Discrete Fréchet Distance Oracles

Boris Aronov
Department of Computer Science and Engineering, Tandon School of Engineering,
New York University, Brooklyn, NY, USA

Tsuri Farhana
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Matthew J. Katz
Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva, Israel

Indu Ramesh
Department of Computer Science and Engineering, Tandon School of Engineering,
New York University, Brooklyn, NY, USA

Abstract

It is unlikely that the discrete Fréchet distance between two curves of length n can be computed in strictly subquadratic time. We thus consider the setting where one of the curves, P, is known in advance. In particular, we wish to construct data structures (*distance oracles*) of near-linear size that support efficient distance queries with respect to P in sublinear time. Since there is evidence that this is impossible for query curves of length $\Theta(n^\alpha)$, for any $\alpha > 0$, we focus on query curves of (small) constant length, for which we are able to devise distance oracles with the desired bounds.

We extend our tools to handle subcurves of the given curve, and even arbitrary vertex-to-vertex subcurves of a given geometric tree. That is, we construct an oracle that can quickly compute the distance between a short polygonal path (the query) and a path in the preprocessed tree between two query-specified vertices. Moreover, we define a new family of geometric graphs, t-*local* graphs (which strictly contains the family of geometric spanners with constant stretch), for which a similar oracle exists: we can preprocess a graph G in the family, so that, given a query segment and a pair u, v of vertices in G, one can quickly compute the smallest discrete Fréchet distance between the segment and any (u, v)-path in G. The answer is exact, if $t = 1$, and approximate if $t > 1$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases discrete Fréchet distance, distance oracle, heavy-path decomposition, t-local graphs

Digital Object Identifier 10.4230/LIPIcs.SoCG.2024.10

Related Version *Full Version*: `https://arxiv.org/abs/2404.04065` [3]

Funding *Boris Aronov*: Work partially supported by NSF Grant CCF-20-08551. Part of the work was done while visiting Institute of Science and Technology Austria.
Tsuri Farhana: Work partially supported by the Lynne and William Frankel Center for Computer Science and by BSF Grant 2019715.
Matthew J. Katz: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
Indu Ramesh: Work supported by a Tandon School of Engineering Fellowship and by NSF Grant CCF-20-08551.

Acknowledgements We would like to thank an anonymous reviewer of an earlier version of this paper for a suggested improvement.

1 Introduction

The continuous Fréchet distance is often used as a measure of similarity between curves [2]. The discrete Fréchet distance [12] is sometimes viewed as a simplified version of the (continuous) Fréchet distance, but it is also the preferred version in some application domains, such as protein alignment (see, e.g., [19]).

Let $A = (a_1, \ldots, a_m)$ and $B = (b_1, \ldots, b_n)$ be two sequences of points in $\mathbb{R}^d$ representing polygonal curves. A *(monotone) walk* of A and B is a sequence of pairs $(c_1, \ldots, c_l)$, where (i) $c_1 = (a_1, b_1)$, (ii) $c_l = (a_m, b_n)$, and (iii) the pair succeeding $c_k = (a_i, b_j)$, for $1 \leq k < l$, is one of the following: $c_{k+1} = (a_{i+1}, b_j)$ (when $i < m$), $c_{k+1} = (a_i, b_{j+1})$ (when $j < n$), or $c_{k+1} = (a_{i+1}, b_{j+1})$ (when $i < m$ and $j < n$). Each pair $c_k = (a_i, b_j)$ in a walk $(c_1, \ldots, c_l)$ of A and B yields a distance $\|a_i - b_j\|$, and the *cost* of the walk is the maximum of these distances. The *discrete Fréchet distance* between A and B, denoted $d_{\mathrm{dF}}(A, B)$, is the minimum over the cost of all walks of A and B.

The discrete Fréchet distance between A and B can be computed in roughly $O(mn)$ time [1, 12]. It is unlikely that it can be computed exactly, or even approximated within a factor less than 3, in strictly subquadratic time [4, 6, 7]. It is therefore natural to ask whether one can do better when, e.g., one of the curves is given in advance. Indeed, let G be a geometric graph, that is, G's vertices correspond to points in the plane, and the weight of an edge of G is the Euclidean distance between the points represented by its vertices. Denote the set of paths from u to v in G, where u and v are vertices of G, by $\mathcal{P}_G(u, v)$. (If G is a tree, then $\mathcal{P}_G(u, v) = \{\Pi_{uv}\}$, where Π_{uv} is the unique path in G from u to v.) The *discrete Fréchet distance between a polygonal curve Q and G* (with respect to u and v) is $\min_{\Pi \in \mathcal{P}_G(u,v)} d_{\mathrm{dF}}(Q, \Pi)$, and we denote this distance by $d_{\mathrm{dF}}(Q, \mathcal{P}_G(u, v))$. Now, assume that we are expecting a stream of polygonal curves $Q_1, Q_2, \ldots$, each with a corresponding pair (u_i, v_i) of vertices of G, and for each arriving Q_i we need to compute the distance $d_{\mathrm{dF}}(Q_i, \mathcal{P}_G(u_i, v_i))$. We thus wish to construct a compact data structure based on G, so that given a query curve Q and two vertices u and v of G, one can compute $d_{\mathrm{dF}}(Q, \mathcal{P}_G(u, v))$ efficiently. In other words, we wish to construct a *distance oracle* for G.

To construct such a data structure, we focus on the case where the curves Q_i are of constant size, i.e., consist of a constant number of vertices (at the other extreme, when queries have size $\Theta(n^\alpha)$, for $\alpha > 0$, it may be impossible to gain anything by polynomial-time preprocessing [5, 15]). In this case, the challenge is to construct a near-linear size data structure such that given a curve Q, one can compute $d_{\mathrm{dF}}(Q, \mathcal{P}_G(u, v))$ in sublinear time. We identify several rather general settings where this is possible. Specifically, if G is a tree with n nodes, we can process query curves of size up to three in $O(\text{polylog}\, n)$ time and curves of size four in $O^*(n^{1/2})$ time; the $O^*(\cdot)$ notation hides subpolynomial factors. (We get slightly better bounds for the special case where the tree is actually a polygonal curve.) Moreover, we define a class of geometric graphs, called *1-local* graphs, which includes the Delaunay graph, for which we can answer segment queries in $O^*(n^{1/2})$ time.

Our results. We first formally state the main problem studied in this paper.

▶ Problem (Distance Oracle). Let G be a geometric graph. Construct a compact data structure such that, given a query polygonal curve Q of length (i.e., number of vertices) k and two vertices u and v of G, one can quickly compute $d_{\mathrm{dF}}(Q, \mathcal{P}_G(u, v))$.

We assume that the sets of points underlying G and Q are in the plane, and we focus on the case where k is a small constant, often between two and four. We consider three main versions of the problem, depending on the graph G.

Table 1 Distance oracles for curves. Decision problem answers questions of the form: "given Q and r, is $d_{\mathrm{dF}}(P,Q) \leq r$?" Optimization problem computes the discrete Fréchet distance. We also offer a variant where at query time one can restrict the query to an arbitrary vertex-to-vertex subcurve of P.

size of Q	decision problem		optimization problem	
k	for P	for subcurve of P	for P	for subcurve of P
1	$O(\log n)$	$O(\log^2 n)$	$O(\log n)$	$O(\log^2 n)$
2	$O(\log^2 n)$	$O(\log^2 n)$	$O(\log^2 n)$	$O(\log^2 n)$
3	$O(\log^2 n)$	$O(\log^2 n)$	$O(\log^3 n)$	$O(\log^3 n)$
4	$O^*(n^{1/2})$	$O^*(n^{1/2})$	$O^*(n^{1/2})$	$O^*(n^{1/2})$

(i) G is a polygonal curve P of length n. This is the most basic version of the problem; we summarize the results in Table 1. We state running times for both decision and optimization algorithms, depending on the number k of vertices in the query curve. For $k = 2$ (i.e., directed segments) and for $k = 3$ (i.e., three-vertex curves), we construct data structures of size $O(n \log n)$, so that $d_{\mathrm{dF}}(P,Q)$ can be computed in $O(\log^3 n)$ time, see Sections 4.1 and 4.2, respectively. The description of the data structure of size $O^*(n)$ for $k = 4$ (i.e., four-vertex curves), so that $d_{\mathrm{dF}}(P,Q)$ can be computed in $O^*(n^{1/2})$ time is deferred to the full version of the paper [3, Section 4.3]. In each of these cases, one can restrict the query to a vertex-to-vertex subcurve of P, specified at query time.

The case where the query curves are line segments was studied by Buchin et al. [8] for the continuous (rather than discrete) Fréchet distance. They presented an $O(n\kappa^{3+\varepsilon} + n^2)$-size data structure, where $\kappa \in [1,n]$ is a parameter set by the user and $\varepsilon > 0$ is an arbitrarily small constant, such that given a query segment ab one can compute the Fréchet distance between ab and P in $O((n/\kappa)\log^2 n)$ time (alternatively, between ab and a point-to-point subcurve of P, specified at query time, in $O((n/\kappa)\log^2 n + \log^4 n)$ time). Thus, to achieve polylogarithmic query time, they need a data structure of size roughly $O(n^4)$, in contrast to $O(n \log n)$ for the discrete Fréchet distance (see Sections 4.1–4.2).

It is not surprising that the bound on the size of the data structure that we obtain in the case of segment queries is much better than the bound of Buchin et al. [8]. On the other hand, it is somewhat surprising that one can obtain a polylogarithmic bound (for queries of up to three vertices) and a sublinear bound (for four-vertex queries), using near-linear space (see Table 1).

Our results for the curve version are relatively technical and we defer them to Section 4. They form the basis for the following, more general version.

(ii) G is a tree T with n nodes. The main idea is to decompose T into heavy paths [18] and use the aforementioned curve oracles. We show in Section 2 that one can construct discrete Fréchet distance oracles for a tree with n vertices with query times $O(\log^3 n)$, $O(\log^3 n)$, $O(\log^4 n)$, and $O^*(\sqrt{n})$ for query sizes one, two, three, and four, respectively, where the structures require $O(n \log n)$ space for query sizes up to three and $O^*(n)$ for query size four.

(iii) G is a local graph with n vertices and m edges. Let $t \geq 1$ be a real parameter. We say that G is *t-local* if the following condition holds: For any disk D and for any two points $p, q \in P \cap D$ (where P is the point set underlying G), there exists a path in G between p and q that does not exit tD, where tD is the disk obtained from D by scaling it by a factor of t around its center, that is, p and q are connected in the subgraph of G that is induced by the set $P \cap tD$. We say that a graph is *local* if it is t-local for some constant $t \geq 1$.

In Section 3, we first show that the class of local graphs *strictly* contains the class of geometric spanners. Next, we show that any 1-local graph contains the Delaunay triangulation (which itself is 1-local). We construct an $O^*(n+m)$-size distance oracle for a given 1-local graph (and in particular an $O^*(n)$-size oracle for the Delaunay triangulation), which handles a segment query in $O^*(n^{1/2})$ time. When $t > 1$, the oracle returns an approximation of the requested distance which depends on t.

More related work. We restrict our discussion of related work to distance oracles. In general, most of the related work deals with the continuous (rather than discrete) Fréchet distance, and with the construction of approximate oracles that return an approximation of the requested distance (rather than the exact distance). All the results below are for the continuous Fréchet distance unless mentioned otherwise.

As for exact oracles, we already mentioned the result of Buchin et al. [8] for arbitrary segment queries with respect to a given curve P. For earlier results geared to horizontal segment queries see [8, 10, 17]. Recently, Cheng and Huang [9] described a distance oracle for k-vertex query curves of size $O(kn)^{\mathrm{poly}(d,k)}$ than can process a query with respect to a point-to-point subcurve of P (specified at query time) in time $O((dk)^{O(1)}\log(kn))$.

As for approximate oracles, Filtser and Filtser [13] construct a $(1+\varepsilon)$-approximate distance oracle for a given n-vertex curve P and k-vertex query curves. Its size is $O(\frac{1}{\varepsilon})^{kd}\log\frac{1}{\varepsilon}$ and it computes a $(1+\varepsilon)$-approximation of the discrete Fréchet distance between P and a k-vertex query curve in $O^*(kd)$ time. Driemel and Har-Peled [11] present a $(1+\varepsilon)$-approximate distance oracle for segment queries (i.e., $k=2$) of size $O((\frac{1}{\varepsilon})^{2d}\cdot\log^2\frac{1}{\varepsilon})$ and query time $O(d)$. They also consider the version in which the query is with respect to a point-to-point subcurve of P, specified at query time. For this version, the size of their data structure is $O(n(\frac{1}{\varepsilon})^{2d}\cdot\log^2\frac{1}{\varepsilon})$ and the query time is $O(\varepsilon^{-2}\log n\log\log n)$. Filtser [14] considered the latter version for the discrete Fréchet distance. By adapting techniques from Driemel and Har-Peled, she constructed a data structure of the same size and query time $O(\log n)$. Finally, for general k, Driemel and Har-Peled [11] provide a constant-factor approximate distance oracle of size $O(nd\log n)$, which can answer distance queries between any subcurve of P and a k-curve query in $O(k^2 d\log n\log(k\log n))$ time.

A problem closely related to ours is the following. Construct a compact data structure for a geometric graph G, such that given a query polygonal curve Q of length k one can quickly compute the minimum Fréchet distance between Q and *any* vertex-to-vertex path in G. This is the query version of the well-known *map matching* problem. Gudmundsson and Smid [16] studied the problem for a c-packed tree T. (A set of edges is c-packed if for any disk the total length of the portions of the edges contained in the disk is at most c times the radius of the disk.) More precisely, they studied a corresponding decision problem with some additional restrictions. Recently, Gudmundsson et al. [15] studied this problem for c-packed graphs. As an intermediate result, they construct a data structure of size $O(c\,m\log m)$ for a c-packed graph G of complexity m, so that given a pair of query vertices u and v, one can return in $O(\log m)$ time a 3-approximation of the Fréchet distance between Q and $\mathcal{P}_G(u,v)$. The preprocessing time is $O(c\,m^2\log^2 m)$.

2 Distance oracles for trees

A geometric tree is a tree whose vertices are points in the plane and whose edges are line segments connecting the corresponding points. In this section, we construct a discrete Fréchet distance oracle for a given geometric tree T. In other words, we describe how to preprocess a tree T on n vertices, so that, given a polygonal curve Q of size k and two vertices u and v of T, one can efficiently compute the discrete Fréchet distance between Q and the path Π_{uv} in T from u to v (which is a polygonal curve), that is, one can quickly return $d_{\mathrm{dF}}(Q, \Pi_{uv})$.

As mentioned, we focus on the case where k is a small constant. In this case, one can compute $d_{\mathrm{dF}}(Q, \Pi_{uv})$ without any preprocessing in linear time, so the goal is to do it in sublinear time after some preprocessing. More precisely, we only allow near-linear time preprocessing and storage.

We present a reduction of our problem (discrete Fréchet distance oracle for trees) to that for polygonal curves. Specifically, assuming we already know how to construct a discrete Fréchet distance oracle for a polygonal curve and queries of size at most k, we construct a discrete Fréchet distance oracle for T and queries of size k, at the cost of an additional logarithmic factor in the query bound. In Section 4, we obtain discrete Fréchet distance oracles for polygonal curves that accept queries of size one, two, three, and four. Thus, by our reduction, we immediately obtain the corresponding discrete Fréchet distance oracles for trees.

Black box: Distance oracle for curves. Fix $k \in \mathbb{N}$. Assume we have a black box that preprocesses a polygonal curve $P = (p_1, \dots, p_n)$ in near-linear time such that, given a subcurve of P between vertices i and j, denoted $P[i, j]$ with $1 \le i \le j \le n$, and a query curve Q of size at most k, it computes the discrete Fréchet distance between Q and $P[i, j]$ in $t_k(n)$ time. We assume that $t_{k-1}(n) \le t_k(n)$. Section 4 presents an implementation of the black box for query size up to four.

2.1 Data structures

Let $T = (V, E)$ be a geometric tree on n vertices. We first pick a root of T arbitrarily and decompose T into *heavy paths* [18]. The *heavy-path decomposition* of a rooted tree T has the following properties: it is a collection of "heavy paths;" each heavy path is a (possibly degenerate) subpath of a leaf-to-root path in T, beginning at a leaf; the top endpoint of each path (unless it is the root of T) links to a node in another heavy path, in such a way that for any two vertices u, v of T the path between them in T switches between at most $O(\log n)$ heavy paths; every link in T is either a heavy-path link or a link between the top node of a heavy path and its parent in T. See Figure 1.

Recall that, for vertices u, v of the T, Π_{uv} denotes the path from u to v in T. Given u, v, one can compute the list of the $O(\log n)$ subpaths (of the decomposition's paths) whose concatenation is Π_{uv} in $O(\log n)$ time, where each subpath is specified by the indices of its endpoints, as shown in [18]. (The simpler data structure in [18] supporting $O(\log^2 n)$-time query is sufficient for our purposes, as this is not the bottleneck in our approach.)

Next, for each path in the decomposition, we construct a discrete Fréchet distance oracle for polygonal curves for queries of size at most k.

2.2 The Algorithm

Let $u, v \in V$ and let $Q = (q_1, q_2, \dots, q_k)$. We describe how to compute $d_{\mathrm{dF}}(\Pi_{uv}, Q)$, using the data structures above. We begin by computing the representation of Π_{uv} as the concatenation of $O(\log n)$ subpaths $P_1, P_2, \dots, P_m$ (i.e., $\Pi_{uv} = P_1 \cdot P_2 \cdot \dots \cdot P_m$, where "$\cdot$" denotes concatenation of paths). If $k = 1$, then $Q = (q_1)$ and

$$d_{\mathrm{dF}}(\Pi_{uv}, (q_1)) = \max_{i \in \{1,\dots,m\}} \{d_{\mathrm{dF}}(P_i, (q_1))\}, \tag{1}$$

which can be computed in time $mt_1(n) = O(t_1(n) \log n)$.

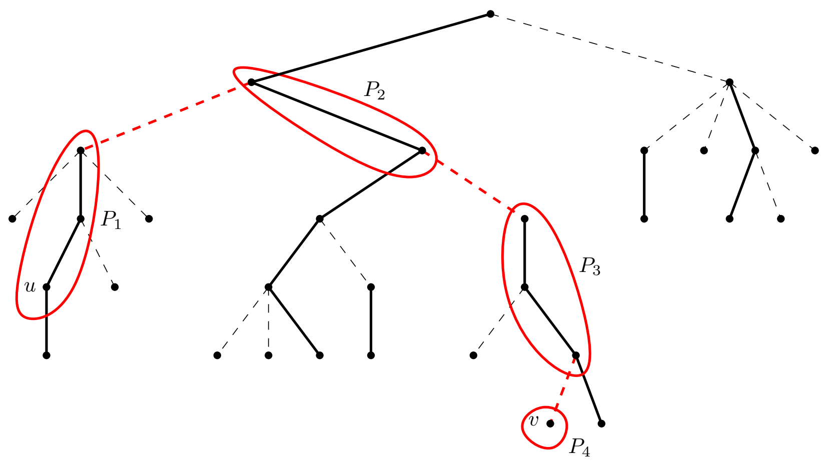

Figure 1 The heavy-path decomposition. The heavy paths are drawn in bold; the path Π_{uv} is the concatenation of the subpaths $P_1, \ldots, P_4$.

For $k > 1$, consider an optimal walk of Π_{uv} and Q, and let P_j be the last subpath to which q_1 is assigned, let q_ℓ be the last point of Q that is assigned to P_j (it is possible that $\ell = 1$), and let $q_{\ell'}$ be the first point of Q that is assigned to P_{j+1}. Then $1 < \ell' \leq k$ and $\ell' \in \{\ell, \ell+1\}$, and

$$\begin{aligned} d_{\mathrm{dF}}(\Pi_{uv}, (q_1 \ldots q_k)) = \max\{ & d_{\mathrm{dF}}(P_1 \cdot P_2 \cdot \ldots \cdot P_{j-1}, (q_1)), \\ & d_{\mathrm{dF}}(P_j, (q_1, \ldots, q_\ell)), \\ & d_{\mathrm{dF}}(P_{j+1} \cdot \ldots \cdot P_m, (q_{\ell'}, \ldots, q_k))\}. \end{aligned} \tag{2}$$

A recursive algorithm. If $k = 1$, return $\max_{i \in \{1,\ldots,m\}} \{d_{\mathrm{dF}}(P_i, (q_1))\}$, according to Eq. (1). Otherwise, according to Eq. (2), for each $j \in \{1, \ldots, m\}$ and for each $\ell \in \{1, \ldots, k\}$ and $\ell' \in \{\ell, \ell+1\}$, $1 < \ell' \leq k$, compute $\max\{d_{\mathrm{dF}}(P_1 \cdot P_2 \cdot \ldots \cdot P_{j-1}, (q_1)), d_{\mathrm{dF}}(P_j, (q_1, \ldots, q_\ell)), d_{\mathrm{dF}}(P_{j+1} \cdot \ldots \cdot P_m, (q_{\ell'}, \ldots, q_k))\}$ (when $j+1 > m$ or $j-1 < 1$, we ignore the relevant component) and return the smallest of all these values.

Dynamic programming procedure. Our algorithm computes the values in Eqs. (1) and (2) bottom up. For each subpath P_j of P and $Q[\ell_1, \ell_2]$ of Q, we calculate $d_{\mathrm{dF}}(P_j, (q_{\ell_1}, \ldots, q_{\ell_2}))$ using the curve oracle black box. Since k is a constant, these calculations take $O(t_k(n) \log n + t_{k-1}(n) \log n + \cdots + t_1(n) \log n) = O(t_k(n) \log n)$, since we assumed $t_{k-1}(n) \leq t_k(n)$.

Then, for each $1 \leq j_1 < j_2 \leq m$ and for each $1 \leq \ell \leq k$, we calculate $d_{\mathrm{dF}}(P_{j_1} \cdot \ldots \cdot P_{j_2}, (q_\ell))$. Again, we calculate the values bottom up, starting from $j_2 - j_1 = 1$ (computing $d_{\mathrm{dF}}(P_{j_1} \cdot \ldots \cdot P_{j_2+1}, (q_\ell))$ takes $O(1)$ time if the answer to $d_{\mathrm{dF}}(P_{j_1} \cdot \ldots \cdot P_{j_2}, (q_\ell))$ and $d_{\mathrm{dF}}(P_{j_2+1}, (q_\ell))$ is known). We calculate $O(\log^2 n)$ values in $O(1)$ time each, so this step takes $O(\log^2 n)$ time in total.

Next, we calculate $d_{\mathrm{dF}}(P_{j_1} \cdot \ldots \cdot P_m, (q_{\ell_1}, \ldots, q_k))$ for each $1 \leq j_1 < m$ and each $1 \leq \ell_1 \leq k$, starting from $j_1 = m-1$ and $\ell_1 = k$, then $j_1 = m-1$ and $\ell_1 = k-1$, etc. Note that if all "smaller" values are already calculated, computing

$$d_{\mathrm{dF}}(P_{j_1} \cdot \ldots \cdot P_m, (q_{\ell_1}, \ldots, q_k)) = \min_{\substack{j \in \{j_1, \ldots, m\} \\ \ell \in \{\ell_1, \ldots, k\} \\ \ell' \in \{\ell, \ell+1\} \wedge \ell_1 < \ell' \leq k}} \left\{ \begin{aligned} \max\{ & d_{\mathrm{dF}}(P_{j_1} \cdot P_{j_1+1} \cdot \ldots \cdot P_{j-1}, (q_{\ell_1})), \\ & d_{\mathrm{dF}}(P_j, (q_{\ell_1}, \ldots, q_\ell)), \\ & d_{\mathrm{dF}}(P_{j+1} \cdot \ldots \cdot P_m, (q_{\ell'}, \ldots, q_k))\} \end{aligned} \right\}$$

takes $O(\log n)$ time. There are $O(\log n)$ such computations, so all of them together take $O(\log^2 n)$ time. Hence, the total running time of the algorithm is $O(\log^2 n + t_k(n) \log n)$.

Using the results from Section 4 (see Table 1) for the black box implementation, we can therefore conclude with the following summary; for the construction time and space, preprocessing for individual heavy paths dominates the costs.

▶ **Theorem 2.1.** *For a geometric tree on n vertices, one can construct discrete Fréchet distance oracles with query times $O(\log^3 n)$, $O(\log^3 n)$, $O(\log^4 n)$, and $O^*(\sqrt{n})$ for query sizes one, two, three, and four, respectively. The structures require $O(n \log n)$ space for query sizes up to three and $O^*(n)$ for query size four.*

3 Distance oracles for local graphs

A *geometric graph* G is a graph defined over a (finite) set P of points in $\mathbb{R}^d$ as vertices, and in which the weight of an edge $e = (p, q)$, $p, q \in P$, is the Euclidean distance $\|p - q\|$ between p and q. Let $t \geq 1$ be a real parameter. We say that G is *t-local* if the following condition holds: For any ball B and for any two points $p, q \in P \cap B$, there exists a path in G between p and q that does not leave tB, where tB is the ball obtained from B by scaling it by a factor of t around its center, that is, p and q are connected in the subgraph of G induced by the set $P \cap tB$. We say that a graph is *local* if it is t-local for some constant $t \geq 1$.

We first examine the connection between geometric spanners and local graphs. Recall that $G = G(P, E)$ is a *t-spanner*, if for any any two points $p, q \in P$, there exists a path in G between p and q of length at most $t \cdot \|p - q\|$ where the length of a path is the sum of the lengths of its edges. A *spanner* is a graph that is a t-spanner for some constant $t \geq 1$.

▶ **Observation 3.1.** *Any geometric t-spanner is $2t$-local.*[1]

Proof. Suppose $G = G(P, E)$ is a t-spanner. We will show that G is $2t$-local.

Consider an arbitrary pair of points $p, q \in P$ and put $d := \|p - q\|$. By assumption, there is a path $P(p, q)$ of length at most td in G. Let D be the disk with segment pq as the diameter. Let x be a point of $P(p, q)$. By the triangle inequality, we have

$$\|p - x\| + \|q - x\| \leq |P(p, x)| + |P(q, x)| = |P(p, q)| \leq td\,, \tag{3}$$

where $P(p, x)$ and $P(q, x)$ are the appropriate subpaths of $P(p, q)$. The locus of points x satisfying Eq. (3) is an elliptical region E with foci p and q and major axis td. In particular, it fits into tD. This proves the observation for disk D, as clearly $tD \subseteq 2tD$.

Now, let D' be any other disk containing both p and q. We show that $tD \subseteq 2tD'$ and therefore $P(p, q) \subseteq 2tD'$. Let o' and d' be the center and diameter of D', respectively. Then, $d' \geq d$ and $\|o' - p\|, \|o' - q\| \leq d'/2$ (and therefore also $\|o' - o\| \leq d'/2$, where o is D's center). Let a be any point on the boundary of tD, then, by triangle inequality, $\|o' - a\| \leq \|o' - o\| + \|o - a\| \leq d'/2 + td/2 = d'(1/2 + t/2) \leq td'$ (since $t \geq 1$), and therefore $a \in 2tD'$, completing the proof. ◀

The opposite implication does not hold, as formalized in Theorem 3.2 below, the proof of which is deferred to the full version of the paper [3, Section 3]. We conclude that the locality property is weaker than the spanning property, i.e., the class of local graphs strictly contains the class of spanner graphs.

[1] We believe that the constant 2 can be improved with some additional effort.

▶ **Theorem 3.2.** *There exists a constant $t > 1$, such that for any $t' \geq 1$, one can construct a graph that is t-local, but not a t'-spanner.*

Hereafter, we assume that $d = 2$ and that the points in P are in *general position*, i.e, no line passes through three or more points of P and no circle passes through four or more of them.

1-local graphs. We begin with the case $t = 1$, which is especially interesting. Let $DT(P)$ denote the Delaunay triangulation of P. We think of $DT(P)$ as a graph over P, and prove below that any 1-local graph over P contains $DT(P)$ as a subgraph, and that $DT(P)$ itself is 1-local.

▶ **Observation 3.3.**
(i) *$DT(P)$ is 1-local.*
(ii) *Any 1-local graph G over P contains $DT(P)$.*

Proof. **(i)** Let D be a disk such that $|P \cap D| \geq 2$ and let $p, q \in P \cap D$. We need to show that there exists a path in $DT(P)$ between p and q that does not leave D, but this is a known property of $DT(P)$. **(ii)** Let $e = pq$ be an edge of $DT(P)$. Then, there exists a disk D such that $P \cap D = \{p, q\}$. Since G is 1-local, there exists a path in G between p and q that does not leave D, so e is an edge of G. We thus conclude that G contains $DT(P)$. ◀

We now return to our main topic, namely, discrete Fréchet distance oracles, and study the following problem. Let G be a t-local graph defined over a set P of n points in the plane. For any two vertices u and v of G, let $\mathcal{P}_G(u, v)$ denote the set of all paths between u and v in G. Then, we define the *discrete Fréchet distance between a polygonal curve Q and G* (with respect to u and v) to be $\min_{\Pi \in \mathcal{P}_G(u,v)} d_{\mathrm{dF}}(Q, \Pi)$, denoted by $d_{\mathrm{dF}}(Q, \mathcal{P}_G(u, v))$. We wish to preprocess G, so that given a query curve Q and two vertices u and v of G, one can compute $d_{\mathrm{dF}}(Q, \mathcal{P}_G(u, v))$ efficiently.

We begin with the special case where the queries are line segments connecting two vertices of G.

G is 1-local and $Q = uv$, where u, v are vertices of G. Let r be the smallest radius for which there exist two vertices u', v' of G, such that $u' \in \mathrm{disk}_r(u)$, $v' \in \mathrm{disk}_r(v)$, and (u', v') is an edge of G, where $\mathrm{disk}_r(w)$ denotes the disk of radius r centered at w. (Notice that if (u, v) is an edge of G, then $r = 0$.) Our solution is based on the following claim.

▷ Claim 3.4. $d_{\mathrm{dF}}(uv, \mathcal{P}_G(u, v)) = r$.

Proof. Let $\Pi = (u = w_1, \ldots, w_k = v) \in \mathcal{P}_G(u, v)$ be a path between u and v such that $d_{\mathrm{dF}}(uv, \mathcal{P}_G(u, v)) = d_{\mathrm{dF}}(uv, \Pi)$, and set $d^* = d_{\mathrm{dF}}(uv, \Pi)$. Let ℓ, $1 \leq \ell < k$, be a split index, i.e., if we associate $(w_1, \ldots, w_\ell)$ with u and $(w_{\ell+1}, \ldots, w_k)$ with v, then

$$\max\{\max_{1 \leq i \leq \ell} \|u - w_i\|, \max_{\ell+1 \leq i \leq k} \|v - w_i\|\} = d^* .$$

We first observe that d^* is at least r. Indeed $w_\ell \in \mathrm{disk}_{d^*}(u)$, $w_{l+1} \in \mathrm{disk}_{d^*}(v)$, and (w_l, w_{l+1}) is an edge of G.

To complete the proof we show that $r \geq d^*$. For a contradiction, assume that $r < d^*$. Let u' and v' be two vertices such that $u' \in \mathrm{disk}_r(u)$, $v' \in \mathrm{disk}_r(v)$, and (u', v') is an edge of G. Then, since G is 1-local, there exists a path from u to u' contained in the disk around u of radius $\|u - u'\| \leq r$, and there exists a path from v to v' contained in the disk around v of radius $\|v - v'\| \leq r$. Therefore, there exists a path from u to v whose discrete Fréchet distance from uv is at most r – a contradiction. ◁

The data structure. In the preprocessing stage, we first construct a data structure T_{EDGE} of near-linear size for disk range searching in P (see also the data structure description and references in the full version [3, Section 4.3]). This data structure allows us to compute the set $P \cap D$, for a query disk D, as the union of pre-stored pairwise-disjoint canonical subsets, in $O^*(\sqrt{n})$ time. For a subset P' of P, let $N(P')$ be the neighbor set of P' in $G = G(P, E)$; that is, $N(P') = \{q \in P \mid \exists p \in P' \text{ with } (p,q) \in E\}$. We now augment T_{EDGE} as follows. For each canonical subset P', we compute $V(P' \cup N(P'))$, the Voronoi diagram of $P' \cup N(P')$, and associate it with P'. The final size of T_{EDGE} is therefore near-linear in $n + m$, where $m = |E|$.

We also construct a second data structure T_{ANNU} of near-linear size for annulus range searching in P. This data structure allows us to compute the set $P \cap A$, for a query annulus A, in $O^*(\sqrt{n} + k)$ time, where $k = |P \cap A|$.

The decision problem. We describe how to determine whether $d_{\text{dF}}(uv, \mathcal{P}_G(u,v)) \le d$, where u and v are any two vertices of G and $d > 0$ is a given value, in $O^*(\sqrt{n})$ time. By arguments similar to those given above, $d_{\text{dF}}(uv, \mathcal{P}_G(u,v)) \le d$ if and only if either $P \cap \text{disk}_d(u) \cap \text{disk}_d(v) \neq \emptyset$, or there exist points $u' \in \text{disk}_d(u)$ and $v' \in \text{disk}_d(v)$ such that (u', v') is an edge of G. We thus use T_{EDGE} to compute a representation of $P \cap \text{disk}_d(u)$ as the union of pairwise-disjoint canonical subsets, and for each of these subsets P', we search in its associated Voronoi diagram $V(P' \cup N(P'))$ for the point that is closest to v. Finally, if at least one of the closest points that were found is within distance d of v, then we return YES, and otherwise we return NO.

Optimization. We now describe how to compute $d_{\text{dF}}(uv, \mathcal{P}_G(u,v))$, which is one of the $O(n)$ distances between u or v and a point in P. Let S be a random sample of size $\sqrt{n}$ of these $O(n)$ distances. We find a pair of consecutive distances $d_1 < d_2$ in $S \cup \{0, \infty\}$, by a binary search using the decision procedure. The expected number of distances between u or v and a point in P that lie in the range $(d_1, d_2]$ is $O(\sqrt{n})$, and we can find them in $O^*(\sqrt{n})$ expected time by querying the second data structure T_{ANNU} with the annuli $A(u, d_1, d_2)$ and $A(v, d_1, d_2)$. Once we have these distances, we can find the smallest among them, d^*, that is still greater or equal than $d_{\text{dF}}(uv, \mathcal{P}_G(u,v))$, by another binary search. We conclude that $d_{\text{dF}}(uv, \mathcal{P}_G(u,v)) = d^*$.

***G* is 1-local and *Q* = *ab*, where *a*, *b* are arbitrary points in the plane.** We remark that the query segment Q does not have to be the segment between the specified vertices (u and v) of G. The only difference is in the algorithm for the decision problem, where we need to take into account that a must be "matched" to u and b must be "matched" to v. In particular, if $d < \max\{\|a - u\|, \|b - v\|\}$, then we immediately return NO. Otherwise, we proceed as above, except that we consider the disk $\text{disk}_d(a)$ (rather than $\text{disk}_d(u)$) and search in the Voronoi diagrams for the point closest to b (rather than to v).

The following theorem summarizes our result for $t = 1$.

▶ **Theorem 3.5.** *Let $G = (P, E)$ be a 1-local graph. Then, we can compute $d_{\text{dF}}(ab, \mathcal{P}_G(u,v))$, for any pair of vertices $u, v \in P$ and any pair of points $a, b \in \mathbb{R}^2$, in $O^*(\sqrt{n})$ expected time, after a preprocessing stage in which we construct data structures of size $O^*(n + m)$. In particular, if G is $DT(P)$, then the size of the data structures is $O^*(n)$.*

***t*-local graphs, *t* > 1.** For $t > 1$, we use the same data structures and query algorithm to obtain an oracle that returns an approximation of the desired distance. More precisely, given two vertices u and v of G, the value r returned by the query algorithm is such that

$r \le d_{\mathrm{dF}}(uv, \mathcal{P}_G(u,v)) \le (t+1)r/2$. The proof is identical to the proof of Claim 3.4, except that now we only know that there exists a path from u to u' that is contained in the disk centered at u of radius $(t+1)\|u-u'\|/2 \le (t+1)r/2$ and similarly for v and v'. This follows from the t-locality property of G applied to the disk of radius $r/2$ centered at the midpoint between u and u'.

As for the case $t=1$, we can also handle arbitrary segment queries. The following theorem summarizes our result for $t > 1$.

▶ **Theorem 3.6.** *Let $G=(P,E)$ be a t-local graph, $t>1$. Then, for any pair of vertices $u,v \in P$ and any pair of points $a,b \in \mathbb{R}^2$, we can compute a value r such that $r \le d_{\mathrm{dF}}(uv, \mathcal{P}_G(u,v)) \le (t+1)r/2$ in $O^*(\sqrt{n})$ expected time, after $O^*(n+m)$ time and space preprocessing.*

By Observation 3.1 we obtain the following corollary.

▶ **Corollary 3.7.** *Let $G=(P,E)$ be a t-spanner, $t>1$. Then, given a query as above, we can compute a value r such that $r \le d_{\mathrm{dF}}(uv, \mathcal{P}_G(u,v)) \le (2t+1)r/2$ in $O^*(\sqrt{n})$ expected time, after a preprocessing stage as above.*

4 Black box revealed: Distance oracles for curves

Notation and definitions. Recall that we write $P[k,\ell]$, for $1 \le k \le \ell \le n$, to denote the (contiguous) subcurve $(p_k, p_{k+1}, \dots, p_\ell)$ of P. For a point q in the plane, the distance from q to the vertex of $P[k,\ell]$ farthest from (nearest to) it, is denoted $d_{\max}(P[k,\ell],q)$ ($d_{\min}(P[k,\ell],q)$).

Consider another curve $Q=(q_1,\dots,q_{\mathrm{last}})$. Put $\Delta = \Delta(P,Q) := \max\{\|p_1-q_1\|, \|p_n - q_{\mathrm{last}}\|\}$. From the definition of a walk, it follows that $d_{\mathrm{dF}}(P,Q) \ge \Delta(P,Q)$.

For two points a and b (which will usually be q_1 and q_{last}) and a real number $r \ge \max\{\|p_1-a\|, \|p_n-b\|\}$, let $P_\vdash(r)$ denote the longest prefix of P, for which $d_{\max}(P_\vdash(r),a) \le r$, and let $P_\dashv(r)$ denote the longest suffix of P for which $d_{\max}(P_\dashv(r),b) \le r$.

As a warm-up, we show how to solve the distance oracle problem for $k=1$. In this case $d_{\mathrm{dF}}(P,(a)) = d_{\max}(P,a)$ and can be computed in logarithmic time by precomputing the farthest-neighbor Voronoi diagram of P and preprocessing it for point-location queries. The subcurve version of the problem (that is, computing $d_{\mathrm{dF}}(P[k,\ell],(a)) = d_{\max}(P[k,\ell],a)$) can be solved in $O(\log^2 n)$ time using the T_{FVD} structure defined below. Clearly, solving the optimization problem answers the decision question within the same time bound. This completes row $k=1$ in Table 1.

4.1 $k = 2$

Let $P=(p_1,\dots,p_n)$ be a sequence of points in the plane representing a polygonal curve. We construct a near-linear size data structure, that, given a 2-vertex query curve $Q=(a,b)$, can compute in $O(\log^2 n)$ time the discrete Fréchet distance $d_{\mathrm{dF}}(P,Q)$ between P and Q.

We begin with some definitions. We say a distance d satisfying $d \ge d_{\mathrm{dF}}(P,(a,b))$ is *feasible*. As already observed, $d < \Delta := \max(\|p_1-a\|, \|p_n-b\|)$ is not feasible. If $d \ge \Delta$ is a feasible distance with $d_{\max}(P_\vdash(d),a) = d$, we say that d is *prefix-feasible*. Alternatively, if d is feasible with $d_{\max}(P_\dashv(d),b)=d$, we say that it is *suffix-feasible*.

▶ **Observation 4.1.** *A distance $d \ge \Delta$ is feasible if and only if $P_\vdash(d)$ and $P_\dashv(d)$ cover P.*

Proof. As already observed, any $d < \Delta$ is infeasible. A feasible d corresponds to a walk that assigns a (non-empty) prefix of P to a and a (non-empty) suffix to b, covering P. The prefix is contained in $P_\vdash(d)$ and the suffix in $P_\dashv(d)$, completing one direction of the proof.

Conversely, if $P_{\vdash}(d)$ and $P_{\dashv}(d)$ cover P (neither can be empty, as $d \geq \Delta$), there is a (prefix,suffix) pair that can be assigned to a and b respectively, producing a walk of cost at most d, completing the proof. ◀

The data structure. We construct a binary tree, T_{FVD}, which stores the farthest-neighbor Voronoi diagram (FVD) of subsequences of P. At the root of T_{FVD}, we store the FVD diagram of $P[1,n]$ together with a corresponding point location structure. In the left and right children of the root, we store the FVDs of $P[1,\lceil \frac{n}{2} \rceil]$ and $P[\lceil \frac{n}{2} \rceil + 1, n]$, respectively, etc. It is easy to see that the size of T_{FVD} is $O(n \log n)$, and that given a query point q and indices k, ℓ $(1 \leq k \leq \ell \leq n)$, one can find the distance $d_{\max}(P[k,\ell], q)$ in $O(\log^2 n)$ time. Moreover, given *any* distance $d \geq \Delta$, one can compute $P_{\vdash}(d)$ and $P_{\dashv}(d)$ in $O(\log^2 n)$ time (roughly speaking, we descend T_{FVD} from the root checking canonical subsets for being within distance d of a or b); we refer to this as a *prefix* (resp., *suffix*) computation. Together with Observation 4.1, this gives an $O(\log^2 n)$ time test for feasibility, both for full P and a subsequene of P.

Optimization. The algorithm consists of two symmetric parts, a left-to-right part and a right-to-left part.

The left-to-right part performs a binary search in P to find the smallest prefix-feasible distance d_L. We start by finding the distance $d_{\lceil \frac{n}{2} \rceil} = d_{\max}(P[1, \lceil \frac{n}{2} \rceil], a)$ (if $d_{\lceil \frac{n}{2} \rceil} < \Delta$, it is not prefix-feasible). Next, we compute $P_{\vdash}(d_{\lceil \frac{n}{2} \rceil})$ and $P_{\dashv}(d_{\lceil \frac{n}{2} \rceil})$, by performing a prefix and a suffix computation using T_{FVD}. If together they cover P, then $d_{\lceil \frac{n}{2} \rceil}$ is prefix-feasible. If so, we guessed too high. If not, we guessed too low. We continue with the binary search on half of the remaining sequence.

In the right-to-left-part, we perform a binary search in P to find the smallest distance d_R which is suffix-feasible. Finally, we output $d_{\text{dF}}(P,Q) = \min\{d_L, d_R\}$.

▶ **Lemma 4.2.** *The algorithm above outputs the correct discrete Fréchet distance between P and Q in time $O(\log^2 n)$.*

Proof. Assume, without loss of generality, that $d^* = d_{\text{dF}}(P,Q)$ is determined by the distance between a and p_k, for some $1 \leq k \leq n-1$. Then d^* is prefix-feasible, and, since d_L is the smallest distance which is prefix-feasible, we have $d_L \leq d^*$. On the other hand, we have $d_L, d_R \geq d^*$. We conclude that the algorithm returns d^* as claimed.

As for the running time, each iteration of the main binary search costs $O(\log^2 n)$ time, which can be improved to $O(\log n)$ time per iteration with a little more care (see the full version of the paper [3, Section 4.1]). Thus, the total cost is $O(\log^2 n)$ time. ◀

The following theorem summarizes the main result of this section.

▶ **Theorem 4.3.** *Given a curve $P = (p_1, \ldots, p_n)$ in the plane, one can construct a data structure of size $O(n \log n)$ such that, for any 2-vertex query curve Q, $d_{\text{dF}}(P,Q)$ can be computed in $O(\log^2 n)$ time. The same running time can be obtained with a subcurve of P specified at query time.*

4.2 $k = 3$

Let $P = (p_1, \ldots, p_n)$ be a sequence of points in the plane representing a polygonal curve. We construct a near-linear size data structure that, given a 3-vertex query curve $Q = (a,b,c)$, computes in $O(\log^3 n)$ time the discrete Fréchet distance $d_{\text{dF}}(P,Q)$ between P and Q.

▷ Claim 4.4 (Feasibility Test). For a distance $d \geq \Delta := \max\{\|p_1 - a\|, \|p_n - c\|\}$, the following procedure decides if d is *feasible*, that is, if $d \geq d_{\mathrm{dF}}(P, Q)$:

> Let i, j be the indices defined by $P_{\vdash}(d) = P[1, i]$ and $P_{\dashv}(d) = P[j, n]$. Now, (i) if $j > i+1$, then d is feasible if $d_{\max}(P[i+1, j-1], b) \leq d$, (ii) if $j = i+1$, then d is feasible if $d_{\min}(P[i, j], b) \leq d$, and (iii) if $j < i + 1$, then d is feasible if $d_{\min}(P[j, i], b) \leq d$.

Indeed, it is easy to verify that in each of the three cases, d is feasible if and only if the appropriate condition holds.

We begin by adapting the definitions of prefix-feasible and suffix-feasible from Section 4.1. We say that $d \geq \Delta$ is *prefix-feasible* if d is feasible and $d_{\max}(P_{\vdash}(d), a) = d$. Alternatively, it is *suffix-feasible* if it is feasible and $d_{\max}(P_{\dashv}(d), c) = d$.

The data structure. We construct two binary trees, T_{FVD} and T_{VD}. The former has been described in Section 4.1 and the latter is its analog for nearest-neighbor Voronoi diagrams.

▶ **Observation 4.5.** *The feasibility test takes $O(\log^2 n)$ time. In particular, given a distance $d \geq \Delta$, such that $d_{\max}(P_{\vdash}(d), a) = d$ (resp. $d_{\max}(P_{\dashv}(d), c) = d$), one can determine whether d is prefix-feasible (resp., suffix-feasible) in $O(\log^2 n)$ time.*

Proof. Find in $O(\log^2 n)$ time the indices i and j, such that $P_{\vdash}(d) = P[1, i]$ and $P_{\dashv}(d) = P[j, n]$, as in the previous section. Next, depending on whether $j > i + 1$, $j = i + 1$, or $j < i + 1$, we verify the appropriate condition in $O(\log^2 n)$ time using T_{FVD} or T_{VD}. ◀

Optimization. We assume for simplicity that all $3n$ distances are distinct and $d_{\mathrm{dF}}(P, Q) > \Delta$. (We can check whether $d_{\mathrm{dF}}(P, Q) = \Delta$, by checking if $\|p_1 - a\|$ is prefix-feasible or if $\|p_n - c\|$ is suffix-feasible, depending on which of the distances determines Δ.)

The algorithm consists of two symmetric parts, a left-to-right part and a right-to-left part. Each part outputs a distance, and the smaller of these two distances is the desired distance, i.e., $d_{\mathrm{dF}}(P, Q)$.

We describe the left-to-right part. We first perform a binary search to find the smallest distance d_L which is prefix-feasible. Next, we find the largest distance $\overline{d}_L$ which is *not* prefix-feasible. More precisely, let p_L be the point of P for which $\|p_L - a\| = d_L$. Then $\overline{d}_L$ is determined by the point farthest from a among the points of $P[1, L - 1]$. Clearly, $\overline{d}_L < d_L$ and $P_{\vdash}(\overline{d}_L)$ is strictly shorter than $P_{\vdash}(d_L)$. Next, we perform the process described in Claim 4.4 above with the distance $\overline{d}_L$ to obtain the value d'. That is, let $1 \leq i, j \leq n$ be the indices such that $P_{\vdash}(\overline{d}_L) = P[1, i]$ and $P_{\dashv}(\overline{d}_L) = P[j, n]$. Now, (i) if $j > i + 1$, then set $d' = d_{\max}(P[i + 1, j - 1], b)$, (ii) if $j = i + 1$, then set $d' = d_{\min}(P[i, j], b)$, and (iii) if $j < i + 1$, then set $d' = d_{\min}(P[j, i], b)$. Finally, the output of this part of the algorithm is $d_1 = \min\{d_L, d'\}$.

The output of the second part of the algorithm is $d_2 = \min\{d_R, d''\}$, where d_R is the smallest distance which is suffix-feasible, and d'' is the value obtained by performing the analogous process with the distance $\overline{d}_R$. Given the outputs of both parts, we conclude that $d_{\mathrm{dF}}(P, Q) = \min\{d_1, d_2\}$.

▶ **Lemma 4.6.** *The algorithm above is correct, i.e., it outputs the discrete Fréchet distance between P and Q. Moreover, its running time is $O(\log^3 n)$.*

Proof. Let d^* denote the discrete Fréchet distance between P and Q, i.e., $d^* = d_{\mathrm{dF}}(P, Q)$. We distinguish between three cases, depending on which vertex of Q defines d^*. Cases I and II are symmetric and easy, while Case III is more involved.

Case I: $d^* = \|p_k - a\|$. This implies that d^* is prefix-feasible. Moreover, it is clearly the smallest distance which is prefix-feasible, so d^* will be found in the first part of the algorithm.

Case II: $d^* = \|p_k - c\|$. The argument is entirely symmetric to Case I.

Case III: $d^* = \|p_k - b\|$. This implies that $d^* < d_L, d_R$. On the other hand, $d^* > \overline{d}_L, \overline{d}_R$, since $\overline{d}_L$ is not prefix-feasible and $\overline{d}_R$ is not suffix-feasible. Assume without loss of generality that $\overline{d}_L \geq \overline{d}_R$. We now claim that $P_{\vdash}(\overline{d}_L) = P_{\vdash}(d^*)$. This is true, since $\overline{d}_L$ is the second largest distance among the distances between a and the vertices of $P_{\vdash}(d_L)$ up to the vertex that determines d_L and $d^* < d_L$. Similarly, we get that $P_{\dashv}(\overline{d}_R) = P_{\dashv}(d^*)$, and therefore also $P_{\dashv}(\overline{d}_L) = P_{\dashv}(d^*)$. This implies that the distance $d_1 = d'$ returned by the left-to-right part of the algorithm is equal to d^*. ◀

The following theorem summarizes the main result of this section.

▶ **Theorem 4.7.** *Given a curve $P = (p_1, \ldots, p_n)$ in the plane, one can construct a data structure of size $O(n \log n)$ such that for any 3-vertex query curve Q, $d_{\mathrm{dF}}(P, Q)$ can be computed in $O(\log^3 n)$ time. The same running time can be obtained with a subcurve of P specified at query time.*

We defer to the full version [3, Section 4.3] the proof of the following theorem:

▶ **Theorem 4.8.** *Given a curve $P = (p_1, \ldots, p_n)$ in the plane, one can construct a data structure of expected size $O^*(n)$ such that for any 4-vertex query curve Q, $d_{\mathrm{dF}}(P, Q)$ can be computed in $O^*(n^{1/2})$ time. The same running time can be obtained with a subcurve of P specified at query time.*

References

1 P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014. doi:10.1137/130920526.

2 H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

3 Boris Aronov, Tsuri Farhana, Matthew J. Katz, and Indu Ramesh. Discrete Fréchet distance oracles. *arXiv*, 2024. doi:10.48550/arXiv.2404.04065.

4 K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.76.

5 K. Bringmann, M. Künnemann, and A. Nusser. Discrete Fréchet distance under translation: Conditional hardness and an improved algorithm. *ACM Trans. Algorithms*, 17(3):25:1–25:42, 2021. doi:10.1145/3460656.

6 K. Bringmann and W. Mulzer. Approximability of the discrete Fréchet distance. *J. Comput. Geom.*, 7(2):46–76, 2016. doi:10.20382/jocg.v7i2a4.

7 K. Buchin, T. Ophelders, and B. Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In Timothy M. Chan, editor, *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2887–2901. SIAM, 2019. doi:10.1137/1.9781611975482.179.

8 M. Buchin, I. van der Hoog, T. Ophelders, L. Schlipf, R. I. Silveira, and F. Staals. Efficient Fréchet distance queries for segments. In *30th Annual European Symposium on Algorithms, ESA 2022, September 5-9, 2022, Berlin/Potsdam, Germany*, volume 244 of *LIPIcs*, pages 29:1–29:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ESA.2022.29.

9 S.-W. Cheng and H. Huang. Solving Fréchet distance problems by algebraic geometric methods. *CoRR*, abs/2308.14569, 2023.

10 M. de Berg, A. D. Mehrabi, and T. Ophelders. Data structures for Fréchet queries in trajectory data. In J. Gudmundsson and M. H. M. Smid, editors, *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017, July 26-28, 2017, Carleton University, Ottawa, Ontario, Canada*, pages 214–219, 2017.

11 A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

12 T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

13 A. Filtser and O. Filtser. Static and streaming data structures for Fréchet distance queries. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1150–1170. SIAM, 2021. doi:10.1137/1.9781611976465.71.

14 O. Filtser. Universal approximate simplification under the discrete Fréchet distance. *Inf. Process. Lett.*, 132:22–27, 2018. doi:10.1016/j.ipl.2017.10.002.

15 J. Gudmundsson, M. P. Seybold, and S. Wong. Map matching queries on realistic input graphs under the Fréchet distance. In N. Bansal and V. Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 1464–1492. SIAM, 2023. doi:10.1137/1.9781611977554.ch53.

16 J. Gudmundsson and M. H. M. Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Comput. Geom.*, 48(6):479–494, 2015. doi:10.1016/J.COMGEO.2015.02.003.

17 J. Gudmundsson, A. van Renssen, Z. Saeidi, and S. Wong. Translation invariant Fréchet distance queries. *Algorithmica*, 83(11):3514–3533, 2021. doi:10.1007/s00453-021-00865-0.

18 D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

19 T. Wylie and B. Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE ACM Trans. Comput. Biol. Bioinform.*, 10(6):1372–1383, 2013. doi:10.1109/TCBB.2013.17.