Improved Algorithms for Partial Curve Matching*

Anil Maheshwari[1], Jörg-Rüdiger Sack[1], Kaveh Shahbaz[1], and Hamid Zarrabi-Zadeh[1]

School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada.
Email: {anil,sack,kshahbaz,zarrabi}@scs.carleton.ca

Abstract. Back in 1995, Alt and Godau gave an efficient algorithm for deciding whether a given curve resembles some *part* of a larger curve under a fixed Fréchet distance, achieving a running time of $O(nm \log(nm))$, for n and m being the number of segments in the two curves, respectively. We improve this long-standing result by presenting an algorithm that solves this decision problem in $O(nm)$ time. Our solution is based on constructing a simple data structure which we call *free-space map*. Using this data structure, we obtain improved algorithms for several variants of the Fréchet distance problem, including the Fréchet distance between two closed curves, and the so-called minimum/maximum walk problems. We also improve the map matching algorithm of Alt *et al.* for the case when the map is a directed acyclic graph.

1 Introduction

The Fréchet distance is a widely-used metric for measuring the similarity of the curves. It finds applications in morphing [8], handwriting recognition [12], protein structure alignment [9], etc. This measure is often illustrated as the minimum-length leash needed for a person to walk a dog, while each of them is traversing a pre-specified polygonal curve without backtracking.

Alt and Godau [3] showed how the Fréchet distance between two polygonal curves with n and m vertices can be computed in $O(nm \log(nm))$ time. For their solution, they introduced a data structure, called *free-space diagram*. The free-space diagram and its variants have been proved to be useful in other applications involving the Fréchet distance (see e.g. [2, 4, 7]).

In their seminal work, Alt and Godau [3] also studied a *partial curve matching* problem in which one wants to see if a given curve resembles some "part" of a larger curve. Given two polygonal curves P and Q of size n and m, respectively, they presented an algorithm that decides in $O(nm \log(nm))$ time whether there is a subcurve R of P whose Fréchet distance to Q is at most ε, for a given $\varepsilon \geqslant 0$. Using parametric search, they solved the optimization problem of finding the minimum such ε in $O(nm \log^2(nm))$ time .

* Research supported by NSERC, HPCVL, and SUN Microsystems.

Later, Alt *et al.* [2] proposed a generalization of the partial curve matching problem to measure the similarity of a curve to some part of a graph. Given a polygonal curve P and a graph G, they presented an $O(nm \log m)$-time algorithm to decide whether there is a path π in G whose Fréchet distance to P is at most ε, where n is the size of P and m is the complexity of G. A variant of the partial curve matching in the presence of outliers is studied by Buchin *et al.* [6], leading to an algorithm with $O(nm(n+m)\log(nm))$ running time.

Our results. In this paper, we present a simple data structure, which we call *free-space map*, that enables us to solve several variants of the Fréchet distance problem efficiently. The results we obtain using this data structure are summarized below. In the following, n and m represent the size of the two given polygonal curves P and Q, respectively, and $\varepsilon \geqslant 0$ is a fixed input parameter.

- *Partial curve matching.* Given two polygonal curves P and Q, we present an algorithm to decide in $O(nm)$ time whether there is a subcurve $R \subseteq P$ whose Fréchet distance to Q is at most ε. This improves the best previous algorithm for this decision problem due to Alt and Godau [3], that requires $O(nm\log(nm))$ time. This also leads to an $O(\log(nm))$ faster algorithm for solving the optimization version of the problem, using parametric search.
- *Closed Fréchet metric.* Alt and Godau [3] showed that for two closed curves P and Q, the decision problem of whether the closed Fréchet distance between P and Q is at most ε can be solved in $O(nm\log(nm))$ time. We improve this long-standing result by giving an algorithm that runs in $O(nm)$ time. As a result, we also improve by a $\log(nm)$-factor the running time of the optimization algorithm for computing the minimum such ε.
- *Maximum walk.* Given two curves P and Q and a fixed $\varepsilon \geqslant 0$, the *maximum walk* problem asks for the maximum-length subcurve of Q whose Fréchet distance to P is at most ε. We show that this optimization problem can be solved efficiently in $O(nm)$ time, without additional $\log(nm)$ factors. The *minimum walk* problem is analogously defined, and can be solved by an extension of the free-space map, as described in the appendix.
- *Graph matching.* Given a directed acyclic graph G, we present an algorithm to decide in $O(nm)$ time whether a curve P matches some part of G under a Fréchet distance of ε, for n and m being the size of P and the complexity of G, respectively. This improves the map matching algorithm of Alt *et al.* [2] for the case of DAGs. Note that Alt *et al.*'s algorithm has a running time of $\Theta(nm\log m)$ in the worst case even if the input graph is a DAG or a simple path.

The above improved results are obtained using a novel simple approach for propagating the reachability information "sequentially" from bottom side to the top side of the free-space diagram. Our approach is different from and simpler than the divide-and-conquer approach used by Alt and Godau [3], and also, the approach taken by Alt *et al.* [2] which is a mixture of line sweep, dynamic programming, and Dijsktra's algorithm.

The free-space map introduced in this paper encapsulates all the information available in the standard free-space diagram, yet it is capable of answering a more general type of queries efficiently. Namely, for any query point on the bottom side of the free-space diagram, our data structure can efficiently report all points on the top side of the diagram which are reachable from that query point. Given that our data structure has the same size and construction time as the standard free-space diagram, it can be viewed as a powerful alternative.

The current lower bound for deciding whether the Fréchet distance between two polygonal curves with total n vertices is at most a given value ε, is $\Omega(n \log(n))$ [5]. However, no subquadratic algorithm is known for this decision problem, and hence, one might conjecture that the problem may be 3SUM-hard (see [1]). If this conjecture holds, then the results obtained in this paper do not only represent improvements, but are also optimal.

The remainder of the paper is organized as follows. In Section 3, we define the free-space map and show how it can be efficiently constructed. In Section 4, we present some applications of the free-space map to problems such as partial curve matching, maximum walk, and closed Fréchet metric. In Section 5, we provide an improved algorithm for matching a curve in a DAG. We conclude in Section 6 with some open problems.

2 Preliminaries

A *polygonal curve* in $\mathbb{R}^d$ is a continuous function $P : [0, n] \to \mathbb{R}^d$ such that for each $i \in \{1, \ldots, n\}$, the restriction of P to the interval $[i-1, i]$ is affine (i.e., forms a line segment). The integer n is called the *size* of P. For each $i \in \{1, \ldots, n\}$, we denote the line segment $P|_{[i-1,i]}$ by P_i.

A *monotone reparametrization* 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 size n and m, respectively, the *Fréchet distance* between P and Q is defined as

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

where $\|\cdot\|$ denotes the Euclidean metric, and α and β range over all monotone reparameterizations of $[0, n]$ and $[0, m]$, respectively. Given a parameter $\varepsilon \geqslant 0$, the *free space* of the two curves P and Q is defined as

$$\mathcal{F}_\varepsilon(P, Q) = \{(s, t) \in [0, n] \times [0, m] \mid \|P(s), Q(t)\| \leqslant \varepsilon\}.$$

We call points in $\mathcal{F}_\varepsilon(P, Q)$ *feasible*. The partition of the rectangle $[0, n] \times [0, m]$ into regions formed by feasible and infeasible points is called the *free-space diagram* of P and Q, denoted by $\mathcal{FD}_\varepsilon(P, Q)$ (see Figure 1.a). For $0 \leqslant j \leqslant m$, we denote by $\mathcal{FD}_j$ the one-dimensional free-space diagram $\mathcal{FD}_\varepsilon(P, Q) \cap ([0, n] \times \{j\})$, corresponding to the curve P and the point $Q(j)$. For each $(i, j) \in \{1 \cdots n\} \times \{1 \cdots m\}$, the intersection of the free space diagram with the square $[i - 1, i] \times [j - 1, j]$ is called a *cell* of the diagram. Likewise, we call the intersection of $\mathcal{FD}_j$ with each interval $[i - 1, i]$ a cell (or more precisely, the i-th cell) of $\mathcal{FD}_j$.

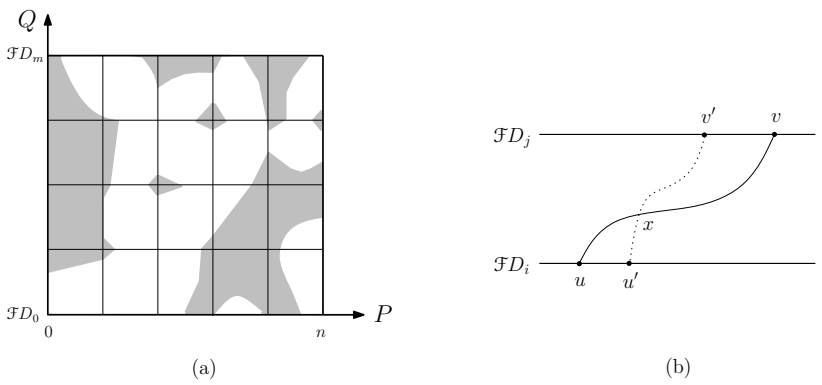

Fig. 1. (a) An example of a free-space diagram. (b) Proof of the crossing lemma.

For an interval I on the line, we denote by $\text{left}(I)$ and $\text{right}(I)$ the left and the right endpoint of I, respectively. Given two points a and b in the plane, we write $a < b$ if $a_x < b_x$. The following simple lemma serves as a building block in our algorithm.

Lemma 1. *Given two sequences A and B of points on a line sorted from left to right, we can preprocess the two sequences into a data structure of size $O(|A|)$ in $O(|A| + |B|)$ time, such that for any query point $a \in A$, the leftmost point $b \in B$ with $a < b$ can be reported in $O(1)$ time.*

3 The Data Structure

Throughout this section, let P and Q be two polygonal curves of size n and m, respectively, and $\varepsilon \geqslant 0$ be a fixed parameter. We call a curve *feasible* if it lies completely within $\mathcal{F}_\varepsilon(P, Q)$. We call the curve *monotone* if it is monotone in both x- and y-coordinates. Alt and Godau [3] showed that $\delta_F(P, Q) \leqslant \varepsilon$ if and only if there is a monotone feasible curve in $\mathcal{F}_\varepsilon(P, Q)$ from $(0, 0)$ to (n, m).

For $0 \leqslant j \leqslant m$, let $\mathcal{F}_j$ be the set of feasible points in $\mathcal{FD}_j$. $\mathcal{F}_j$ consists of $O(n)$ feasible intervals, where each feasible interval is a maximal continuous set of feasible points, restricted to be within a cell.

Given two points u and v in the free space, we say that v is *reachable* from u, denoted by $u \rightsquigarrow v$, if there is a monotone feasible curve in $\mathcal{F}_\varepsilon(P, Q)$ from u to v. Clearly, reachability is "transitive": if $u \rightsquigarrow v$ and $v \rightsquigarrow w$, then $u \rightsquigarrow w$.

Lemma 2 (Crossing Lemma). *Let $u, u' \in \mathcal{F}_i$ and $v, v' \in \mathcal{F}_j$ $(i < j)$ such that $u \leqslant u'$ and $v' \leqslant v$. If $u \rightsquigarrow v$ and $u' \rightsquigarrow v'$, then $u \rightsquigarrow v'$ and $u' \rightsquigarrow v$.*

Proof. Let π be a monotone feasible curve that connects u to v. Since u' and v' are in different sides of π, any monotone curve that connects u' to v' in $\mathcal{F}_\varepsilon(P, Q)$ intersects π at some point x (see Figure 1.b). The concatenation of the subcurve from u to x and the one from x to v' gives a monotone feasible curve from u to v'. Similarly, v is connected to u' by a monotone feasible curve through x. □

Let S be a set of points in $\mathcal{F}_\varepsilon(P,Q)$. For $0 \leqslant j \leqslant m$, we define

$$\mathcal{R}_j(S) := \{v \in \mathcal{F}_j \mid \exists u \in S \text{ s.t. } u \rightsquigarrow v\}.$$

For an interval I on $\mathcal{F}_i$ with $i \leqslant j$, we define the *left pointer* of I on $\mathcal{F}_j$, denoted by $\ell_j(I)$, to be the leftmost point in $\mathcal{R}_j(I)$. Similarly, the *right pointer* of I on $\mathcal{F}_j$, denoted by $r_j(I)$, is defined to be the rightmost point in $\mathcal{R}_j(I)$. If $\mathcal{R}_j(I)$ is empty, both pointers $\ell_j(I)$ and $r_j(I)$ are set to null. Such pointers have been previously used in [2,3]. For a single point u, we use $\mathcal{R}_j(u)$, $\ell_j(u)$, and $r_j(u)$ instead of $\mathcal{R}_j(\{u\})$, $\ell_j(\{u\})$, and $r_j(\{u\})$, respectively.

For $0 \leqslant j \leqslant m$, we define the *reachable set* $\mathcal{R}(j) := \mathcal{R}_j(\mathcal{F}_0)$ to be the set of all points in $\mathcal{F}_j$ reachable from $\mathcal{F}_0$. We call each interval of $\mathcal{R}(j)$, contained in a feasible interval of $\mathcal{F}_j$, a *reachable interval*. It is clear by our definition that $\mathcal{R}(0) = \mathcal{F}_0$.

Observation 1 *For $0 \leqslant i \leqslant j \leqslant m$, we have $\mathcal{R}(j) = \mathcal{R}_j(\mathcal{R}(i))$.*

The following lemma describes an important property of reachable sets.

Lemma 3. *For any two indices i, j $(0 \leqslant i \leqslant j \leqslant m)$ and any point $u \in \mathcal{R}(i)$, $\mathcal{R}_j(u) = \mathcal{R}(j) \cap [\ell_j(u), r_j(u)]$.*

Proof. Let $S = [\ell_j(u), r_j(u)]$. By Observation 1, $\mathcal{R}(j) = \mathcal{R}_j(\mathcal{R}(i))$. Thus, it is clear by the definition of pointers that $\mathcal{R}_j(u) \subseteq \mathcal{R}(j) \cap S$. Therefore, it remains to show that $\mathcal{R}(j) \cap S \subseteq \mathcal{R}_j(u)$. Suppose, by way of contradiction, that there is a point $v \in \mathcal{R}(j) \cap S$ such that $v \notin \mathcal{R}_j(u)$. Since $v \in \mathcal{R}(j)$, there exists some point $u' \in \mathcal{R}(i)$ such that $u' \rightsquigarrow v$. If u' is to the left (resp., to the right) of u, then the points u, u', v, and $\ell_j(u)$ (resp., $r_j(u)$) satisfy the conditions of Lemma 2. Therefore, by Lemma 2, $u \rightsquigarrow v$, which implies that $v \in \mathcal{R}_j(u)$; a contradiction. □

Lemma 3 provides an efficient way for storing the reachable sets $\mathcal{R}_j(I)$, for all feasible intervals I on $\mathcal{F}_0$: instead of storing each reachable set $\mathcal{R}_j(I)$ separately, one set per feasible interval I, which takes up to $\Theta(n^2)$ space, we only need to store a single set $\mathcal{R}(j)$, along with the pointers $\ell_j(I)$ and $r_j(I)$ which takes only $O(n)$ space in total. The reachable set $\mathcal{R}_j(I)$, for each interval I on $\mathcal{F}_0$, can be then obtained by $\mathcal{R}(j) \cap [\ell_j(I), r_j(I)]$. For each interval I on $\mathcal{F}_0$, we call the set $\{\ell_j(I), r_j(I)\}$ a *compact representation* of $\mathcal{R}_j(I)$.

Lemma 4. *For $0 < j \leqslant m$, if $\mathcal{R}(j-1)$ is given, then $\mathcal{R}(j)$ can be computed in $O(n)$ time.*

Proof. Let $\mathcal{D}$ be the restriction of the free space diagram to the rectangle $[1,n] \times [j-1, j]$. Alt *et al.* [2] showed that for all feasible intervals I on the bottom side of $\mathcal{D}$ (i.e., on $\mathcal{F}_{j-1}$), the left and the right pointers of I on the top side of $\mathcal{D}$ (i.e., on $\mathcal{F}_j$) can be computed using a series of linear scans in $O(n)$ time. Let $\mathcal{D}'$ be a copy of $\mathcal{D}$ in which all points in $\mathcal{F}_{j-1} \setminus \mathcal{R}(j-1)$ are marked infeasible. $\mathcal{D}'$ can be computed from $\mathcal{D}$ in $O(n)$ time. By running the algorithm of [2] on $\mathcal{D}'$, we

obtain all pointers $\ell_j(I)$ and $r_j(I)$, for all reachable intervals I on $\mathcal{R}(j-1)$, in $O(n)$ total time. Now, we can produce $\mathcal{R}_j(\mathcal{R}(j-1))$ easily by identifying those (portions of) intervals on $\mathcal{F}_j$ that lie in at least one interval $[\ell_j(I), r_j(I)]$. Since for all intervals I on $\mathcal{R}(j-1)$ sorted from left to right, $\ell_j(I)$'s and $r_j(I)$'s are in sorted order (this is an easy corollary of the Lemma 2), we can accomplish this step by a linear scan over the left and right pointers in $O(n)$ time. The proof is complete, as $\mathcal{R}(j) = \mathcal{R}_j(\mathcal{R}(j-1))$ by Observation 1. □

We now describe our main data structure, which we call *free-space map*. The data structure maintains reachability information on each row of the free-space diagram, using some additional pointers that help answering reachability queries efficiently. The free-space map of two curves P and Q consists of the following:

(i) the reachable sets $\mathcal{R}(j)$, for $0 \leqslant j \leqslant m$,
(ii) the right pointer $r_j(I)$ for each reachable interval I on $\mathcal{R}(j-1)$, $0 < j \leqslant m$,
(iii) the next reachable point for each cell in $\mathcal{FD}_j$, for $0 < j \leqslant m$, and
(iv) the previous take-off point for each cell in $\mathcal{FD}_j$, for $0 \leqslant j < m$,

where a *take-off* point on $\mathcal{FD}_j$ is a reachable point in $\mathcal{R}(j)$ from which a point on $\mathcal{FD}_{j+1}$ is reachable. For example, in Figure 2, ℓ_j is the next reachable point of ℓ', and r' is the previous take-off point of r_{j-1}.

Theorem 1. *Given two polygonal curves P and Q of sizes n and m, respectively, we can build the free-space map of P and Q in $O(nm)$ time.*

Proof. We start from $\mathcal{R}(0) = \mathcal{F}_0$, and construct each $\mathcal{R}(j)$ iteratively from $\mathcal{R}(j-1)$, for j from 1 to m, using Lemma 4. The total time needed for this step is $O(nm)$. The construction of $\mathcal{R}(j)$, as seen in the proof of Lemma 4, involves computing all right (and left) pointers, for all reachable intervals on $\mathcal{R}(j-1)$. Therefore, item (ii) of the data structure can be obtained at no additional cost. Item (iii) is computed as follows. Let S be the set of all left pointers obtained upon constructing $\mathcal{R}(j)$. For each cell c in $\mathcal{FD}_j$, the next reachable point of c, if any, is a member of S. We can therefore compute item (iii) for each row $\mathcal{FD}_j$ by a linear scan over the cells and the set S using Lemma 1 in $O(n)$ time. For each row, item (iv) can be computed analogous to item (iii), but in a reverse order. Namely, given the set $\mathcal{R}(j)$, we compute the set of points on $\mathcal{FD}_{j-1}$ reachable from $\mathcal{R}(j)$ in the free-space diagram rotated by 180 degrees. Let S be the set of all left pointers obtained in this reverse computation. For each cell c in $\mathcal{FD}_{j-1}$, the previous take-off point of c, if there is any, is a member of S. We can therefore compute item (iv) for each row by a linear scan over the cells and the set S using Lemma 1 in $O(n)$ time. The total time for constructing the free-space map is therefore $O(nm)$. □

In the following, we show how the reachability queries can be efficiently answered, using the free-space map. For the sake of describing the query algorithm, we introduce two functions as follows. Given a point $u \in \mathcal{FD}_j$, we define LEFTMOST-REACHABLE(u) to be the leftmost point on or after u on $\mathcal{FD}_j$. Analogously, we define RIGHTMOST-TAKE-OFF(u) to be the rightmost take-off point

Algorithm 1 QUERY(u), where $u \in \mathcal{F}_0$

```
1: let ℓ_0 = r_0 = u
2: for j = 1 to m do
3:     let ℓ' be the orthogonal projection of ℓ_{j−1} onto FD_j
4:     ℓ_j ← LEFTMOST-REACHABLE(ℓ')
5:     let r' = RIGHTMOST-TAKE-OFF(r_{j−1})
6:     if r' < ℓ_{j−1} or r' = null then
7:         r_j ← null
8:     else
9:         r_j ← r_j(I), for I being the reachable interval containing r'
10:    if ℓ_j or r_j is null then
11:        return null
12: return ℓ_m, r_m
```

on or before u on $\mathcal{FD}_j$. Note that both these functions can be computed in $O(1)$ time using the pointers stored in the free-space map.

Theorem 2. *Let the free-space map of P and Q be given. Then, for any query point $u \in \mathcal{F}_0$, $\ell_m(u)$ and $r_m(u)$ can be computed in $O(m)$ time.*

Proof. The procedure for computing $\ell_m(u)$ and $r_m(u)$ for a query point $u \in \mathcal{F}_0$ is described in Algorithm 1. The following invariant holds during the execution of the algorithm: After the j-th iteration, $\ell_j = \ell_j(u)$ and $r_j = r_j(u)$. We prove this by induction on j. The base case, $\ell_0 = r_0 = u$, trivially holds. Now, suppose inductively that $\ell_{j-1} = \ell_{j-1}(u)$ and $r_{j-1} = r_{j-1}(u)$. We show that after the j-th iteration, the invariant holds for j. We assume, w.l.o.g., that $\mathcal{R}_j(u)$ is non-empty, i.e., $\ell_j(u) \leqslant r_j(u)$. Otherwise, the last take-off point from $\mathcal{R}(j-1)$ will be either null, or smaller than r_{j-1}, which is then detected and handled by lines 6–7.

We first show that $\ell_j = \ell_j(u)$. Suppose by contradiction that $\ell_j \neq \ell_j(u)$. If $\ell_j < \ell_j(u)$, then we draw a vertical line from ℓ_j to $\mathcal{FD}_{j-1}$ (see Figure 2). This line crosses any monotone path from $\ell_{j-1} = \ell_{j-1}(u)$ to $\ell_j(u)$ at a point x. The line segment $x\ell_j$ is completely in the free space, because otherwise, it must be cut by an obstacle, which contradicts the fact that the free space inside a cell is convex. But then, ℓ_j becomes reachable from ℓ_{j-1} through x, contradicting

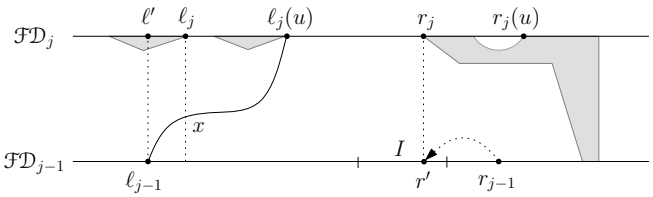

Fig. 2. Proof of Theorem 2.

the fact that $\ell_j(u)$ is the leftmost reachable point in $\mathcal{R}(j)$. The case, $\ell_j > \ell_j(u)$, cannot arise, because then, $\ell_j(u)$ is a reachable point after ℓ' and before ℓ_j, which contradicts our selection of ℓ_j as the leftmost reachable point of ℓ' in line 4.

We can similarly show that $r_j = r_j(u)$. Suppose by contradiction that $r_j \neq r_j(u)$. The case $r_j > r_j(u)$ is impossible, because then, r_j is a point on $\mathcal{R}(j)$ reachable from $\mathcal{R}(j-1)$ which is after $r_j(u)$. This contradicts the fact that $r_j(u)$ is the rightmost point on $\mathcal{R}(j)$. If $r_j < r_j(u)$ (see Figure 2), then $r_j(u)$ is reachable from a point $x \in \mathcal{R}(j-1)$ with $x < r'$, because r' is the rightmost take-off point on or before r_{j-1}. But then, by Lemma 2, $r_j(u)$ is reachable from r', which contradicts the fact that r_j is the left pointer of the reachable interval I containing r'. □

Theorem 3. *Given two polygonal curves P and Q of size n and m, respectively, we can build in $O(nm)$ time a data structure of size $O(nm)$, such that for any query point $u \in \mathcal{F}_0$, a compact representation of $\mathcal{R}_m(u)$ can be reported in $O(m)$ time.*

Remark The query time in Theorem 3 can be improved to $O(\log m)$, as shown in the appendix. However, we only use the $O(m)$ query time for the applications provided in the next section.

4 Applications

In this section, we provide some of the applications of the free-space map.

Partial Curve Matching Given two polygonal curves P and Q, and an $\varepsilon \geqslant 0$, the partial curve matching problem involves deciding whether there exists a subcurve $R \subseteq P$ such that $\delta_F(R, Q) \leqslant \varepsilon$. As noted in [3], this is equivalent to deciding whether there exists a monotone path in the free space from $\mathcal{FD}_0$ to $\mathcal{FD}_m$. This decision problem can be efficiently solved using the free-space map. For each feasible intervals I on $\mathcal{FD}_0$, we compute a compact representation of $\mathcal{R}_m(\text{left}(I))$ using Theorem 3 in $O(m)$ time. Observe that $\mathcal{R}_m(I) = \emptyset$ if and only if $\mathcal{R}_m(\text{left}(I)) = \emptyset$. Therefore, we can decide in $O(nm)$ time whether there exists a point on $\mathcal{FD}_m$ reachable from $\mathcal{FD}_0$. Furthermore, we can use parametric search as in [3] to find the smallest ε for which the answer to the above decision problem is "yes" in $O(nm \log(nm))$ time. Therefore, we obtain:

Theorem 4. *Given two polygonal curves P and Q of size n and m, respectively, we can decide in $O(nm)$ time whether there exists a subcurve $R \subseteq P$ such that $\delta_F(R, Q) \leqslant \varepsilon$, for a given $\varepsilon \geqslant 0$. A subcurve $R \subseteq P$ minimizing $\delta_F(R, Q)$ can be computed in $O(nm \log(nm))$ time.*

Closed Curves An important variant of the Fréchet metric considered by Alt and Godau [3] is the following. Given two closed curves P and Q, define

$$\delta_C(P, Q) = \inf_{s_1, s_2 \in \mathbb{R}} \delta_F(R \text{ shifted by } s_1, Q \text{ shifted by } s_2)$$

to be the *closed Fréchet metric* between P and Q. This metric is of significant importance for comparing shapes.

Consider a diagram $\mathcal{D}$ of size $2n \times m$ obtained from concatenating two copies of the standard free-space diagram of P and Q. Alt and Godau showed that $\delta_C(P,Q) \leqslant \varepsilon$ if and only if there exists a monotone feasible path in $\mathcal{D}$ from $(t,0)$ to $(n+t,m)$, for a value $t \in [0,n]$. We show how such a value t, if there is any, can be found efficiently using a free-space map built on top of $\mathcal{D}$.

Observation 2 *Let i be a fixed integer $(0 < i \leqslant n)$, $I = [a,b]$ be the feasible interval on the i-th cell of $\mathcal{FD}_0$, and $J = [c,d]$ be the feasible interval on the $(i+n)$-th cell of $\mathcal{FD}_m$. Then there is a value $t \in [i-1,i]$ with $(t,0) \rightsquigarrow (n+t,m)$ if and only if $\max((\ell_m(I))_x, c) \leqslant b+n$ and $\min((r_m(I))_x, d) \geqslant a+n$.*

We iterate on i from 1 to n, and check for each i if a desired value $t \in [i-1,i]$ exists using Observation 2. Each iteration involves the computation of $\ell_m(I)$ and $r_m(I)$ that can be done in $O(m)$ time using the free-space map. The total time is therefore $O(nm)$.

Theorem 5. *Given two closed polygonal curves P and Q of size n and m, respectively, we can decide in $O(nm)$ time whether $\delta_C(P,Q) \leqslant \varepsilon$, for a given $\varepsilon \geqslant 0$. Furthermore, $\delta_C(P,Q)$ can be computed in $O(nm \log(nm))$ time.*

Maximum Walk An interesting variant of the Fréchet distance problem is the following: Given two curves P and Q and a fixed $\varepsilon \geqslant 0$, find a maximum-length subcurve of Q whose Fréchet distance to P does not exceed ε. In the dog-person illustration, this problem corresponds to finding the best starting point on P such that when the person walks the whole curve Q, his or her dog can walk the maximum length on P, without exceeding a leash of length ε. We show that this optimization problem can be solved efficiently in $O(nm)$ time using the free space map. The following observation is the main ingredient.

Observation 3 *Let R be a maximum-length subcurve of P such that $\delta_F(R,Q) \leqslant \varepsilon$. The starting point of R corresponds to the left endpoint of a feasible interval I on $\mathcal{FD}_0$, and its ending point corresponds to $r_m(I)$.*

By Observation 3, we only need to test n feasible intervals on $\mathcal{FD}_0$, and their right pointer on $\mathcal{FD}_m$ to find the best subcurve R. Note that, given the two endpoints of a subcurve R on the free-space map, finding the actual length of R on the original curve P can take up to $O(n)$ time. To speed up the length computation step, we can use a table lookup method used by the authors in [10] to answer each length query in $O(1)$ time, after $O(n)$ preprocessing. The total time for computing the maximum-length subcurve R will be therefore $O(nm)$.

Theorem 6. *Given two polygonal curves P and Q of size n and m, respectively, and a parameter $\varepsilon \geqslant 0$, we can find in $O(nm)$ time a maximum-length subcurve $R \subseteq P$ such that $\delta_F(R,Q) \leqslant \varepsilon$.*

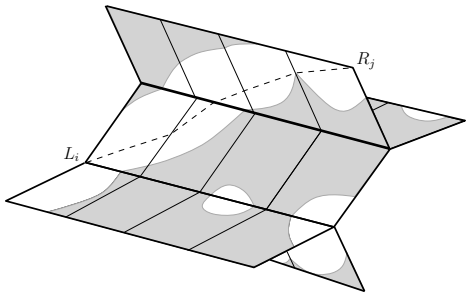

Fig. 3. An example of a free-space surface.

5 Matching a Curve in a DAG

Let P be a polygonal curve of size n, and G be a connected graph with m edges. Alt *et al.* [2] presented an $O(nm \log m)$-time algorithm to decide whether there is a path π in G with Fréchet distance at most ε to P, for a given $\varepsilon \geqslant 0$. In this section, we improve this result for the case when G is a directed acyclic graph (DAG), by giving an algorithm that runs in only $O(nm)$ time. The idea is to use a sequential reachability propagation approach similar to the one used in Section 3. Our approach is structurally different from the one used by Alt *et al.* [2]. In particular, their algorithm has $\Theta(nm \log m)$ running time in the worst case even if the input graph is a DAG or a simple path.

We first borrow some notation from [2]. Let $G = (V, E)$ be a connected DAG with m edges, such that $V = \{1, \ldots, \nu\}$ corresponds to points $\{v_1, \ldots, v_\nu\} \subseteq \mathbb{R}^2$, for $\nu \leqslant m+1$. We assume, w.l.o.g., that the elements of V are numbered according to a topological ordering of the vertices of G. Such a topological ordering can be computed in $O(m)$ time. We embed each edge $(i, j) \in E$ as an oriented line segment s_{ij} from v_i to v_j. Each s_{ij} is continuously parametrized by values in $[0, 1]$ according to its natural parametrization, namely, $s_{ij} : [0, 1] \to \mathbb{R}^2$.

For each vertex $j \in V$, let $\mathcal{FD}_j := \mathcal{FD}_\varepsilon(P, v_j)$ be the one-dimensional free-space diagram corresponding to the path P and the vertex j. We denote by L_j and R_j the left endpoint and the right endpoint of $\mathcal{FD}_j$, respectively. Moreover, we denote by $\mathcal{F}_j$ the set of feasible points on $\mathcal{FD}_j$. For each $(i, j) \in E$, let $\mathcal{FD}_{ij} := \mathcal{FD}_\varepsilon(P, s_{ij})$ be a two-dimensional free-space diagram, which consists of a row of n cells. We glue together the two-dimensional free-space diagrams according to the adjacency information of G, as shown in Figure 3. The resulting structure is called the *free-space surface* of P and G, denoted by $\mathcal{FS}_\varepsilon(P, G)$. We denote the set of feasible points in $\mathcal{FS}_\varepsilon(P, G)$ by $\mathcal{F}_\varepsilon(P, G)$.

Given two points $u, v \in \mathcal{F}_\varepsilon(P, G)$, we say that v is *reachable* from u, denoted by $u \rightsquigarrow v$, if there is a monotone feasible curve from u to v in $\mathcal{F}_\varepsilon(P, G)$, where monotonicity in each cell of the surface is with respect to the orientation of the edges of P and G defining that cell. Given a set of points $S \subseteq \mathcal{F}_\varepsilon(P, G)$, we define $\mathcal{R}_j(S) := \{v \in \mathcal{F}_j \mid \exists u \in S \text{ s.t. } u \rightsquigarrow v\}$. Let $\mathbb{L} = \cup_{j \in V}(L_j \cap F_j)$. For each

Algorithm 2 DECISION-DAG-MATCHING(P, G, ε)

1: **for each** $j \in V$ in a topological order **do**
2: $\quad \mathcal{R}(j) \leftarrow \mathcal{R}_j(L_j \cap \mathcal{F}_j) \cup (\cup_{(i,j)\in E} \mathcal{R}_j(\mathcal{R}(i)))$
3: let $S = \cup_{j\in V}(R_j \cap \mathcal{R}(j))$
4: return TRUE if $S \neq \emptyset$, otherwise return FALSE

$j \in V$, we define the *reachable set* $\mathcal{R}(j) := \mathcal{R}_j(\mathbb{L})$. Observe that there is a path π in G with $\delta_F(P, \pi) \leqslant \varepsilon$ if and only if there is a vertex $j \in V$ with $R_j \in \mathcal{R}(j)$.

Theorem 7. *Given a polygonal curve P of size n and a directed acyclic graph G of size m, we can decide in $O(nm)$ time whether there is a path π in G with $\delta_F(P, \pi) \leqslant \varepsilon$, for a given $\varepsilon \geqslant 0$. A path π in G minimizing $\delta_F(P, \pi)$ can be computed in $O(nm \log(nm))$ time.*

Proof. Algorithm 2 computes, for each vertex $j \in V$, the reachable set $\mathcal{R}(j)$ in a topological order. It then returns true only if there is a vertex $j \in V$ such that R_j is reachable which indicates the existence of a path π in G with $\delta_F(P, \pi) \leqslant \varepsilon$. To prove the correctness, we only need to show that for every vertex $j \in V$, the algorithm computes $\mathcal{R}(j)$ correctly. We prove this by induction on j. Suppose by induction that the set $\mathcal{R}(i)$ for all $i < j$ is computed correctly. Now consider a point $u \in F_j$. If $u \in \mathcal{R}(j)$, then there exists a vertex $k < j$ such that L_k is connected to u by a monotone feasible curve $\mathcal{C}$ in $\mathcal{FS}_\varepsilon(P, G)$. If $k = j$, then $u \in \mathcal{R}(j)$ because $\mathcal{R}_j(L_j \cap \mathcal{F}_j)$ is added to $\mathcal{R}(j)$ in line 2. If $k < j$, then the curve $\mathcal{C}$ must pass through a vertex i with $(i, j) \in E$. Since the vertices of V are sorted in a topological order, we have $i < j$, and hence, $\mathcal{R}(i)$ is computed correctly by the induction hypothesis. Hence, letting $x = \mathcal{C} \cap \mathcal{F}_i$, we have $x \in \mathcal{R}(i)$. Furthermore, we know that x is connected to u using the curve $\mathcal{C}$. Therefore, the point u is in $\mathcal{R}_j(\mathcal{R}(i))$, and hence, is added to $\mathcal{R}(j)$ in line 2. Similarly, we can show that if $u \notin \mathcal{R}(j)$, then u is not added to $\mathcal{R}(j)$ by the algorithm. Suppose by contradiction that u is added to $\mathcal{R}(j)$ in line 2. Then either $u \in \mathcal{R}_j(L_j \cap \mathcal{F}_j)$ or $u \in \mathcal{R}_j(\mathcal{R}(i))$, for some $i < j$. But by the definition of reachability, both cases imply that u is reachable from a point in $\mathbb{L}$, which is a contradiction.

For the time complexity, note that each $\mathcal{R}_j(\mathcal{R}(i))$ in line 2 can be computed in $O(n)$ time using Lemma 4. Moreover, $\mathcal{R}_j(L_j \cap \mathcal{F}_j)$, for each $j \in V$, can be computed by finding the largest feasible interval on F_j containing L_j in $O(n)$ time. Therefore, processing each edge (i, j) takes $O(n)$ time, and hence, the whole computation takes $O(nm)$ time. Once the algorithm finds a reachable left endpoint v, we can construct a feasible monotone path connecting a right endpoint $u \in \mathbb{L}$ to v by keeping, for each reachable interval I on $R(j)$, a back pointer to a reachable interval J on $R(i)$, $(i, j) \in E$, from which I is reachable. The path $u \rightsquigarrow v$ can be constructed by following the back pointers from v to u, in $O(m)$ time. For the optimization problem, we use parametric search as in [2, 3], to find the value of $\delta_F(P, \pi)$ by an extra $\log(nm)$-factor, namely, in $O(nm \log(nm))$ time. □

Remark As noted in [2], it is straight-forward to modify the algorithm to allow paths in G to start and end anywhere inside edges of the graph, not necessarily at the vertices. This can be easily done by allowing the feasible path found by our algorithm to start and end at any feasible point on the left and right boundary of $\mathcal{FD}_{ij}$, for each edge $(i, j) \in E$.

6 Conclusions

In this paper, we presented improved algorithms for some variants of the Fréchet distance problem, including partial curve matching, closed Fréchet distance, maximum walk, and matching a curve in a DAG. Our improved results are based on a new data structure, called free-space map, that might be applicable to other problems involving the Fréchet metric. It remains open whether the same improvements obtained here can be achieved for matching curves inside general graphs (a similar improvement is recently obtained for complete graphs in [11]). Proving a lower bound better than $\Omega(n \log n)$ for variants of the Fréchet distance studied in this paper is another major problem left open.

References

1. H. Alt. The computational geometry of comparing shapes. *Efficient Algorithms*, pages 235–248, 2009.
2. H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. Algorithms*, 49(2):262–283, 2003.
3. H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. of Comput. Geom. Appl.*, 5:75–91, 1995.
4. K. Buchin, M. Buchin, and J. Gudmundsson. Constrained free space diagrams: a tool for trajectory analysis. *Int. J. of Geogr. Inform. Sci.*, 24(7):1101–1125, 2010.
5. K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? In *Proc. 23rd EWCG*, pages 170–173, 2007.
6. K. Buchin, M. Buchin, and Y. Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th ACM-SIAM Sympos. Discrete Algorithms*, pages 645–654, 2009.
7. A. F. Cook and C. Wenk. Geodesic Fréchet distance inside a simple polygon. In *Proc. 25th Sympos. Theoret. Aspects Comput. Sci.*, volume 5664 of *Lecture Notes Comput. Sci.*, pages 193–204, 2008.
8. A. Efrat, L. J. Guibas, S. Har-Peled, J. S. B. Mitchell, and T. M. Murali. New similarity measures between polylines with applications to morphing and polygon sweeping. *Discrete Comput. Geom.*, 28(4):535–569, 2002.
9. M. Jiang, Y. Xu, and B. Zhu. Protein structure-structure alignment with discrete Fréchet distance. *J. Bioinform. Comput. Biol.*, 6(1):51–64, 2008.
10. A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Fréchet distance with speed limits. *Comput. Geom. Theory Appl.*, 44(2):110–120, 2011.
11. A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh. Staying close to a curve. In *Proc. 23rd Canad. Conf. Computat. Geom.*, 2011, to appear.
12. E. Sriraghavendra, K. Karthik, and C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proc. 9th Internat. Conf. Document Anal. Recognition*, pages 461–465, 2007.

A Improved Query Time

In this appendix, we show how the query time in the free-space map can be improved from $O(m)$ to $O(\log m)$, without increasing the preprocessing time and space. This improved query time is crucial for applications such as the minimum walk problem.

As a subproblem in our improved solution, we need to efficiently solve a special case of an offline vertical ray shooting problem, which is defined as follows. Consider a vertical slab $[0,1]\times[0,m]$ (see Figure 4). For each $0 \leqslant i \leqslant m$, there are two (possibly empty) segments in the slab at height i, attached to the boundary of the slab, one from left and the other from right. Given a query point q, the vertical ray shooting problem involves finding the first segment in the slab directly above q. If the query points are restricted to be among the endpoints of the segments, we show that the vertical ray shooting queries can be answered in $O(1)$ time, after $O(m)$ preprocessing time.

Lemma 5. *Let S be a set of segments $s_i = [0, a_i] \times \{i\}$, and T be a set of segments $t_i = [b_i, 1] \times \{i\}$ with $0 \leqslant a_i \leqslant b_i \leqslant 1$, for $0 \leqslant i \leqslant m$. We can find for each segment $s_i \in S$ (resp., $t_i \in T$), the first segment in $S \cup T$ directly above right(s_i) (resp., left(t_i)) in $O(m)$ total time.*

Proof. Algorithm 3 performs the task. It assigns to each segment s_i, an *up* pointer that points to the first segment directly above right(s_i), if such a segment exists. The *up* pointers for segments of T can be computed analogously. The algorithm makes use of a queue Q, which is empty at the beginning, and supports the standard operations PUSH(), POP(), and TOP(), along with two additional operations BOTTOM() and BOTTOM-POP(), that are analogous to TOP() and POP(), respectively, but applied to the bottom of the queue.

We say that a segment $s \in S$ is *covered* by a segment $t \in S \cup T$, if a vertical ray from s intersects t. Let $S_i = \{s_0, \ldots, s_i\}$ and $T_i = \{t_0, \ldots, t_i\}$, for $0 \leqslant i \leqslant m$. The following invariant is maintained by the algorithm: At the end of iteration i, Q contains a subset of segments from S_i that are not covered by any segment from $S_i \cup T_i$, in a decreasing order of their lengths, from bottom to the top. The invariant clearly holds for $i = 0$. Suppose by induction that the

Algorithm 3 RAY-SHOOTING(S, T)

```
1: Q ← ∅
2: for i from 0 to m do
3:     while |Q.TOP()| ⩽ |s_i| do
4:         Q.POP().up ← s_i
5:     while |Q.BOTTOM()| ⩾ 1 − |t_i| do
6:         Q.BOTTOM-POP().up ← t_i
7:     Q.PUSH(s_i)
```

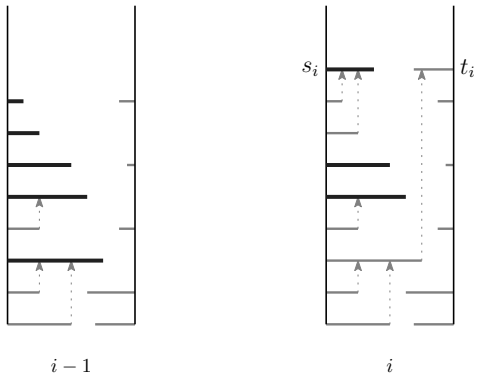

Fig. 4. An example of the execution of Algorithm 3 for two steps $i-1$ and i. Segments in the queue are shown in bold.

invariant holds for $i-1$. In the i-th iteration, we first pop out from the top of the queue all segments that are covered by s_i, in lines 3–4. Then, we remove from the bottom of the queue all segments covered by t_i, in lines 5–6. Finally, we add s_i to the top of the queue. (See Figure 4 for an illustration.) It is easy to verify that after the insertion of s_i, the segments of Q are still sorted in a decreasing order of their lengths (because we have already removed segments smaller than s_i from Q), and that, no segment of Q is covered by a segment in $S_i \cup T_i$ (because we have removed covered segments from Q). Furthermore, it is clear that any segment s removed from Q is assigned to the first segment that is directly above $\mathrm{right}(s)$, because we are processing segments in order from bottom to the top. The correctness of the algorithm therefore follows. Note that after the termination of the algorithm, Q still contains some uncovered segments from S, whose *up* pointers are assumed to be null, as they are not covered by any segment in $S \cup T$. Since each segment of S is inserted into and removed from the queue at most once, lines 4 and 6 of the algorithm are executed at most m times, and hence, the whole algorithm runs in $O(m)$ time. □

Remark Alt *et al.* [2] have employed a very similar idea to compute the left and right pointers in a row of cells in the free-space diagram.

Theorem 8. *Given two polygonal curves P and Q of size n and m, respectively, we can build in $O(nm)$ time a data structure of size $O(nm)$, such that for any query point $u \in \mathcal{F}_0$, a compact representation of $\mathcal{R}_m(u)$ can be reported in $O(\log m)$ time.*

Proof. We first build a free-space map as in Theorem 3 in $O(nm)$ time. Let I be a feasible interval on $\mathcal{F}_0$. For each $u \in I$, we have $r_m(u) = r_m(I) = r_m(\mathrm{right}(I))$. Therefore, by storing $r_m(I)$ for all feasible intervals I on $\mathcal{F}_0$, we can report $r_m(u)$ for each query point $u \in \mathcal{F}_0$ in $O(1)$ time. Since there are $O(n)$ feasible intervals

on $\mathcal{F}_0$, and computing each right pointer takes $O(m)$ time by Theorem 2, this step takes $O(nm)$ time in total. To report $\ell_m(u)$ quickly, we store for each reachable interval $I \in \mathcal{R}(j)$, $0 < j < m$, the pointer $\ell_m(I)$ in the data structure. We can compute all these left pointers in $O(nm)$ time as follows. We first preprocess each column of the free-space map for vertical ray shooting as in Lemma 5, by assuming horizontal segments to be non-reachable intervals on each row $\mathcal{FD}_j$. To compute left pointers, we inductively process the free-space map from top to bottom. Suppose that the left pointers are computed and stored for all reachable intervals above $\mathcal{FD}_j$, and let I be a reachable interval on $\mathcal{FD}_j$, with $q = \text{left}(I)$. We can find the first non-reachable segment s above q using our ray shooting data structure in $O(1)$ time. If no such s exists, $\ell_m(q)$ is directly above q on $\mathcal{R}(m)$. Otherwise, as in Algorithm 1, we project q directly to a point $q' \in s$, and then, find the first reachable point p after q'. If such a point p exists, it should be the left endpoint of a reachable interval I', for which we have already stored the pointer $\ell_m(\text{left}(I'))$. Therefore, $\ell_m(q) = \ell_m(\text{left}(I'))$ can be computed in $O(1)$ time. As a result, finding all left pointers takes $O(n)$ time for each $\mathcal{FD}_j$, and $O(nm)$ time for the whole free-space map.

For each feasible interval I on $\mathcal{F}_0$, we partition I into $O(m)$ subintervals, such that for all points u in a subinterval, the first segment directly above u (in the ray shooting data structure) is the same. Such a partitioning can be easily obtained by scanning each column of the free-space map from bottom to the top. The total number of subintervals obtained this way is $O(nm)$. Now, for each subinterval J on $\mathcal{F}_0$, we compute $\ell_m(J)$ in the same way described above in $O(1)$ time. Namely, we find the unique segment s above J, find the first reachable point p after s, and take the pointer $\ell_m(p)$, which is stored in the data structure. The total time and space needed for this step is therefore $O(nm)$. For any query point $u \in \mathcal{F}_0$, we first locate the subinterval J containing u in $O(\log m)$ time. Now, $\ell_m(u) = \ell_m(J)$ and $r_m(u) = r_m(I)$ for the feasible interval I containing subinterval J, both accessible in $O(1)$ time. □

Note that the only expensive operation in our query algorithm is to locate the subinterval containing the query point. If the subinterval is given, then the query can be answered in $O(1)$ time.

Minimum Walk Given two curves P and Q and a fixed $\varepsilon \geqslant 0$, the *minimum walk* problem asks for the minimum-length subcurve of P that a person can walk while his/her dog walks the whole curve Q without exceeding a leash of length ε. This optimization problem can be solved efficiently using our extended data structure.

Theorem 9. *Given two polygonal curves P and Q of size n and m, respectively, and a parameter $\varepsilon \geqslant 0$, we can find in $O(nm)$ time a minimum-length subcurve $R \subseteq P$ such that $\delta_F(R, Q) \leqslant \varepsilon$.*

Proof. Let R be a minimum-length subcurve of P such that $\delta_F(R, Q) \leqslant \varepsilon$. It is easy to verify that the starting point of R corresponds to the right endpoint of a subinterval J on $\mathcal{F}_0$, and its ending point corresponds to $\ell_m(J)$. Therefore, to

find the best subcurve R, we only need to check the right endpoints of $O(nm)$ subintervals on $\mathcal{FD}_0$ and their corresponding left pointers. Since the left pointer for each subinterval is already stored, as shown in the proof of Theorem 8, the query time for each subinterval is $O(1)$. The total time needed is therefore $O(nm)$. □