Efficient Fréchet Distance Queries for Segments

Maike Buchin ✉
Ruhr Universität Bochum, Germany

Ivor van der Hoog ✉
Utrecht University, The Netherlands

Tim Ophelders ✉
Utrecht University, The Netherlands
TU Eindhoven, The Netherlands

Lena Schlipf ✉
Universität Tübingen, Germany

Rodrigo I. Silveira ✉
Polytechnic University of Catalonia, Barcelona, Spain

Frank Staals ✉
Utrecht University, The Netherlands

Abstract

We study the problem of constructing a data structure that can store a two-dimensional polygonal curve P, such that for any query segment $\overline{ab}$ one can efficiently compute the Fréchet distance between P and $\overline{ab}$. First we present a data structure of size $O(n \log n)$ that can compute the Fréchet distance between P and a horizontal query segment $\overline{ab}$ in $O(\log n)$ time, where n is the number of vertices of P. In comparison to prior work, this significantly reduces the required space. We extend the type of queries allowed, as we allow a query to be a horizontal segment $\overline{ab}$ together with two points $s, t \in P$ (not necessarily vertices), and ask for the Fréchet distance between $\overline{ab}$ and the curve of P in between s and t. Using $O(n \log^2 n)$ storage, such queries take $O(\log^3 n)$ time, simplifying and significantly improving previous results. We then generalize our results to query segments of arbitrary orientation. We present an $O(nk^{3+\varepsilon} + n^2)$ size data structure, where $k \in [1, n]$ is a parameter the user can choose, and $\varepsilon > 0$ is an arbitrarily small constant, such that given any segment $\overline{ab}$ and two points $s, t \in P$ we can compute the Fréchet distance between $\overline{ab}$ and the curve of P in between s and t in $O((n/k) \log^2 n + \log^4 n)$ time. This is the first result that allows efficient exact Fréchet distance queries for arbitrarily oriented segments.

We also present two applications of our data structure. First, we show that our data structure allows us to compute a local δ-simplification (with respect to the Fréchet distance) of a polygonal curve in $O(n^{5/2+\varepsilon})$ time, improving a previous $O(n^3)$ time algorithm. Second, we show that we can efficiently find a translation of an arbitrary query segment $\overline{ab}$ that minimizes the Fréchet distance with respect to a subcurve of P.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational Geometry, Data Structures, Fréchet distance

Digital Object Identifier 10.4230/LIPIcs.ESA.2022.29

Related Version An abstract with preliminary results to this paper waspresented at EuroCG 2020 [10]. A full version with all omitted proofs is available on ArXiv [11].
Preliminary Version: `http://www1.pub.informatik.uni-wuerzburg.de/eurocg2020/data/uploads/papers/eurocg20_paper_65.pdf` [10]
Full Version: `https://arxiv.org/abs/2203.01794` [11]

Funding Research of Schlipf was supported by the Ministry of Science, Research and the Arts Baden-Württemberg (Germany).

Acknowledgements This work started at Dagstuhl workshop 19352, "Computation in Low-Dimensional Geometry and Topology." We thank Dagstuhl, the organizers, and the other participants for a stimulating workshop.

1 Introduction

Comparing the shape of polygonal curves is an important task that arises in many contexts such as GIS applications [2, 9], protein classification [20], curve simplification [7], curve clustering [1] and even speech recognition [21]. Within computational geometry, there are two well studied distance measures for polygonal curves: the Hausdorff and the Fréchet distance. The Fréchet distance has proven particularly useful as it takes the course of the curves into account. However, the Fréchet distance between curves is costly to compute: as its computation requires roughly quadratic time [3, 8]. When a large number of Fréchet distance queries are required, we would like to have a data structure, a so-called *distance oracle*, to answer these queries more efficiently. This leads to a fundamental data structuring problem: preprocess a polygonal curve such that, given a query polygonal curve, their Fréchet distance can be computed efficiently (query curves are assumed to be comparatively small). It turns out that this problem is extremely challenging. Indeed, even though great efforts have been devoted to design data structures to answer Fréchet distance queries, there is still no distance oracle known that is able to answer *exact* queries for a general query curve.

To make progress on this important problem, in this work we focus on a more restrictive but fundamental setting: when the query curve is a single segment. The reasons to study this variant of the problem are twofold. On the one hand, it is a necessary step to solve the general problem. On the other hand, it is a setting that has its own applications. For example, in trajectory simplification, or when trying to find subtrajectories that are geometrically close to a given query segment (e.g. when computing shortcut-variants of the Fréchet distance [12], or in trajectory analysis [5] on sports data). A similar strategy of tackling segment queries has also been successfully applied in nearest neighbor queries with the Fréchet distance [4].

We study preprocessing a polygonal curve P to determine the *exact continuous* Fréchet distance between P and a query segment in sublinear time. Specifically, we study preprocessing a polygonal curve P of n vertices in the plane, such that given a query segment $\overline{ab}$, traversed from a to b, the Fréchet distance between P and $\overline{ab}$ can be computed in sublinear time. Without preprocessing, this problem can be solved in $O(n \log n)$ time.

Related work. Data structures that support (approximate) nearest neighbor queries with respect to the Fréchet distance have received considerable attention throughout the years [4, 13, 15]. In these problems, the goal is typically to store a set of polygonal curves such that given a query curve and a query threshold Δ one can quickly report (or count) the curves that are (approximately) within (discrete) Fréchet distance Δ of the query curve. Some of these data structures even allow approximately counting the number of curves that have a subcurve within Fréchet distance Δ [5]. Highlighting its practical importance, the near neighbor problem using Fréchet distance was posed as ACM Sigspatial GIS Cup in 2017 [24].

Here, we want to compute the Fréchet distance of (part of) a curve to a low complexity *query* curve. For the *discrete* Fréchet distance, efficient $(1+\varepsilon)$-approximate distance oracles are known, even when P is given in an online fashion [14]. For the *continuous* Fréchet distance, Driemel and Har-Peled [12] present an $O(n\varepsilon^{-4} \log \varepsilon^{-1})$ size data structure that given a query segment $\overline{ab}$ can compute a $(1+\varepsilon)$-approximation of the Fréchet distance between P and $\overline{ab}$ in $O(\varepsilon^{-2} \log n \log \log n)$ time. Their approach extends to higher dimensions and low complexity polygonal query curves. However, in contrast to our solution, this approximation uses techniques that do not provide insight into the structure of the algorithmic problem.

Gudmundsson et al. [17] present an $O(n \log n)$ sized data structure that can *decide* if the Fréchet distance to $\overline{ab}$ is smaller than a given value Δ in $O(\log^2 n)$ time (so with some parametric search approach one could consider computing the Fréchet distance itself). However, their result holds only when the length of $\overline{ab}$ and all edges in P is relatively large compared to Δ. De Berg et al. [6] presented an $O(n^2)$ size data structure that does not have any restrictions on the length of the query segment or the edges of P. However, the orientation of the query segment is restricted to be horizontal. Queries are supported in $O(\log^2 n)$ time, and even queries asking for the Fréchet distance to a vertex-to-vertex subcurve are allowed (in that case, using $O(n^2 \log^2 n)$ space). Recently, Gudmundsson et al. [18] extended this result to allow the subcurve to start and end anywhere within P. Their data structure has size $O(n^2 \log^2 n)$ and queries take $O(\log^8 n)$ time. In their journal version, Gudmundsson et al. [19] directly apply the main result of a preliminary version of this paper [10] to immediately improve space usage of their data structure to $O(n^{3/2})$; their preprocessing time remains $O(n^2 \log^2 n)$. The current version of this paper significantly improves these results. Moreover, we present data structures that allow for arbitrarily oriented query segments.

Problem statement & our results. Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices $p_1, \ldots, p_n$. For ease of exposition, we assume that the vertices of P are in general position, i.e., all x- and y-coordinates are unique, no three points lie on a line, and no four points are cocircular. We consider P as a function mapping any time $t \in [0, 1]$ to a point $P(t)$ in the plane. Our ultimate goal is to store P such that we can quickly compute the *Fréchet distance* $\mathcal{D}_\mathcal{F}(P, Q)$ between P and a query curve Q. The Fréchet distance is defined as

$$\mathcal{D}_\mathcal{F}(P, Q) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \|P(\alpha(t)) - Q(\beta(t))\|,$$

where $\alpha, \beta \colon [0, 1] \to [0, 1]$ are nondecreasing surjections, also called reparameterizations of P and Q, respectively, and $\|p - q\|$ denotes the Euclidean distance between p and q.

In this work we focus on the case where Q is a single line segment $\overline{ab}$ starting at a and ending at b. Note that P may self-intersect and $\overline{ab}$ may intersect P. Our first main result deals with the case where $\overline{ab}$ is horizontal:

▶ **Theorem 1.** *Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices. There is an $O(n \log n)$ size data structure that can be built in $O(n \log^2 n)$ time such that given a horizontal query segment $\overline{ab}$ it can report $\mathcal{D}_\mathcal{F}(P, \overline{ab})$ in $O(\log n)$ time.*

This significantly improves over the earlier result of de Berg et al. [6], as we reduce the required space and preprocessing time from quadratic to near linear. We simultaneously improve the query time from $O(\log^2 n)$ to $O(\log n)$. We further extend our results to allow queries against subcurves of P. Let s, t be two points on P, we use $P[s, t]$ to denote the subcurve of P from s to t. For horizontal query segments we then get:

▶ **Theorem 2.** *Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices. There is an $O(n \log^2 n)$ size data structure that can be built in $O(n \log^2 n)$ time such that given a horizontal query segment $\overline{ab}$ and two query points s and t on P it can report $\mathcal{D}_\mathcal{F}(P[s, t], \overline{ab})$ in $O(\log^3 n)$ time.*

De Berg et al. presented a data structure that could handle such queries in $O(\log^2 n)$ time (using $O(n^2 \log^2 n)$ space), provided that s and t were vertices of P. Compared to their data structure we thus again significantly improve the space usage, while allowing more general queries. The recently presented data structure of Gudmundsson et al. [18] does allow s and t

to lie on the interior of edges of P (and thus supports queries against arbitrary subcurves). Their original data structure uses $O(n^2 \log^2 n)$ space and allows for $O(\log^8 n)$ time queries. Compared to their result we use significantly less space, while also improving the query time.

Using the insights gained in this restricted setting, we then present the first data structure that allows exact Fréchet distance queries with arbitrarily oriented query segments in sublinear time. With near quadratic space we obtain a query time of $O(n^{2/3} \log^2 n)$. If we insist on logarithmic query time the space usage increases to $O(n^{4+\varepsilon})$. In particular, we present a data structure with the following time-space tradeoff. At only a small additional cost we can also support subcurve queries.

▶ **Theorem 3.** *Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices, and let $k \in [1..n]$ be a parameter. There is an $O(nk^{3+\varepsilon} + n^2)$ size data structure that can be built in $O(nk^{3+\varepsilon} + n^2)$ time such that given an arbitrary query segment $\overline{ab}$ it can report $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ in $O((n/k) \log^2 n)$ time. In addition, given two query points s and t on P, it can report $\mathcal{D}_{\mathcal{F}}(P[s,t], \overline{ab})$ in $O((n/k) \log^2 n + \log^4 n)$ time.*

In Theorem 3 and throughout the rest of the paper $\varepsilon > 0$ is an arbitrarily small constant. In both Theorem 2 and Theorem 3 the query time can be made sensitive to the number of vertices $m = |P[s,t]|$ in the query subcurve $P[s,t]$. That is, we can get query times $O(\log^3 m)$ and $O((m/k) \log^2 m + \log^4 m)$, respectively.

To achieve our results, we also develop data structures that allow us to efficiently query the *directed Hausdorff distance* $\overrightarrow{\mathcal{D}}_H(P[s,t], \overline{ab}) = \max_{p \in P[s,t]} \min_{q \in \overline{ab}} \|p - q\|$ from (a subcurve $P[s,t]$ of) P to the query segment $\overline{ab}$. For an arbitrarily oriented query segment $\overline{ab}$ and a query subcurve $P[s,t]$ our data structure uses $O(n \log n)$ space and can answer such queries in $O(\log^2 n)$ time. Using more space, queries can be answered in $O(\log n)$ time, see Section 4.

Applications. In the full version [11] we show how to efficiently solve two problems using our data structure. First, we show how to compute a local δ-simplification of P – that is, a minimum complexity curve whose edges are within Fréchet distance δ to the corresponding subcurve of P – in $O(n^{5/2+\varepsilon})$ time.This improves existing $O(n^3)$ time algorithms [16].

Recently, Gudmundsson et al. [18] (full version [19]) studied the query version of this problem, where the goal is to preprocess P, such that given a query curve Q and two points s and t on P, one can find the translation of Q that minimizes the Fréchet distance between (a subcurve) $P[s,t]$ and Q efficiently. They study this query version in a restricted setting, where Q is a horizontal segment. Their original data structure uses $O(n^2 \log^2 n)$ space and allows for $O(\log^{32} n)$ time queries. By applying our data structure, we solve their problem using $O(n \log^2 n)$ space whilst supporting $O(\log^{12} n)$ time queries. In addition, we answer one of their open question to find optimal translations for arbitrarily oriented query segments. Finally, we answer another of their open problems by showing how to find a scaling of a query segment that minimizes the Fréchet distance. Specifically, we show a $O(n \log^2 n)$ size data structure that for any horizontal query segment, can compute the scaling of the segment that minimizes its Fréchet distance to $P[s,t]$ in $O(\log^4 n)$ time.We also show a version for arbitrarily oriented queries.

2 Global approach

We illustrate the main ideas of our approach, in particular for the case where the query segment $\overline{ab}$ is horizontal, with a left of b. We can build a symmetric data structure in case a lies right of b. We now first review some definitions based on those in [6].

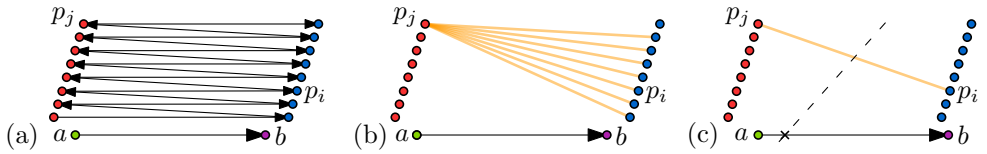

Figure 1 (a) A polygonal curve and query segment. (b) The red vertex p_j forms a backward pair with all but one blue vertex. (c) For a fixed backward pair (p_i, p_j), we consider the distance between the intersection (cross) of their bisector (dashed) and $\overline{ab}$, and either p_i or p_j.

Let $P^{\leq}$ be the set of ordered pairs of vertices $(p, q) \in P \times P$ where p precedes or equals q along P. An ordered pair $(p, q) \in P^{\leq}$ forms a *backward pair* if $x_q \leq x_p$. Throughout the rest of the paper, x_p and y_p denote the x- and y-coordinates of point p, respectively. We denote by $\mathcal{B}(P)$ the backward pairs and say $(p, q) \in \mathcal{B}(P)$ is *trivial* if $p = q$ (Figure 1). For two points $p, q \in P$, we then define $\delta_{pq}(y) = \min_x \max\{\|(x, y) - p\|, \|(x, y) - q\|\}$ (See Figure 1(c)). That is, $\delta_{pq}(y)$ is a function that for any y, gives the minimum distance between a point at height y and both p and q. We will use the function δ_{pq} only when $(p, q) \in \mathcal{B}(P)$ is a backward pair. We define the function $\mathcal{D}_B(y) = \max\{\delta_{pq}(y) \mid (p, q) \in \mathcal{B}(P)\}$, which we refer to as the *backward pair distance* of a horizontal segment at height y with respect to P. Note that $\mathcal{D}_B(y)$ is the upper envelope of the functions δ_{pq} for all backward pairs (p, q) of P. De Berg et al. [6] prove that the Fréchet distance is the maximum of four terms:

$$\mathcal{D}_{\mathcal{F}}(P, \overline{ab}) = \max\left\{||p_1 - a||, \quad ||p_n - b||, \quad \overrightarrow{\mathcal{D}}_H(P, \overline{ab}), \quad \mathcal{D}_B(y_a)\right\}. \tag{1}$$

The first two terms are trivial to compute in $O(1)$ time. Like de Berg et al., we build separate data structures that allow us to efficiently compute the third and fourth terms.

A key insight is that we can compute $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ by building the furthest segment Voronoi diagrams (FSVD) of two sets of horizontal halflines, and querying these diagrams with the endpoints a and b. See Section 3.1. This allows for a linear space data structure that supports querying $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ in $O(\log n)$ time, improving both the space and query time over [6].

However, in [6] the data structure that supports computing the backward pair distance dominates the required space and preprocessing time, as there may be $\Omega(n^2)$ backward pairs, see Figure 1. Via a divide and conquer argument we show that the number of backward pairs that show up on the upper envelope $\mathcal{D}_B$ is only $O(n \log n)$, see Section 3.2. The crucial ingredient is that there are only $O(n)$ backward pairs (p, q) contributing to $\mathcal{D}_B$ in which p is a vertex among the first $n/2$ vertices of P, and q is a vertex in the remaining $n/2$ vertices. Surprisingly, we can again argue this using furthest segment Voronoi diagrams of sets of horizontal halflines. This allows us to build $\mathcal{D}_B$ in $O(n \log^2 n)$ time in total. We can then extend these results to support queries against an arbitrary subcurve $P[s, t]$ of P (see [11]).

For arbitrarily oriented query segments we similarly decompose $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ into four terms (Section 4). The directed Hausdorff term can still be queried efficiently using an $O(n \log^2 n)$ size data structure. However, our initial data structure for the backward pair distance uses $O(n^{4+\varepsilon})$ space. The main reason for this is that functions δ_{pq} expressing the cost of a backward pair are now bivariate, depending on both the slope and intercept of the supporting line of $\overline{ab}$. The upper envelope of a set of n such functions may have quadratic complexity.

While our divide and conquer strategy does not help us to directly bound the complexity of the (appropriately generalized function) $\mathcal{D}_B$ in this case, it does allow us to support queries against subcurves of P. Moreover, we can use it to obtain a favourable query time vs. space trade off. In the full version [11] we then apply our data structure to efficiently solve various Fréchet distance related problems. Omitted proofs can be found in the full version [11].

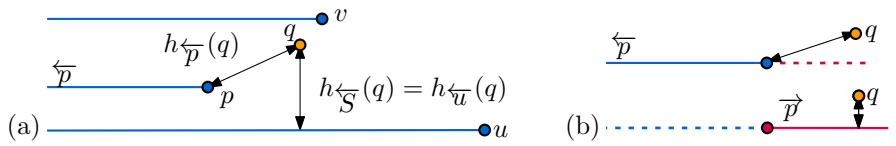

Figure 2 (a) A set $S := \{p, u, v\}$ with $\overleftarrow{S}$ in blue. The distance $h_{\overleftarrow{p}}(q)$ is the distance to the apex of $\overleftarrow{p}$, whilst the distance $h_{\overleftarrow{u}}(q)$ is their vertical difference. (b) the distance between a point q and p is the maximum of the distance to the left and right halfline from p.

3 Horizontal queries

3.1 The Hausdorff term

In this section we introduce some definitions that will be used throughout the paper, and state the fact that there is a linear-size data structure to query the Hausdorff term in $O(\log n)$ time, which can be built in $O(n \log n)$ time. The proof of this result is in the full version [11].

For a point $p \in \mathbb{R}^2$, define $\overleftarrow{p}$ to be the "leftward" horizontal halfline starting at p and containing all points directly to the left of p. Refer to Figure 2. Analogously, we define $\overrightarrow{p}$ as the "rightward" horizontal halfline starting at p, so that $p = \overleftarrow{p} \cap \overrightarrow{p}$. We extend this notation to any set of points S, that is, $\overleftarrow{S} = \{\overleftarrow{s} \mid s \in S\}$ denotes the set of "leftward" halflines starting at the points in $S \subseteq \mathbb{R}^2$. We define $\overrightarrow{S}$ analogously. Let S and T be two (possibly overlapping) point sets in the plane. We define the following distance functions for rays $\overleftarrow{p}, \overleftarrow{S}$ (definitions for $\overrightarrow{p}, \overrightarrow{S}$ are analogous):

$$h_{\overleftarrow{p}}(q) = \overrightarrow{\mathcal{D}}_H(\{q\}, \overleftarrow{p}) = \min\left\{\|p' - q\| \mid p' \in \overleftarrow{p}\right\}, \qquad h_{\overleftarrow{S}}(q) = \max\left\{h_{\overleftarrow{p}}(q) \mid p \in S\right\}$$

Note that $h_{\overrightarrow{S}}$ (resp., $h_{\overleftarrow{S}}$) is the upper envelope of the distance functions to the halflines in $\overrightarrow{S}$ (resp., $\overleftarrow{S}$). Since $h_{\overrightarrow{S}}$ and $h_{\overleftarrow{S}}$ map each point in the plane to a distance, the envelopes live in $\mathbb{R}^3$. Combining furthest segment Voronoi diagrams with point location data structures, we can show how to compute $\overrightarrow{\mathcal{D}}_H(S, \overline{ab})$ efficiently:

▶ **Theorem 4.** *Let S be a set of n points in $\mathbb{R}^2$. In $O(n \log n)$ time we can build a data structure of linear size so that given a horizontal query segment $\overline{ab}$, $\overrightarrow{\mathcal{D}}_H(S, \overline{ab})$ can be computed in $O(\log n)$ time.*

Note that the directed Hausdorff distance from a polygonal curve P to a (horizontal) line segment is attained at a vertex of P [6], thus, we can use Theorem 4 to compute it.

3.2 The backward pairs term

Here we show that the function $\mathcal{D}_B$, representing the backward pair distance, has complexity $O(n \log n)$, can be computed in $O(n \log^2 n)$ time, and evaluated for some query value y in $O(\log n)$ time. This leads to an efficient data structure for querying P for the Fréchet distance to a horizontal query segment $\overline{ab}$, proving Theorem 1. See the full version [11] for details.

Recall that $\mathcal{D}_B(y)$ is the maximum over all function values $\delta_{pq}(y)$ for all backward pairs $(p, q) \in \mathcal{B}(P)$. To avoid computing $\mathcal{B}(P)$, we define a new function $\delta'_{pq}(y)$ that applies to any ordered pair of points $(p, q) \in P^{\leq}$. We show that for all backward pairs $(p, q) \in \mathcal{B}(P)$, we have $\delta'_{pq}(y) = \delta_{pq}(y)$. For any pair $(p, q) \in P^{\leq}$ that is not a backward pair, we show that there exists a (trivial) backward pair $(p', q') \in \mathcal{B}(P)$ such that $\delta'_{pq}(y) \leq \delta'_{p'q'}(y) = \delta_{p'q'}(y)$. Consequently, we can compute the value $\mathcal{D}_B(y)$ by computing the maximum value of $\delta'_{pq}(y)$ over all pairs in $P^{\leq}$. We will show how to do this in an efficient manner.

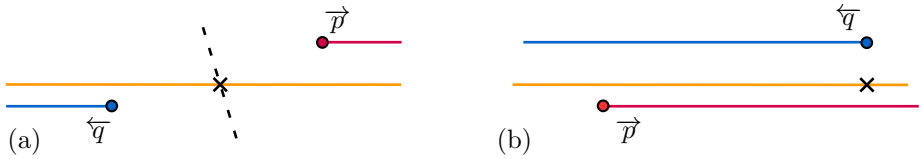

Figure 3 The distance $\delta'_{pq}(y)$ is realised by either (a) the point of intersection between y and the bisector between $\overleftarrow{q}$ and $\overrightarrow{p}$ or, (b) the vertical distance between a line at height y and p or q.

For a pair of points $(p, q) \in P^{\leq}$, we define the *pair distance* between a query $\overline{ab}$ at height y and (p, q) as the Hausdorff distance from a horizontal line of height y to $(\overrightarrow{p} \cup \overleftarrow{q})$, that is:

$$\delta'_{pq}(y) = \min_x \max \left\{ h_{\overleftarrow{q}}((x, y)), h_{\overrightarrow{p}}((x, y)) \right\}.$$

See Figure 3. The following key lemma states that, for our purposes we can use δ' instead of δ.

▶ **Lemma 5.** *For any y, $\mathcal{D}_B(y) = \max\{\delta_{pq}(y) \mid (p, q) \in \mathcal{B}(P)\} = \max\{\delta'_{pq}(y) \mid (p, q) \in P^{\leq}\}$.*

3.2.1 Relating $\mathcal{D}_B(y)$ to furthest segment Voronoi diagrams

Now we devise a divide and conquer algorithm that computes $\mathcal{D}_B(y)$ by computing it for subsets of vertices of P. Lemma 5 allows us to express $\mathcal{D}_B(y)$ in terms of $P^{\leq}$ instead of $\mathcal{B}(P)$. Next we refine the definition of $\mathcal{D}_B(y)$ to make it decomposable. To that end, we define $\mathcal{D}_B(y)$ on pairs of subsets of P. Let S, T be any two subsets of vertices of P, we define:

$$\mathcal{D}_B^{S\times T}(y) = \max\left\{\delta'_{pq}(y) \mid (p, q) \in (S \times T) \cap P^{\leq}\right\}.$$

We show that we can compute $\mathcal{D}_B^{S\times T}(y)$ efficiently using the δ' functions. For this, we fix a value of y and show that computing $\mathcal{D}_B^{S\times T}(y)$ is equivalent to computing an intersection between two curves that consist of a linear number of pieces, each of constant complexity. We then argue that as y changes, the intersection point moves along a linear complexity curve that can be computed in $O(n \log n)$ time. This will allow us to query $\mathcal{D}_B(y) = \mathcal{D}_B^{P\times P}(y)$ in $O(\log n)$ time, for any query height y.

From distance to intersections. For a fixed value y', computing $\mathcal{D}_B^{S\times T}(y')$ is equivalent to computing an intersection point between two curves:

▶ **Lemma 6.** *Let $y' \in \mathbb{R}$ be a fixed height, let p be a point in P, and let T be a subset of the vertices of $P[p, p_n]$. The graphs of the functions $x \mapsto h_{\overrightarrow{p}}((x, y'))$ and $x \mapsto h_{\overleftarrow{T}}((x, y'))$ intersect at a single point (x^*, y'). Moreover, $\mathcal{D}_B^{\{p\}\times T}(y') = h_{\overleftarrow{T}}((x^*, y')) = h_{\overrightarrow{p}}((x^*, y'))$*

▶ **Lemma 7.** *Let $y' \in \mathbb{R}$ be a fixed height, and let S, T be subsets of vertices of P such that all vertices in S precede all vertices in T. The graphs of the functions $x \mapsto h_{\overrightarrow{S}}((x, y'))$ and $x \mapsto h_{\overleftarrow{T}}((x, y'))$ intersect at a single point (x^*, y'). Moreover, $\mathcal{D}_B^{S\times T}(y') = h_{\overrightarrow{S}}((x^*, y')) = h_{\overleftarrow{T}}((x^*, y'))$.*

Proof. If all points in S precede all points in T, then all elements in $S \times T$ are in $P^{\leq}$ and we note: $\mathcal{D}_B^{S\times T}(y') = \max_{p\in S}\left\{\mathcal{D}_B^{\{p\}\times T}(y')\right\}$. The equality then follows from Lemma 6. ◀

Given such a pair S, T, for a fixed value y', we can compute a linear-size representation of $x \mapsto h_{\overleftarrow{T}}((x, y'))$ in $O(n \log n)$ time as follows (see Figure 4). We compute the *FSVD* of $\overleftarrow{T}$ in $O(n \log n)$ time. Then, we compute the Voronoi cells intersected by a line of height y' (denoted by $\ell_{y'}$) in left-to-right order in $O(n \log n)$ time. Suppose that a segment of $\ell_{y'}$ intersects only the Voronoi cell belonging to a halfline $\overleftarrow{q} \in \overleftarrow{T}$, then on this domain the function $h_{\overleftarrow{T}}((x, y')) = h_{\overleftarrow{q}}((x, y'))$, and thus it has constant complexity. A horizontal line can intersect at most a linear number of Voronoi cells, hence the function has total linear complexity. Analogous arguments apply to $x \mapsto h_{\overrightarrow{S}}((x, y'))$.

Varying the y-coordinate. Let $f_{(\overrightarrow{S},\overleftarrow{T})} : y \mapsto x^*$ be the function that for each y gives the intersection point x^* such that $h_{\overrightarrow{S}}((x^*, y)) = h_{\overleftarrow{T}}((x^*, y))$. The intersection point (x^*, y') lies on a Voronoi edge of the *FSVD* of $(\overleftarrow{T} \cup \overrightarrow{S})$. More precisely, it lies on the bichromatic bisector of the *FSVD* of $\overleftarrow{T}$ and the one of $\overrightarrow{S}$.

When we vary the y-coordinate, the intersection point traces this bisector. This implies that, given the *FSVD* of $\overrightarrow{S}$ and the *FSVD* of $\overleftarrow{T}$, the graph of $f_{(\overrightarrow{S},\overleftarrow{T})}$ can be computed in $O(n)$ time. Using these properties, we can prove the following.

▶ **Lemma 8.** *Let S, T be subsets of vertices of P such that all vertices in S precede all vertices in T. The function $\mathcal{D}_B^{S \times T}$ has complexity $O(n)$ and can be computed in $O(n \log n)$ time. Evaluating $\mathcal{D}_B^{S \times T}(y)$, for some query value $y \in \mathbb{R}$, takes $O(\log n)$ time.*

3.2.2 Applying divide and conquer

We begin by analyzing the complexity of the function $\mathcal{D}_B(y)$. Consider a partition of P into subcurves S and T with at most $\lceil n/2 \rceil$ vertices each, and with S occurring before T along P. Our approach relies on the following fact.

▶ **Observation 9.** *Let P be partitioned into two subcurves S and T with all vertices in S occurring on P before the vertices of T. We have that*

$$\mathcal{D}_B(y) = \mathcal{D}_B^{P \times P}(y) = \max\left\{\mathcal{D}_B^{S \times S}(y), \mathcal{D}_B^{S \times T}(y), \mathcal{D}_B^{T \times T}(y)\right\}.$$

Note that we can omit $\mathcal{D}_B^{T \times S}$ because $(T \times S) \cap P^{\leq} = \emptyset$. We obtain the following lemma.

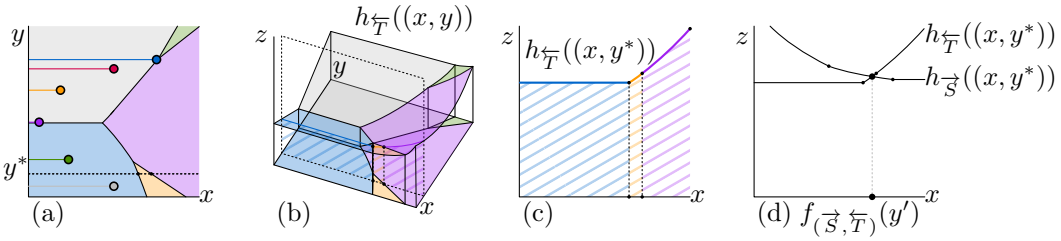

Figure 4 (a) A set $\overleftarrow{T}$ of rays arising from a set T of points, with their FSVD. (b) $h_{\overleftarrow{T}}((x, y))$ is the distance from (x, y) to the ray corresponding to the Voronoi cell at (x, y). (c) For a fixed y', $x \mapsto h_{\overleftarrow{T}}((x, y'))$ is monotonically increasing. (d) The value x for which $h_{\overleftarrow{T}}((x, y')) = h_{\overrightarrow{S}}((x, y'))$ corresponds to $f_{(\overrightarrow{S},\overleftarrow{T})}(y')$, and to $\mathcal{D}_B^{S \times T}(y')$.

▶ **Lemma 10.** *Let P be a polygonal curve with n vertices. The function $\mathcal{D}_B$ has complexity $O(n \log n)$ and can be computed in $O(n \log^2 n)$ time. Evaluating $\mathcal{D}_B(y)$, for a given $y \in \mathbb{R}$, takes $O(\log n)$ time.*

Eq. 1 together with Theorem 4 and Lemma 10 thus imply that we can store P in an $O(n \log n)$ size data structure, so that we can compute $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ for some horizontal query segment $\overline{ab}$ in $O(\log n)$ time. That is, we have established Theorem 1.

Subcurve queries. We can extend our data structure to support Fréchet distance queries to subcurves of P, establishing Theorem 2. The two main ideas are *(i)* to store all intermediate data structures constructed in the above divide and conquer algorithm, and that *(ii)* we can actually achieve the result of Lemma 8 – evaluating $\mathcal{D}_B^{S \times T}(y')$ for some query value y' in $O(\log n)$ time – by *separately* storing a data structure on S and a data structure on T. See the full version [11] for details. Our data structure has total size $O(n \log^2 n)$ and allows us to find $O(\log n)$ nodes whose associated subcurves make up the query subcurve $P[s,t]$. For each such pair of nodes μ, ν we use the (extended) Lemma 8 data structures associated with these nodes to compute the contribution $\mathcal{D}_B^{P_\mu \times P_\nu}(y)$ of backward pairs with one vertex in subcurve P_ν and one vertex in P_μ. We thus spend $O(\log^3 n)$ time to compute the backward pair distance. This dominates the $O(\log^2 n)$ time required to query the Theorem 4 data structures associated with each node to compute the Hausdorff distance term.

4 Arbitrary orientation queries

In this section we extend our results to arbitrarily oriented query segments, proving Theorem 3. Let α be the slope of the line containing the query segment $\overline{ab}$, and let β be its intercept (note that vertical query segments can be handled by a rotated version of our data structure for horizontal queries). We again consider the case where a is left of b; the other case is symmetric. Following Eq. 1, we can write $\mathcal{D}_{\mathcal{F}}(P, \overline{ab})$ as the maximum of four terms: $\|p_1 - a\|$, $\|p_n - b\|$, $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$, and the *backward pair distance* $\mathcal{D}_B(\alpha, \beta)$ with respect to α. The backward pair distance is now defined as

$$\mathcal{D}_B(\alpha, \beta) = \max\{\delta_{pq}(\alpha, \beta) \mid (p, q) \in \mathcal{B}(P)\}, \qquad \text{where}$$
$$\delta_{pq}(\alpha, \beta) = \min_x \max\{\|(x, \alpha x + \beta) - p\|, \|(x, \alpha x + \beta) - q\|\}.$$

In Section 4.1 we present an $O(n \log n)$ size data structure that supports querying $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ in $O(\log^2 n)$ time. The key insight is that we can use furthest *point* Voronoi diagrams instead of furthest *segment* Voronoi diagrams. In Section 4.2 we present a data structure that efficiently supports querying $\mathcal{D}_B(\alpha, \beta)$. In Section 4.3 we extend our results to support queries against subcurves of P as well. This combines our insights from the horizontal queries with our results from Sections 4.1 and 4.2. Finally, in Section 4.4 we then show how this also leads to a space-time trade off.

4.1 The Hausdorff distance term

For any point p and slope α we will denote by ${}^{\leftarrow\alpha}p$ the ray with apex p and slope α that points in the leftward direction. Similarly, for any point set T, we define ${}^{\leftarrow\alpha}T = \left\{ {}^{\leftarrow\alpha}p \mid p \in T \right\}$. Furthermore, we define $h_{\leftarrow_p^\alpha}(x, y)$ to be the directed Hausdorff distance from (x, y) to the ray ${}^{\leftarrow\alpha}p$, and $h_{\leftarrow_T^\alpha}(x, y) = \max\{h_{\leftarrow_p^\alpha}(x, y) \mid p \in T\}$.

We can show that we can use the furthest point Voronoi diagram instead of the furthest segment Voronoi diagram and obtain the following results.

▶ **Lemma 11.** *Let T be a set of n points in $\mathbb{R}^2$. In $O(n^2)$ time we can construct an $O(n^2)$ size data structure so that given a query point (x, y) and slope α we can compute $h_{\underset{T}{\leftarrow}\alpha}(x, y)$ in $O(\log n)$ time.*

▶ **Theorem 12.** *Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices.*

- *In $O(n \log n)$ time we can construct a data structure of size $O(n \log n)$ so that given a query segment $\overline{ab}$, $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ can be computed in $O(\log^2 n)$ time.*
- *In $O(n^2)$ time we can construct a data structure of size $O(n^2)$ so that given a query segment $\overline{ab}$, $\overrightarrow{\mathcal{D}}_H(P, \overline{ab})$ can be computed in $O(\log n)$ time.*

4.2 The backward pair distance term

Let $(p, q) \in P^{\leq}$ be an ordered pair. We restrict $\delta_{pq}(\alpha, \beta)$ to the interval of α values for which (p, q) is a backward pair with respect to orientation α. Hence, each δ_{pq} is a partially defined, constant algebraic degree, constant complexity, bivariate function. The backward pair distance $\mathcal{D}_B$ is the upper envelope of $O(n^2)$ such functions. This envelope has complexity $O(n^{4+\varepsilon})$, for some arbitrarily small $\varepsilon > 0$, and can be computed in $O(n^{4+\varepsilon})$ time [23]. Evaluating $\mathcal{D}_B(\alpha, \beta)$ for some given α, β takes $O(\log n)$ time. The following lemma, together with Theorem 12 then gives an $O(n^{4+\varepsilon})$ size data structure that supports $O(\log n)$ time Fréchet distance queries.

▶ **Lemma 13.** *Let P be an n-vertex polygonal curve in $\mathbb{R}^2$. In $O(n^{4+\varepsilon})$ time we can construct a size $O(n^{4+\varepsilon})$ data structure so that given a query segment $\overline{ab}$, $\mathcal{D}_B(\overline{ab})$ can be computed in $O(\log n)$ time.*

4.3 Subcurve queries

Next, we sketch how to support querying against subcurves $P[s, t]$ of P in $O(\log^4 n)$ time as well. Refer to the full version [11] for details. We use the same approach as in the horizontal query segment case: we store the vertices of P into the leaves of a range tree where each internal node ν corresponds to some canonical subcurve P_ν, so that any subcurve $P[s, t]$ can be represented by $O(\log n)$ nodes.

The Hausdorff distance term. Computing the directed Hausdorff distance is decomposable, so using this approach with the data structure of Theorem 12 immediately gives us a data structure that allows us to compute $\overrightarrow{\mathcal{D}}_H(P[s, t], \overline{ab})$ in $O(\log^2 n)$ time. Since the space usage satisfies the recurrence $S(n) = 2S(n/2) + O(n^2)$, this uses $O(n^2)$ space in total.

The backward pair distance term. By storing the data structure of Lemma 13 at every node of the tree, we can efficiently compute the contribution of the backward pairs inside each of the $O(\log n)$ canonical subcurves that make up $P[s, t]$. However, as before, we are still missing the contribution of the backward pairs from different canonical subcurves. We again store additional data structures at every node of the tree that allow us to efficiently compute this contribution.

Let S and T be (the vertices of) two such canonical subcurves, with all vertices of S occurring before T along P. As before we will argue that for some given α and β the functions $x \mapsto h_{\underset{T}{\leftarrow}\alpha}(x, \alpha x + \beta)$ and $x \mapsto h_{\underset{S}{\rightarrow}\alpha}(x, \alpha x + \beta)$ are monotonically increasing and decreasing,

respectively, and that the intersection point of (the graphs of) these functions corresponds to the contribution of the backward pairs in $S \times T$. So, our goal is to build data structures storing S and T that given a query α, β allow us to compute the intersection point of these functions. We will show that we can use the data structure of Lemma 11 to support such queries in $O(\log^2 n)$ time.

We generalize some of our earlier geometric observations to arbitrary orientations. Let p, q be vertices of P, and let S and T be subsets of vertices of P. We define

$$\delta'_{pq}(\alpha,\beta) = \min_x \max\left\{h_{\underset{q}{\leftarrow}\alpha}((x,\alpha x+\beta)), h_{\underset{p}{\rightarrow}\alpha}((x,\alpha x+\beta))\right\} \qquad \text{and}$$
$$\mathcal{D}_B^{S\times T}(\alpha,\beta) = \max\left\{\delta'_{pq}(\alpha,\beta) \mid (p,q) \in (S\times T) \cap P^{\leq}\right\}.$$

and prove the following lemmas (by essentially rotating the plane so that the query segment becomes horizontal, and applying the appropriate lemmas from earlier sections):

▶ **Lemma 14.** *Let P be partitioned into two subcurves S and T with all vertices in S occurring on P before the vertices of T. We have that*

$$\mathcal{D}_B(\alpha,\beta) = \mathcal{D}_B^{P\times P}(\alpha,\beta) = \max\left\{\mathcal{D}_B^{S\times S}(\alpha,\beta), \mathcal{D}_B^{T\times T}(\alpha,\beta), \mathcal{D}_B^{S\times T}(\alpha,\beta)\right\}.$$

▶ **Lemma 15.** *Let S and T be subsets of vertices of P, with S occurring before T along P, and let α, β denote some query parameters. The function $x \mapsto h_{\underset{T}{\leftarrow}\alpha}(x, \alpha x+\beta)$ is monotonically increasing, whereas $x \mapsto h_{\underset{S}{\rightarrow}\alpha}(x, \alpha x+\beta)$ is monotonically decreasing. These functions intersect at a point $(x^*, \alpha x^*+\beta)$, for which $\mathcal{D}_B^{S\times T}(\alpha,\beta) = h_{\underset{T}{\leftarrow}\alpha}(x^*, \alpha x^*+\beta) = h_{\underset{S}{\rightarrow}\alpha}(x^*, \alpha x^*+\beta)$.*

Querying $\mathcal{D}_B^{S\times T}(\alpha,\beta)$. Consider the predicate $Q(x) = h_{\underset{T}{\leftarrow}\alpha}(x, \alpha x+\beta) < h_{\underset{S}{\rightarrow}\alpha}(x, \alpha x+\beta)$. It follows from Lemma 15 that there is a single value x^* so that $Q(x) = \textsc{False}$ for all $x < x^*$ and $Q(x) = \textsc{True}$ for all $x \geq x^*$. Moreover, x^* realizes $\mathcal{D}_B^{S\times T}(\alpha,\beta)$. By storing S and T, each in a separate copy of the data structure of Lemma 11, we can evaluate $Q(x)$, for any value x, in $O(\log n)$ time. We then use parametric search [22] to find x^* in $O(\log^2 n)$ time. Note that this approach is an $O(\log n)$ factor slower compared to the approach we used for horizontal queries only.

▶ **Lemma 16.** *Let S, T be subsets of vertices of P such that all vertices in S precede all vertices in T, stored in the data structure of Lemma 11. For any query α, β we can compute $\mathcal{D}_B^{S\times T}(\alpha,\beta)$ in $O(\log^2 n)$ time.*

For every node ν of the recursion tree on P we store: (i) the data structure of Lemma 13 built on its canonical subcurve P_ν, and (ii) the data structure of Lemma 11 built on the vertices of P_ν. The total space usage of the data structure follows the recurrence $S(n) = 2S(n/2) + O(n^{4+\varepsilon})$, which solves to $O(n^{4+\varepsilon})$. To query the data structure with some subcurve $P[s,t]$ from some vertex s to a vertex t we again find the $O(\log n)$ nodes whose canonical subcurves together define $P[s,t]$, query the Lemma 13 data structure for each of them, and run the algorithm from Lemma 16 for each pair. The total running time is then $O(\log^4 n)$. As before, the procedure can be easily extended to the case where s and t lie on the interior of an edge. We conclude:

▶ **Lemma 17.** *Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices, and $\varepsilon > 0$. There is an $O(n^{4+\varepsilon})$ size data structure that can be built in $O(n^{4+\varepsilon})$ time that for an arbitrary query segment $\overline{ab}$ (and two points s and t on P) can report $\mathcal{D}_B^{P[s,t]\times P[s,t]}(\alpha,\beta)$ in $O(\log^4 n)$ time.*

Since we can compute all four terms $\|s-a\|$, $\|t-b\|$, $\overrightarrow{\mathcal{D}}_H(P[s,t], \overline{ab})$, and $\mathcal{D}_B^{P[s,t]\times P[s,t]}(\alpha,\beta)$ in $O(\log^4 n)$ time, it follows that we can efficiently answer Fréchet distance queries against subcurves.

4.4 Space vs. Query time tradeoff

We can use our approach for subcurve queries from Section 4.3 to obtain a space vs. query time trade off for queries against the entire curve. Let $k \in [1..n]$ be a parameter. We trim the recursion tree on P at a node ν of size $O(k)$. Let $\mathcal{T}$ denote the resulting tree (i.e., the top $\log(n/k)$ levels of the full recursion tree), and let $L(\mathcal{T})$ denote the set of leaves of $\mathcal{T}$, each of which thus corresponds to a subcurve of length $O(k)$. Let $\ell(\nu)$ and $r(\nu)$ be the left and right child of ν, respectively. By repeated application of the second equality in Lemma 14 we have

$$\mathcal{D}_B^{P\times P}(\alpha,\beta) = \max\left\{\max_{\nu\in\mathcal{T}} \mathcal{D}_B^{P_{\ell(\nu)}\times P_{r(\nu)}}(\alpha,\beta), \max_{\nu\in L(\mathcal{T})} \mathcal{D}_B^{P_\nu\times P_\nu}(\alpha,\beta)\right\}.$$

At every leaf of $\mathcal{T}$ we now store the data structure of Lemma 13, and at every internal node the data structure of Lemma 11. The space required by all Lemma 13 data structures is $O((n/k)k^{4+\varepsilon}) = O(nk^{3+\varepsilon})$. The total size for all Lemma 11 data structures follows the recurrence $S(n) = 2S(n/2) + O(n^2)$ which solves to $O(n^2)$. Hence, the total space used is $O(nk^{3+\varepsilon} + n^2)$. The preprocessing time is $O(nk^{3+\varepsilon} + n^2)$ as well.

To answer a query (α, β) we query the Lemma 13 data structures at the leaves of $\mathcal{T}$ in $O(\log k)$ time each. For every internal node ν we use Lemma 16 to compute the contribution of $\mathcal{D}_B^{P_{\ell(\nu)}\times P_{r(\nu)}}(\alpha,\beta)$ in $O(\log^2 n)$ time. Hence, the total query time is $O((n/k)\log k + (n/k)\log^2 n) = O((n/k)\log^2 n)$. So, e.g., choosing $k = n^{1/3}$ yields an $O(n^{2+\varepsilon})$ size data structure supporting $O(n^{2/3}\log^2 n)$ time queries. We can extend this idea to support subcurve queries in $O((n/k)\log^2 n + \log^4 n)$ time as well, giving us the following result:

▶ **Lemma 18.** *Let P be a polygonal curve in $\mathbb{R}^2$ with n vertices, and let $k \in [1..n]$ be a parameter. In $O(nk^{3+\varepsilon} + n^2)$ time we can construct a data structure of size $O(nk^{3+\varepsilon} + n^2)$ so that given a query segment $\overline{ab}$, $\mathcal{D}_B(\overline{ab})$ can be computed in $O((n/k)\log^2 n)$ time. If, in addition we are also given two points s and t on P, $\mathcal{D}_B^{P[s,t]\times P[s,t]}(\overline{ab})$ can be computed in $O((n/k)\log^2 n + \log^4 n)$ time.*

Since computing $\overrightarrow{\mathcal{D}}_H(P[s,t],\overline{ab})$ can be done in $O(\log^2 n)$ time using only $O(n^2)$ space, we thus established Theorem 3. Once again, it is possible to make the query time proportional to the complexity of $P[s,t]$ rather than to n.

5 Concluding Remarks

We presented data structures for efficiently computing the Fréchet distance of (part of) a curve to a query segment. This constitutes an important step towards the more ambitious goal of finding data structures to efficiently answer queries for general polygonal curves.

Our results improve over previous work for horizontal segments and are the first for arbitrarily oriented segments. However, we are left with the challenge of reducing the space used for arbitrary orientations. There are two main issues. The first issue is that even for a small interval of query orientations (e.g., one of the $O(n^2)$ angular intervals defined by lines through a pair of points) it is difficult to limit the number of relevant backward pairs to $o(n^2)$. The second issue is how to combine the backward pair distance values contributed by various subcurves. For (low algebraic degree) univariate functions, the upper envelope has near linear complexity, whereas for bivariate functions the complexity is near quadratic. The combination of these issues makes it hard to improve over the somewhat straightforward $O(n^{4+\varepsilon})$ space bound we build upon.

References

1 Pankaj K Agarwal, Sariel Har-Peled, Nabil H Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005.

2 H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003.

3 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.

4 Boris Aronov, Omrit Filtser, Michael Horton, Matthew J. Katz, and Khadijeh Sheikhan. Efficient nearest-neighbor query and clustering of planar curves. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R Salavatipour, editors, *Algorithms and Data Structures*, pages 28–42. Springer International Publishing, 2019.

5 M. de Berg, A. F Cook IV, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013.

6 Mark de Berg, Ali D Mehrabi, and Tim Ophelders. Data structures for Fréchet queries in trajectory data. In *29th Canadian Conference on Computational Geometry (CCCG'17)*, pages 214–219, 2017.

7 K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, 2011.

8 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the fréchet distance. *Discret. Comput. Geom.*, 58(1):180–216, 2017. doi:10.1007/s00454-017-9878-7.

9 Kevin Buchin, Maike Buchin, Marc van Kreveld, Maarten Löffler, Rodrigo I Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

10 Maike Buchin, Ivor van der Hoog, Tim Ophelders, Rodrigo I. Silveira, Lena Schlipf, and Frank Staals. Improved space bounds for Fréchet distance queries. In *36th European Workshop on Computational Geometry (EuroCG'20)*, 2020.

11 Maike Buchin, Ivor van der Hoog, Tim Ophelders, Rodrigo I. Silveira, Lena Schlipf, and Frank Staals. Efficient Fréchet distance queries for segments. *CoRR*, abs/2203.01794, 2022. arXiv:2203.01794.

12 A. Driemel and S. Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

13 Anne Driemel and Ioannis Psarros. $(2+\epsilon)$-ANN for time series under the Fréchet distance. *arXiv preprint*, 2020. arXiv:2008.09406.

14 Arnold Filtser and Omrit Filtser. Static and streaming data structures for Fréchet distance queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1150–1170. SIAM, 2021.

15 Arnold Filtser, Omrit Filtser, and Matthew J. Katz. Approximate nearest neighbor for curves – simple, efficient, and deterministic. In *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 48:1–48:19, 2020. doi:10.4230/LIPIcs.ICALP.2020.48.

16 Michael" Godau. A natural metric for curves — computing the distance for polygonal chains and approximation algorithms. In *8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991.

17 J. Gudmundsson, M. Mirzanezhad, A. Mohades, and C. Wenk. Fast Fréchet distance between curves with long edges. *International Journal of Computational Geometry & Applications*, 29(2):161–187, 2019.

18 Joachim Gudmundsson, André van Renssen, Zeinab Saeidi, and Sampson Wong. Fréchet distance queries in trajectory data. In *The Third Iranian Conference on Computational Geometry (ICCG 2020)*, pages 29–32, 2020.

19 Joachim Gudmundsson, André van Renssen, Zeinab Saeidi, and Sampson Wong. Translation invariant Fréchet distance queries. *Algorithmica*, 83(11):3514–3533, 2021. doi:10.1007/s00453-021-00865-0.

20 M. Jiang, Y. Xu, and B. Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of Bioinformatics and Computational Biology*, 6(01):51–64, 2008.

21 S. Kwong, QH He, K. Man, KS Tang, and CW Chau. Parallel genetic-based hybrid pattern matching algorithm for isolated word recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 12(05):573–594, 1998.

22 N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

23 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry*, 12(3):327–345, 1994.

24 Martin Werner and Dev Oliver. ACM SIGSPATIAL GIS cup 2017: Range queries under Fréchet distance. *SIGSPATIAL Special*, 10(1):24–27, June 2018. doi:10.1145/3231541.3231549.