow A_2 from l_2 to r_2. If $r_1 \cap l_2$ is nonempty, we define the merged arrow A by way of two sub-arrows A'_1 and A'_2 such that A'_1 ends at, and A'_2 starts at $r_1 \cap l_2$. A'_1 starts from the same interval that A starts and A'_2 ends at the same interval A ends at. Note that the start and end intervals of A come from the definition of an arrow; and the c-intervals defined by A can be modified by the interim interval, $r_1 \cap l_2$.

After merging, A_1 or A_2 may need to be kept. Consider the example in Figure 4.15, and note that the gray arrows were used to construct the blue arrows. If we removed the gray arrows before merging the other arrows, we would not end up with every blue arrow that we should. After two cells have been merged, however, the gray arrows are no longer necessary.

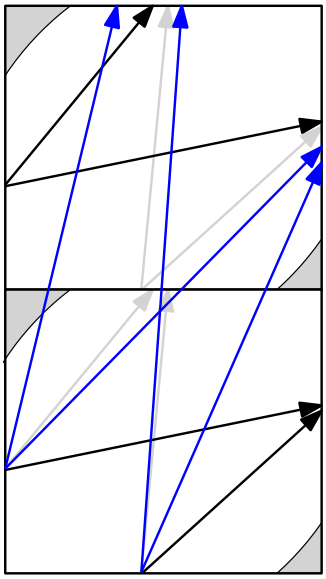

Figure 4.15: Two cells after merging arrows. The blue arrows are new, the gray arrows are removed.

The idea that not all arrows need to be kept is the basis for our later runtime analysis. It allows us to argue about the combinatorial complexity of intersecting c-intervals without the number of comparisons becoming exponential. This will be discussed in depth in Section 4.3.4.

We note, here, that the above merging strategy means that two arrows cannot be merged in constant time. The time it takes to merge will be proportional to the number of sub-arrows, which is in the worst case $O(p + q)$.

It may be apparent that, when merging large numbers of cells together, there may be multiple arrows that point from the same start interval (or subset of the same start interval) to the same end interval (or subset of the same end interval). We would like to ensure that, in a case like that, we can pick exactly one of those arrows to keep without losing any information for the decision problem.

Theorem 4.16. *Suppose there exists two arrows A and B such that both start from the same side of the one cell and end on the same side of another cell. It suffices to keep only the arrow whose final interval is the largest.*

Proof. Assume that the end interval E is on the right side of the final cell, let $I \subseteq E$ be the final interval of A and let $J \subseteq E$ be the final interval of B, without loss of generality. There are three cases:

1. A and B both pass through (i.e., the path they define must pass through) the bottom interval of the final cell. Then $I = J = E$, and we can pick either to keep. *However, this can never happen, because if A and B both pass through the bottom, then one of the two would have been invalid earlier.*

2. A passes through the bottom interval of the final cell, and B passes through the left interval of the final cell. Then $I = E$ and $J \subseteq I$ is connected. Then we keep A over B always, and we are still guaranteed a monotone path through the free space between the same intervals.

3. A and B both pass through the left side of the final cell. Then either $I \subseteq J$ or $J \subseteq I$, and we keep the arrow of whichever is larger. *However, this can never happen, because if A and B both pass through the left, then one of the two would have been invalid earlier.*

□

Theorem 4.16 gives us an equivalence between the maximum number of distinct arrows that could represent monotone paths, and the maximum number of distinct arrows that point monotonically from edges of the free space diagram. I.e., if there is a possible path between two intervals in the free space, it suffices to count only one arrow between them.

4.3.3 Diagonals

We discussed before how diagonals correspond to columns in the free space, and that they must be merged in a specific order. The switch from intervals and pointers to arrows does not change this requirement.

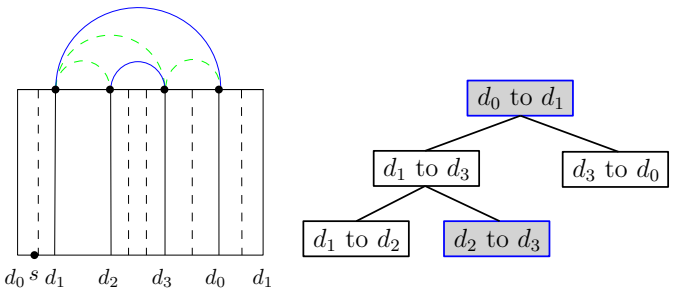

Figure 4.16: The diagonals drawn over the free space, which corresponds to the nodes of the diagonal tree.

We define a *diagonal tree* to be a tree structure that encodes an ordering of the diagonals such that a depth-first traversal of the tree yields the correct merge order. Figure 4.16 shows how a diagonal tree might look with respect to a given diagonal nesting structure.

Note that we can populate the tree in $O(p)$ time. There are p nodes, d of which are true diagonals, and $p - d$ of which are not true diagonals but aid in the correct merge order (i.e., no shortest path checks must be performed).

4.3.4 Runtime Preliminaries

Before we give the full algorithm and its runtime, we consider smaller parts of the algorithm and compute the runtimes of those parts first. When the final algorithm is presented, it should be easy to cross-reference with this section.

The analyses below are not complicated — they are, for the most part, counting arguments — but they are somewhat involved. For the following, assume that we are working with two polygons P and Q with p and q vertices, respectively.

Merging a Column of Cells

The naïve approach to merging a column of cells vertically would simply be to check every arrow in the ith cell against every arrow in the $(i+1)$th cell for every $0 \leq i \leq q$. This would yield an undesirable runtime, and it forgets what we've already shown,

which is that after merging two cells, some are no longer needed.

The approach is to count how many arrows remain after merging a column of cells of height q. By height, we mean the number of vertical cells spanned by the arrow. By enumerating all of the arrows that are possible after a column is merged, we will know how many interval comparisons are actually necessary for merging a column.

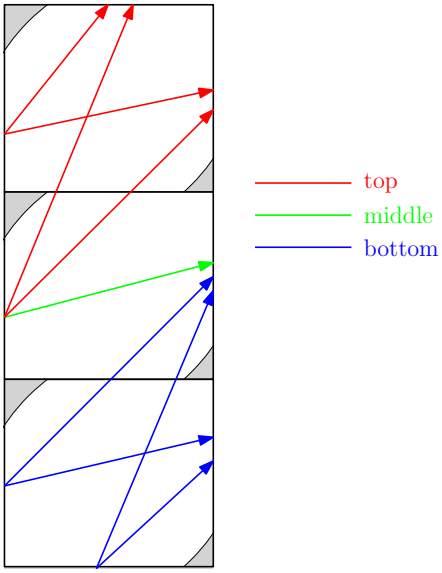

Figure 4.17: A column of height q with top, bottom, and middle arrows drawn in. Arrows of height q omitted.

We introduce a notion of *top*, *bottom*, and *middle* arrows. For a column of height q and arrows of height $h < q$, top arrows are those that end in the top cell, bottom arrows are arrows that begin in the bottom cell, and the rest are middle arrows. The distinction makes counting easier, which we will see in a moment. Figure 4.17 shows an example of a top, middle, and bottom arrow. Note that for $h = q$, it is unclear how to classify the arrows; for this reason we remove them from our analysis, noting that there are always at most four arrows of height q in a given column.

For every $h < q$ there are at most two top arrows, one for the top and right sides each of the top cell. Similarly, for every $h < q$, there are at most two bottom arrows, one for the bottom and left sides each of the bottom cell. See Figure 4.17 for an illustration. Therefore there are are a total of $4(q - 1)$ bottom and top arrows for a

single column.

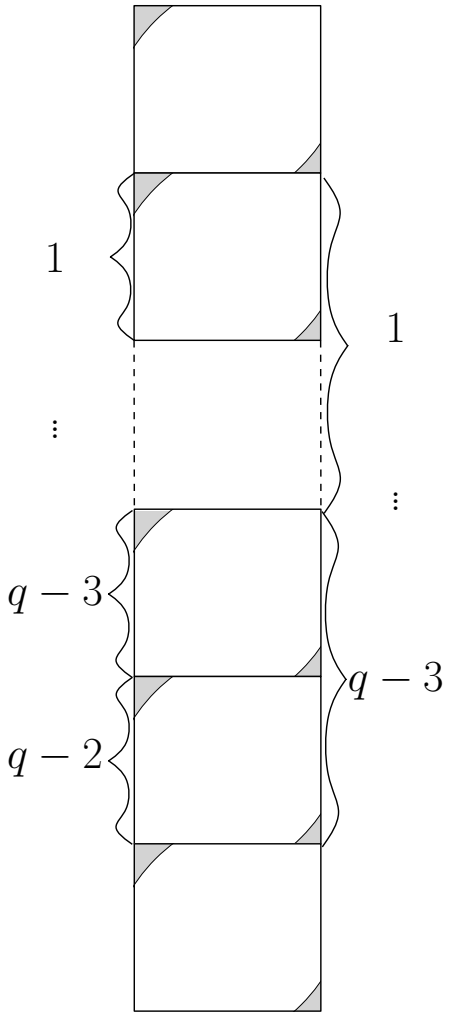

Figure 4.18: A column of height q showing how many middle arrows of height 1 and 2 can fit.

Counting the middle arrows requires some small amount of work. No arrows of height q or $q-1$ can fit in a column. However, we can fit one arrow of height $q-2$, two arrows of height $q-3$, three arrows of height $q-4$, and so on, until we see that we can fit $q-1$ middle arrows of height $q-2$. See Figure 4.18 for a visual representation of how arrows "fit." The number of middle arrows that can fit in a column of height q is then

$$\sum_{1}^{q-2} k = \frac{1}{2}(q-1)(q-2)$$

Therefore see see that for a column of height q, we have that the number of arrows is $4 + 4(q-1) + \frac{1}{2}(q-1)(q-2)$, which is $O(q^2)$.

Merging a column of cells must therefore require $O(q^2)$ arrow merges. As stated before, two arrows cannot be merged in constant time, since their arrow trees must be appropriately modified to enforce monotonicity. In a single column of cells, the maximum number of leaves in the arrow tree (and therefore the maximum number of nodes in the tree as well) is $O(q)$.

Thus a columns of cells requires $O(q^2)$ arrow merges which each costs $O(q)$ each, so that the expected runtime for merging a single column of cells is $O(q^3)$.

Merging Two Columns of Cells

Now that we know how long it takes to merge a single column of cells, we wish to know how long it takes to merge two adjacent columns of cells, and, further, that this time does not increase on consecutive column merges.

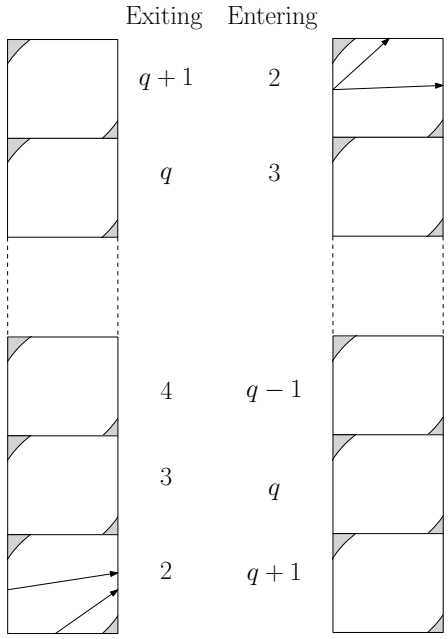

Figure 4.19: Counting how many arrows enter and exit at each row. Some arrows have been drawn in for reference.

The proof, as before, is a counting argument, though we take a slightly different

approach. If we have two columns of cells, and we count the number of arrows exiting and entering at each row, then we know how many arrows might intersect, and therefore how long it will take to merge.

We assume that our columns are already merged as described in the previous section. It is a fairly simple matter to enumerate the number of arrows entering on the left side of each cell and exiting on the right side of each cell. Figure 4.19 illustrates the situation, and Table 4.1 shows exactly how many arrows enter and exit each row, and the product of the two, which is the total number of arrows to be checked per row.

Table 4.1: Number of arrows entering and exiting at each row.

	q	$q-1$	$q-2$	$q-3$	$\cdots$	2	1
enter	2	3	4	5		q	q+1
exit	$q+1$	q	$q-1$	$q-2$		3	2
checks	$2(q+1)$	$3q$	$4(q-1)$	$5(q-2)$		$3q$	2(q+1)

It is clear from Table 4.1 that the total number of arrow merges required for merging two columns is

$$\sum_{k=2}^{q+1} k(q-(k-3)) = \frac{1}{6}(q^3+9q^2-16q-12)$$

Therefore merging two columns requires $O(q^3)$ arrows to merged. As before, the time to merge two arrows is not constant, but in this case each arrow tree has a maximum of $p+q$ leaves. Therefore the time two merge two columns is $O(q^3(p+q))$.

It is important to make the point that, although we have shown that merging two columns takes $O(q^3(p+q))$, we have not shown that successive merges do not increase the runtime. In other words, we need to demonstrate that merging 3 columns takes asymptotically the same time as simply merging 2 columns.

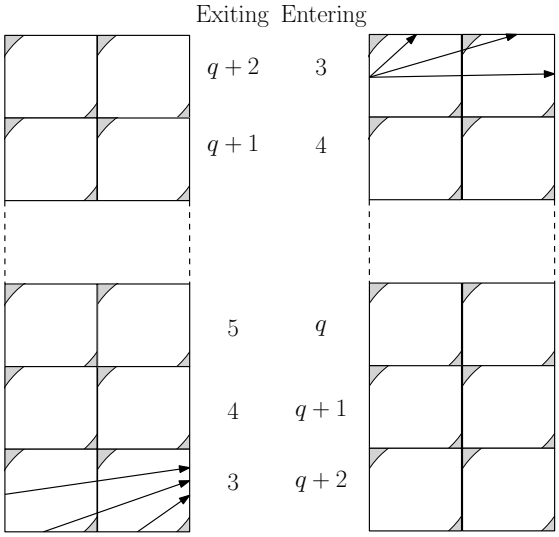

Figure 4.20: Counting how many arrows enter and exit at each row for pre-merged columns. Some arrows have been drawn in for reference.

The proof follows the same argument as before. We assume we have two columns already merged and wish to merge a third with it. We again count how many arrows enter and exit at each row; we know these values for a single column of height q and so must only count how many arrows enter and exit the pre-merged columns. Figure 4.20 enumerates these arrows explicitly by height and width of the arrow.

We can see from Figure 4.20 that we have the same relationship as before, in fact the same numbers as before increased by a constant factor. However, this constant factor depends on the number of previously merged columns, and therefore on p.

If we merge one column onto two pre-merged columns (either on the left or right, order doesn't matter), then we find the following number of arrow checks:

$$\sum_{k=2}^{q+1} k(q-(k-4))$$

For completeness' sake, merging the pth column onto $p+1$ pre-merged columns takes:

$$\sum_{k=2}^{q+1} k(q-(k-(p+1))) = \frac{1}{6}q(3p(q+3)+q^2+3q-4)$$

We can then sum over all merges:

$$\sum_{j=3}^{p+1}\sum_{k=2}^{q+1} k(q-(k-(2+j))) = \frac{1}{12}q(3p^2(q+3)+p(2q^2+21q+37)-2(q^2+12q+23))$$

so that the total number of arrow checks for merging p columns is $O(q^2p^2+q^3p)$.

We noted earlier that the each arrow merge takes $O(p+q)$ in the worst case; however, the worst does not occur in every merge. We would then like to include the cost of merging arrows in the overall runtime calculation.

Note that the maximum number of leaves that an arrow tree can have is $q+m$ where m is the number of already merged columns. Using our previous notation, $m=j-2$. Therefore the total runtime for merging all of the arows in p columns is

$$\sum_{j=3}^{p+1}\sum_{k=2}^{q+1}(q+(j-2))(k(q-(k-(2+j)))) =$$

$$\frac{1}{6}q(p^3(q+3)+p^2(2q^2+9q+7)+p(q^3+10q^2+13q-10)-q(q^2+12q+23))$$

which is $O(q^2p^3+p^2q^3)$. This is the runtime for merging p columns together.

4.3.5 Algorithm

We now know everything we need to perform an analysis of an arrow-based algorithm.

Algorithm 4.2 Compute the Fréchet distance between simple polygons (decision problem)

Compute the convex decomposition of P with d diagonals and compute the free space diagram of the boundary curves of P and Q.

for all diagonals in the convex decomposition of P **do**

  for all possible hourglasses for the given diagonal **do**

    Decide if $\delta_F(\text{diagonal}, \text{shortest path}) \leq \varepsilon$ for a shotest path in the hourglass

    Store the result so that we can find valid intervals in $O(1)$.

  end for

end for

for all columns in the free space **do**

  Merge column down

  Record feasible start points

  Build the diagonal tree if a feasible point exists

end for

for every diagonal tree **do**

  Walk each tree, merge columns, perform shortest path checks if necessary

end for

for every remaining arrow **do**

  Check if arrow spans P and Q, if so, a feasible path has been found

end for

Answer "yes" if a feasible path has been found, else "no"

Algorithm 4.2 is dominated by the merge step. There are d diagonal trees, each with p columns which need to be merged. Each merging of p columns takes $O(q^2p^3 + p^2q^3)$ time. We assumed, as before, that we can pre-compute the diagonal-to-hourglass mappings so that they can be determined in constant time. Therefore the total runtime is $O(d(q^2p^3 + p^2q^3))$.

If we allow that $p = q = n$, this algorithm has a runtime of $O(n^6)$, which is (at most) a linear factor faster than Algorithm 4.1.

5

Applications and Further Research

In this section we discuss several applications of the ideas outlined above, and areas that require further consideration or reasearch.

5.1 Morphing

One of the motivations for the arrow-based approach of Algorithm 4.2 was so that one could recover the original path through the free space and therefore a mapping between P and Q. It was postulated that the restrictions on the parameterizations might lead to some nice class of maps that could be useful in morphing problems. Morphing refers to continuously deforming one surface into another. [EGHP+01] provides a good overview of the problems, approaches, algorithms, and applications.

As it happens, there appears to be no appreciable link between a Fréchet distance realizing map and any qualitative properties of a morphing based on that map.

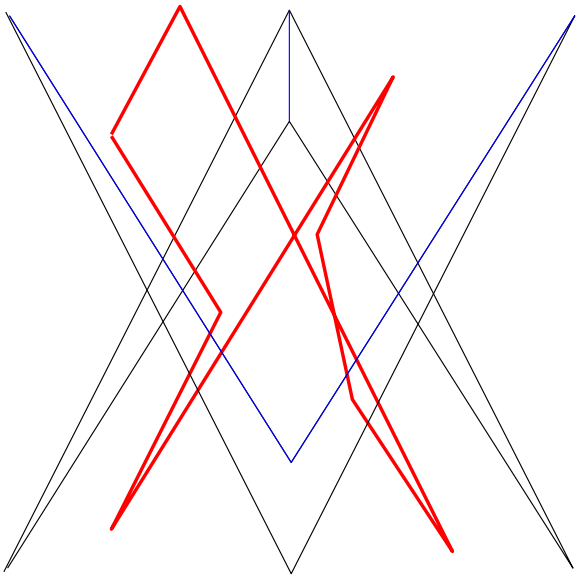

Figure 5.1: Two polygons, with diagonal and shortest path in blue, and an intermediate polygon between them based on the Fréchet distance.

Figure 5.1 shows one of the results of our experiment. The mapping defined by the Fréchet distance yields the intermediate polygon pictured when interpolating linearly.

It should be noted that our approach allowed us to recover the original path through the free space. Figure 5.2 shows the polygons P and Q with their intermediate polygon, as well as the path through the free space that defined the mapping. The path is drawn at the ε value that realizes the Fréchet distance.

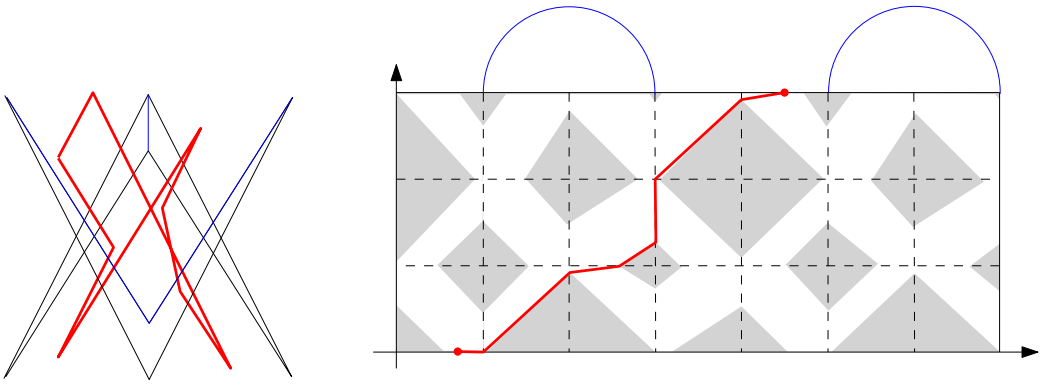

Figure 5.2: Morphing defined by a path in the free space. The diagonals have been drawn in.

5.1.1 Isotopic Fréchet Distance

Although we saw no useful results with the Fréchet distance as a morphing map, there is some related work that links the Fréchet distance and morphing.

The isotopic Fréchet distance (Definition 2.20) is a variant of the Fréchet distance which forces the motion between the input objects to follow an ambient isotopy (Definition 2.19). This maintains topologically equivalent shapes throughout a deformation; i.e., intermediate polygons of a morphing between simple polygons defined by the isotopic Fréchet distance would also be simple [CJLL11].

Chambers et al. show that the isotopic Fréchet distance is not equivalent to the homotopic Fréchet distance (for a which a polynomial time algorithm exists, as described in [CVE+10].) Currently no algorithm exists to compute the isotopic Fréchet distance. It should also be noted that a polynomial time algorithm exists to compute a morphing between polygons that does have simple intermediate polygons [EGHP+01].

5.2 ICP

Algorithm 4.2 could have applications in Iterative Closest Point algorithms, using the Fréchet distance instead of some other metric. ICP algorithms match shapes by moving one to another in small increments [ESE08]. It would be interesting to see when such an ICP algorithm reaches the optimal solution.

5.3 Future Research

There are several interesting paths that arise simply relating to the work done for Algorithm 4.2. It would appear, for example, that there is nothing preventing the use of the matrix multiplication approach of Algorithm 4.1. This could bring the

runtime down by another small factor (depending, as discussed in Section 4.2.3, on the multiplication algorithm).

It also seems that the arrow trees are only necessary if one wishes to recover the path through the free space, as we did for morphing. If it can be shown that an arrow does not need to keep a reference to its sub-arrows, then arrow merges can be done in $O(1)$ time, and the runtime of the algorithm drops by another linear factor, for a total of $O(n^5)$ for the decision problem.

The approach of using arrows seems to relate to graph-traversal, but with monotonicity enforced. If the approach could be generalized it may be more broadly applicable than just to Fréchet distance problems. Already, it seems that the arrow approach can improve the runtime of any algorithm that would otherwise require the combined reachability graph.

Bibliography

[AB10] Helmut Alt and Maike Buchin, *Can we compute the similarity between surfaces?*, Discrete & Computational Geometry **43** (2010), no. 1, 78–99.

[ABB95] Helmut Alt, Bernd Behrends, and Johannes Blömer, *Approximate matching of polygonal shapes*, Annals of Mathematics and Artificial Intelligence **13** (1995), no. 3-4, 251–265.

[ABG+03] Helmut Alt, Peter Braß, Michael Godau, Christian Knauer, and Carola Wenk, *Computing the hausdorff distance of geometric patterns and shapes*, Discrete and Computational Geometry (Boris Aronov, Saugata Basu, János Pach, and Micha Sharir, eds.), Algorithms and Combinatorics, vol. 25, Springer Berlin Heidelberg, 2003, pp. 65–76 (English).

[ABW90] Helmut Alt, Johannes Blömer, and Hubert Wagener, *Approximation of convex polygons*, Automata, Languages and Programming (MichaelS. Paterson, ed.), Lecture Notes in Computer Science, vol. 443, Springer Berlin Heidelberg, 1990, pp. 703–716.

[AG95] Helmut Alt and Michael Godau, *Computing the Fréchet distance between two polygonal curves*, Internat. J. Comput. Geom. Appl. **5** (1995), 75–91.

[AG99] H. Alt and L. Guibas, *Discrete geometric shapes: Matching, interpolation, and approximation*, Handbook of Computational Geometry (Jörg-

Rüdiger Sack and Jorge Urrutia, eds.), Elsevier Science Publishers, 1999, pp. 121 – 153.

[AST92] Pankaj K. Agarwal, Micha Sharir, and Sivan Toledo, *Applications of parametric searching in geometric optimization*, Proceedings of the third annual ACM-SIAM symposium on discrete algorithms (Philadelphia, PA, USA), SODA '92, Society for Industrial and Applied Mathematics, 1992, pp. 72–82.

[AW12] Mahmuda Ahmed and Carola Wenk, *Constructing street networks from GPS trajectories*, European Symposium on Algorithms (Leah Epstein and Paolo Ferragina, eds.), Lecture Notes in Computer Science, vol. 7501, Springer, 2012, pp. 60–71.

[BBG08a] K. Buchin, M. Buchin, and J. Gudmundsson, *Detecting single file movement*, Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems (New York, NY, USA), GIS '08, ACM, 2008, pp. 33:1–33:10.

[BBG+08b] Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo, *Detecting commuting patterns by clustering subtrajectories*, Algorithms and Computation (Seok-Hee Hong, Hiroshi Nagamochi, and Takuro Fukunaga, eds.), Lecture Notes in Computer Science, vol. 5369, Springer Berlin Heidelberg, 2008, pp. 644–655.

[BBS10] Kevin Buchin, Maike Buchin, and André Schulz, *Fréchet distance of surfaces: Some simple hard cases*, European Symposium on Algorithms (Mark Berg and Ulrich Mayer, eds.), Lecture Notes in Computer Science, vol. 6347, Springer Berlin Heidelberg, 2010, pp. 63–74.

[BBW07] Kevin Buchin, Maike Buchin, and Carola Wenk, *Computing the Fréchet distance between simple polygons*, 2007.

[CDG+11] Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk, *Approximate map matching with respect to the Fréchet distance*, ALENEX (Matthias Müller-Hannemann and Renato Fonseca F. Werneck, eds.), SIAM, 2011, pp. 75–83.

[CDHP+11] Atlas F. Cook, IV, Anne Driemel, Sariel Har-Peled, Jessica Sherette, and Carola Wenk, *Computing the Fréchet distance between folded polygons*, Proceedings of the 12th international conference on algorithms and data structures (Berlin, Heidelberg), WADS'11, Springer-Verlag, 2011, pp. 267–278.

[CJLL11] Eric W. Chambers, Tao Ju, Devid Letscher, and Lu Liu, *Isotopic Fréchet distance*, CCCG, 2011.

[Col87] Richard Cole, *Slowing down sorting networks to obtain faster sorting algorithms*, J. ACM **34** (1987), no. 1, 200–208.

[CVE+10] Erin Wolf Chambers, Éric Colin De Verdiére, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite, *Homotopic fréchet distance between curves or, walking your dog in the woods in polynomial time*, 2010.

[EEMM94] Thomas Eiter, Thomas Eiter, Heikki Mannila, and Heikki Mannila, *Computing discrete Fréchet distance*, Tech. report, Technische Universität Wien, 1994.

[EGHP+01] Alon Efrat, Leonidas J. Guibas, Sariel Har-Peled, Joseph S. B. Mitchell, and T. M. Murali, *New similarity measures between polylines with applications to morphing and polygon sweeping*, Proceedings of the 12th

Annual ACM-SIAM Symposium on Discrete Algorithms, 2001, pp. 680–689.

[ESE08] Esther Ezra, Micha Sharir, and Alon Efrat, *On the performance of the ICP algorithm*, Computational Geometry **41** (2008), no. 1-2, 77–93.

[GHL+86] L Guibas, J Hershberger, D Leven, M Sharir, and R Tarjan, *Linear time algorithms for visibility and shortest path problems inside simple polygons*, Proceedings of the second annual symposium on Computational geometry (New York, NY, USA), SCG '86, ACM, 1986, pp. 1–13.

[God98] Michael Godau, *On the complexity of measuring the similarity between geometric objects in higher dimensions*, Ph.D. thesis, Freie Universität Berlin, 1998.

[JXZ07] Minghui Jian, Ying Xu, and Binhai Zhu, *Protein structure-structure alignment with discrete Fréchet distance*, Proc. 5th Asia-Pacific Bioinform. Conf., 2007, pp. 131–141.

[KHM+98] 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** (1998), no. 05, 573–594.

[KKS05] Man-Soon Kim, Sang-Wook Kim, and Miyoung Shin, *Optimization of subsequence matching under time warping in time-series databases*, SAC '05: Proceedings of the 2005 ACM symposium on Applied computing (New York, NY, USA), ACM Press, 2005, pp. 581–586.

[Knu98] Donald E. Knuth, *The art of computer programming, volume 3: (2nd ed.) sorting and searching*, Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[KP99] Eamonn J. Keogh and Michael J. Pazzani, *Scaling up dynamic time warping to massive datasets*, Springer, 1999, pp. 1–11.

[MDH06] A. Mascret, T. Devogele, and A. Hénaff, *Coastline matching process based on the discrete Fréchet distance*, Proceedings of the 12th International Symposium on Spatial Data Handling (SDH) (Vienna, Austria), Springer-Verlag, 2006, pp. 383–400.

[Meg83] Nimrod Megiddo, *Applying parallel computation algorithms in the design of serial algorithms*, J. ACM **30** (1983), no. 4, 852–865.

[MP99] Mario E. Munich and Pietro Perona, *Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification*, In Proceedings of 7th International Conference on Computer Vision, 1999, pp. 108–115.

[OV02] René Van Oostrum and Remco C. Veltkamp, *Parametric search made practical*, SoCG: 18th Symposium on Computational Geometry, ACM Press, 2002, pp. 1–9.

[SGHS08] 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 and Language Processing **16** (2008), 1138–1151.

[SKB07] E. Sriraghavendra, Karthik K., and C. Bhattacharyya, *Fréchet distance based approach for searching online handwritten documents*, Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 01 (Washington, DC, USA), ICDAR '07, IEEE Computer Society, 2007, pp. 461–465.

[Str69] Volker Strassen, *Gaussian elimination is not optimal*, Numerische Mathematik **13** (1969), no. 4, 354–356.

[SW12] Jessica Sherette and Carola Wenk, *Partial matching between surfaces using Fréchet distance*, Algorithm Theory - SWAT 2012 (FedorV. Fomin and Petteri Kaski, eds.), Lecture Notes in Computer Science, vol. 7357, Springer Berlin Heidelberg, 2012, pp. 13–23.

[Vas11] Virginia Vassilevska Williams, *Breaking the Coppersmith-Winograd barrier*, 2011.

[WS06] Carola Wenk and Randall Salas, *Addressing the need for map-matching speed: Localizing global curve-matching algorithms*, In SSDBM, 2006, pp. 379–388.