GEODESIC FRÉCHET AND HAUSDORFF DISTANCE INSIDE A SIMPLE POLYGON

ATLAS F. COOK IV AND CAROLA WENK

Abstract. We unveil an alluring alternative to parametric search that applies to both the non-geodesic and geodesic Fréchet optimization problems in the plane. This randomized approach is based on a variant of red-blue intersections and is appealing due to its elegance and practical efficiency when compared to parametric search.

The frontiers of knowledge are expanded by our debut of the first algorithm for the geodesic Fréchet decision problem between two polygonal curves A and B inside a simple bounding polygon P. The geodesic decision problem is asymptotically almost as fast as its non-geodesic sibling and requires $O(N^2 \log k)$ time and $O(k+N)$ space after $O(k)$ preprocessing, where N is the larger of the complexities of A and B and k is the complexity of P. The culmination of our work is a randomized solution to the geodesic Fréchet optimization problem that runs in $O(k + (N^2 \log kN) \log N)$ expected time and $O(k+N^2)$ space. This run time is within a logarithmic factor of being asymptotically equivalent to the run time of the non-geodesic Fréchet optimization problem [3].

The algorithm for the geodesic Fréchet decision problem rests on a foundation of several key properties. We prove that a geodesic cell for polygonal curves inside a simple polygon P has at most one free region and that this region is monotone. This allows reachability information to be propagated through a cell in constant time once its boundaries are known. We also show how to compute a cell's boundaries in $O(\log k)$ time after preprocessing P in $O(k)$ time and space.

Other interesting and related results are that the geodesic Hausdorff distance between point sets inside a simple polygon P can be computed in $O((k+N)\log(k+N))$ time and $O(k+N)$ space. The approach relies on geodesic Voronoi diagrams and geodesic distance queries inside P. The geodesic Hausdorff distance for line segments inside P can be found in $O(k+N^2 \log k)$ time and $O(k+N)$ space.

1. Introduction

This section reviews the approach commonly used to compute the *non-geodesic* Fréchet distance, discusses related work, and outlines the rest of the paper. This information serves as essential background for computing the geodesic Fréchet distance.

1.1. Fréchet distance background.

The Fréchet distance is a similarity metric that returns a value indicating how similar two curves are to each other. Applications of the Fréchet distance include map matching [20] and shape similarity.

The Fréchet distance is commonly illustrated by a person walking a dog on a leash [3]. The person walks forward on one curve, and the dog walks forward on the

Date: August 18, 2007.

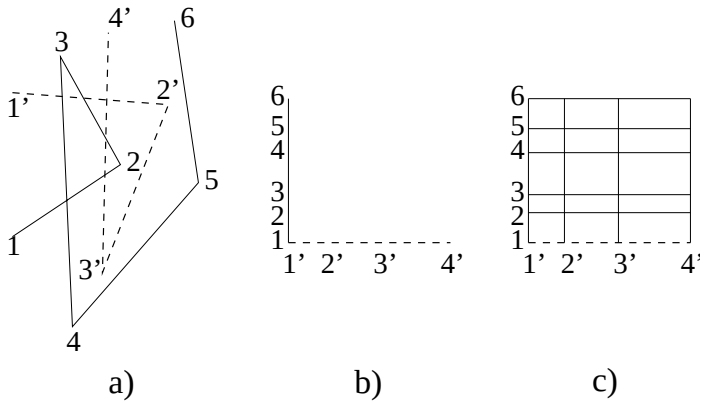


FIGURE 1. Two polygonal curves (a) are mapped onto the axes of a two-dimensional graph (b). For polygonal curves, the graph is subdivided into cells based on the positions of the curve vertices (c).

other curve. As the person and dog move along their respective curves, a leash is maintained to keep track of the separation between them. The maximum separation attained during the walk defines the required length of the leash.

The Fréchet distance is the length of the *shortest* leash that makes it possible for the person and dog to walk from beginning to end on their respective curves without breaking the leash. The leash's length is a measure of how similar the two curves are to each other. Short leashes mean the curves are similar; long leashes mean the curves are different. See section 2.2 for a formal definition of the Fréchet distance.

To compute the Fréchet distance, a way of representing all possible person and dog walks is needed. Alt and Godau's standard representation [3] maps the two curves (e.g., Figure 1a) onto the axes of a two-dimensional graph (e.g., Figure 1b). Parametrizing these curves into the range $[0, 1]$ forces the graph to be defined in a unit square. For polygonal curves, the graph is partitioned into cells by cutting the graph at every position where a curve has a vertex. This partitioned graph is called the *free space diagram* and is illustrated in Figure 1c.

The free space diagram represents all possible person and dog walks along their respective curves. At the beginning of the walk, the person and dog are positioned at the start of their curves. This position occurs at the bottom-left corner of the free space diagram. At the end of the walk, the person and dog are positioned at the end of their curves, and this occurs at the upper-right corner of the free space diagram. All walks relevant to the Fréchet distance are paths in the free space diagram from the bottom-left corner to the upper-right corner. If the person and dog are only allowed to move forward on their curves (i.e., not backward), then only monotone paths in the free space diagram should be considered.

The free space diagram provides a means of solving a subproblem of the Fréchet distance called the *decision problem*. The decision problem assumes a leash of length ε is given. True is returned if a leash of length ε is long enough to permit a monotone person and dog walk from start to finish. Otherwise, false is returned.

Notice that any point (p, d) in the free space diagram is associated with a point p on the person curve and a point d on the dog curve. Consequently, the point (p, d) can be mapped to the distance $d(p, d)$ between p and d, where this distance must be either $\leq \varepsilon$ or $> \varepsilon$.

To solve the decision problem, distances in the free space diagram are categorized. *Free space* consists of all points (p, d) in the free space diagram with $d(p, d) \leq \varepsilon$. Intuitively, points in the free space are associated with positions during the walk where the person and dog are close together. *Constrained space* consists of all points in the free space diagram with $d(p, d) > \varepsilon$. Intuitively, points in the constrained space are associated with positions during the walk where the person and dog are far apart.

The Fréchet decision problem returns true only when a path exists that satisfies two conditions. First, the path must be monotone and travel from the bottom-left corner to the the upper-right corner of the free space diagram. Second, the path must only travel through free space. We decide if such a path exists (for polygonal curves) by subdividing the free space diagram into cells and computing each cell's free space. Dynamic programming is used to propagate reachability information on a cell-by-cell basis.

After solving the decision problem, the idea of binary search allows converging to the shortest leash length $\varepsilon*$ such that the decision problem is still true. This length $\varepsilon*$ is the Fréchet distance and is the solution to the Fréchet optimization problem. To guarantee converging to the exact value of $\varepsilon*$ in a continuous domain, parametric search (see [2] and [3]) is often used in lieu of binary search.

1.2. **Related Work.** Most previous work assumes an obstacle-free environment where the leash connecting the person to the dog has its length defined by an L_p metric. In [3] the Fréchet distance between polygonal curves A and B is computed in arbitrary dimensions for obstacle-free environments in $O(N^2 \log N)$ time, where N is the larger of the complexities of A and B. Variations of the Fréchet distance have allowed the curves to be simple polygons [7] or piecewise smooth [18] instead of polygonal. Fréchet distance has also been used successfully in the practical domain of map matching [20]. All these works assume a leash length that is defined by an L_p metric.

This paper's contribution is to measure the leash length by its geodesic distance inside a simple polygon P (instead of by its L_p distance). To our knowledge, there are only two other works that employ such a leash. One is a workshop article [15] that computes the Fréchet distance for polygonal curves A and B on the surface of a convex polyhedron, but their method requires $O(N^3k^4 \log(kN))$ time. The other paper [9] applies the Fréchet distance to morphing by considering the polygonal curves A and B to be obstacles that the leash must go around. Their method works in $O(N^2 \log^2 N)$ time but only applies when A and B both lie on the boundary of the simple polygon P. Our work can handle both this case and more general cases. We consider P as the only obstacle, and the curves are allowed to occur at arbitrary positions inside P.

1.3. **Outline.** A core idea of this paper is that the free space in a geodesic cell is monotone. We show how to quickly compute a cell boundary and how to propagate reachability through a cell in constant time. This is sufficient to solve the geodesic Fréchet decision problem. To solve the geodesic Fréchet optimization problem, we

replace the standard parametric search approach by a novel and asymptotically faster (in the expected case) randomized algorithm that is based on red-blue intersection counting. It is notable that the randomized algorithm also applies to the non-geodesic Fréchet optimization problem in the plane.

In section 2, the hourglasses and funnels of Guibas et al. [10] are discussed. These structures represent shortest paths inside a simple polygon and are used to prove that any horizontal or vertical line segment in a geodesic cell has at most one connected set of free space values. We also show how to count and report certain types of red-blue intersections. Section 3 extends the results on hourglasses and funnels to prove that a geodesic cell has at most one free region. This region must be monotone, and reachability information can be propagated through this region in constant time once the cell boundaries are known.

Section 4 shows how to compute the boundaries of a geodesic cell, the geodesic Fréchet decision problem and the geodesic Fréchet optimization problem. The decision problem can be solved in $O(N^2 \log k)$ time after preprocessing. The main result of this paper is that the geodesic Fréchet distance between two polygonal curves inside a simple bounding polygon can be computed in $O(k + (N^2 \log kN) \log N)$ expected time and $O(k + N^3 \log kN)$ worst-case time, where N is the larger of the complexities of A and B and k is the complexity of the simple polygon. This expected run time is almost a quadratic factor in k faster than the straightforward approach, similar to [9], of partitioning each cell into $O(k^2)$ subcells. Briefly, these subcells are simple combinatorial regions based on *pairs* of hourglass intervals. Section 5 shows how to compute the geodesic Hausdorff distance for sets of points or sets of line segments inside a simple polygon.

2. Preliminaries

To compute the geodesic Fréchet distance for two polygonal curves A and B inside a simple polygon P, a few concepts need to be defined. Sections 2.1 and 2.2 introduce notation and definitions. In section 2.3, the hourglasses and funnels of [10] are described. Sections 2.4 and 2.5 show that funnels have a simple structure. Section 2.6 introduces a distance function for an hourglass that also has a simple structure. Section 2.7 shows how to perform red-blue intersection counting and reporting. Such intersections are theoretically interesting and will also have practical implications for solving the geodesic Fréchet optimization problem.

2.1. **Notation.** Let k be the complexity of a simple polygon P that contains the polygonal curves A and B in its interior. A *geodesic* is a path that avoids all obstacles and cannot be shortened by slight perturbations [16]. Let $\pi(a, b)$ denote the geodesic inside P between two points a and b. The *geodesic distance* $d(a, b)$ is the length of a shortest path between a and b that avoids all obstacles, where length is measured by L_2 distance.

Let $\downarrow$, $\uparrow$, and $\downarrow\uparrow$ denote decreasing, increasing, and decreasing then increasing functions, respectively. For example, "H is $\downarrow\uparrow$-bitone" means that H is a bitone function that is first monotone decreasing then monotone increasing.

2.2. **Definitions.** The Fréchet distance is formally defined as

$$\delta_F(A, B) = \inf_{f,g:[0,1]\to[0,1]} \sup_{t\in[0,1]} d(\ A(f(t)), B(g(t))\ )$$

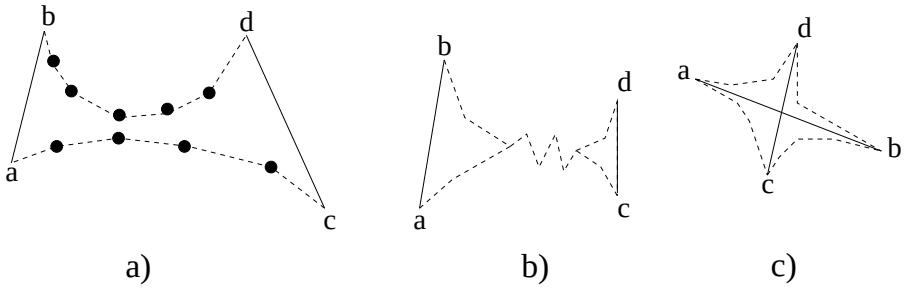


FIGURE 2. a) An open hourglass with marked hourglass vertices, b) a closed hourglass, and c) an intersecting hourglass.

where f and g range over continuous non-decreasing reparametrizations. Intuitively, a free space cell C is defined by two line segments $\overline{ab} \in A$ and $\overline{cd} \in B$. More formally, suppose the polygonal curves are defined as $A : [0, m] \to V$ and $B : [0, n] \to V$, where V is a Euclidean vector space in the plane and m and n are, respectively, the number of line segments defining A and B. Cell $C_{ij} = [i-1, i] \times [j-1, j]$ for $1 \leq i \leq m$ and $1 \leq j \leq n$ (see [3]).

2.3. **Funnels and Hourglasses.** All geodesics in a free space cell C can be described by either the funnel or hourglass structure of [10]. A funnel describes all shortest paths between a point and a line segment, so it represents a horizontal (or vertical) line segment in a free space cell. An hourglass describes all shortest paths between two line segments and represents an entire free space cell.

Let the *funnel* $\mathcal{F}_{p,\overline{cd}}$ represent all shortest paths between an apex point p and a line segment $\overline{cd}$. $\mathcal{F}_{p,\overline{cd}}$ is the region bounded by the line segment $\overline{cd}$ and the shortest path chains $\pi(p, c)$ and $\pi(p, d)$. That is, $\mathcal{F}_{p,\overline{cd}} = \overline{cd} \cup \pi(p, c) \cup \pi(p, d)$. The shortest path chains $\pi(p, c)$ and $\pi(p, d)$ are "outward convex" by [10]; in other words, the convex hulls of $\pi(p, c)$ and $\pi(p, d)$ lie outside $\mathcal{F}_{p,\overline{cd}}$.

There are three types of hourglasses: open, closed, and intersecting. An *open hourglass* is defined by non-crossing $\overline{ab}$ and $\overline{cd}$ and two disjoint shortest path chains $\pi(a, c)$ and $\pi(b, d)$. A *closed hourglass* is an open hourglass but with a collapsed interior: $\pi(a, c)$ and $\pi(b, d)$ share a common polygonal path that is traversed by all shortest paths between $\overline{ab}$ and $\overline{cd}$. An *intersecting hourglass* is defined when $\overline{ab}$ and $\overline{cd}$ cross and defines four shortest path chains $\pi(a, c)$, $\pi(a, d)$, $\pi(b, c)$, and $\pi(b, d)$. Open, closed, and intersecting hourglasses are illustrated in Figures 2a, 2b, and 2c.

In general, any type of hourglass $\mathcal{H}_{\overline{ab},\overline{cd}}$ can be represented as the region bounded by shortest path chains and line segments. That is, $\mathcal{H}_{\overline{ab},\overline{cd}} = \pi(a, c) \cup \pi(a, d) \cup \pi(b, c) \cup \pi(b, d) \cup \overline{ab} \cup \overline{cd}$.

We have seen that any geodesic cell C can be described by either an open hourglass, a closed hourglass, or an intersecting hourglass. All three of these structures have $O(k)$ vertices, where k is the complexity of the simple polygon P. This follows because all shortest paths (e.g., $\pi(a, c)$) inside a simple polygon P are acyclic, polygonal, and only have corners at vertices of P [12].

2.4. **Intervals.** Any geodesic $\pi(p, q)$ in P between a fixed point p and any point $q \in \overline{cd}$ can be described by a funnel $\mathcal{F}_{p,\overline{cd}}$ with outward convex chains $\pi(p, c)$,

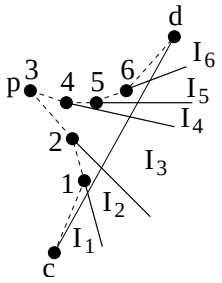


FIGURE 3. A funnel can be partitioned into $O(k)$ intervals such that chain vertex j defines interval I_j.

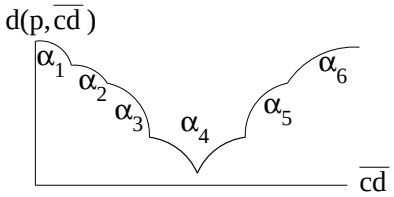


FIGURE 4. The geodesic distance function $F_{p,\,\overline{cd}}$ is piecewise hyperbolic.

$\pi(p, d)$. By extending all line segments on these polygonal chains into lines and intersecting these lines with $\overline{cd}$, a partition of $\overline{cd}$ into $O(k)$ intervals $I_1,\ I_2, ...,\ I_R$ is obtained. Shortest paths from p to any point $q \in I_j$ are combinatorially the same for $1 \leq j \leq R$. See Figure 3.

All shortest paths from p to any point $q \in I_j$ are polygonal and have the form $p,\ p_i,\ p_{i+1}, ...,\ j,\ q$ where $p_i, ...,\ j$ are the funnel chain vertices that the path visits. Let L be the length of the path $p,\ p_i,\ p_{i+1}, ...,\ j$. The length of a shortest path from p to q is $L + d(j, q)$, where $d(j, q)$ is equal to the L_2 distance between j and q. Varying q along I_j yields the distance function $d(j, I_j) + L$. This function is a hyperbolic arc α_j since $d(j, I_j)$ equals the L_2 distance from a point to a line segment and L is a constant. α_j achieves its minimum distance at either an endpoint of I_j or the perpendicular from j to I_j.

Since $d(p, I_j)$ is a hyperbolic arc, the distance function $F_{p,\,\overline{cd}}$ from p to the entire line segment $\overline{cd}$ is piecewise hyperbolic. In section 2.5 we will see that $F_{p,\,\overline{cd}}$ is bitone, so at most one of the hyperbolic arcs is bitone; all other arcs are monotone.

2.5. **Funnel Bitonicity.** Any horizontal (or vertical) line segment inside a geodesic free space cell has a distance function $F_{p,\,\overline{cd}} : [c, d] \to \mathbb{R}$ with $F_{p,\,\overline{cd}}(q) = d(p, q)$. In section 2.4, we saw that $F_{p,\,\overline{cd}}$ is piecewise hyperbolic, and this behavior is illustrated in Figure 4.

Lemma 1. *$F_{p,\,\overline{cd}}$ is $\downarrow\uparrow$-bitone.*

Proof. Without loss of generality, assume that $\overline{cd}$ is vertical. If we examine the *slopes* of the line segments defining the funnel chains in order from c to p along $\pi(c, p)$ and continuing from p to d along $\pi(p, d)$, we see that the sequence of these slopes is monotone. The two chains separately have monotone slopes due to their

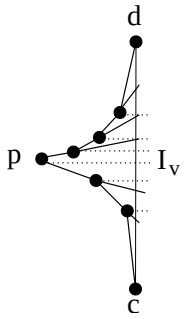


FIGURE 5. $F_{p,\ \overline{cd}}$ is $\downarrow\uparrow$-bitone because at most one interval I_v can be bitone. Dotted line segments are perpendiculars from each chain vertex to $\overline{cd}$.

convexity. The slopes where the two chains meet at apex p are also monotone since the two chains never cross each other.

Partition $\overline{cd}$ into $O(k)$ intervals $I_1,\ I_2, ...,\ I_R$ as in section 2.4. Each interval I_j for $1 \leq j \leq R$ is defined by two rays R_{j-1} and R_j that originate at chain vertex j and intersect $\overline{cd}$. Let $\alpha_j \in F_{p,\ \overline{cd}}$ be the hyperbolic arc for I_j.

α_j is bitone if and only if the perpendicular $\perp_j$ from j to the line ζ supporting $\overline{cd}$ lies strictly in the interior of I_v. Otherwise, α_j is monotone. Let the slope of $\perp_j$ be μ, and note that μ is constant over all chain vertices. Since I_j is defined by two rays $R_{j-1},\ R_j$ from j to ζ, $\perp_j$ will only intersect ζ in I_j when the slope μ lies between the slopes of R_{j-1} and R_j. Since the ray slopes are monotone through the intervals $I_{1...R}$, at most one bitone arc α_v for $1 \leq v \leq R$ exists. Hence, at most one arc of $F_{p,\ \overline{cd}}$ is bitone; the rest are monotone.

Suppose $\perp_v$ lies in the interior of I_v so that α_v is $\downarrow\uparrow$-bitone. By the monotonicity of the rays defining the intervals, $\alpha_{1...(v-1)}$ is $\downarrow$-monotone and $\alpha_{(v+1)...R}$ is $\uparrow$-monotone. If $\perp_v$ lies on the common boundary of I_{v-1} and I_v, then $\alpha_{1...(v-1)}$ is $\downarrow$-monotone and $\alpha_{v...R}$ is $\uparrow$-monotone. Hence, $F_{p,\ \overline{cd}}$ is $\downarrow\uparrow$-bitone. See Figure 5.

□

Corollary 1. *Any horizontal (or vertical) line segment in a free space cell has at most one connected set of free space values.*

Proof. A horizontal (or vertical) line segment in a geodesic free space cell has a distance function $F_{p,\ \overline{cd}}$. Free space consists of all values less than or equal to a given distance ε. Since Lemma 1 ensures that $F_{p,\ \overline{cd}}$ is $\downarrow\uparrow$-bitone, $F_{p,\ \overline{cd}}$ has at most one connected set of free space values.

□

2.6. **Hourglass Bitonicity.** This section introduces a $\downarrow\uparrow$-bitone distance function $H_{\overline{ab},\ \overline{cd}}$ for the hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$. $H_{\overline{ab},\ \overline{cd}}$ will be useful in section 3.1 for analyzing the structure of a geodesic free space cell.

Consider the hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$. Let the *shortest* distance from a to any point on $\overline{cd}$ occur at the point $M_a \in \overline{cd}$. Define M_b similarly.

As p is varied from a to b, the *minimum* distance from p to $\overline{cd}$ traces out a function $H_{\overline{ab},\ \overline{cd}} : [a, b] \to \mathbb{R}$ with $H_{\overline{ab},\ \overline{cd}}(p) = \min_{q \in [c,d]} d(p, q)$. See Figure 6.

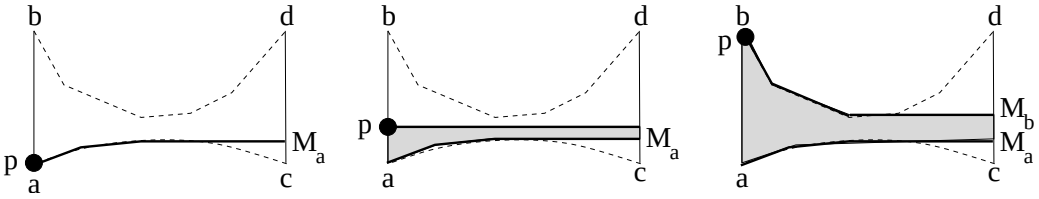


FIGURE 6. The shortest distance from p to any point on $\overline{cd}$ defines $H_{\overline{ab},\ \overline{M_aM_b}}$ as p is varied from a to b.

Lemma 2. *$H_{\overline{ab},\ \overline{cd}}$ is $\downarrow\uparrow$-bitone.*

Proof. By optimal substructure, shortest paths from p to $\overline{cd}$ will not cross as p is varied from a to b, so $H_{\overline{ab},\ \overline{cd}}$ and $H_{\overline{ab},\ \overline{M_aM_b}}$ are identical functions. The task is to show that $H_{\overline{ab},\ \overline{M_aM_b}}$ is $\downarrow\uparrow$-bitone regardless of whether the hourglass $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is open, closed, or intersecting (cf. section 2.3).

Suppose that $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is an open hourglass. Without loss of generality, assume that $\overline{M_aM_b}$ is vertical. Let $\mu_{\overline{ab}}$ be the slope of $\overline{ab}$. Let S_a and S_b be the points where horizontals from M_a and M_b intersect $\overline{ab}$. S_a and S_b will exist for any *open* hourglass $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ where $M_a \neq M_b$ because the position of the minimal distance from p to $\overline{cd}$ always occurs at either c, d, or a perpendicular to the interior of $\overline{cd}$ by [16].[1]

$\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ can be split into three parts: two funnels $\mathcal{F}_{\overline{aS_a},\ M_a}$, $\mathcal{F}_{\overline{S_bb},\ M_b}$, and an L_2-section L that lies in-between the two funnels[2]. Let $F_{\overline{aS_a},\ M_a}$, $F_{\overline{S_bb},\ M_b}$, and F_L denote distance functions for these three structures so that $H_{\overline{ab},\ \overline{M_aM_b}}$ has the form $F_{\overline{aS_a},\ M_a} \circ F_L \circ F_{\overline{S_bb},\ M_b}$, where $\circ$ denotes concatenation.

At most one bitone arc α_v defines $H_{\overline{ab},\ \overline{M_aM_b}}$ (cf. section 2.5).[3] This follows from the proof of Lemma 1 because the line segment slopes on the chains form a monotone sequence from a to M_a along $\pi(a, M_a)$ and continuing from M_b to b along $\pi(M_b, b)$.

If $\alpha_v \in F_{\overline{aS_a},\ M_a}$ as illustrated in Figure 7a, then clearly the slope $\mu_{\overline{ab}} < 0$, so $F_{\overline{aS_a},\ M_a}$ is $\downarrow\uparrow$-bitone and F_L and $F_{\overline{S_bb},\ M_b}$ are $\uparrow$-monotone. When $\alpha_v \in F_{\overline{S_bb},\ M_b}$, Figure 7b shows that $\mu_{\overline{ab}} > 0$, so both $F_{\overline{aS_a},\ M_a}$ and F_L are $\downarrow$-monotone and $F_{\overline{S_bb},\ M_b}$ is $\downarrow\uparrow$-bitone.[4] If α_v is part of F_L, then $\overline{ab}$ and $\overline{cd}$ are parallel, so $F_{\overline{aS_a},\ M_a}$ is $\downarrow$-monotone, F_L is constant, and $F_{\overline{S_bb},\ M_b}$ is $\uparrow$-monotone (see Figure 7c). Hence, $H_{\overline{ab},\ \overline{cd}}$ is always $\downarrow\uparrow$-bitone for any open hourglass.

Suppose that $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is a closed hourglass. *All* shortest paths from $p \in \overline{ab}$ to $\overline{cd}$ will end at the same point M_a as shown in Figure 8a. Hence, $H_{\overline{ab},\ \overline{M_aM_b}}$ equals $F_{\overline{ab},\ M_a}$ and is $\downarrow\uparrow$-bitone by Lemma 1.

When $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ is an intersecting hourglass, $\overline{ab}$ and $\overline{cd}$ will cross at the point ι as illustrated in Figure 8b. $H_{\overline{ab},\ \overline{M_aM_b}}$ has the form $H_{\overline{a\iota},\ \overline{M_a\iota}} \circ H_{\overline{\iota b},\ \overline{\iota M_b}}$, where

[1] If $M_a = M_b$, then $H_{\overline{ab},\ \overline{M_aM_b}}$ is trivially $\downarrow\uparrow$-bitone because it equals $F_{\overline{ab},\ M_a}$.

[2] Notice that the funnel $\mathcal{F}_{\overline{aS_a},\ M_a}$ uses the *second* subscript for the apex. This emphasizes that the apex occurs on $\overline{cd}$ instead of on $\overline{ab}$.

[3] If *no* bitone arc helps define $H_{\overline{ab},\ \overline{M_aM_b}}$, then clearly $H_{\overline{ab},\ \overline{M_aM_b}}$ is monotone.

[4] If $\mu_{\overline{ab}} = 0$, then $M_a = M_b$, and $H_{\overline{ab},\ \overline{cd}}$ is $\downarrow\uparrow$-bitone because it equals $F_{\overline{ab},\ M_a}$.

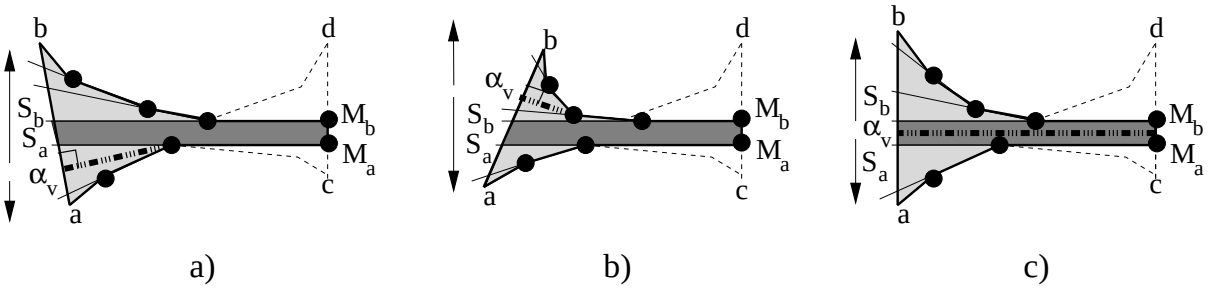


FIGURE 7. An *open* hourglass $\mathcal{H}_{\overline{ab},\ \overline{M_aM_b}}$ has at most one bitone arc α_v. Regardless of the position of α_v, $H_{\overline{ab},\ \overline{cd}}$ is $\downarrow$-monotone for all arcs from a to α_v and $\uparrow$-monotone for all arcs from α_v to b as indicated by the arrows in the diagrams. $\mathcal{F}_{\overline{aS_a},\ M_a}$ and $\mathcal{F}_{\overline{S_bb},\ M_b}$ are lightly shaded; L is heavily shaded.

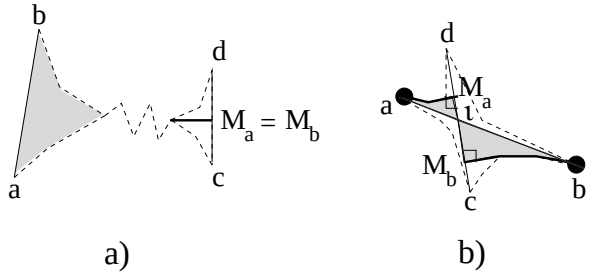


FIGURE 8. a) A *closed* hourglass always has $M_a = M_b$. b) An *intersecting* hourglass can be split into two (shaded) open hourglasses.

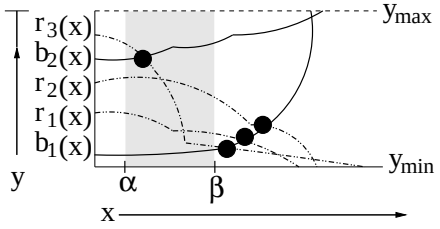
$H_{\overline{a\iota},\ \overline{M_a\iota}}$ and $H_{\overline{\iota b},\ \overline{\iota M_b}}$ are distance functions for *open* hourglasses. By the above arguments on open hourglasses, $H_{\overline{a\iota},\ \overline{M_a\iota}}$ and $H_{\overline{\iota b},\ \overline{\iota M_b}}$ are each (at-worst) $\downarrow\uparrow$-bitone. Consider varying p from a to ι. $d(a, M_a)$ is positive and $d(\iota, \iota) = 0$. Hence, $H_{\overline{a\iota},\ \overline{M_a\iota}}$ is actually $\downarrow$-monotone. Varying p from ι to b is similar: $d(\iota, \iota) = 0$ and $d(b, M_b)$ is positive, so $H_{\overline{\iota b},\ \overline{\iota M_b}}$ is $\uparrow$-monotone. Therefore, $H_{\overline{ab},\ \overline{cd}}$ is $\downarrow\uparrow$-bitone.

□

2.7. **Red-Blue Intersections.** This section shows how to efficiently count and report a certain type of red-blue intersections in the plane in an arbitrary interval $\alpha \leq x \leq \beta$. This problem is interesting both from theoretical and applied stances and will prove useful in section 4.3.1 for the Fréchet optimization problem.

Let $R = \{r_1(x), r_2(x), ..., r_m(x)\}$ be a set of m "red" curves in the plane such that each red curve is monotone *decreasing* and has $O(k)$ complexity. Let $B = \{b_1(x), b_2(x), ..., b_n(x)\}$ be a set of n "blue" curves in the plane where each blue curve is monotone *increasing* and has $O(k)$ complexity. Let $V(k)$ be the time to compute the *value* of any red or blue curve at a given x-position.

Let $I(k)$ be the time to find the *intersection* of any $r_i(x)$ and $b_j(x)$. Observe that any monotone decreasing curve $r_i(x) \in R$ intersects a monotone increasing curve $b_j(x) \in B$ in at most one contiguous sequence of points (as a consequence



Index	L_α	$r_i(\alpha)$ ->n	L_β	$r_i(\beta)$->n
5	$r_3(x)$	2	$b_2(x)$	
4	$b_2(x)$		$r_2(x)$	1
3	$r_2(x)$	1	$r_1(x)$	1
2	$r_1(x)$	1	$r_3(x)$	1
1	$b_1(x)$		$b_1(x)$	

FIGURE 9. Intersection counting calculates that $r_3(x)$ intersects two blue curves for $x \geq \alpha$ but only intersects one blue curve for $x \geq \beta$. Subtracting these quantities and accounting for $x = \beta$ intersections reveals that $r_3(x)$ must have one intersection in the interval $\alpha \leq x \leq \beta$.

of the monotonicities). For counting purposes, this sequence is considered to be a single intersection.

A few assumptions are required for the below counting and reporting algorithms. All red and blue curves should have values within an arbitrary range $y_{\min} \leq y \leq y_{\max}$ such that $\min(r_i(x)) = y_{\min}$ for all $1 \leq i \leq m$ and $\max(b_j(x)) = y_{\max}$ for all $1 \leq j \leq n$. In addition, the (open) interval $\alpha < x < \beta$ must contain no left endpoint of any red or blue curve.

Theorem 1. *The number of intersections in the interval $\alpha \leq x \leq \beta$ between every red curve $r_i(x) \in R$ and all $b_j(x) \in B$ can be* counted *in $O(N(V(k) + \log N))$ total time, where $N = \max(m, n)$.*

Proof. A key idea is that if $r_i(p) \geq b_j(p)$ at $x = p$, then the curves $r_i(x)$ and $b_j(x)$ will intersect for some $x \geq p$. This follows because $\min(r_i(x)) \leq \max(b_j(x))$.

Intersections in the interval $\alpha \leq x \leq \beta$ can be counted by taking "snapshots" of the curve positions at the endpoints α and β. The α-snapshot is found by computing the values of all red and blue curves at α in $O(N * V(k))$ time and sorting these values to create the list L_α in $O(N \log N)$ time. The list L_β can be found similarly in $O(N(V(k) + \log N))$ additional time.

If a curve is defined entirely outside the interval $\alpha \leq x \leq \beta$, then it can be safely ignored. If a curve's right endpoint lies strictly inside the interval, then this right endpoint can conceptually be extended horizontally to β. This extension will only create false red-blue intersections when the left endpoint of a $b_j(x)$ curve appears at $x = \beta$ such that $b_j(\beta) = y_{\max}$, and this special case can easily be handled without increasing the time bounds.

For counting queries, the sorted list L_α is preprocessed in $O(N)$ time by one linear scan over L_α. For each $r_i(\alpha) \in L_\alpha$, let $r_i(\alpha)$->n be the number of $b_j(\alpha)$ such that $r_i(\alpha) > b_j(\alpha)$. Define $r_i(\beta)$->n similarly. In essence, each red curve $r_i(x)$ keeps track of how many blue curves $b_j(x)$ lie below it because these curves will intersect for some $x \geq \alpha$. Preprocessing L_β yields red-blue intersections for $x \geq \beta$. For counting purposes, the number of $x \geq \alpha$ intersections minus the number of $x \geq \beta$ intersections yields the number of intersections in $\alpha \leq x < \beta$. Since it is a simple matter to compute the intersections at $x = \beta$ from the sorted list L_β, all intersections of $r_i(x)$ with B for $\alpha \leq x \leq \beta$ can be counted in $O(N(V(k) + \log N))$ time. See Figure 9.

□

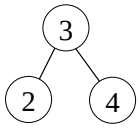
Step	Current Curve	Action	T_R
1	$r_3(x)$	Insert 2	
2	$b_2(x)$	Query 5: $r_3(x)$ intersects $b_2(x)$.	
3	$r_2(x)$	Insert 4	
4	$r_1(x)$	Insert 3	
5	$b_1(x)$	Query 1: No new intersections.	

FIGURE 10. In the situation depicted by Figure 9, intersection reporting finds that $r_3(x)$ intersects $b_2(x)$ in the interval $\alpha \leq x \leq \beta$.

Theorem 2. *The intersections in the interval $\alpha \leq x \leq \beta$ between every red curve $r_i(x) \in R$ and all $b_j(x) \in B$ can be* reported *in $O(N(V(k) + \log N + I(k)) + K)$ total time, where K is the total number of intersections reported.*

Proof. The goal is to compute for every $r_i(x)$ a list of all $b_j(x)$ such that $r_i(x) \cap b_j(x)$ for $\alpha \leq x \leq \beta$. Testing if two curves are equal at $x = \alpha$ or $x = \beta$ is straightforward, so we do not mention this case further. An intersection occurs for $\alpha < x < \beta$ exactly when $r_i(\alpha) > b_j(\alpha)$ and $r_i(\beta) < b_j(\beta)$ are both true. Intuitively, the first condition means that $r_i(x)$ must intersect $b_j(x)$ for some $x > \alpha$. The second condition means that there is no intersection for $x \geq \beta$. This implies that $r_i(x) \cap b_j(x)$ for $\alpha < x < \beta$.

The sorted lists L_α, L_β are together sufficient to report all red-blue intersection pairs. To extract these pairs efficiently, a balanced binary search tree T_R is incrementally constructed from red curve indices. Each blue curve is handled by querying T_R.

Let $I_\beta(r_i(x))$ denote the index of $r_i(x)$ in the list L_β. Similarly define $I_\beta(b_i(x))$. Begin with $T_R = \varnothing$. March through L_α in decreasing order (i.e., top-to-bottom in Figure 9). Process each $r_i(x)$ by inserting $I_\beta(r_i(x))$ into T_R in $O(\log N)$ time. For each $b_j(x)$, query T_R with $I_\beta(b_j(x))$. All $r_i(x)$ with indices in T_R less than this query index will intersect $b_j(x)$. See Figure 10.

□

3. Geodesic Cell Properties

All results in this section are for polygonal curves inside a simple polygon. In section 3.1, we show that a geodesic free space cell has at most one free region and that this region is monotone. Section 3.2 extends this result to show that reachability can be propagated through a cell in constant time once its boundaries are known.

3.1. Cell Free Space.

Consider a geodesic free space cell C for polygonal curves A and B inside a simple polygon. The goal of this section is to show that C has at most one free region and that this region is x and y monotone.

Recall that a free space cell C is defined by two line segments: $\overline{ab} \in A$ and $\overline{cd} \in B$. A horizontal (or vertical) line segment in C has a distance function $F_{p,\ \overline{cd}}$ between a fixed point $p \in \overline{ab}$ and the line segment $\overline{cd}$.

Lemma 3. *For any ε, C has at most one free space region R. R must be monotone and connected.*

Proof. A simple polygon is x-monotone if any vertical line intersects it in at most one connected interval. A polygon is y-monotone if any horizontal line intersects it in at most one connected interval. By Corollary 1, any free space region R in C must be x and y monotone. Next, it is proven that all free space points are connected so that C has at most one free space region.

Let $\varepsilon > 0$ be given. Take any two points $(p_1, q_1), (p_2, q_2)$ in the free space, i.e., $d(p_1, q_1) \leq \varepsilon$ and $d(p_2, q_2) \leq \varepsilon$. We need to show that they are connected in the free space. For this, first move vertically from (p_1, q_1) to the minimum point on its vertical. Do the same for (p_2, q_2). By Lemma 1, this movement causes the distance to decrease monotonically. By Lemma 2, any two minimum points are connected by a $\downarrow\uparrow$-bitone distance function $H_{\overline{ab},\ \overline{cd}}$ (cf. section 2.6), but as the starting points are $\leq \varepsilon$ all points on this constructed path are $\leq \varepsilon$.

□

3.2. **Cell Reachability.** This section proves that given C's boundaries, it is possible to propagate reachability information through C in constant time. In other words, the space inside C's boundaries is not required to compute the Fréchet distance.

Reachability information is crucial to solving the Fréchet decision problem. Recall from section 1 that in order to solve the Fréchet decision problem we need to know if a monotone path exists through the free space diagram from the bottom-left corner to the upper-right corner. Reachability information is a way of locally encoding on a cell-by-cell basis those free space points that are reachable by a monotone path from the bottom-left corner of the free space diagram. Let C_L, C_T, C_R, C_B be the left, top, right, and bottom boundaries of C, respectively.

Lemma 4. *Given the reachable points for C_L and C_B plus the free space points on C_R and C_T, reachability information can be propagated to C_R and C_T in constant time.*

Proof. Lemma 1 ensures that C_L and C_B each have at most one connected set of reachable points. If some point $p \in C_L$ is reachable, then all free space on C_T is reachable. This is true because C's free region R is both connected and monotone. Consequently, there must be some path $\pi(p, l)$ from $p \in C_L$ to C_T's leftmost free space point l. $\pi(p, l)$ is monotone because one can follow the monotone free space boundary between these two points. Symmetrically, if any point on C_B is reachable, then all free space on C_R is reachable. See Figure 11a.

If no point on C_L is reachable, then the only possible monotone path to C_T must originate at C_B. Imagine shooting a vertical ray V upward from C_B's leftmost reachable point. Let V intersect C_T at the point q. If q is a free space point, then q is the leftmost reachable point on C_T, and all free space on C_T to the right of q is reachable. If q is a constrained space point, then C_T's free interval either lies completely left of q or completely right of q. The former case is completely unreachable from C_B. The latter case is completely reachable since one can follow

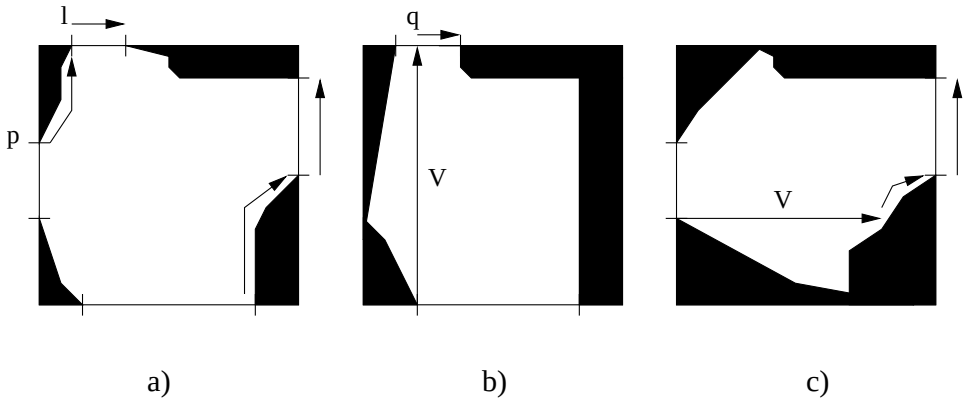


FIGURE 11. Propagating reachability information through cell C takes constant time once C's boundaries are known.

V until it hits constrained space and then follow the monotone free space boundary to the leftmost free space point on C_T. Propagating reachability from C_L to C_R is a symmetric case. See Figures 11b and 11c.

□

4. Geodesic Fréchet Algorithm

The goal of this section is to compute the exact geodesic Fréchet distance δ_F between two polygonal curves inside a simple polygon. Section 4.1 shows how to compute one cell's boundaries in $O(\log k)$ time, and section 4.2 uses this result to solve the geodesic Fréchet decision problem. Sections 4.3, 4.4, and 4.5 use the decision problem and red-blue intersection counting to solve the Fréchet optimization problem. This approach is novel in that it is a practical alternative to parametric search for both the geodesic and non-geodesic Fréchet optimization problems in the plane.

4.1. **Computing one cell's boundaries in $O(\log k)$ time.** A cell boundary is a horizontal (or vertical) line segment in a free space cell. This boundary can be associated with a funnel $\mathcal{F}_{p,\overline{cd}}$ with a distance function $F_{p,\ \overline{cd}}$ that is $\downarrow\uparrow$-bitone (cf. Lemma 1). Given $\varepsilon \geq 0$, computing the free space on a cell boundary requires finding the (at most two) intersections t_1, t_2 of $F_{p,\ \overline{cd}}$ and $y = \varepsilon$.

$F_{p,\ \overline{cd}}$ is defined by $O(k)$ arcs $\alpha_{1\ldots R}$ as shown in section 2.4. Any arc α_j for $1 \leq j \leq R$ is defined by both a point and a line segment. The point is a chain vertex $j \in \pi(p,c) \cup \pi(p,d)$ (excluding endpoints c, d). The line segment is an interval $I_j \in \overline{cd}$. α_j can be computed in constant time once j and I_j are known because α_j represents the L_2 distance between the point j and the line segment I_j. See Figure 12a.

Suppose the chains $\pi(p,c)$ and $\pi(p,d)$ are known. A straightforward way to compute the free space on a cell boundary is to sequentially compute all of the arcs $\alpha_{1\ldots R}$ as follows. For each chain vertex j, compute I_j by extending the two chain line segments adjacent to j into lines and intersecting them with $\overline{cd}$. α_j can then easily be constructed from j and I_j. Repeating this process for all chain vertices suffices to construct $\alpha_{1\ldots R}$ in $O(k)$ time. See Figure 12b. Once all the arcs are

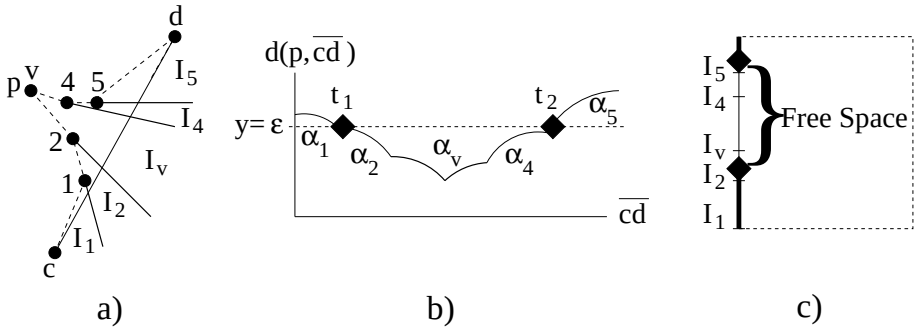


FIGURE 12. a) A funnel $\mathcal{F}_{p,\,\overline{cd}}$ is associated with a cell boundary. b) The arcs $\alpha_{1\ldots R}$ define $F_{p,\,\overline{cd}}$. c) Once the (at most two) intersections t_1, t_2 of $F_{p,\,\overline{cd}}$ with the line $y = \varepsilon$ have been found, the free space on a cell boundary is immediately known.

known, each arc can be intersected with the line $y = \varepsilon$ in constant time. These intersections are sufficient to define the free space on a cell boundary as illustrated in Figure 12c.

To improve the run time of the above approach from $O(k)$ to $O(\log k)$ time, realize that it is not necessary to explicitly construct *all* of the arcs of $F_{p,\,\overline{cd}}$. The arcs themselves are unimportant. Only the intersections t_1 and t_2 of $\alpha_{1\ldots R}$ with $y = \varepsilon$ are required, and a binary search can find these intersections by examining only $O(\log k)$ arcs. To perform this search, the chains $\pi(p, c)$ and $\pi(p, d)$ must be available in $O(\log k)$ time, and the chain data structure must support logarithmic searches over the chain vertices.

The chains $\pi(p, c)$ and $\pi(p, d)$ can be constructed in $O(\log k)$ time (after $O(k)$ preprocessing) by the algorithms of Guibas and Hershberger [11], [13]. These algorithms represent a chain by a binary search tree $\mathcal{T}$ with $O(\log k)$ height.

Even though $\mathcal{T}$ can have $O(k)$ complexity, it can be constructed in only $O(\log k)$ time through clever use of preprocessing structures [11]. A critical property shown in [13, p. 232] is that two adjacent trees of height $h(\mathcal{T}_1)$ and $h(\mathcal{T}_2)$ can be concatenated (i.e., merged) into a new tree with height $h(\mathcal{T}_3) \leq \max(h(\mathcal{T}_1),\ h(\mathcal{T}_2)) + 1$. The preprocessing step of [11] triangulates the simple polygon P and computes constant-size trees for each triangle. These trees are repeatedly merged in bottom-up fashion to create a "balanced hierarchical decomposition" of P [11]. Each of these precomputed trees is the result of $O(\log k)$ merge operations, so the height of any precomputed tree is $O(\log k)$.

Queries are handled by concatenating at most a logarithmic number of precomputed trees together [11, p. 56]. Since each concatenation increases the height of the tree by at most one, the final tree $\mathcal{T}$ will have $O(\log k)$ height. The query time to construct $\mathcal{T}$ is $O(\log k)$ because [11, p. 61-2] ensures that during the query a significant number of concatenations occur on trees of small height.

To find $\pi(p, c)$, perform a query on the points p and c. The query creates a binary search tree $\mathcal{T}_c$ that represents $\pi(p, c)$ in $O(\log k)$ time. The construction of $\mathcal{T}_d$ for $\pi(p, d)$ is similar.

Binary searches on $\mathcal{T}_c$ and $\mathcal{T}_d$ can be performed to find the intersections t_1, t_2 if arcs are available to guide the search. Since nodes in $\mathcal{T}_c$ and $\mathcal{T}_d$ can be directly associated with edges but not with arcs [13, p. 233], a conversion is needed.

Converting a node into an arc is possible in constant time. An arc α_v is defined by a chain vertex v and a line segment I_v (cf. section 2.4). Let an arbitrary tree node n represent the edge $\overline{vw}$, where v and w are adjacent chain vertices. To compute I_v, the two chain line segments adjacent to v must be found. Clearly, $\overline{vw}$ is one of these edges. The other edge $\overline{uv}$ is the predecessor of $\overline{vw}$.

Lemma 5. *For any edge $\overline{vw}$ in $\mathcal{T}_c$ or $\mathcal{T}_d$, its predecessor $\overline{uv}$ can be found in constant time.*

Proof. To find a node's predecessor in constant time, the algorithms of [11] and [13] can be extended so that every node n has a pointer n_ρ to its immediate predecessor. To support constant time updates to n_ρ, n also needs a pointer n_l to the largest valued node in the tree rooted at n. Adding these pointers requires modifying the preprocessing step of [11] so that these pointers are initialized for all nodes in a constant-sized chain. In addition to this base case, the concatenation process is also updated.

The concatenation process merges trees $\mathcal{T}_1$ and $\mathcal{T}_2$ into $\mathcal{T}_3$. It creates a new root node r and modifies nodes on two root-to-leaf paths: one in $\mathcal{T}_1$ and one in $\mathcal{T}_2$ [13, p. 233-4]. Begin by updating n_l for each of these nodes. n_l either points to n itself or to o_l, where o is n's right child. Each of these updates takes constant time if performed in bottom-up fashion.

Updating n_ρ on a root-to-leaf path should also be performed bottom-up. If an arbitrary node n (e.g., node 4 in Figure 13) has a left child m, then n_ρ is simply m_l. If n has no left child (e.g., node 5 in Figure 13), then n_ρ is the first ancestor $\hat{n}$ such that n is a right descendant of $\hat{n}$. This ancestor can be maintained during the traversal so that $\hat{n}$ is always available in constant time. If no such ancestor exists, then n is the smallest node in the tree and correctly has no predecessor.

There are two final predecessors to update to complete the concatenation. Let r be the new root and q be r's left child. Also let w be the smallest node in the tree rooted at r's right child (e.g., node 9 in Figure 13). r_ρ equals q_l, and $w_\rho$equals r. All pointer updates can be performed in the $O(h(\mathcal{T}_1) + h(\mathcal{T}_2))$ time that is allowed per concatenation [13, p. 232].

□

Lemma 6. *Both the minimum value of $F_{p,\,\overline{cd}}$ and the (at most two) intersections t_1, t_2 of $F_{p,\,\overline{cd}}$ with the line $y = \varepsilon$ can be found for any $\varepsilon \geq 0$ in $O(\log k)$ time (after preprocessing).*

Proof. Binary searches on $\mathcal{T}_c$ and $\mathcal{T}_d$ to find t_1 and t_2 can start by searching for the bitone arc α_v in both trees. α_v contains the minimum value of $F_{p,\,\overline{cd}}$, and there is at most one bitone arc in $\mathcal{T}_c \cup \mathcal{T}_d$ because the trees together represent the chains of the funnel $\mathcal{F}_{p,\overline{cd}}$ (cf. Lemma 1).

Following the binary search paradigm, separately traverse each tree in search of the bitone arc α_v. At the root node r use r_ρ to construct α_r in constant time (cf. Lemma 5). If α_r is bitone, then α_v has been found, and this step is complete. If α_r is $\uparrow$-monotone, then recurse on the left child of r. Otherwise, recurse on the right child of r.

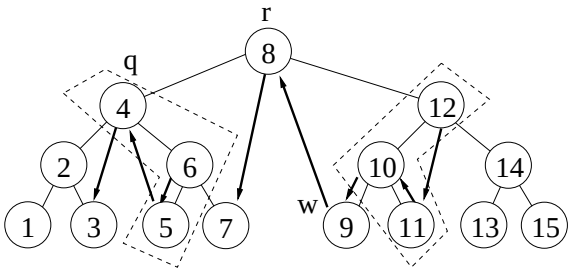


FIGURE 13. Concatenation involves updating predecessors in three locations: the new root r, the smallest valued node w in r's right subtree, and two (boxed) root-to-leaf paths. Although nodes are associated with edges, they are shown with integer values to simplify the predecessor relationship that is indicated by the arrows.

The tree that does not contain α_v has a monotone sequence of arcs. The other tree has at most two monotone arc sequences: one on each side of α_v. The intersections t_1 and t_2 of these monotone arc sequences with $y = \varepsilon$ can be found in logarithmic time. Simply test the endpoints of the (monotone) current arc α_r in constant time. If the endpoints of α_r define a range that contains ε, then find the exact intersection of α_r with $y = \varepsilon$. Otherwise, recurse on the appropriate child of r.

□

Once the intersections t_1 and t_2 are known, they must be mapped onto the cell boundary. Once this mapping is known, it is trivial to define the free space for any cell boundary.

Lemma 7. *Let t be an arbitrary point on any arc $\alpha_r \in F_{p,\,\overline{cd}}$ that is defined by the interval I_r. The position of t on the cell boundary can be found in $O(1)$ time.*

Proof. Let the endpoints of I_r define the (closed) range $[i, j]$ on the cell boundary. Compute the arc-length of α_r from its left endpoint to t and divide this value by the total arc-length of α_r. This quotient supplies the position in the range $[i, j]$ where t occurs. Since all of these steps take constant time, the position of t on the cell boundary is available in constant time.

□

Corollary 2. *The free space on all four boundaries of a single cell can be found in $O(\log k)$ time.*

Proof. This result follows immediately from Lemmas 6 and 7.

□

4.2. **Geodesic Fréchet Decision Problem.**

Theorem 3. *After a one-time preprocessing step of $O(k)$ time* [11]*, the geodesic Fréchet decision problem for polygonal curves A and B inside a simple polygon P can be solved for any $\varepsilon \geq 0$ in $O(N^2 \log k)$ time and $O(k + N)$ space.*

Proof. There are $O(N^2)$ cells in the free space diagram. Compute all cell boundaries in $O(N^2 \log k)$ time (cf. Corollary 2) and propagate reachability information through all cells in $O(N^2)$ time (cf. Lemma 4). Return true if the upper right corner of the free space diagram is reachable. Return false otherwise.

The space bounds follow because the cells can be handled via dynamic programming such that only two rows need to reside in memory at any one time. These two rows require only $O(N)$ storage because only $O(1)$ space per cell is needed to define the cell boundaries. The $O(k)$ term comes from storing the preprocessing structures of [11] throughout the algorithm's execution.

□

4.3. **Geodesic Fréchet Optimization Problem.** Let $\varepsilon*$ be the minimum value of ε such that the Fréchet decision problem returns true. That is, $\varepsilon*$ equals the Fréchet distance δ_F. Parametric Search is a technique commonly used to find $\varepsilon*$ (see [3], [19], [2], and [8]).[5]. The typical approach to find $\varepsilon*$ is to sort all the cell boundary functions based on the unknown parameter $\varepsilon*$. The comparisons performed during the sort guarantee that the result of the decision problem is known for all "critical values" [3] that could potentially define $\varepsilon*$.

Previous sorting algorithms have operated on cell boundaries of constant complexity. The geodesic case is different because each cell boundary has $O(k)$ complexity. As a result, a straightforward parametric search based on sorting these values would require $O(kN^2 \log kN)$ time even when using Cole's [8] optimization.[6]

We present a randomized algorithm with expected run time $O(k+(N^2 \log kN) \log N)$ and worst-case run time $O(k+N^3 \log kN)$. This algorithm is an order of magnitude faster than parametric search in the expected case. Both algorithms involve cubic factors in the worst-case.

Each cell boundary has at most one free space interval. The upper boundary of this interval is a function $b_{ij}(\varepsilon)$, and the lower boundary of this interval is a function $a_{ij}(\varepsilon)$. See Figure 14a.

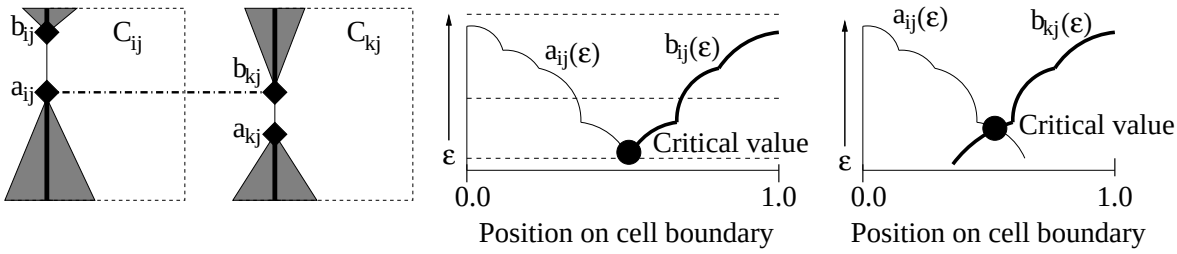
The seminal work of Alt and Godau [3] defines three types of critical values for the Fréchet distance. There are exactly two type (a) critical values associated with distances between the starting points of A and B and the ending points of A and B. Type (b) critical values occur $O(N^2)$ times when $a_{ij}(\varepsilon) = b_{ij}(\varepsilon)$. See Figure 14b.

Type (a) and (b) critical values occur $O(N^2)$ times and are easily handled in $O((N^2 \log k) \log N)$ time. This process involves computing values in $O(N^2 \log k)$ time, sorting in $O(N^2 \log N)$ time, and then running the decision problem $O(\log N)$ times. Each execution of the decision problem resolves half of the remaining critical values. Resolving the type (a) and (b) critical values as a first step will lead to an observation that simplifies the randomized algorithm on type (c) critical values.

Alt and Godau [3] show that type (c) critical values occur when the position of $a_{ij}(\varepsilon)$ in cell C_{ij} equals the position of $b_{kj}(\varepsilon)$ in cell C_{kj} in the free space diagram. See Figure 14a.

[5]An easier to implement alternative to parametric search is to run the decision problem once for every bit of accuracy that is desired. This approach runs in $O((N^2 \log k)B)$ time, where B is the desired number of bits of accuracy [19]. This approach requires only $O(k + N)$ space using row-based dynamic programming for the decision problem.

[6]A variation of the general sorting problem called the "nuts and bolts" problem (see [14]) is tantalizingly close to an acceptable $O(N^2 \log N)$ sort, but it is not solvable in the general case.



a) Free Space Diagram

b) Distance function with a type (b) critical value

c) Distance function with a type (c) critical value

FIGURE 14. There are $O(N^2)$ type (b) critical values and $O(N^3)$ type (c) critical values.

As ε increases, Figure 14b shows that $a_{ij}(\varepsilon)$ is $\downarrow$-monotone on the cell boundary and $b_{ij}(\varepsilon)$ is $\uparrow$-monotone. This follows from Lemma 1. As illustrated in Figure 14c, $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$ intersect at most once. This follows from the monotonicities and piecewise hyperbolic structures of $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$. Hence, there are $O(N^2)$ intersections of $a_{ij}(\varepsilon)$ and $b_{kj}(\varepsilon)$ in row j and a total of $O(N^3)$ type (c) critical values over all rows. There are also $O(N^2)$ intersections of $a_{ij}(\varepsilon)$ and $b_{ik}(\varepsilon)$ in *column* i and a total of $O(N^3)$ additional type (c) critical values over all columns.

Lemma 8. *The intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ can be found for any $\varepsilon \geq 0$ in $O(\log k)$ time after preprocessing.*

Proof. Using the approach of section 4.1, construct the binary search trees $\mathcal{T}_a$ and $\mathcal{T}_b$ in $O(\log k)$ time that are, respectively, associated with the monotone functions $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$. We show that a logarithmic search over $\mathcal{T}_a$ and $\mathcal{T}_b$ is sufficient to find the intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ or report that no intersection exists.

Start at the roots of both trees. In $O(1)$ time build the arc α_a for the current node in $\mathcal{T}_a$. Using the monotonicity of α_a, construct two axis-parallel rectangles r_{a1}, r_{a2} in constant time such that $r_{a1} \cup r_{a2}$ contains all potential coordinates for the other arcs in $\mathcal{T}_a$. Repeat this process for α_b, r_{b1}, and r_{b2}.

Figure 15 illustrates the general idea. In Figure 15a, it should be clear that neither α_b nor any arc in r_{b2} can be involved in an intersection because $\alpha_b \cup r_{b2}$ is disjoint from $r_{a1} \cup \alpha_a \cup r_{a2}$. Consequently, it is correct to move to the left child of the current node in $\mathcal{T}_b$ and update α_b. Figure 15b shows that in the next step, neither α_a nor any arc in r_{a1} is involved in an intersection. Consequently, move to the right child of the current node in $\mathcal{T}_a$ and update α_a. The third step in Figure 15c shows that α_b is the only arc in $\mathcal{T}_b$ that can intersect an arc in $\mathcal{T}_a$. Continuing the search on $\mathcal{T}_a$ is sufficient to find this intersection or determine that it does not exist.

Each iteration, an algorithm can either return the intersection of $\alpha_a \cap \alpha_b$ if it exists, report that there is no intersection, or update α_a or α_b and continue with the next iteration. We show next that at each step it is always possible to update either α_a or α_b.

Suppose that p is an endpoint of α_a. Let $\mathcal{A} = r_{a1} \cup p \cup r_{a2}$ and note that $\mathcal{A} \supset (r_{a1} \cup \alpha_a \cup r_{a2})$. Let $\mathcal{B} = r_{b1} \cup \alpha_b \cup r_{b2}$ for an arbitrary α_b. Several observations follow directly from the monotonicities of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$. If p is disjoint from $\mathcal{B}$,

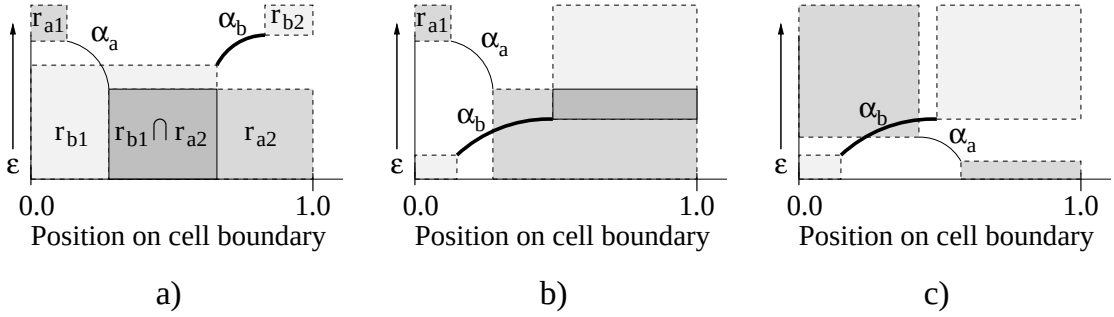


FIGURE 15. Example steps to find the intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$.

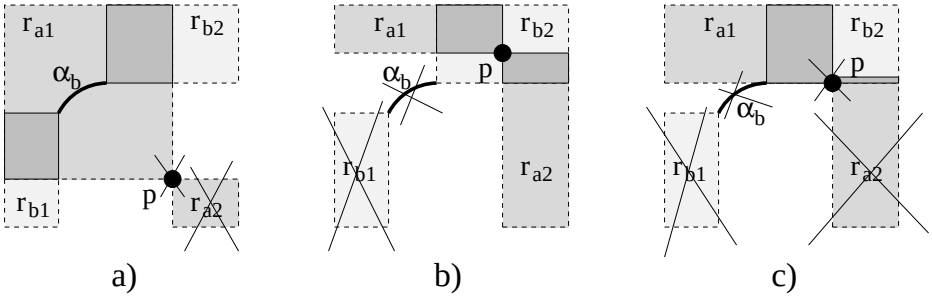


FIGURE 16. Intersecting $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ takes $O(\log k)$ time. Arcs that can be safely discarded are crossed out.

then $\mathcal{B}$ can intersect at most one of r_{a1}, r_{a2}. Hence, either $p \cup r_{a1}$ or $p \cup r_{a2}$ can be discarded (see Figure 16a). Similarly, if p lies strictly in the interior of $r_{b1} \cup r_{b2}$, then either $\alpha_b \cup r_{b1}$ or $\alpha_b \cup r_{b2}$ can be discarded (see Figure 16b). When p lies precisely on the boundary of r_{b1} or r_{b2}, then it is possible for $\mathcal{B}$ to intersect r_{a1}, p, and r_{a2} (see Figure 16c). However, since endpoints are shared by adjacent arcs, it is permissible to discard either $p \cup r_{a1}$ or $p \cup r_{a2}$ and also either $\alpha_b \cup r_{b1}$ or $\alpha_b \cup r_{b2}$. Hence, at each step it is always possible to update either α_a or α_b.

The run time follows because each step performs constant work on four rectangles and two arcs to determine how to update the arcs for the next step. Since the trees $\mathcal{T}_a$ and $\mathcal{T}_b$ have $O(\log k)$ height, the total number of steps is $O(\log k)$. Hence, an algorithm can find the intersection of $a_{ij}(\varepsilon)$ and $b_{kl}(\varepsilon)$ (or determine that no such intersection exists) in $O(\log k)$ time.

□

The below observations imply that Theorems 1 and 2 can be applied to count and report the number of type (c) critical values in the closed interval $[\alpha, \beta]$.

Observation 1. *Precomputing the type (a) and type (b) critical values of* [3] *shrinks the (closed) interval* $[\alpha, \beta]$ *containing* $\varepsilon*$ *such that no new* $a_{ij}(\varepsilon)$, $b_{kl}(\varepsilon)$ *appear in the open interval* (α, β) *when processing the type (c) critical values.*

Observation 2. *All* $a_{ij}(\varepsilon)$ *have minimum values at the bottom of the cell boundary. All* $b_{ij}(\varepsilon)$ *have maximum values at the top of the cell boundary. That is,* $\min(a_{ij}(\varepsilon)) = 0.0$ *and* $\max(b_{ij}(\varepsilon)) = 1.0$ *for all* $1 \leq i,\ j \leq N$.

4.3.1. *Randomized Algorithm.* The below randomized algorithm solves the geodesic Fréchet optimization problem. This algorithm is asymptotically faster than parametric search by an order of magnitude in the expected case and shares a cubic run time with parametric search in the worst-case.

(1) Precompute and sort all type (a) and type (b) critical values in $O(N^2 \log kN)$ time (cf. Lemma 6). Run the decision problem $O(\log N)$ times to resolve these values and shrink the $\varepsilon*$ interval down to $[\alpha, \beta]$ in $O((N^2 \log k) \log N)$ time.
(2) Let j represent an arbitrary row in the free space diagram. Count the number κ_j of type (c) critical values for each row j in the interval $[\alpha, \beta]$ using Theorem 1. Intersection counting requires $O(N \log kN)$ time per row for a total of $O(N^2 \log kN)$ time for all rows. Let C_j be the counting data structure for row j.
(3) To achieve a fast *expected* run time, use Quicksort's paradigm to pick a random intersection for each row.[7] To find a random intersection for row j, pick a random number between 1 and κ_j. Since every $a_{ij}(\varepsilon) \in C_j$ stores the number of intersections in which it is involved, a search through C_j can determine the particular $a_{Rj}(\varepsilon)$ that is involved in the randomly selected intersection. Once $a_{Rj}(\varepsilon)$ is known, its $O(N)$ intersections in $[\alpha, \beta]$ can be determined in $O(N \log k)$ time by testing all $b_{kj}(\varepsilon)$ that lie below $a_{Rj}(\varepsilon)$ in C_j's list L_α (cf. Lemma 8). The randomly selected intersection ϑ_j is then immediately available and can be stored for later use.[8]
(4) To achieve a fast *worst-case* run time, also pick the $a_{Mj}(\varepsilon)$ in each row that has the most intersections. Add all intersections in $[\alpha, \beta]$ that involve $a_{Mj}(\varepsilon)$ to a global pool P of unresolved critical values[9] and delete $a_{Mj}(\varepsilon)$ from any future consideration. If desired, the intersections for the randomly selected $a_{Rj}(\varepsilon)$ can also be added to P.
(5) $O(N^2)$ values are added to P each step after finding $O(N)$ intersections for each row. Sort all values in P, and find the median Ξ of these values. Also find the median Ψ of the $O(N)$ randomly selected ϑ_j in $O(N)$ time using the standard median algorithm mentioned in [14].
(6) Run the decision problem twice: once on Ξ; once on Ψ. This shrinks the interval $[\alpha, \beta]$ and halves the size of P. Repeat steps 2 through 6 until all *row*-based type (c) critical values have been resolved.
(7) Resolve all *column*-based type (c) critical values in the same spirit as steps 2 through 6.
(8) Return the smallest critical value that satisfies the decision problem (i.e., $\varepsilon*$) as the value of the geodesic Fréchet distance.

4.4. **Geodesic Fréchet Distance Run Time.**

Theorem 4. *The exact* geodesic *Fréchet distance between two polygonal curves A and B inside a simple bounding polygon P can be computed in* $O(k+(N^2 \log kN) \log N)$

[7] Picking a critical value at random is related to the distance selection problem [6] and is mentioned in [1], but to our knowledge, this alternative to parametric search has never been applied to the Fréchet distance.

[8] In practice, the *median* of the intersections is a better choice for ϑ_j.

[9] The idea of a global pool is similar to Cole's optimization for parametric search [8].

expected time and $O(k + N^3 \log kN)$ worst-case time, where N is the larger of the complexities of A and B and k is the complexity of P. $O(k+N^2)$ space is required.

Proof. Preprocess P once for shortest path queries in $O(k)$ time [11]. In the average case, each execution of the decision problem will essentially cut the total number of unresolved type (c) critical values in half. This follows from the well-known proof of Quicksort's expected run time. Consequently, the expected number of iterations of the algorithm is $O(\log N^3) = O(\log N)$.

In the worst-case, each of the $O(N)$ $a_{ij}(\varepsilon)$ in a row will be picked as $a_{Mj}(\varepsilon)$. Therefore, each row can require at most $O(N)$ iterations. Since *all* rows are processed each iteration, the entire algorithm requires at most $O(N)$ iterations for *row*-based critical values. By a similar argument, *column*-based critical values also require at most $O(N)$ iterations.

The size of the pool P is expressed by the recurrence $S(x) = \frac{S(x-1)+O(N^2)}{2}$, where x is the current step number, and $S(0) = 0$. Intuitively, each steps adds $O(N^2)$ values to P and then half the values in P are always resolved. It is not difficult to see that $S(x) \in O(N^2)$ for any step number x.

Each iteration of the algorithm requires intersection counting and intersection calculations for $O(N)$ rows (or columns) at a cost of $O(N^2 \log kN)$ time. In addition, the global pool P is sorted in $O(N^2 \log N)$ time, and the decision problem is executed in $O(N^2 \log k)$ time. Consequently, the expected run time is $O(k+(N^2 \log kN) \log N)$ and the worst-case run time is $O(k+N^3 \log kN)$ including $O(k)$ preprocessing time for geodesics.

The preprocessing step of [11] requires $O(k)$ space, and this space must remain allocated throughout the algorithm. $O(N^2)$ additional space is sufficient for the remaining steps.

□

4.5. **Non-Geodesic Fréchet Distance Run Time.** Although the exact non-geodesic Fréchet distance is normally computed in $O(N^2 \log N)$ time using parametric search (see [3]), the constant factors involved in parametric search can be enormous [8]. To mitigate these expensive constant factors, Oostrum and Veltkamp [19] have implemented a Quicksort-based parametric search algorithm.

To the best of our knowledge, the randomized algorithm in section 4.3.1 provides the first practical alternative to parametric search for solving the Fréchet optimization problem.

Theorem 5. *The exact* non-geodesic *Fréchet distance between two polygonal curves A and B in the plane can be computed in $O(N^2 \log^2 N)$ expected time, where N is the larger of the complexities of A and B. $O(N^2)$ space is required.*

Proof. The argument is very similar to the proof of Theorem 4. The main difference is that non-geodesic distances can be computed in $O(1)$ time (instead of the $O(\log k)$ time needed for geodesic distances).

□

5. Geodesic Hausdorff Distance

Hausdorff distance is a similarity metric commonly used to compare sets of points or sets of line segments. The directed Hausdorff distance can be formally defined as $\tilde{\delta}_H(A, B) = \sup_{a \in A} \inf_{b \in B} d(a, b)$, where A and B are sets and $d(a, b)$ is the geodesic

distance between a and b (see [3] and [5]). Intuitively, the Hausdorff distance finds for each $a \in A$ the distance to its nearest neighbor in B. The supremum of these nearest neighbor distances is $\tilde{\delta}_H(A,B)$. The undirected Hausdorff distance is the larger of the two directed distances: $\delta_H(A,B) = \max(\tilde{\delta}_H(A,B),\ \tilde{\delta}_H(B,A))$. Sections 5.1 and 5.2 show how to compute δ_H inside a simple polygon for sets of points or sets of line segments.

5.1. **Points.**

Theorem 6. *$\delta_H(A,B)$ for point sets A,B inside a simple polygon P can be computed in $O((k+N)\log(k+N))$ time and $O(k+N)$ space, where N is the larger of the complexities of A and B and k is the complexity of P.*

Proof. Precompute the geodesic Voronoi diagrams VD_A, VD_B for A and B inside P. These can be found in $O((k+N)\log(k+N))$ time and $O(k+N)$ space using the algorithm of [17]. Also preprocess P for shortest path queries in $O(k)$ time and space using the algorithm of [13].

For each point $a \in A$, find its nearest neighbor $a' \in B$ in $O(\log k)$ time via point location in VD_B and compute the geodesic distance $d(a,a')$ in $O(\log k)$ additional time using the algorithm of [13]. Return the maximum of these distances as the value of $\tilde{\delta}_H(A,B)$. Compute $\tilde{\delta}_H(B,A)$ is a similar manner.

Calculating $\tilde{\delta}_H(A,B)$ requires $O(\log k)$ time for each point in A; this is $O(N \log k)$ total time after preprocessing. Including preprocessing yields a run time of $O(k+N)\log(k+N))$. The space bounds are also dominated by the $O(k+N)$ preprocessing. $\tilde{\delta}_H(B,A)$ requires identical time and space bounds as does $\delta_H(A,B)$ since it is the larger of $\tilde{\delta}_H(A,B)$ and $\tilde{\delta}_H(B,A)$.

□

5.2. **Line Segments.** The directed Hausdorff distance $\tilde{\delta}_H(A,B)$ for sets of line segments A and B is computed by finding for each $a \in A$ the nearest neighbor point on any line segment in B to any point on a. The result is a set of nearest neighbor distances, and $\tilde{\delta}_H(A,B)$ is the supremum of these distances. It has been shown in [4] that by intersecting line segments with Voronoi edges the number of critical points that must be considered is $O(1)$ per line segment. However, no geodesic Voronoi diagram for line segments has been published to our knowledge, so the below algorithm essentially computes geodesic distances between all pairs $a \in A,\ b \in B$ of line segments.

Theorem 7. *$\delta_H(A,B)$ for sets of line segments A,B inside a simple polygon P can be computed in $O(k+N^2\log k)$ time and $O(k+N)$ space, where N is the larger of the complexities of A and B and k is the complexity of P.*

Proof. Consider first the simple case of computing $\tilde{\delta}_H(\overline{ab},\overline{cd})$ between two line segments. $\tilde{\delta}_H(\overline{ab},\overline{cd})$ is exactly the minimum value of $H_{\overline{ab},\ \overline{cd}}$, where $H_{\overline{ab},\ \overline{cd}}$ is a distance function defined in section 2.6 for the hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$.

The task is to find the minimum value of $H_{\overline{ab},\ \overline{cd}}$ for any type of hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$. For an intersecting hourglass, (cf. 2.3), clearly $\tilde{\delta}_H(\overline{ab},\overline{cd}) = 0$ since $\overline{ab}$ and $\overline{cd}$ intersect.

For a closed hourglass, $H_{\overline{ab},\ \overline{cd}}$ equals the $\downarrow\uparrow$-bitone distance function $F_{\overline{ab},\ M_a}$ (cf. section 2.6). The minimum value of $F_{\overline{ab},\ M_a}$ is available in $O(\log k)$ time by Lemma

6 once the position of M_a is known. $M_a \in \overline{cd}$ (cf. section 2.6) is the *position* on $\overline{cd}$ where the minimum distance in $F_{a,\ \overline{cd}}$ occurs. Therefore, it can also be found in $O(\log\ k)$ time by Lemma 6.

An open hourglass is easier to handle once it is split into three pieces defined by $M_a,\ M_b \in \overline{cd}$ as demonstrated in section 2.6. M_a can be found in $O(\log k)$ time as described above, and M_b can be found similarly.

For the *open* hourglass $\mathcal{H}_{\overline{ab},\ \overline{cd}}$, $H_{\overline{ab},\ \overline{cd}}$ is the concatenation of distance functions for two funnels and an L_2-section (cf. section 2.6). The minimum value for $H_{\overline{ab},\ \overline{cd}}$ can be found in $O(\log k)$ time by simply finding the minimum distance for both funnels and returning the smaller value. The L_2-section need not be considered since its distance function is monotone. This means that given any two line segments $\overline{ab}$ and $\overline{cd}$, $\tilde{\delta}_H(\overline{ab}, \overline{cd})$ can be computed in $O(\log k)$ time after preprocessing.

$\tilde{\delta}_H(A, B)$ can be computed for *sets* A, B as follows. For a single line segment $a \in A$ compute the minimum distance to every $b \in B$. This yields the distance to a's nearest neighbor in B in $O(N \log k)$ time. Repeating this step for every $a \in A$ and returning the supremum of all the nearest neighbor distances yields $\tilde{\delta}_H(A, B)$ in $O(N^2 \log k)$ time after $O(k)$ preprocessing (see [13]) for shortest paths.

Only $O(k)$ space is needed for preprocessing but clearly the simple polygon and sets A, B must be stored, so the space requirement is $O(k + N)$. $\tilde{\delta}_H(B, A)$ and $\delta_H(A, B)$ have identical time and space bounds.

□

6. Conclusion

To compute the geodesic Fréchet distance between two polygonal curves inside a simple polygon, we have proven that a geodesic cell has at most one free space region R and that R must be monotone. It follows from the monotonicity of R that reachability information can be propagated through a cell in constant time once the cell boundaries are known. By extending the shortest path algorithm of [11] and [13], the boundaries of a single cell can be computed in logarithmic time, and this approach leads to an efficient algorithm to solve the geodesic Fréchet decision problem.

A randomized algorithm based on counting red-blue intersections inside an interval $[\alpha, \beta]$ is used to solve the geodesic Fréchet optimization problem in lieu of the standard parametric search approach. The randomized algorithm is also a practical alternative to parametric search for the non-geodesic Fréchet optimization problem in the plane.

These results allow computing the geodesic Fréchet distance between two polygonal curves A and B inside a simple bounding polygon P in $O(k+(N^2 \log kN) \log N)$ expected time, where N is the larger of the complexities of A and B and k is the complexity of P. In the worst-case, both the randomized algorithm and parametric search include cubic terms. In the expected case, the randomized algorithm is an order of magnitude faster because a straightforward parametric search (even with Cole's [8] optimization) would need to sort $O(kN^2)$ values.

The beauty of the geodesic Fréchet decision problem is that cell boundaries can be computed in the same asymptotic time that it takes to compute a shortest path. By [13], the algorithm used to compute these shortest paths is optimal. Therefore, it is unlikely that a single cell's boundaries can be computed asymptotically faster

than we have shown. An attempt to cluster cells together to achieve a superior run time has potential but would likely lose the ability to perform logarithmic searches.

The geodesic Hausdorff distance for point sets inside a simple polygon can be computed in $O((k+N)\log(k+N))$ time and $O(k+N)$ space. The approach is based on geodesic Voronoi diagrams and geodesic distance queries. As we know of no published algorithm to create the geodesic Voronoi diagram for line segments, the geodesic Hausdorff distance for line segments is more difficult to compute. Our approach uses $O(k+N^2\log k)$ time and $O(k+N)$ space. The development of a geodesic Voronoi diagram for line segments would almost certainly improve this run time.

References

[1] Pankaj K. Agarwal and Micha Sharir. Efficient algorithms for geometric optimization. *ACM Comput. Surv.*, 30(4):412–458, 1998.

[2] Pankaj K. Agarwal, Micha Sharir, and Sivan Toledo. Applications of parametric searching in geometric optimization. volume 17, pages 292–318, Duluth, MN, USA, 1994. Academic Press, Inc.

[3] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. volume 5, pages 75–91, 1995.

[4] Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate matching of polygonal shapes (extended abstract). In *SCG '91: Proceedings of the seventh annual symposium on Computational geometry*, pages 186–193, New York, NY, USA, 1991. ACM Press.

[5] Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2003.

[6] Sergei Bespamyatnikh and Michael Segal. Selecting distances in arrangements of hyperplanes spanned by points. volume 2, pages 333–345, September 2004.

[7] Kevin Buchin, Maike Buchin, and Carola Wenk. Computing the Fréchet distance between simple polygons in polynomial time. In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 80–87, New York, NY, USA, 2006. ACM Press.

[8] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[9] 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. *Discrete & Computational Geometry*, 28(4):535–569, 2002.

[10] L Guibas, J Hershberger, D Leven, M Sharir, and R Tarjan. Linear time algorithms for visibility and shortest path problems inside simple polygons. pages 1–13, 1986.

[11] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *J. Comput. Syst. Sci.*, 39(2):126–152, 1989.

[12] Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert Endre Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987.

[13] John Hershberger. A new data structure for shortest path queries in a simple polygon. *Inf. Process. Lett.*, 38(5):231–235, 1991.

[14] János Komlós, Yuan Ma, and Endre Szemerédi. Matching nuts and bolts in o(n log n) time. In *SODA '96: Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 232–241, Philadelphia, PA, USA, 1996. Society for Industrial and Applied Mathematics.

[15] A Maheshwari and J Yi. On computing Fréchet distance of two paths on a convex polyhedron. *EWCG 2005*, pages 41–4, 2005.

[16] Joseph S. B. Mitchell, David M. Mount, and Christos H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16(4):647–668, 1987.

[17] Evanthia Papadopoulou and D. T. Lee. A new approach for the geodesic Voronoi diagram of points in a simple polygon and other restricted polygonal domains. *Algorithmica*, 20(4):319–352, 1998.

[18] Günter Rote. Computing the Fréchet distance between piecewise smooth curves. Technical Report ECG-TR-241108-01, May 2005.

[19] René van Oostrum and Remco C. Veltkamp. Parametric search made practical. In *SCG '02: Proceedings of the eighteenth annual symposium on Computational geometry*, pages 1–9, New York, NY, USA, 2002. ACM Press.

[20] Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: Localizing global curve-matching algorithms. In *18th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 379–388, 2006.