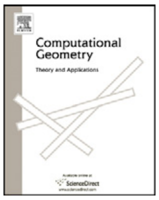
Computational Geometry 44 (2011) 110–120

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo



Fréchet distance with speed limits [☆]

Anil Maheshwari [*], Jörg-Rüdiger Sack, Kaveh Shahbaz, Hamid Zarrabi-Zadeh

School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada

A R T I C L E   I N F O

Article history:
Received 17 November 2009
Accepted 23 September 2010
Available online 25 September 2010
Communicated by W. Evans

Keywords:
Fréchet distance
Speed constraints

A B S T R A C T

In this paper, we introduce a new generalization of the well-known Fréchet distance between two polygonal curves, and provide an efficient algorithm for computing it. The classical Fréchet distance between two polygonal curves corresponds to the maximum distance between two point objects that traverse the curves with arbitrary non-negative speeds. Here, we consider a problem instance in which the speed of traversal along each segment of the curves is restricted to be within a specified range. We provide an efficient algorithm that decides in $O(n^2 \log n)$ time whether the Fréchet distance with speed limits between two polygonal curves is at most ε, where n is the number of segments in the curves, and $\varepsilon \geqslant 0$ is an input parameter. We then use our solution to this decision problem to find the exact Fréchet distance with speed limits in $O(n^2 \log^2 n)$ time.

© 2010 Elsevier B.V. All rights reserved.

1. Introduction

Fréchet distance [2] is a metric to measure the similarity of polygonal curves. It finds application in several problems, such as morphing [7], handwriting recognition [13], and protein structure alignment [8]. The Fréchet distance between two curves is often referred to as the "dog-leash distance" because it can be interpreted as the minimum-length leash required for a person to walk a dog, if the person and the dog, each travels from its respective starting position to its ending position, without ever letting go off the leash or backtracking. The length of the leash determines how similar the two curves are to each other: a short leash means the curves are similar, and a long leash means that the curves are different from each other.

Two problem instances naturally arise: decision and optimization. In the *decision problem*, one wants to decide whether two polygonal curves P and Q are within ε Fréchet distance from each other, i.e., if a leash of length ε suffices. In the *optimization problem*, one wants to determine the minimum such ε. In [2], Alt and Godau gave a quadratic-time algorithm for the decision problem, where n is the total number of segments in the curves. They also solved the corresponding optimization problem in $O(n^2 \log n)$ time.

In the classical problem, the speed of motion on the two polygonal curves is unbounded. Motivated by practical importance of similarity measures, we here consider a problem variant in which motion speeds are bounded, both from below and from above. More precisely, associated to each segment of the curves, there is a speed range that specifies the minimum and the maximum speed allowed for travelling along that segment. We say that a point object traverses a curve with permissible speed, if it traverses the polygonal curve from start to end so that the speed used on each segment falls within its permissible range.

[☆] Research supported by NSERC and SUN Microsystems.
[*] Corresponding author.
E-mail addresses: anil@scs.carleton.ca (A. Maheshwari), sack@scs.carleton.ca (J.-R. Sack), kshahbaz@scs.carleton.ca (K. Shahbaz), zarrabi@scs.carleton.ca (H. Zarrabi-Zadeh).

0925-7721/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.comgeo.2010.09.008

The decision version of the Fréchet distance problem with speed limits is formulated as follows: Let P and Q be two polygonal curves with minimum and maximum permissible speeds assigned to each segment of P and Q. For a given $\varepsilon \geqslant 0$, is there an assignment of speeds so that two point objects can traverse P and Q with permissible speed and, throughout the entire traversal, remain at distance at most ε from each other? The objective in the optimization problem is to find the smallest such ε.

In this paper, we present a new algorithm that solves the decision version of the Fréchet distance problem with speed limits in $O(n^2 \log n)$ time. Our main approach is to compute a free-space diagram similar to the one used in the standard Fréchet distance algorithm [2]. However, since the complexity of the free-space diagram in our problem is cubic – in contrast to the standard free-space diagram that has quadratic complexity – we use a "lazy computation" technique to avoid computing unneeded portions of the free space, and still be able to solve the decision problem correctly. Combined with a parametric search technique, we then use our algorithm for the decision problem to solve the optimization problem exactly in $O(n^2 \log^2 n)$ time.

Different variants of the Fréchet distance have been studied in the literature, including Fréchet distance for closed curves [2], Fréchet distance between two curves inside a simple polygon [5], Fréchet distance between two paths on a polyhedral surface [6,10], and the so-called homotopic Fréchet distance [3]. The Fréchet distance with speed limits we consider in this paper is a natural generalization of the classical Fréchet distance, and has potential applications in GIS, when the speed of moving objects is considered in addition to the geometric structure of the trajectories.

This paper is organized as follows. The problem is formally defined in the next section. In Section 3, we describe a simple algorithm that solves the decision problem in $O(n^3)$ time. In Section 4, we provide an improved algorithm for the decision problem that runs in $O(n^2 \log n)$ time. Section 5 describes how the optimization problem can be solved efficiently. Finally, we summarize in Section 6 and outline directions for future work.

2. Preliminaries

A *polygonal curve* in $\mathbb{R}^d$ is a continuous function $P : [0, n] \to \mathbb{R}^d$ with $n \in \mathbb{N}$, such that for each $i \in \{0, \dots, n-1\}$, the restriction of P to the interval $[i, i+1]$ is affine (i.e., forms a line segment). The integer n is called the *length* of P. Moreover, the sequence $P(0), \dots, P(n)$ represents the set of *vertices* of P. For each $i \in \{1, \dots, n\}$, we denote the line segment $P(i-1)P(i)$ by P_i.

Fréchet distance. A *monotone parametrization* of $[0, n]$ is a continuous non-decreasing function $\alpha : [0, 1] \to [0, n]$ with $\alpha(0) = 0$ and $\alpha(1) = n$. Given two polygonal curves P and Q of lengths n and m respectively, the *Fréchet distance* between P and Q is defined as

$$\delta_F(P, Q) = \inf_{\alpha, \beta} \max_{t \in [0,1]} d\big(P\big(\alpha(t)\big), Q\big(\beta(t)\big)\big),$$

where d is the Euclidean distance, and α and β range over all monotone parameterizations of $[0, n]$ and $[0, m]$, respectively.

Fréchet distance with speed limits. Consider two point objects $\mathcal{O}_P$ and $\mathcal{O}_Q$ that traverse P and Q respectively from start to the end. If we think of the parameter t in the parameterizations α and β as "time", then $P(\alpha(t))$ and $Q(\beta(t))$ specify the positions of $\mathcal{O}_P$ and $\mathcal{O}_Q$ on P and Q respectively at time t. The preimages of $\mathcal{O}_P$ and $\mathcal{O}_Q$ can be viewed as two point objects $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$ traversing $[0, n]$ and $[0, m]$, respectively, with their positions at time t being specified by $\alpha(t)$ and $\beta(t)$.

In the classical definition of Fréchet distance, the parameterizations α and β are arbitrary non-decreasing functions, meaning that $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$ (and therefore, $\mathcal{O}_P$ and $\mathcal{O}_Q$) can move with arbitrary speeds in the range $[0, \infty]$. In our variant of the Fréchet distance with speed limits, each segment S of the curves P and Q is assigned a pair of non-negative real numbers $(v_{\min}(S), v_{\max}(S))$ that specify the minimum and the maximum permissible speed for moving along S. The speed limits on each segment is independent of the limits of other segments. When $\mathcal{O}_P$ moves along a segment S with speed v, $\bar{\mathcal{O}}_P$ moves along the preimage of S (which is a unit segment) with speed $v/\|S\|$. Therefore, the speed limit $(v_{\min}(S), v_{\max}(S))$ on a segment S, forces a speed limit on the preimage of S bounded by the following two values:

$$\bar{v}_{\min}(S) = \frac{v_{\min}(S)}{\|S\|} \quad \text{and} \quad \bar{v}_{\max}(S) = \frac{v_{\max}(S)}{\|S\|}.$$

We define a *speed-constrained parametrization of P* to be a continuous surjective function $f : [0, T] \to [0, n]$ with $T > 0$ such that for any $i \in \{1, \dots, n\}$, the slope of f at all points $t \in [f^{-1}(i-1), f^{-1}(i)]$ is within $[\bar{v}_{\min}(P_i), \bar{v}_{\max}(P_i)]$. Here, we define the slope of a function f at a point t to be $\lim_{h \to 0^+} f(t+h)/h$, where h approaches 0 only from above (right). By this definition, if f is a continuous function, then the slope of f at any point t in its domain is well defined, even if f is not differentiable at t.

Given two polygonal curves P and Q of lengths n and m respectively with speed limits on their segments, the *speed-constrained Fréchet distance* between P and Q is defined as

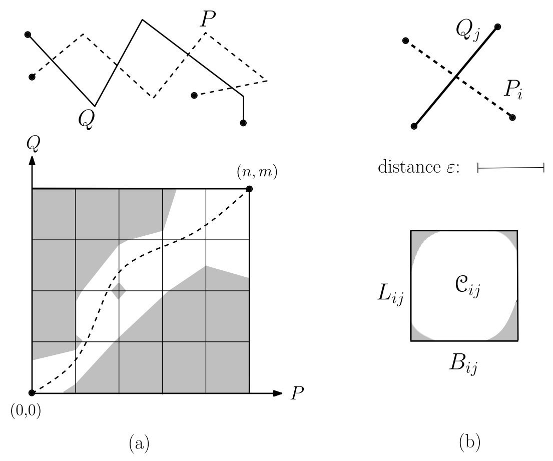


Fig. 1. (a) The free-space diagram for two polygonal curves P and Q; (b) two segments P_i and Q_j and their corresponding free space. The diagram was generated using a Java applet developed by S. Pelletier [12].

$$\delta_{\bar{F}}(P,Q)=\inf_{\alpha,\beta}\max_{t\in[0,T]} d\big(P\big(\alpha(t)\big),\,Q\big(\beta(t)\big)\big),$$

where $\alpha:[0,T]\to[0,n]$ ranges over all speed-constrained parameterizations of P and $\beta:[0,T]\to[0,m]$ ranges over all speed-constrained parameterizations of Q. Note that this new formulation of Fréchet distance is similar to the classical one, with the only difference that the parameterizations here are restricted to have limited slopes, reflecting the speed limits on the segments of the input polygonal curves.

Free-space diagram. Let $\mathcal{B}_{n\times m}=[0,n]\times[0,m]$ be an n by m rectangle in the plane. Each point $(s,t)\in\mathcal{B}_{n\times m}$ uniquely represents a pair of points $(P(s),Q(t))$ on the polygonal curves P and Q. We decompose $\mathcal{B}_{n\times m}$ into $n\cdot m$ unit grid cells $\mathcal{C}_{ij}=[i-1,i]\times[j-1,j]$ for $(i,j)\in\{1,\dots,n\}\times\{1,\dots,m\}$, where each cell $\mathcal{C}_{ij}$ corresponds to a segment P_i on P and a segment Q_j on Q. Given a parameter $\varepsilon\geqslant 0$, the *free space* $\mathcal{F}_\varepsilon$ is defined as

$$\mathcal{F}_\varepsilon=\big\{(s,t)\in\mathcal{B}_{n\times m}\;\big|\;d\big(P(s),Q(t)\big)\leqslant\varepsilon\big\}.$$

We call any point $p\in\mathcal{F}_\varepsilon$ a *feasible* point. An example of the free-space diagram for two curves P and Q is illustrated in Fig. 1(a). The free-space diagram was first used in [2] to find the standard Fréchet distance in near quadratic time.

Notations. We introduce some notation used throughout the paper. Each line segment bounding a cell in $\mathcal{B}_{n\times m}$ is called an *edge* of $\mathcal{B}_{n\times m}$. We denote by L_{ij} (resp., by B_{ij}) the left (resp., bottom) line segment bounding $\mathcal{C}_{ij}$. For a cell $\mathcal{C}_{ij}$, we define the *entry side* of $\mathcal{C}_{ij}$ to be $\mathrm{entry}(\mathcal{C}_{ij})=L_{ij}\cup B_{ij}$, and its *exit side* to be $\mathrm{exit}(\mathcal{C}_{ij})=B_{i,j+1}\cup L_{i+1,j}$. Throughout this paper, we process the cells in a *cell-wise* order, in which a cell $\mathcal{C}_{ij}$ precedes a cell $\mathcal{C}_{k\ell}$ if either $i<k$ or $i=k\ \&\ j<\ell$ (this corresponds to the row-wise order of the cells, from the first cell, $\mathcal{C}_{0,0}$, to the last cell, $\mathcal{C}_{nm}$).

For an easier manipulation of the points and intervals on the boundary of the cells, we define the following orders: Given two points p and q in the plane, we say that p is *before* q, and denote it by $p\prec q$, if either $p_x<q_x$ or $p_x=q_x\ \&\ p_y>q_y$. For an interval I of points in the plane, the *left endpoint* of I, denoted by $\mathrm{left}(I)$, is a point p such that $p\prec q$ for all $q\in I$, $q\neq p$. The *right endpoint* of I, denoted by $\mathrm{right}(I)$, is defined analogously. Given two intervals I_1 and I_2 in the plane, we say that I_1 is *before* I_2, and denote it by $I_1\prec I_2$, if $\mathrm{left}(I_1)\prec\mathrm{left}(I_2)$ and $\mathrm{right}(I_1)\prec\mathrm{right}(I_2)$. Note that $I_1\prec I_2$ implies that none of the intervals I_1 and I_2 can be properly contained in the other.

3. The decision problem

In this section, we provide an algorithm for solving the following decision problem: Given two polygonal curves P and Q of lengths n and m respectively ($n\geqslant m$) with speed limits on their segments, and a parameter $\varepsilon\geqslant 0$, decide whether $\delta_{\bar{F}}(P,Q)\leqslant\varepsilon$. We use a free-space diagram approach, similar to the one used in the standard Fréchet distance problem [2]. However, the complexity of the "reachable portion" on the cell boundaries is different in our problem; namely, each cell boundary in our problem has a complexity of $O(n^2)$, while in the original problem cell boundaries have $O(1)$ complexity. This calls for a more detailed construction of the free space.

Consider two point objects, $\mathcal{O}_P$ and $\mathcal{O}_Q$, traversing P and Q, with their preimages, $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$, traversing $[0,n]$ and $[0,m]$, respectively. When $\mathcal{O}_P$ and $\mathcal{O}_Q$ traverse P and Q from beginning to the end, the trajectories of $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$ on $[0,n]$ and $[0,m]$ specify a path $\mathcal{P}$ in $\mathcal{B}_{n\times m}$ from $(0,0)$ to (n,m). Suppose that $\mathcal{P}$ passes through a point $(s,t)\in\mathcal{C}_{ij}$. The slope of $\mathcal{P}$ at point (s,t) is equal to the ratio of the speed of $\bar{\mathcal{O}}_Q$ at point t to the speed of $\bar{\mathcal{O}}_P$ at point s. Therefore, the minimum slope at (s,t) is obtained when $\bar{\mathcal{O}}_Q$ moves with its minimum speed at point t, and $\bar{\mathcal{O}}_P$ moves with its maximum speed

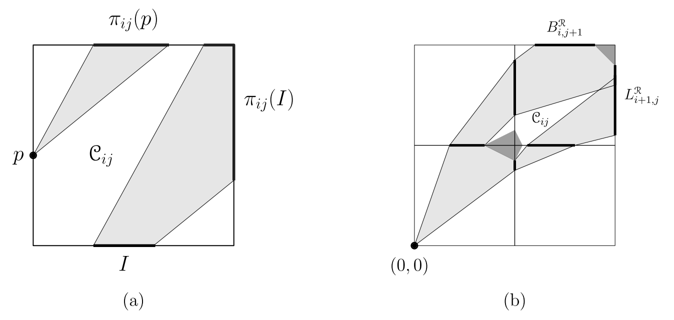


Fig. 2. (a) Projecting a point p and an interval I onto the exit side of $\mathcal{C}_{ij}$; (b) computing reachable intervals on the exit side of a cell $\mathcal{C}_{ij}$. Dark gray areas represent infeasible (obstacles) regions. Reachable intervals are shown with bold line segments.

at point s. Similarly, the maximum slope is obtained when $\bar{\mathcal{O}}_Q$ moves with its maximum speed, and $\bar{\mathcal{O}}_P$ moves with its minimum speed. We define

$$\text{minSlope}_{ij} = \frac{\bar{v}_{\min}(Q_j)}{\bar{v}_{\max}(P_i)} \quad \text{and} \quad \text{maxSlope}_{ij} = \frac{\bar{v}_{\max}(Q_j)}{\bar{v}_{\min}(P_i)},$$

where $\bar{v}_{\min}(\cdot)$ and $\bar{v}_{\max}(\cdot)$ are the speed limits for $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$ as defined in Section 2. Indeed, minSlope$_{ij}$ and maxSlope$_{ij}$ specify the minimum and the maximum "permissible" slopes for $\mathcal{P}$ at any point inside $\mathcal{C}_{ij}$. A path $\mathcal{P} \subset \mathcal{B}_{n\times m}$ is called *slope-constrained* if for any point $(s,t) \in \mathcal{P} \cap \mathcal{C}_{ij}$, the slope of $\mathcal{P}$ at (s,t) is within [minSlope$_{ij}$, maxSlope$_{ij}$]. A point $(s,t) \in \mathcal{F}_\varepsilon$ is called *reachable* if there is a slope-constrained path from $(0,0)$ to (s,t) in $\mathcal{F}_\varepsilon$.

Lemma 1. *$\delta_{\bar{F}}(P,Q) \leqslant \varepsilon$ iff (n,m) is reachable.*

The statement of the lemma is similar to the one used in [2]; however, since the setting is different, we have provided a proof in [9].

A simple algorithm. We now describe a simple algorithm for the decision problem. As a preprocessing step, the free space, $\mathcal{F}_\varepsilon$, is computed by the algorithm. Let $L^{\mathcal{F}}_{ij} = L_{ij} \cap \mathcal{F}_\varepsilon$ and $B^{\mathcal{F}}_{ij} = B_{ij} \cap \mathcal{F}_\varepsilon$. Since F_ε is convex within $\mathcal{C}_{ij}$ [2], each of $L^{\mathcal{F}}_{ij}$ and $B^{\mathcal{F}}_{ij}$ is a line segment. The preprocessing step therefore involves computing line segments $L^{\mathcal{F}}_{ij}$ and $B^{\mathcal{F}}_{ij}$ for all feasible pairs (i,j), which can be done in $O(n^2)$ time. We then compute the reachability information on the boundary of each cell. Let $L^{\mathcal{R}}_{ij}$ be the set of reachable points in L_{ij}, and $B^{\mathcal{R}}_{ij}$ be the set of reachable points in B_{ij}. We process the cells in the cell-wise order, from $\mathcal{C}_{0,0}$ to $\mathcal{C}_{nm}$, and at each cell $\mathcal{C}_{ij}$, we propagate the reachability information from the entry side of the cell to its exit side, using the following projection function. Given a point $p \in \text{entry}(\mathcal{C}_{ij})$, the *projection* of p onto the exit side of $\mathcal{C}_{ij}$ is defined as

$$\pi_{ij}(p) = \{q \in \text{exit}(\mathcal{C}_{ij}) \mid \text{the slope of } \overline{pq} \text{ is within } [\text{minSlope}_{ij}, \text{maxSlope}_{ij}]\}.$$

For a point set $S \subseteq \text{entry}(\mathcal{C}_{ij})$, we define $\pi_{ij}(S) = \bigcup_{p\in S} \pi_{ij}(p)$ (see Fig. 2(a)). To compute the set of reachable points on the exit side of a cell $\mathcal{C}_{ij}$, the algorithm first projects $L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}$ to the exit side of $\mathcal{C}_{ij}$, and takes its intersection with $\mathcal{F}_\varepsilon$. More precisely, the algorithm computes $L^{\mathcal{R}}_{i+1,j}$ and $B^{\mathcal{R}}_{i,j+1}$ from $L^{\mathcal{R}}_{ij}$, $B^{\mathcal{R}}_{ij}$, $L^{\mathcal{F}}_{i+1,j}$, and $B^{\mathcal{F}}_{i,j+1}$, using the following formula: $B^{\mathcal{R}}_{i,j+1} \cup L^{\mathcal{R}}_{i+1,j} = \pi_{ij}(L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}) \cap (B^{\mathcal{F}}_{i,j+1} \cup L^{\mathcal{F}}_{i+1,j})$ (see Fig. 2(b)). Details are provided in Algorithm 1.

Algorithm 1 Decision Algorithm

```
1:  Compute the free space, $\mathcal{F}_\varepsilon$
2:  Set $L^{\mathcal{R}}_{0,0} = B^{\mathcal{R}}_{0,0} = \{(0,0)\}$,  $L^{\mathcal{R}}_{i,0} = \emptyset$ for $i \in \{1,\dots,n\}$,  $B^{\mathcal{R}}_{0,j} = \emptyset$ for $j \in \{1,\dots,m\}$
3:  for $i = 0$ to $n$ do
4:      for $j = 0$ to $m$ do
5:          $\sigma = L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}$
6:          $\lambda = \pi_{ij}(\sigma)$
7:          $B^{\mathcal{R}}_{i,j+1} = \lambda \cap B^{\mathcal{F}}_{i,j+1}$
8:          $L^{\mathcal{R}}_{i+1,j} = \lambda \cap L^{\mathcal{F}}_{i+1,j}$
9:      end for
10: end for
11: Return YES if $(n,m) \in L^{\mathcal{R}}_{n+1,m}$, NO otherwise.
```

Lemma 2. *After the execution of Algorithm* 1, *a point* $q \in \operatorname{exit}(\mathcal{C}_{ij})$ *is reachable iff* $q \in B^{\mathcal{R}}_{i,j+1} \cup L^{\mathcal{R}}_{i+1,j}$.

Proof. We prove by induction on the cells in the cell-wise order. ($\Leftarrow$) Let $q \in B^{\mathcal{R}}_{i,j+1} \cup L^{\mathcal{R}}_{i+1,j}$. Then by our construction, there is a point $p \in L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}$ such that $q \in \pi_{ij}(p)$. By induction hypothesis, p is reachable, and therefore, there is a slope-constrained path $\mathcal{P}$ in $\mathcal{F}_\varepsilon$ connecting $(0,0)$ to p. Now, $\mathcal{P} + \overline{pq}$ is a slope-constrained path from $(0,0)$ to q, implying that q is reachable. ($\Rightarrow$) We show that any point $q \in \operatorname{exit}(\mathcal{C}_{ij})$ which is not in $B^{\mathcal{R}}_{i,j+1} \cup L^{\mathcal{R}}_{i+1,j}$ is unreachable. Suppose on the contrary that q is reachable. Then, there should exist a slope-constrained path $\mathcal{P}$ in $\mathcal{F}_\varepsilon$ that connects $(0,0)$ to q. Because the slope of $\mathcal{P}$ cannot be negative, $\mathcal{P}$ must cross $\operatorname{entry}(\mathcal{C}_{ij})$ at some point p. Now, p is reachable from $(0,0)$, because it is on a slope-constrained path from $(0,0)$ to p. Therefore, $p \in L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}$ by induction. Consider two line segments s_1 and s_2 that connect p to $\operatorname{exit}(\mathcal{C}_{ij})$ with slopes minSlope_{ij} and maxSlope_{ij}, respectively. Since $q \notin \pi_{ij}(p)$, the portion of $\mathcal{P}$ that lies between p and q must cross either s_1 or s_2. But, it implies that the slope of $\mathcal{P}$ at the cross point falls out of the permissible range $[\text{minSlope}_{ij}, \text{maxSlope}_{ij}]$, and thus, $\mathcal{P}$ cannot be slope-constrained: a contradiction. □

Corollary 3. *Algorithm* 1 *returns* YES *iff* $\delta_{\bar{F}}(P, Q) \leqslant \varepsilon$.

Proof. This follows immediately from Lemmas 1 and 2. □

We now show how Algorithm 1 can be implemented efficiently. Let a *reachable interval* be a maximal continuous subset of reachable points on the entry side (or the exit side) of a cell. Therefore, each of $L^{\mathcal{R}}_{ij}$ and $B^{\mathcal{R}}_{ij}$ can be represented as a sequence of reachable intervals. We make two observations:

Observation 1. *For each cell* $\mathcal{C}_{ij}$, *the number of reachable intervals on* $\operatorname{exit}(\mathcal{C}_{ij})$ *is at most one more than the number of reachable intervals on* $\operatorname{entry}(\mathcal{C}_{ij})$.

Proof. Let $\sigma = L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}$ be the set of reachable points on $\operatorname{entry}(\mathcal{C}_{ij})$, and let $\lambda = \pi_{ij}(\sigma)$ be the projection of σ onto $\operatorname{exit}(\mathcal{C}_{ij})$. Since the projection on each reachable interval on the exit side is continuous, no reachable interval in σ can contribute to more than one reachable interval in λ. Therefore, the number of intervals in λ is at most equal to the number of intervals in σ. (Note that projected intervals can merge.) However, after splitting λ between $L_{i+1,j}$ and $B_{i,j+1}$, at most one of the intervals in λ (the one containing $L_{i+1,j} \cap B_{i,j+1}$) may split into two, which increases the number of intervals by at most one. □

Corollary 4. *The number of reachable intervals on the entry side of each cell is* $O(n^2)$.

The above upper bound of $O(n^2)$ is indeed tight as proved in Section 4.

Observation 2. *Let* $\langle I_1, I_2, \ldots, I_k\rangle$ *be a sequence of intervals on the entry side of a cell* $\mathcal{C}_{ij}$. *If* $I_1 \prec I_2 \prec \cdots \prec I_k$ *then* $\pi_{ij}(I_1) \prec \pi_{ij}(I_2) \prec \cdots \prec \pi_{ij}(I_k)$.

Proof. For all $t \in \{1, \ldots, k\}$, let ℓ_t be the line segment connecting $\operatorname{left}(I_t)$ to $\operatorname{left}(\pi_{ij}(I_t))$, and r_t be the line segment connecting $\operatorname{right}(I_t)$ to $\operatorname{right}(\pi_{ij}(I_t))$. The observation immediately follows from the fact that all segments in the set $\{\ell_t\}_{1 \leqslant t \leqslant k}$ have slope maxSlope_{ij} (and thus are parallel), and all segments in $\{r_t\}_{1 \leqslant t \leqslant k}$ have slope minSlope_{ij}. Note that this proof holds even if the intervals in the original sequence and/or intervals in the projected sequence overlap each other. □

Theorem 5. *Algorithm* 1 *solves the decision problem in* $O(n^3)$ *time.*

Proof. The correctness of the algorithm follows from Corollary 3. For the running time, we first compute the time needed for processing a cell $\mathcal{C}_{ij}$. Let r_{ij} be the number of reachable intervals on the entry side of $\mathcal{C}_{ij}$. We use a simple data structure, like a linked list, to store each $L^{\mathcal{R}}_{ij}$ and $B^{\mathcal{R}}_{ij}$ as a sequence of its reachable intervals (sorted in $\prec$ order). It is easy to observe that lines 5–8 can be performed in $O(r_{ij})$ time. In particular, line 5 can be performed by a simple concatenation of two lists in O(1) time; and lines 7 and 8 involve an easy intersection test for each of the intervals in λ, which takes $O(r_{ij})$ time. The crucial part is line 6 at which reachable intervals are projected. Computing the projection of each interval takes constant time. However, we need to merge intersecting intervals afterwards. By Observation 2, the merge step can be performed via a linear scan, which takes $O(r_{ij})$ time. The overall running time of the algorithm is therefore $O(\sum_{i,j} r_{ij})$.

Since $r_{ij} = O(n^2)$ by Corollary 4, and there are $O(n^2)$ cells, a running time of $O(n^4)$ is immediately implied. We can obtain a tighter bound by computing $\sum_{i,j} r_{ij}$ explicitly. Define $R_k = \sum_{i+j=k} r_{ij}$, for $0 \leqslant k \leqslant 2n$. R_k denotes the number of reachable intervals on the entry side of all cells $\mathcal{C}_{ij}$ with $i + j = k$. By Observation 1, each of the $k+1$ cells contributing to R_k can produce at most 1 new interval. Therefore, $R_{k+1} \leqslant R_k + k + 1$. Starting with $R_0 = 1$, we get $R_k \leqslant \sum_{\ell=0}^{k} (\ell + 1) = O(k^2)$. Thus,

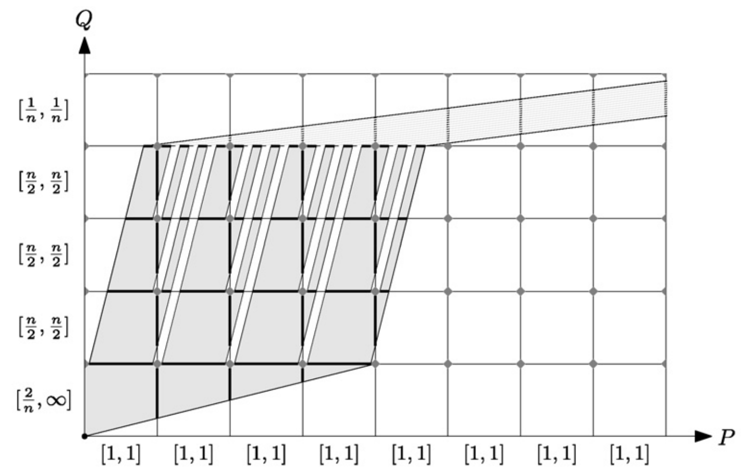

Q
$[\frac{1}{n},\frac{1}{n}]$
$[\frac{n}{2},\frac{n}{2}]$
$[\frac{n}{2},\frac{n}{2}]$
$[\frac{n}{2},\frac{n}{2}]$
$[\frac{2}{n},\infty]$
[1, 1]
[1, 1]
[1, 1]
[1, 1]
[1, 1]
[1, 1]
[1, 1]
[1, 1]
P

Fig. 3. A lower bound example for $n = 8$. The small gray diamonds represent obstacles in the free-space diagram. Reachable intervals are shown with bold black line segments. The numbers shown at each row and column represent speed limits on the corresponding segment.

$$\sum_{0\leqslant i,j\leqslant n} r_{ij} \leqslant \sum_{0\leqslant k\leqslant 2n} R_k = \sum_{0\leqslant k\leqslant 2n} O\big(k^2\big) = O\big(n^3\big). \qquad \square$$

4. An improved algorithm

In the previous section, we provided an algorithm that solves the decision problem in $O(n^3)$ time. The following lemma shows that any algorithm which is based on computing the reachability information on all cells has $\Omega(n^3)$ time complexity.

Lemma 6. *For any $n > 0$, there exist two polygonal curves P and Q of size $O(n)$ such that in the free-space diagram corresponding to P and Q, there are $\Theta(n)$ cells, where each cell has $\Theta(n^2)$ reachable intervals on its boundary.*

Proof. Let P be a polygonal curve consisting of n horizontal segments of unit length centered at the origin, and let Q be a polygonal curve consisting of $n/2+1$ vertical segments, where each segment Q_2 to $Q_{n/2+1}$ has unit length centered at the origin, and Q_1 has length $1-\delta$, for a sufficiently small $\delta \ll 1/n$. Let $\varepsilon = \sqrt{1/2-\delta+\delta^2}$. The free-space diagram $\mathcal{F}_\varepsilon$ for the two curves has a shape like Fig. 3 (the gray diamond-shape regions show obstacles in the free space each having a width of 2δ in x direction). We assign the following speed limits to the segments of P and Q. All segments of P have speed limits $[1,1]$, Q_1 has speed limits $[2/n,\infty]$, Q_2 to $Q_{n/2}$ have limits $[n/2,n/2]$, and $Q_{n/2+1}$ has limits $[1/n,1/n]$. The number of reachable intervals on each horizontal line $y=i$ is increased by $n/2$ at each row i, for i from 1 to $n/2$, yielding a total number of $\Theta(n^2)$ reachable intervals on the line $y=n/2$. Since all these reachable intervals are projected to the right side in the last row, each cell $\mathcal{C}_{i,n/2+1}$ for $i \in \{n/2+1,\dots,n\}$ has $\Theta(n^2)$ reachable intervals on its entry side. $\square$

While the complexity of the free space is cubic by the previous lemma, we show in this section that it is possible to eliminate some of the unneeded computations, and obtain an improved algorithm that solves the decision problem in $O(n^2\log n)$ time. The key idea behind our faster algorithm is to use a “lazy computation” technique: we delay the computation of reachable intervals until they are actually required. In our new algorithm, instead of computing the projection of all reachable intervals one by one from the entry side of each cell to its exit side, we only keep a sorted order of projected intervals, along with some minimal information that enables us to compute the exact location of the intervals whenever necessary.

To this end, we distinguish between two types of reachable intervals. Given a reachable interval I in exit($\mathcal{C}_{ij}$), we call I an *interior interval* if there is a reachable interval I' in entry($\mathcal{C}_{ij}$) such that $I = \pi_{ij}(I')$, and we call I a *boundary interval* otherwise. The main gain, as we see later in this section, is that the exact location of interior intervals can be computed efficiently based on the location of the boundary intervals. The following iterated projection is a main tool that we will use.

Iterated projections. Let I_1 be a reachable interval on the entry side of a cell $\mathcal{C}_{i_1j_1}$, and I_k be an interval on the exit side of a cell $\mathcal{C}_{i_kj_k}$. We say that I_k is an *iterated projection* of I_1, if there is a sequence of cells $\mathcal{C}_{i_2j_2},\dots,\mathcal{C}_{i_{k-1}j_{k-1}}$ and a sequence of intervals $I_2,\dots,I_{k-1}$ such that for all $1 \leqslant t \leqslant k-1$, $I_t \subseteq \text{entry}(\mathcal{C}_{i_tj_t})$ and $I_{t+1} = \pi_{i_tj_t}(I_t)$ (see Fig. 4). In the following, we show that I_k can be computed efficiently from I_1.

Given two points $p \in \mathcal{C}_{ij}$ and $q \in \mathcal{C}_{i'j'}$, we say that q is the *min projection* of p, if there is a polygonal path $\mathcal{P}$ from p to q passing through a sequence of cells $\mathcal{C}_{i_1j_1},\mathcal{C}_{i_2j_2},\dots,\mathcal{C}_{i_kj_k}$ $(k \geqslant 1)$, such that $(i_1,j_1)=(i,j)$, $(i_k,j_k)=(i',j')$, and $\mathcal{P}\cap\mathcal{C}_{i_tj_t}$ is a line segment whose slope is $\text{minSlope}_{i_tj_t}$, for all $1 \leqslant t \leqslant k$. The *max projection* of a point p is defined analogously.

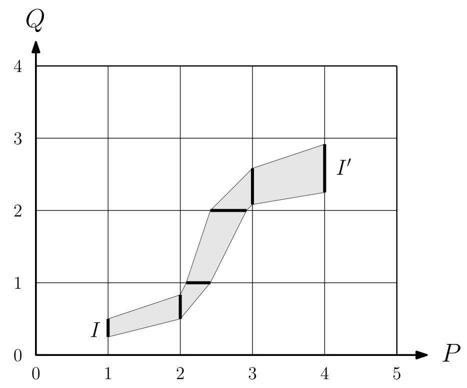


Fig. 4. I' is an iterated projection of I.

Lemma 7. *Using $O(n)$ preprocessing time and space, we can build a data structure that for any point $p \in \mathcal{B}_{n\times m}$ and any edge e of $\mathcal{B}_{n\times m}$, determines in $O(1)$ time if the min (or the max) projection of p onto the line containing e lies before, after, or on e; and in the latter case, computes the exact projection of p onto e in constant time.*

Proof. Suppose, w.l.o.g., that e is a vertical edge of $\mathcal{B}_{n\times m}$, corresponding to a vertex $P(i)$ of P and a segment $Q(j-1)Q(j)$ of Q. Then $e = \{i\} \times [j-1, j]$. Let q be the min projection of p on the line $x = i$. Let $p = (p_x, p_y)$ and $q = (q_x, q_y)$. The path connecting p to q in the definition of the min projection has slope minSlope$_{ij}$ in each cell $\mathcal{C}_{ij}$ it passes through. Such a path corresponds to the traversals of two point objects $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$, where $\bar{\mathcal{O}}_P$ traverses $[p_x, q_x]$ with its maximum permissible speed, and $\bar{\mathcal{O}}_Q$ traverses $[p_y, q_y]$ with its minimum permissible speed. Since each of the point objects $\bar{\mathcal{O}}_P$ and $\bar{\mathcal{O}}_Q$ can traverse $O(n)$ segments, computing the min projection can be easily done in $O(n)$ time. However, we can speedup the computation using a simple table lookup technique. For $\bar{\mathcal{O}}_P$, we keep two arrays $T^P_{\min}$ and $T^P_{\max}$ of size n, where for each $i \in \{1,\dots,n\}$, $T^P_{\min}[i]$ (resp., $T^P_{\max}[i]$) represents the minimum (resp., maximum) time needed for $\bar{\mathcal{O}}_P$ to traverse the interval $[0, i]$. Similarly, we keep two arrays $T^Q_{\min}$ and $T^Q_{\max}$ for $\bar{\mathcal{O}}_Q$. These four tables can be easily constructed in $O(n)$ time. To find time t needed for $\bar{\mathcal{O}}_P$ to traverse $[p_x, q_x]$ with its maximum speed, we do the following: we first lookup $a = T^P_{\max}[\lceil p_x \rceil]$ and $b = T^P_{\max}[q_x]$ in $O(1)$ time. Clearly, $b - a$ is equal to the time needed for $\bar{\mathcal{O}}_P$ to traverse $[\lceil p_x \rceil, q_x]$ (note that q_x is an integer). We also compute the time t' needed for $\bar{\mathcal{O}}_P$ to traverse $[p_x, \lceil p_x \rceil]$ directly from the length of the interval, and the maximum speed of $\bar{\mathcal{O}}_P$ in interval $[\lceil p_x \rceil - 1, \lceil p_x \rceil]$. Therefore, $t = t' + b - a$ can be computed in $O(1)$ time total. By similar table lookups, we compute the times t_1 and t_2 needed for $\bar{\mathcal{O}}_Q$ to traverse $[p_y, j-1]$ and $[p_y, j]$, respectively, with its minimum speed. If $t_1 \leqslant t \leqslant t_2$, then we conclude that q_y lies in e, and we can easily compute its exact location on e by computing the distance that $\bar{\mathcal{O}}_Q$ traverses in $t - t_1$ time using its minimum speed on interval $[j-1, j]$. Otherwise, we output that q is before or after e, depending on whether $t < t_1$ or $t > t_2$, all in $O(1)$ time. □

Corollary 8. *If I' is an iterated projection of I, then I' can be computed from I in $O(1)$ time, after $O(n)$ preprocessing time.*

Proof. This is a direct corollary of Lemma 7 and the fact that if $I' = [a', b']$ is an iterated projection of $I = [a, b]$, then a' is the max projection of a, and b' is the min projection of b. □

The data structure. The main data structure that we need in our algorithm is a dictionary for storing a sorted sequence of intervals. A balanced binary search tree can be used for this purpose. Let T be the data structure that stores a sequence $\langle I_1, I_2, \dots, I_k\rangle$ of intervals in $\prec$ order. We need the following operations to be supported by T.

Search: Given a point x, find the leftmost interval I in T such that $x \leqslant \text{left}(I)$.
Insert: Insert a new interval I into T, right before T.Search(left(I)), or at the end of T if I is to the right of all existing intervals in T. In our algorithm, inserted intervals are not properly contained in any existing interval of T, and therefore, the resulting sequence is always sorted.
Delete: Delete an existing interval I from T.
Split: Given an interval $I = I_j$, $1 < j \leqslant k$, split T into two data structures T_1 and T_2, containing $\langle I_1, \dots, I_{j-1}\rangle$ and $\langle I_j, \dots, I_k\rangle$, respectively.
Join: Given two data structures with interval sequences $\mathcal{I}_1$ and $\mathcal{I}_2$, where each interval in $\mathcal{I}_1$ is before any interval in $\mathcal{I}_2$, join the two structures to obtain a single structure T containing the concatenated sequence $\mathcal{I}_1 \cdot \mathcal{I}_2$.

It is pretty straightforward to modify a standard balanced binary search tree to perform all the above operations in $O(\log|T|)$ time (for example, see Chapter 4 in [14]). Note that the exact coordinates of the interior intervals are not explicitly

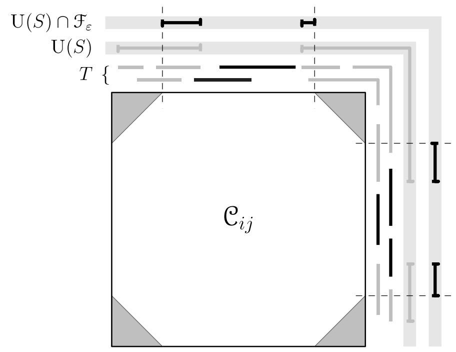


Fig. 5. An example of the execution of Algorithm 2 on a cell $\mathcal{C}_{ij}$. The intervals of $S \subseteq T$ are shown in gray. The black intervals in T represent the interior intervals. The intervals in $\mathrm{U}(S) \cap \mathcal{F}_\varepsilon$ are boundary intervals which are inserted in lines 12–13.

stored in the data structure. Rather, we compute the coordinates on the fly whenever a comparison is made, in $O(1)$ time per comparison, using Corollary 10.

The algorithm. Let L^T_{ij} (resp., B^T_{ij}) denote the balanced search tree storing the sequence of reachable intervals on L_{ij} (resp., on B_{ij}). The reachable intervals stored in the trees are not necessarily disjoint. In particular, we allow interior intervals to have overlaps with each other, but not with boundary intervals. Moreover, the exact locations of the interior intervals are not explicitly stored. However, we maintain the invariant that each interior interval can be computed in $O(1)$ time, and that the union of the reachable intervals stored in L^T_{ij} (resp., in B^T_{ij}) at each time is equal to $L^{\mathcal{R}}_{ij}$ (resp., $B^{\mathcal{R}}_{ij}$).

The overall structure of the algorithm is similar to that of Algorithm 1. We process the cells in the cell-wise order, and propagate the reachability information through each cell by projecting the reachable intervals from the entry side to the exit side. However, to get a better performance, cells are processed in a slightly different manner, as presented in Algorithm 2. In this algorithm, $\mathrm{exit}(\mathcal{C}_{ij})$ is considered as a single line segment whose points are ordered by $\prec$ relation. For a set S of intervals, we define $\mathrm{U}(S) = \bigcup_{I \in S} I$. Given a data structure T as defined in the previous subsection, we use T to refer to both the data structure and the set of intervals stored in T. Given a point set S on a line, by an interval (or a segment) of S we mean a maximal continuous subset of points contained in S.

Algorithm 2 IMPROVED DECISION ALGORITHM

```
1: Compute the free space, $\mathcal{F}_\varepsilon$
2: for $i \in \{0,\dots,n\}$ do $L^T_{i,0} = \emptyset$
3: for $j \in \{0,\dots,m\}$ do $B^T_{0,j} = \emptyset$
4: $L^T_{0,0}$.INSERT($[o,o]$) where $o = (0,0)$
5: for $i = 0$ to $n$ do
6:     for $j = 0$ to $m$ do
7:         $T = \mathrm{JOIN}(L^T_{ij}, B^T_{ij})$
8:         Project $T$ to the exit side of $\mathcal{C}_{ij}$
9:         $S = \{I \in T \mid I \not\subseteq B^{\mathcal{F}}_{i,j+1}$ and $I \not\subseteq L^{\mathcal{F}}_{i+1,j}\}$
10:        for each $I \in S$ do $T$.DELETE($I$)
11:        $(B^T_{i,j+1}, L^T_{i+1,j}) = T$.SPLIT($T$.SEARCH($(i,j)$))
12:        for each interval $I$ in $\mathrm{U}(S) \cap B^{\mathcal{F}}_{i,j+1}$ do $B^T_{i,j+1}$.INSERT($I$)
13:        for each interval $I$ in $\mathrm{U}(S) \cap L^{\mathcal{F}}_{i+1,j}$ do $L^T_{i+1,j}$.INSERT($I$)
14:    end for
15: end for
16: Return YES if $(n,m) \in L^T_{n+1,m}$, NO otherwise.
```

The algorithm works as follows. We first compute $\mathcal{F}_\varepsilon$ in line 1. Lines 2–4 initialize the data structures for the first row and the first column of $\mathcal{B}_{n\times m}$. Lines 5–13 process the cells in the cell-wise order. For each cell $\mathcal{C}_{ij}$, lines 7–13 propagate the reachability information through $\mathcal{C}_{ij}$ by creating data structures $B^T_{i,j+1}$ and $L^T_{i+1,j}$ on the exit side of $\mathcal{C}_{ij}$, based on B^T_{ij} and L^T_{ij}, and the feasible intervals $B^{\mathcal{F}}_{i,j+1}$ and $L^{\mathcal{F}}_{i+1,j}$. In line 7, a data structure T is obtained by joining the interval sequences in B^T_{ij} and L^T_{ij}. We then project T to the exit side of $\mathcal{C}_{ij}$ in line 8 by (virtually) transforming each interval $I \in T$ to an interval $\pi_{ij}(I)$ on $\mathrm{exit}(\mathcal{C}_{ij})$. Since the projection preserves the relative order of intervals by Observation 2, and since we do not need to explicitly update the location of interior intervals on the exit side, the projection is simply done by copying T to the exit side of $\mathcal{C}_{ij}$ (boundary intervals will be fixed later in lines 12–13). Furthermore, since B^T_{ij} and L^T_{ij} are not needed afterwards in the algorithm, we do not actually duplicate T. Instead, we simply assign T to the exit side, without making a new copy. In line 9, we determine a set S of intervals that are not completely contained in $B^{\mathcal{F}}_{i,j+1}$ or in $L^{\mathcal{F}}_{i+1,j}$. All such intervals are deleted from T in line 10 (see Fig. 5 for an illustration). The remaining intervals in T have no intersection with the corner point (i,j). Therefore, we can easily split T in line 11 into two disjoint data structures, $B^T_{i,j+1}$ and $L^T_{i+1,j}$, each corresponding to one edge of the exit side. In lines 12–13 we insert the boundary intervals into $B^T_{i,j+1}$ and $L^T_{i+1,j}$, which are computed as those portions of $\mathrm{U}(S)$ that lie inside $\mathcal{F}_\varepsilon$. Note that whenever a boundary interval I is inserted into a data structure, its coordinates are stored along with the interval. After processing all cells, the decision problem is easily answered in line 16 of the algorithm by checking if the target point (n, m) is reachable.

Lemma 9. *After processing each cell $\mathcal{C}_{ij}$, the following statements hold true*:

(i) *any interval inserted into* $\mathrm{exit}(\mathcal{C}_{ij})$ *in lines* 12–13 *is a boundary interval,*
(ii) *each interior interval on* $\mathrm{exit}(\mathcal{C}_{ij})$ *can be expressed as an iterated projection of a boundary interval.*

Proof. (i) This is easy by observing that no interior interval is added to S in line 9, and therefore, $\mathrm{U}(S)$ cannot completely contain any interior interval. (ii) The proof is by induction on the cells in the cell-wise order. Let I be an interior interval on $\mathrm{exit}(\mathcal{C}_{ij})$. Then I is a direct projection of an interval $I' \subseteq \mathrm{entry}(\mathcal{C}_{ij})$ obtained in line 8. If I' is a boundary interval, then we are done. Otherwise, I' is an interior interval, and therefore, it is by induction an iterated projection of another boundary interval I''. Since $I = \pi_{ij}(I')$ and $I' \subseteq \mathrm{entry}(\mathcal{C}_{ij})$, I is in turn an iterated projection of I''. □

Corollary 10. *After processing each cell $\mathcal{C}_{ij}$, the exact location of each reachable interval on* $\mathrm{exit}(\mathcal{C}_{ij})$ *is accessible in $O(1)$ time.*

Proof. Fix a reachable interval I on $\mathrm{exit}(\mathcal{C}_{ij})$. If I is a boundary interval, then by Lemma 9(i), it is inserted into a data structure by lines 12–13, and hence, its coordinates are stored in the data structure upon insertion. If I is an interior interval, then by Lemma 9(ii), it is an iterated projection of a boundary interval, and hence, its location can be computed in $O(1)$ time using Corollary 8. □

Lemma 11. *After processing each cell $\mathcal{C}_{ij}$, $B^{\mathcal{R}}_{i,j+1} \cup L^{\mathcal{R}}_{i+1,j} = \mathrm{U}(B^T_{i,j+1} \cup L^T_{i+1,j})$.*

Proof. We prove by induction on the cells in the cell-wise order. Recall from Section 3 (Algorithm 1) that $B^{\mathcal{R}}_{i,j+1} \cup L^{\mathcal{R}}_{i+1,j} = \pi_{ij}(L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}) \cap (B^{\mathcal{F}}_{i,j+1} \cup L^{\mathcal{F}}_{i+1,j})$. Therefore, we just need to show that $\mathrm{U}(B^T_{i,j+1} \cup L^T_{i+1,j}) = \pi_{ij}(L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij}) \cap (B^{\mathcal{F}}_{i,j+1} \cup L^{\mathcal{F}}_{i+1,j})$. By line 7, $\mathrm{U}(T) = \mathrm{U}(L^T_{ij} \cup B^T_{ij})$. Let T_1 be the set of intervals in T right after the execution of line 8, S be the set of intervals deleted in line 10, N be the set of new intervals inserted in lines 12–13, and $T_2 = (T_1 \setminus S) \cup N$. Fix a point $p \in \mathrm{U}(T_1)$, and let K be the set of intervals in T_1 containing p. We distinguish between two cases:

- $p \in \mathcal{F}_\varepsilon$: There are two possibilities: (1) $K \nsubseteq S$: Here, there is an interval in K that remains in T_1 after deletion of S in line 10. Therefore, $p \in \mathrm{U}(T_2)$. (2) $K \subseteq S$: Here, all intervals of K are removed in line 10. However, since $p \in \mathcal{F}_\varepsilon$, there is an interval $I \in N$ such that $p \in I$. Therefore, after insertion of I in lines 12–13, we have $p \in \mathrm{U}(T_2)$.
- $p \notin \mathcal{F}_\varepsilon$: In this case, $K \subseteq S$, and hence $p \notin \mathrm{U}(T_1 \setminus S)$. Moreover, no interval in N can contain p. Therefore, $p \notin \mathrm{U}(T_2)$.

The above two cases together show that $\mathrm{U}(T_2) = \mathrm{U}(T_1) \cap \mathcal{F}_\varepsilon$. Note that, $\mathrm{U}(T_1) = \pi_{ij}(\mathrm{U}(L^T_{ij} \cup B^T_{ij}))$ (by lines 7 and 8), and $T_2 = B^T_{i,j+1} \cup L^T_{i+1,j}$. Therefore, $\mathrm{U}(B^T_{i,j+1} \cup L^T_{i+1,j}) = \pi_{ij}(\mathrm{U}(L^T_{ij} \cup B^T_{ij})) \cap (B^{\mathcal{F}}_{i,j+1} \cup L^{\mathcal{F}}_{i+1,j})$, which completes the proof, because $L^{\mathcal{R}}_{ij} \cup B^{\mathcal{R}}_{ij} = \mathrm{U}(L^T_{ij} \cup B^T_{ij})$ by induction. □

Theorem 12. *Algorithm* 2 *solves the decision problem in $O(n^2 \log n)$ time.*

Proof. The correctness of the algorithm follows from Lemma 11, combined with Lemma 2. For the running time, we compute the number of operations needed to process each cell $\mathcal{C}_{ij}$ in lines 7–13. Let $\mathcal{T}$ denote the time needed for each data structure operation. Line 7 needs one join operation that takes $O(\mathcal{T})$ time. Line 8 consists of a simple assignment taking only $O(1)$ time. To compute the subset S in line 9, we start walking from the two sides of T, and add intervals to S until we reach the first intervals from both sides that do not belong to S. Moreover, we find the interval $I = T.\textsc{Search}((i, j))$, and start walking around I in both directions until we find all consecutive intervals around I that lie in S (see Fig. 5). To check if an interval lies in S or not, we need to compute the coordinates of the interval that can be done in $O(1)$ time. Therefore, computing S takes $O(|S| + \mathcal{T})$ time in total. Line 10 requires $|S|$ delete operation that takes $O(|S| \times \mathcal{T})$ time. Line 11 consists of a split operation taking $O(\mathcal{T})$ time. The set $\mathrm{U}(S)$ used in lines 12–13 can be computed in $O(|S|)$ time by a linear scan over the set S. Since $\mathrm{U}(S)$ consists of at most three segments (see Fig. 5), computing $\mathrm{U}(S) \cap \mathcal{F}_\varepsilon$ in lines 12–13 takes constant time. Moreover, there are at most four insertion operations in lines 12–13 to insert boundary intervals. Therefore, lines 12–13 takes $O(|S| + \mathcal{T})$ time. Thus, letting $s_{ij} = |S|$, processing each cell $\mathcal{C}_{ij}$ takes $O((s_{ij} + 1) \times \mathcal{T})$ time in total. Since at most four new intervals are created at each cell, the total number of intervals created over all cells is $O(n^2)$. Note that any of these $O(n^2)$ intervals can be deleted at most once, meaning that $\sum_{i,j} s_{ij} = O(n^2)$. Moreover, each comparison made in the data structures takes $O(1)$ time by Corollary 10, and hence, $\mathcal{T} = O(\log n)$. Therefore, the total running time of the algorithm is $O(\sum_{i,j}(s_{ij} + 1) \log n) = O(n^2 \log n)$. □

5. Optimization problem

In this section, we describe how our decision algorithm can be used to compute the exact value of the Fréchet distance with speed limits between two polygonal curves. We use the same parametric search technique as used in [2]. Let $L_{ij}^{\mathcal{F}} = [a_{ij}, b_{ij}]$ and $B_{ij}^{\mathcal{F}} = [c_{ij}, d_{ij}]$. Notice that the free space, $\mathcal{F}_\varepsilon$, is an increasing function of ε. That is, for $\varepsilon_1 \leqslant \varepsilon_2$, we have $\mathcal{F}_{\varepsilon_1} \subseteq \mathcal{F}_{\varepsilon_2}$. Therefore, to find the exact value of $\delta = \delta_{\bar{F}}(P, Q)$, we can start from $\varepsilon = 0$, and continuously increase ε until we reach the first point at which $\mathcal{F}_\varepsilon$ contains a slope-constrained path from $(0,0)$ to (n,m). It is not difficult to see that this occurs at only one of the following "critical values":

(A) smallest ε for which $(0,0) \in \mathcal{F}_\varepsilon$ or $(n,m) \in \mathcal{F}_\varepsilon$,
(B) smallest ε at which $L_{ij}^{\mathcal{F}}$ or $B_{ij}^{\mathcal{F}}$ becomes non-empty for some pair (i, j),
(C) smallest ε at which a_{ij} is the min projection of $b_{k\ell}$, or d_{ij} is the max projection of $c_{k\ell}$, for some i, j, k, and ℓ.

Obviously, there are two critical values of type (A), $O(n^2)$ critical values of type (B), and $O(n^4)$ critical values of type (C), each computable in $O(1)$ time (see [2] and Lemma 7). Therefore, to find the exact value of δ, one can compute all these $O(n^4)$ values, sort them, and do a binary search (equipped with our decision algorithm) to find the smallest ε for which $\delta_{\bar{F}}(P, Q) \leqslant \varepsilon$, in $O(n^4 \log n)$ total time. However, as mentioned in [2], a parametric search method [11,4] can be applied to the critical values of type (C) to get a faster algorithm.

The crucial observation made in [2] is that any comparison-based sorting algorithm that sorts a_{ij}, b_{ij}, c_{ij}, and d_{ij} (defined as functions of ε) has critical values that include those of type (C). This is because the critical values of type (C) occur if $a_{ij} = b_{k\ell} + c$ or $d_{ij} = c_{k\ell} + c'$, for some i, j, k, and ℓ, and some constants c and c' (obtained from min and max projections). We can thus use the following refined algorithm:

1. Compute all critical values of types (A) and (B), and sort them.
2. Binary search to find two consecutive values ε_1 and ε_2 in the sorted list such that $\delta \in [\varepsilon_1, \varepsilon_2]$.
3. Let S be the set of endpoints $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ of intervals $L_{ij}^{\mathcal{F}}$ and $B_{ij}^{\mathcal{F}}$ that are non-empty for $\varepsilon \in [\varepsilon_1, \varepsilon_2]$. Use Cole's parametric search method [4] based on sorting the values in S to find the exact value of δ.

Steps 1 and 2 together take $O(n^2 \log n)$. The parametric search in step 3 takes $O((k + T)\log k)$ time, where k is the number of values to be sorted, and T is the time needed by the decision algorithm. In our case, $k = |S| = O(n^2)$, and $T = O(n^2 \log n)$. Therefore, we conclude:

Theorem 13. *The exact Fréchet distance with speed limits can be computed in $O(n^2 \log^2 n)$ time.*

6. Conclusions

In this paper, we introduced a variant of the Fréchet distance between two polygonal curves in which the speed of traversal along each segment of the curves is restricted to be within a specified range. We presented an efficient algorithm to solve the decision problem in $O(n^2 \log n)$ time, which together with a parametric search, led to a $O(n^2 \log^2 n)$ time algorithm for finding the exact value of the Fréchet distance with speed limits.

Several open problems arise from our work. In particular, it is interesting to consider speed limits in other variants of the Fréchet distance studied in the literature, such as the Fréchet distance between two curves lying inside a simple polygon [5], on a convex polyhedron [10], or on a polyhedral surface [6]. Our result can be also useful in matching planar maps, where the objective is to find a curve in a road network that is as close as possible to a vehicle trajectory. In [1], the traditional Fréchet metric is used to match a trajectory to a road network. If the road network is very congested, the Fréchet distance with speed limits introduced here seems to find a more realistic path in the road network, close to the trajectory of the vehicle. It is also interesting to extend our variant of the Fréchet distance to the setting where the speed limits on the segments of the curves change as functions over time.

Acknowledgements

The authors would like to thank the anonymous referees for their helpful comments.

References

[1] H. Alt, A. Efrat, G. Rote, C. Wenk, Matching planar maps, J. Algorithms 49 (2) (2003) 262–283.
[2] H. Alt, M. Godau, Computing the Fréchet distance between two polygonal curves, Internat. J. Comput. Geom. Appl. 5 (1995) 75–91.
[3] E.W. Chambers, E. Colin de Verdière, J. Erickson, S. Lazard, F. Lazarus, S. Thite, Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time, Comput. Geom. Theory Appl. 43 (3) (2010) 295–311.
[4] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, J. ACM 34 (1) (1987) 200–208.

[5] A.F. Cook, C. Wenk, Geodesic Fréchet distance inside a simple polygon, in: Proc. 25th Sympos. Theoret. Aspects Comput. Sci., in: Lecture Notes in Comput. Sci., vol. 5664, 2008, pp. 193–204.
[6] A.F. Cook, C. Wenk, Shortest path problems on a polyhedral surface, in: Proc. 11th Workshop Algorithms Data Struct., in: Lecture Notes in Comput. Sci., vol. 5664, 2009, pp. 156–167.
[7] A. Efrat, L.J. Guibas, S. Har-Peled, J.S.B. Mitchell, T.M. Murali, New similarity measures between polylines with applications to morphing and polygon sweeping, Discrete Comput. Geom. 28 (4) (2002) 535–569.
[8] M. Jiang, Y. Xu, B. Zhu, Protein structure–structure alignment with discrete Fréchet distance, J. Bioinform. Comput. Biol. 6 (1) (2008) 51–64.
[9] A. Maheshwari, J.-R. Sack, K. Shahbaz, H. Zarrabi-Zadeh, Fréchet distance with speed limits, Technical Report TR-10-08, School of Computer Science, Carleton University, 2010.
[10] A. Maheshwari, J. Yi, On computing Fréchet distance of two paths on a convex polyhedron, in: Proc. 21st European Workshop Comput. Geom., 2005, pp. 41–44.
[11] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, J. ACM 30 (4) (1983) 852–865.
[12] S. Pelletier, Computing the Fréchet distance between two polygonal curves, http://www.cim.mcgill.ca/~stephane/cs507/Project.html.
[13] E. Sriraghavendra, K. Karthik, C. Bhattacharyya, Fréchet distance based approach for searching online handwritten documents, in: Proc. 9th Internat. Conf. Document Anal. Recognition, 2007, pp. 461–465.
[14] R.E. Tarjan, Data Structures and Network Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1983.