Jaywalking your dog : computing the Fréchet distance with shortcuts

SIAM J. COMPUT.
Vol. 42, No. 5, pp. 1830–1866

JAYWALKING YOUR DOG: COMPUTING THE FRÉCHET DISTANCE WITH SHORTCUTS*

ANNE DRIEMEL† AND SARIEL HAR-PELED‡

Abstract. The similarity of two polygonal curves can be measured using the Fréchet distance. We introduce the notion of a more robust Fréchet distance, where one is allowed to shortcut between vertices of one of the curves. This is a natural approach for handling noise, in particular batched outliers. We compute a $(3+\varepsilon)$-approximation to the minimum Fréchet distance over all possible such shortcuts, in near linear time, if the curve is c-packed and the number of shortcuts is either small or unbounded. To facilitate the new algorithm we develop several new tools: (a) a data structure for preprocessing a curve (not necessarily c-packed) that supports $(1+\varepsilon)$-approximate Fréchet distance queries between a subcurve (of the original curve) and a line segment; (b) a near linear time algorithm that computes a permutation of the vertices of a curve, such that any prefix of $2k-1$ vertices of this permutation forms an optimal approximation (up to a constant factor) to the original curve compared to any polygonal curve with k vertices, for any $k>0$; and (c) a data structure for preprocessing a curve that supports approximate Fréchet distance queries between a subcurve and query polygonal curve. The query time depends quadratically on the complexity of the query curve and only (roughly) logarithmically on the complexity of the original curve. To our knowledge, these are the first data structures to support these kind of queries efficiently.

Key words. Fréchet distance, geometric similarity, outliers, approximation algorithms

AMS subject classifications. 65D18, 68W25

DOI. 10.1137/120865112

1. Introduction.

Comparing the shapes of polygonal curves—or sequenced data in general—is a challenging task that arises in many different contexts. The Fréchet distance and its variants (e.g., dynamic time warping [30]) have been used as similarity measures in various applications such as matching of time series in databases [31], comparing melodies in music information retrieval [37], and matching coastlines over time [35], as well as in map-matching of vehicle tracking data [9, 38] and moving objects analysis [10, 11]. Informally, the Fréchet distance between two curves is defined as the maximum distance a point on the first curve has to travel as this curve is being continuously deformed into the second curve. Another common description uses the following "leash" metaphor: Imagine traversing the two curves simultaneously and at each point in time the two positions are connected by a leash of a fixed length. During the traversal you can vary the speeds on both curves independently but not walk backward. The Fréchet distance corresponds to the minimum length of a leash that permits such a traversal.

The Fréchet distance captures similarity under small non-affine distortions and for some of its variants also spatiotemporal similarity [33]. However, it is very sensitive to local noise, which is frequent in real data. Unlike similarity measures such as

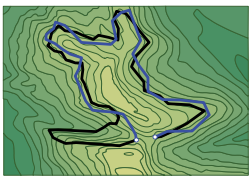
*Received by the editors February 7, 2012; accepted for publication (in revised form) June 24, 2013; published electronically September 26, 2013. A preliminary version of this paper appeared in *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms*, 2012, pp. 318–337. The latest full version of this paper is available online (arXiv:1107:1720).

http://www.siam.org/journals/sicomp/42-5/86511.html

†Department of Information and Computing Sciences, Utrecht University, The Netherlands (anne@cs.uu.nl). This work was supported by the Netherlands Organisation for Scientific Research (NWO) under project 612.065.823.

‡Department of Computer Science, University of Illinois, Urbana, IL, 61801 (sariel@uiuc.edu, http://www.uiuc.edu/∼sariel/). Work on this paper was partially supported by NSF AF awards CCF-0915984 and CCF-1217462.

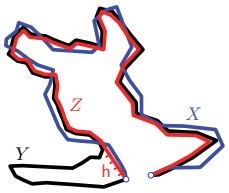
the root-mean-square deviation, which averages over a set of similarity values, and dynamic time warping, which minimizes the sum of distances along the curves, the Fréchet distance is a so-called bottleneck measure and can therefore be affected to an extent which is generally unrelated to the relative amount of noise across the curves. In practice, curves might be generated by physical tracking devices, such as GPS, which is known to be inaccurate when the connection to the satellites is temporarily disturbed due to atmospheric conditions or reflections of the positioning signal on high buildings. Such inaccurate data points are commonly referred to as "outliers." Note that outliers come in batches if they are due to such a temporary external condition. Similarly, in computer vision applications, the silhouette of an object could be partially occluded, and in sound recordings, outliers may be introduced due to background sounds or breathing. Detecting outliers in time series has been studied extensively in the literature [34]. One may also be interested in outliers as a deviation from a certain expected behavior or because they carry some meaning. It could be, for instance, that trajectories of two hikers deviate locally because one hiker chose to take a detour to a panoramic view point, as in the following figure:



Outlier detection is inherently nontrivial if not much is known about the underlying probability distributions and the data is sparse [4]. We circumvent this problem in the computation of the Fréchet distance by minimizing over all possibilities for outlier removal. In a sense, our approach is similar to computing a certain notion of partial similarity. Unlike other partial distance measures, the distance measure we propose is parameter-free. For comparison, in the *partial Fréchet distance*, as studied by Buchin, Buchin, and Wang [14], one is interested in maximizing the portions of the curves which can be matched within a certain Fréchet distance (the parameter). In this case, the dissimilar portions of the curves are ignored. In our case, they are replaced by *shortcuts*, which have to be matched under the Fréchet distance.

The task at hand. We are given two polygonal curves X and Y in $\mathbb{R}^d$, which we perceive as a sequence of linearly interpolated measurement points. We believe that Y is similar to X but it might contain considerable noise that is occluding this similarity. That is, it might contain erroneous measurement points (outliers), which need to be ignored when assessing the similarity. We would like to apply a few edit operations to Y so that it becomes as similar to X as possible, in the process hopefully removing the noise in Y and judging how similar it really is to X. To this end, we—conceptually—remove subsequences of measurement points, which we suspect to be outliers, and minimize over all possibilities for such a removal. This is formalized in the shortcut Fréchet distance.

Shortcut Fréchet distance. A shortcut replaces a subcurve between two vertices by a straight segment that connects these vertices. The part being shortcut is not ignored, but rather the new curve with the shortcuts has to be matched entirely to the other curve under the Fréchet distance. As a concrete example, consider the following figure:



The Fréchet distance between X and Y is quite large, but after we shortcut the outlier "bump" in Y, the resulting new curve Z has a considerably smaller Fréchet distance to X. We are interested in computing the minimum such distance allowing an unbounded number of shortcuts.

Naturally, there are many other possibilities for tackling the task at hand, for example,

(i) bounding the number of shortcuts by a parameter k,
(ii) allowing shortcuts on both curves,
(iii) allowing only shortcuts between vertices that are close by along the curve,
(iv) ignoring the part being shortcut and maximizing the length of the remaining portions,
(v) allowing shortcuts to start and end anywhere along the curve,
(vi) allowing curved shortcuts.

If one is interested in (iii), then the problem turns into a map-matching problem, where the start and end points are fixed and the graph is formed by the curve and its eligible shortcuts. For this problem, results can be found in the literature [17, 5]. A recent result by Har-Peled and Raichel [28] is applicable to the variant where one allows such shortcuts on both curves, i.e., (ii)+(iii). The version in (iv) has been studied under the name of *partial Fréchet distance* [14].

In this paper, we concentrate on the directed vertex-restricted shortcut Fréchet distance (see section 2.2 for the exact definition) because computing it efficiently seems like a first step in understanding how to solve some of the more difficult variants, e.g., (v). Surprisingly, computing this simpler version of the shortcut Fréchet distance is already quite challenging, especially if one is interested in an efficient algorithm. A more recent result by Buchin, Driemel, and Speckmann [16] and Driemel [22] shows that computing the shortcut Fréchet distance exactly is weakly NP-hard for variant (v), where we allow shortcuts to start and end anywhere along the curve. Furthermore, our algorithms can be extended to variant (i), i.e., where at most k shortcuts are allowed; see Remark 4.11.

Note that allowing shortcuts on both curves does not always yield a meaningful measure, especially if shortcuts on both curves may be matched to each other. In particular, if one of the two curves is more accurately sampled and can act as a model curve, allowing shortcuts on only one of the two curves seems reasonable.

Input model. A curve Y is *c-packed* if the total length of Y inside any ball is bounded by c times the radius of the ball. Intuitively, c-packed curves behave reasonably in any resolution. The boundary of convex polygons, algebraic curves of bounded maximum degree, the boundary of (α, β)-covered shapes [25], and the boundary of γ-fat shapes [18] are all c-packed (under the standard assumption that they have bounded complexity). Interestingly, the class of c-packed curves is closed under simplification; see [24]. This makes them attractive for efficient algorithmic manipulation.

Another input model which is commonly used is called low density [20]. We call a set of line segments *ϕ-dense* if for any ball the number of line segments that intersect

this ball and which are longer than the radius of the ball is bounded by ϕ. It is easy to see by a simple packing argument that c-packed curves are $O(c)$-dense.

Informal restatement of the problem. In the parametric space of the two input curves, we are given a terrain defined over a grid partitioning $[0,1]^2$, where the height at each point is defined as the distance between the two associated points on the two curves. The grid is induced by the vertices of the two curves. As in the regular Fréchet distance, we are interested in finding a path between $(0,0)$ and $(1,1)$ on the terrain, such that the maximum height on the path does not exceed some δ. (The minimum such δ is the desired distance.) This might not be possible as there might be "mountain chains" blocking the way. To overcome this, we are allowed to introduce tunnels that go through such obstacles. Each of these tunnels connects two points that lie on the horizontal lines of the grid, as these correspond to the vertices of one curve. Naturally, we require that the starting and ending points of such a tunnel have height at most δ (the current distance threshold being considered), and furthermore, the price of such a tunnel (i.e., the Fréchet distance between the corresponding shortcut and subcurve) is smaller than δ. Once we introduce these tunnels, we need to compute a *monotone* path from $(0,0)$ to $(1,1)$ in the grid which uses tunnels. Finally, we need to search for the minimum δ for which there is a feasible solution.

Challenge and ideas. Let n be the total number of vertices of the input curves. A priori there are potentially $O(n^2)$ horizontal edges of the grid that might contain end points of a tunnel, and as such, there are potentially $O(n^4)$ different families of tunnels that the algorithm might have to consider. A careful analysis of the structure of these families shows that, in general, it is sufficient to consider one (canonical) tunnel per family. Using c-packedness and simplification, we can reduce the number of relevant grid edges to near linear. This in turn reduces the number of potential tunnels that need to be inspected to $O(n^2)$. This is still insufficient to get a near linear time algorithm. Surprisingly, we prove that if we are interested only in a constant factor approximation, for every horizontal edge of the grid we need to inspect only a constant number of tunnels. Thus, we reduce the number of tunnels that the algorithm needs to inspect to near linear. And yet we are not done, as naively computing the price of a tunnel requires time near linear in the size of the associated subcurve. To overcome this, we develop a new data structure, so that after preprocessing we can compute the price of a tunnel in polylogarithmic time per tunnel. Now, carefully putting all these insights together, we get a near linear time algorithm for the approximate decision version of the problem.

However, to compute the minimum δ, for which the decision version returns true—which is the shortcut Fréchet distance—we need to search over the critical values of δ. To this end, we investigate and characterize the critical values introduced by the shortcut version of the problem. Using the decision procedure, we perform a binary search of several stages over these values, in the spirit of [24], to get the required approximation.

Our results.

(A) *Computing the shortcut Fréchet distance.* For a prescribed parameter $\varepsilon > 0$, we present an algorithm for computing a $(3+\varepsilon)$-approximation to the directed vertex-restricted shortcut Fréchet distance between two given c-packed polygonal curves of total complexity n; see Definition 2.5 for the formal definition of the distance being approximated.

If we allow an unbounded number of shortcuts the running time of the new algorithm is $O(c^2 n \log^2 n(\log n + \varepsilon^{-2d} \log(1/\varepsilon)))$; see Theorem 4.10 for the exact result.

A variant of this algorithm can also handle the case where we allow only k shortcuts, with running time $O(c^2kn\log^3 n)$; see Remark 4.11. In the analysis of these problems we use techniques developed by Driemel, Har-Peled, and Wenk in [24] and follow the general approach used in the parametric search technique of devising a decision procedure which is used to search over the critical events for the Fréchet distance. The shortcuts introduce a new type of critical event, which we analyze in section 4.3. The presented approximation algorithms can be easily modified to yield polynomial-time exact algorithms for the same problems (and for general polygonal curves). As such, the main challenge in devising the new algorithm was to achieve near linear time performance. Furthermore, the algorithm uses a new data structure (described next) that is interesting on its own merit.

(B) *Fréchet-distance queries between a segment and a subcurve.* We present a data structure that preprocesses a given polygonal curve Z such that given a query segment h, and two points p, p' on Z (and the edges containing them), it $(1+\varepsilon)$-approximates the Fréchet distance between h and the subcurve of Z between p and p'. Surprisingly, the data structure works for any polygonal curve (not necessarily packed or dense), requires near linear preprocessing time and space, and can answer such queries in polylogarithmic time (ignoring the dependency on ε). See Theorem 5.9 for the exact result.

(C) *Universal vertex permutation for curve simplification.* We show how to preprocess a polygonal curve in near linear time and space such that given a number $k \in \mathbb{N}$, one can compute a simplification in $O(k)$ time which has $K = 2k-1$ vertices (of the original curve) and is optimal up to a constant factor with respect to the Fréchet distance to the original curve, compared to any curve which uses k vertices. Surprisingly, this can be done by computing a permutation of the vertices of the input curve, such that this simplification is the subcurve defined by the first K vertices in this permutation. Namely, we compute an ordering of the vertices of the curves by their Fréchet "significance." See Theorem 6.7 for the exact result.

(D) *Fréchet-distance queries between a curve with k vertices and a subcurve.* We use the above universal vertex permutation to extend the data structure in (B) to support queries with polygonal curves of multiple segments (as opposed to single segments) and obtain a constant factor approximation with polylogarithmic query time; see Theorem 6.9. The query time is quadratic in the query curve complexity and logarithmic in the input curve complexity.

Related work. Assume we are given two polygonal curves of total complexity n and we are interested in computing the Fréchet distance between these curves. The problem has been studied in many variations. We only discuss the results which we deem most relevant and refer the reader to [13] for additional references.

Driemel, Har-Peled, and Wenk presented a near linear time $(1+\varepsilon)$-approximation algorithm for the Fréchet distance assuming the curves are well behaved, that is, c-packed [24]. In general, computing the Fréchet distance exactly takes roughly quadratic time. After publication in the seminal paper by Alt and Godau [6], their $O(n^2\log n)$-time algorithm remained the state of the art for more than a decade. This lead Alt to conjecture that the problem of deciding whether the Fréchet distance between two curves is smaller than or equal to a given value is 3SUM-hard. However, recently, there has been some progress in improving upon the quadratic running time of the decision algorithm. First, Agarwal et al. presented a subquadratic time algorithm for a specific variant of the Fréchet distance [2]. Buchin et al. build upon their work and give an algorithm for the original Fréchet distance [13]. Their algorithm is randomized and takes $o(n^2\log n)$ expected time overall to compute the Fréchet

distance. The decision algorithm they present is deterministic and takes subquadratic time. The only lower bound known for the decision problem is $\Omega(n \log n)$ and was given by Buchin et al. [12]. A randomized algorithm simpler than the one by Alt and Godau, which has the same running time but avoids parametric search, was recently presented by Har-Peled and Raichel [29].

Buchin, Buchin, and Wang [14] showed how to compute the *partial Fréchet distance* under the L_1 and L_∞ metric. Here, one fixes a threshold δ and computes the maximal length of subcurves of the input curves that match under Fréchet distance δ. The running time of their algorithm is roughly $O(n^3 \log n)$.

For the problem of counting the number of subcurves that are within a certain Fréchet distance, a recent result by Gudmundsson, de Berg, and Cook provides a data structure to answer such queries up to a constant approximation factor [32]. To the best of our knowledge the problem of computing the Fréchet distance when one is allowed to introduce shortcuts has not been studied before.

Previous work on curve simplification. There is a large body of literature on curve simplification. Since this is not the main subject of the paper, we only discuss a selection of results which we consider most relevant, since they use the Fréchet distance as a quality measure. Agarwal et al. [3] give a near linear time approximation algorithm to compute a simplification which is in Fréchet distance ε to the original curve and of which the size is at most the size of the optimal simplification with error $\varepsilon/2$. Abam et al. [1] study the problem in the streaming setting, where one wishes to maintain a simplification of the prefix seen so far. Their algorithm achieves an $O(1)$ competitive ratio using $O(k^2)$ additional storage and maintains a curve with $2k$ vertices which has a smaller Fréchet distance to the prefix than the optimal Fréchet simplification with k vertices. Bereg et al. [8] give an exact $O(n \log n)$ algorithm that minimizes the number of vertices in the simplification but using the discrete Fréchet distance, where only distances between the vertices of the curves are considered. Simplification under the Fréchet distance has also been studied by Guibas et al. [27].

Organization. In section 2 we describe some basic definitions and results. In particular, the formal problem statement and the definition of the *directed vertex-restricted shortcut Fréchet distance* between two curves is given in section 2.2. We also discuss some basic tools needed for the algorithms. In section 3, we describe the approximation algorithm for the shortcut Fréchet distance. Here, we devise an approximate decision procedure in section 3.2 that is used in the main algorithm, described in section 3.3, to search over an approximate set of candidate values. The analysis of this algorithm is given in section 4. Since the shortcuts introduce a new set of candidate values, we provide an elaborate study of these new events in section 4.3. The main result for approximating the shortcut Fréchet distance is stated in Theorem 4.10. In the remaining sections we describe the new data structures. In section 5.4 we describe a data structure for a fixed curve that answers queries for the Fréchet distance between a subcurve and a given segment. In section 6, we use this data structure to compute the universal vertex permutation. The extension to query curves with more than two vertices is described in section 6.2. We conclude with discussion and some open problems in section 7.

2. Preliminaries.

Notation. A *curve* X is a continuous mapping from $[0, 1]$ to $\mathbb{R}^d$, where $X(t)$ denotes the point on the curve parameterized by $t \in [0, 1]$. Given two curves X and Y that share an end point, let $X \oplus Y$ denote the *concatenated* curve. We denote with $X[x, x']$ the subcurve of X from $X(x)$ to $X(x')$ and with $X\langle \mathsf{p}, \mathsf{p}' \rangle$ the subcurve of X

between the two points $\mathsf{p}, \mathsf{p}' \in X$. Similarly, $\overline{Y}[y, y']$ denotes the line segment between the points $Y(y)$ to $Y(y')$; we call this a *shortcut* of Y. For a set of numbers U, an *atomic interval* is a maximum interval of $\mathbb{R}$ that does not contain any point of U in its interior.

2.1. Background and standard definitions. Some of the material covered in this section is standard and follows the presentation in Driemel, Har-Peled, and Wenk [24]. A *reparameterization* is a one-to-one and continuous function $f : [0,1] \to [0,1]$. It is *orientation-preserving* if it maps $f(0) = 0$ and $f(1) = 1$. The Fréchet distance is defined only for oriented curves, as we need to match the start and end points of the curves. The orientation of the curves we use would be understood from the context.

DEFINITION 2.1. *Let $X : [0,1] \to \mathbb{R}^d$ and $Y : [0,1] \to \mathbb{R}^d$ be two polygonal curves. We define the* width *of an orientation-preserving reparametrization $f : [0,1] \to [0,1]$, with respect to X and Y, as*

$$\operatorname{width}_f(X,Y) = \max_{\alpha \in [0,1]} \|X(f(\alpha)) - Y(\alpha)\|.$$

The Fréchet distance *between the two curves is*

$$\mathsf{d}_{\mathcal{F}}(X,Y) = \inf_{f:[0,1]\to[0,1]} \operatorname{width}_f(X,Y).$$

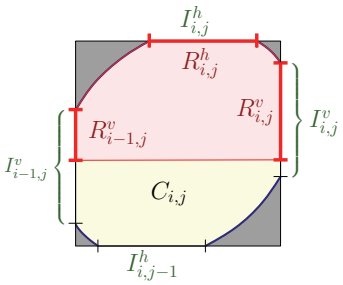
DEFINITION 2.2. *Let $X : [0,1] \to \mathbb{R}^d$ and $Y : [0,1] \to \mathbb{R}^d$ be two polygonal curves. The square $[0,1]^2$ represents their* parametric space. *For a point $\mathsf{p} = (x_{\mathsf{p}}, y_{\mathsf{p}}) \in [0,1]^2$, we define its* elevation *to be $\mathsf{d}(\mathsf{p}) = \|X(x_{\mathsf{p}}) - Y(y_{\mathsf{p}})\|$. Let $\delta > 0$ be a parameter; the* δ-free space *of X and Y is defined as*

$$\mathcal{D}_{\leq\delta}(X,Y) = \{\mathsf{p} \in [0,1]^2 \,\big|\, \mathsf{d}(\mathsf{p}) \leq \delta\}.$$

Free space diagram. We are interested only in polygonal curves, which we assume to have uniform parameterizations. The parametric space can be broken into a (not necessarily uniform) grid called the *free space diagram*, where a vertical line corresponds to a vertex of X and a horizontal line corresponds to a vertex of Y.

Every two segments of X and Y define a *free space cell* in this grid. In particular, let $C_{i,j} = C_{i,j}(X,Y)$ denote the free space cell that corresponds to the ith edge of X and the jth edge of Y. The cell $C_{i,j}$ is located in the ith column and jth row of this grid.

It is known that the free space, for a fixed δ, inside such a cell $C_{i,j}$ (i.e., $\mathcal{D}_{\leq\delta}(X,Y) \cap C_{i,j}$) is the clipping of an affine transformation of a disk to the cell [6], as such, it is convex and of constant complexity:



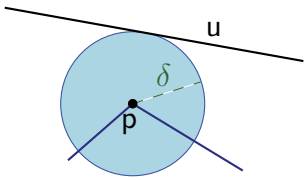
Let $I^h_{i,j}$ denote the horizontal *free space interval* at the top boundary of $C_{i,j}$, and $I^v_{i,j}$ denote the vertical free space interval at the right boundary.

We define the *complexity* of the relevant free space, for distance, δ, denoted by $\mathcal{N}_{\leq\delta}(X,Y)$, as the total number of grid cells that have a nonempty intersection with $\mathcal{D}_{\leq\delta}(X,Y)$.

OBSERVATION 2.3. *Given two segments* pq *and* uv*, it holds that* $\mathsf{d}_{\mathcal{F}}(\mathsf{pq},uv) = \max(\|u-\mathsf{p}\|,\ \|v-\mathsf{q}\|)$. *To see this, consider the uniform parameterization* $\mathsf{p}(t) = t\mathsf{p} + (1-t)\mathsf{q}$ *and* $u(t) = tu + (1-t)v$ *for* $t \in [0,1]$. *It is easy to verify that* $f(t) = \|\mathsf{p}(t) - u(t)\|$ *is convex, and as such* $f(t) \leq \max(f(0), f(1))$*, for any* $t \in [0,1]$.

Free space events. To compute the Fréchet distance consider increasing δ from 0 to ∞. As δ increases, structural changes happen to the free space. We are interested in the radii (i.e., the value of δ) of these events.

Consider a segment $\mathsf{u} \in X$ and a vertex $\mathsf{p} \in Y$; a *vertex-edge event* corresponds to the minimum radius of a ball centered at p, such that u is tangent to the ball:



In the free space diagram, this corresponds to the event that a free space interval consists of one point only. The line supporting this boundary edge corresponds to the vertex, and the other dimension corresponds to the edge. Naturally, the event could happen at a vertex of u. The second type of event, a *monotonicity event*, corresponds to a value δ for which a monotone subpath inside the δ-free space becomes feasible. Geometrically, this corresponds to the common distance of two vertices on one curve to the intersection point of their bisector with a segment on the other curve.

2.2. The *k*-shortcut Fréchet distance.

DEFINITION 2.4. *For a polygonal curve* Y*, we refer to any order-preserving concatenation of* $k+1$ *nonoverlapping (possibly empty) subcurves of* Y *with* k *shortcuts connecting the end points of the subcurves in the order along the curve as a* k-shortcut curve *of* Y. *Formally, for values* $0 \leq y_1 \leq y_2 \leq \cdots \leq y_{2k} \leq 1$*, the shortcut curve is defined as* $Y[0,y_1] + \overline{Y}[y_1,y_2] + Y[y_2,y_3] + \cdots + \overline{Y}[y_{2k-1},y_{2k}] + Y[y_{2k},1]$. *If each* $Y(y_i)$ *is a vertex of* Y*, we refer to the shortcut curve as being* vertex-restricted*; otherwise we say it is* unrestricted.

DEFINITION 2.5. *Given two polygonal curves* X *and* Y*, we define their* continuous k*-shortcut Fréchet distance as the minimal Fréchet distance between the curve* X *and any unrestricted* k*-shortcut curve of* Y. *We denote it with* $\mathsf{d}_{\mathrm{S}}(k,X,Y)$. *If we do not want to bound the number of shortcuts, we omit the parameter* k *and denote it with* $\mathsf{d}_{\mathrm{S}}(X,Y)$. *The* vertex-restricted k-shortcut Fréchet distance *is defined as above using only vertex-restricted shortcut curves of* Y. *Furthermore, note that in all cases we allow only one of the input curves to be shortcut, namely,* Y*, and thus we call the distance measure* directed.

In this paper, we study the directed vertex-restricted k-shortcut Fréchet distance for the case of bounded and unbounded k. In the following, we will omit the predicates *directed* and *vertex-restricted* when it is clear from the context.

Free space. The k*-reachable free space* $\mathcal{R}^k_{\leq\delta}(X,Y)$ is

$$\mathcal{R}^k_{\leq\delta}(X,Y) = \{\mathsf{p} = (x_{\mathsf{p}}, y_{\mathsf{p}}) \in [0,1]^2 |\ \mathsf{d}_{\mathrm{S}}(k, X[0,x_{\mathsf{p}}], Y[0,y_{\mathsf{p}}]) \leq \delta\}.$$

This is the set of points that have an (x, y)-monotone path from $(0, 0)$ that stays inside the free space and otherwise uses at most k tunnels, which are defined in the next subsection.

2.3. Tunnels and gates: Definitions.

2.3.1. Tunnels. In the parametric space, a shortcut $\overline{Y}[y_{\mathsf{p}}, y_{\mathsf{q}}]$ and the subcurve $X[x_{\mathsf{p}}, x_{\mathsf{q}}]$ that it is being matched to correspond to a rectangle with corners p and q, where $\mathsf{p} = (x_{\mathsf{p}}, y_{\mathsf{p}})$ and $\mathsf{q} = (x_{\mathsf{q}}, y_{\mathsf{q}})$. By shortcutting the curve on the vertical axis, we are collapsing this rectangle to a single row; see Figure 2.1(c). More precisely, this is the free space diagram of the shortcut and the subcurve. We call this row a *tunnel* and denote it by $\tau(\mathsf{p}, \mathsf{q})$. We require $x_{\mathsf{p}} \leq x_{\mathsf{q}}$ and $y_{\mathsf{p}} \leq y_{\mathsf{q}}$ for monotonicity. Figure 2.1 shows the full example of a tunnel. We call the Fréchet distance of the shortcut segment to the subcurve the *price* of this tunnel and denote it with $\mathrm{prc}(\tau(\mathsf{p}, \mathsf{q})) = \mathsf{d}_{\mathcal{F}}(X[x_{\mathsf{p}}, x_{\mathsf{q}}], \overline{Y}[y_{\mathsf{p}}, y_{\mathsf{q}}])$. A tunnel $\tau(\mathsf{p}, \mathsf{q})$ is *feasible* for δ if it holds that $\mathsf{d}(\mathsf{p}) \leq \delta$ and $\mathsf{d}(\mathsf{q}) \leq \delta$, i.e., if $\mathsf{p}, \mathsf{q} \in \mathcal{D}_{\leq \delta}(X, Y)$. (Note that in turn the feasibility of a monotone path in the free space of the tunnel is determined by the price of the tunnel.) Now, let $u = Y(y_{\mathsf{p}})$ and $v = Y(y_{\mathsf{q}})$ and let e be the edge of X that contains $X(x_{\mathsf{p}})$ (resp., e' the edge that contains $X(x_{\mathsf{q}})$) for the tunnel $\tau(\mathsf{p}, \mathsf{q})$. We denote with $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$ the *family of tunnels* that $\tau(\mathsf{p}, \mathsf{q})$ belongs to. Furthermore, let $\mathcal{T}_{\leq \delta}(\mathsf{e}, \mathsf{e}', u, v)$ denote the subset of these tunnels that are feasible for δ.

DEFINITION 2.6. *The* canonical tunnel *of the tunnel family $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$, denoted by $\tau_{\min}(\mathsf{e}, \mathsf{e}', u, v)$, is the tunnel that matches the shortcut uv to the subcurve $X[s, t]$ such that s and t are the values realizing*

$$\mathrm{r}_{\min}(\mathsf{e}, \mathsf{e}', u, v) = \min_{\substack{X(s) \in \mathsf{e}, X(t) \in \mathsf{e}', \\ s \leq t}} \max \begin{pmatrix} \|X(s) - u\|, \\ \|X(t) - v\| \end{pmatrix}. \tag{2.1}$$

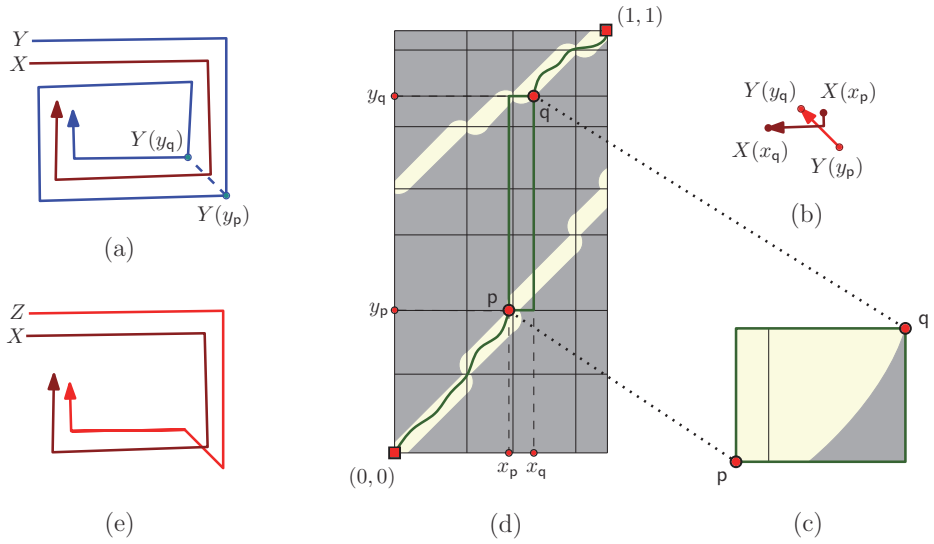


FIG. 2.1. (a) *Example of two dissimilar curves that can be made similar by shortcutting one of them.* (b) *A tunnel $\tau(\mathsf{p}, \mathsf{q})$ corresponds to a shortcut and a subcurve matched to each other and* (c) *their free space diagram.* (d) *The tunnel connects previously disconnected components of the free space.* (e) *The curve Z resulting from shortcutting Y. Its (regular) Fréchet distance from X is dramatically reduced.*

We refer to $\mathrm{r}_{\min}(\mathsf{e}, \mathsf{e}', u, v)$ *as the* minimum radius *of this family. The canonical tunnel may not be uniquely defined if only one of the two values* s *or* t *determines the minimum radius. In this case, we define* s *and* t *as the values minimizing* $\|X(s) - u\|$ *and* $\|X(t) - v\|$ *for* $X(s) \in \mathsf{e}$ *and* $X(t) \in \mathsf{e}'$*, individually. We call the price of the canonical tunnel the* canonical price *of this tunnel family.*

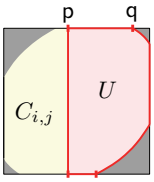
Clearly, one can compute the canonical tunnel $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$ in constant time. In particular, the price of this canonical tunnel is

$$\mathrm{prc}(\tau_{\min}(\mathsf{e}, \mathsf{e}', u, v)) = \mathsf{d}_{\mathcal{F}}(X[s, t], uv). \tag{2.2}$$

We emphasize that a shortcut is always a segment connecting two vertices of the curve Y, and a tunnel always lies in the parametric space; that is, they exist in two completely different domains.

Observation 2.7. *The minimum radius of a tunnel family* $\mathrm{r}_{\min}(\mathsf{e}, \mathsf{e}', u, v)$ *corresponds to either* (i) *the distance of* u *to its closest point on* e*,* (ii) *the distance of* v *to its closest point on* e'*, or* (iii) *the common distance of* u *and* v *to the intersection of their bisector with the edge* e *(i.e., a monotonicity event). Note that the event in case* (iii) *can happen only if* $\mathsf{e} = \mathsf{e}'$*.*

2.3.2. Gates. Let U be a subset of the parametric space that is convex in every cell. Let $I^h_{i,j}$ be a free space interval. We call the left end points of $U \cap I^h_{i,j}$ the *left gate* of U in the cell $C_{i,j}$, and similarly the right end point is the *right gate*. The figure below shows an example of gates p and q:



The *set of gates* of U are the gates with respect to all cells in the free space diagram. We define the *canonical gate* of a vertex-edge pair as the point in parametric space that minimizes the vertex-edge distance. Note that canonical gates serve as end points of canonical tunnels that span across columns in the free space diagram.

2.4. Curve simplification. We use the following simple algorithm for the simplification of the input curves. It is easy to verify that the curve simplified with parameter μ is in Fréchet distance at most μ to the original curve; see [24].

Definition 2.8. *Given a polygonal curve* $\boldsymbol{X}$ *and a parameter* $\mu > 0$*, first mark the initial vertex of* $\boldsymbol{X}$ *and set it as the current vertex. Now scan the polygonal curve from the current vertex until it reaches the first vertex that is in distance at least* μ *from the current vertex. Mark this vertex and set it as the current vertex. Repeat this until reaching the final vertex of the curve and also mark it. We refer to the resulting curve* X *that connects only the marked vertices, in their order along* $\boldsymbol{X}$*, as a* μ-simplification *of* $\boldsymbol{X}$ *and we denote it with* $\mathrm{simpl}(\boldsymbol{X}, \mu)$*.*

During the course of the algorithm we will simplify the input curves in order to reduce the complexity of the free space. The k-shortcut Fréchet distance does not satisfy the triangle inequality, as can be seen by the counterexample shown in Figure 2.2. Therefore, we need the next lemma to ensure that the computed distance between the simplified curves approximates the distance between the original curves. The proof is straightforward and can be found in [22].

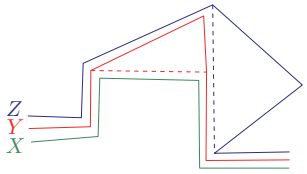


FIG. 2.2. *The directed k-shortcut Fréchet does not satisfy the triangle inequality. In the depicted counterexample it holds that* $\mathsf{d}_{\mathbb{S}}(k,X,Z) > \mathsf{d}_{\mathbb{S}}(k,X,Y) + \mathsf{d}_{\mathbb{S}}(k,Y,Z)$ *for any value of* k *and for* k *unbounded. This holds true in the vertex-restricted and the continuous case.*

LEMMA 2.9 (see [22]). *Given a simplification parameter* μ *and two polygonal curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$*, let* $X = \operatorname{simpl}(\boldsymbol{X},\mu)$ *and* $Y = \operatorname{simpl}(\boldsymbol{Y},\mu)$ *denote their* μ*-simplifications, respectively. For any* $k \in \mathbb{N}$*, it holds that* $\mathsf{d}_{\mathbb{S}}(k,X,Y) - 2\mu \leq \mathsf{d}_{\mathbb{S}}(k,\boldsymbol{X},\boldsymbol{Y}) \leq \mathsf{d}_{\mathbb{S}}(k,X,Y) + 2\mu$*. Similarly,* $\mathsf{d}_{\mathbb{S}}(X,Y) - 2\mu \leq \mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) \leq \mathsf{d}_{\mathbb{S}}(X,Y) + 2\mu$*.*

LEMMA 2.10 (see [24]). *For any two c-packed curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *in* $\mathbb{R}^d$ *of total complexity* n*, and two parameters* $0 < \varepsilon < 1$ *and* $\delta > 0$*, we have that*

$$\mathcal{N}_{\leq\delta}\left(\operatorname{simpl}(\boldsymbol{X},\mu), \operatorname{simpl}(\boldsymbol{Y},\mu)\right) = O(cn/\varepsilon),$$

where $\mu = \Theta(\varepsilon\delta)$*.*

2.5. Building blocks for the algorithm. The algorithm uses the following two nontrivial data structures.

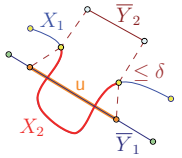
DATA STRUCTURE 2.11. *Given a polygonal curve* Z *with* n *vertices in* $\mathbb{R}^d$*, one can build a data structure, in* $O(\chi^2 n \log^2 n)$ *time, using* $O(\chi^2 n)$ *space, where* $\chi = \varepsilon^{-d}\log(1/\varepsilon)$*, that supports a procedure* **price**(p,q) *which receives two points* p *and* q *in the parametric space of* X *and* Y *and returns a value* ϕ *such that* $\phi \leq \operatorname{prc}(\tau(\mathsf{p},\mathsf{q})) \leq (1+\varepsilon)\phi$ *in* $O(\varepsilon^{-3}\log n \log\log n)$ *time. See section* 5.4 *and Theorem* 5.9*.*

DATA STRUCTURE 2.12. *For given parameters* ε *and* δ *and two c-packed curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *in* $\mathbb{R}^d$*, let* $X = \operatorname{simpl}(\boldsymbol{X},\mu)$ *and* $X = \operatorname{simpl}(\boldsymbol{Y},\mu)$*, where* $\mu = \varepsilon\delta$*. One can compute all the vertex-edge pairs of the two simplified curves* X *and* Y *in distance at most* δ *from each other in time* $O(n\log n + c^2 n/\varepsilon)$*. See below for details.*

We describe here how to realize Data Structure 2.12. Observe that X and Y have density $\phi = O(c)$. Now, we build the data structure of de Berg and Streppel [21] for the segments of Y (with $\varepsilon = 1/2$). For each vertex of X we compute all the segments of Y that are in distance at most δ from it, using the data structure [21]. Each query takes $O(\log n + k\phi)$ time, where k is the number of edges reported. Lemma 2.10 implies that the total sum of the k's is $O(cn/\varepsilon)$. We now repeat this for the other direction. This way, one can realize Data Structure 2.12.

2.6. Monotonicity of the prices of tunnels. The following two lemmas imply readily that under certain conditions the prices of tunnels which share an end point are approximately monotone with respect to the x-coordinate of their starting point. We will exploit this in the approximation algorithm that computes the reachability in the free space diagram. We will see in section 3.1 that this drastically reduces the number of tunnels that need to be inspected in order to decide if a particular cell is reachable.

LEMMA 2.13. *Given a value* $\delta > 0$ *and two curves* X_1 *and* X_2*, such that* X_2 *is a subcurve of* X_1*, and given two line segments* $\overline{Y}_1$ *and* $\overline{Y}_2$*, such that* $\mathsf{d}_{\mathcal{F}}(X_1,\overline{Y}_1) \leq \delta$ *and the start (resp., end) point of* X_2 *is in distance* δ *to the start (resp., end) point of* $\overline{Y}_2$*, then* $\mathsf{d}_{\mathcal{F}}(X_2,\overline{Y}_2) \leq 3\delta$*.*



Proof. Let u denote the subsegment of $\overline{Y}_1$ that is matched to X_2 under an optimal Fréchet mapping between X_1 and $\overline{Y}_1$. We know that $\mathsf{d}_{\mathcal{F}}(X_2, \mathsf{u}) \leq \delta$ by this mapping. The start point of $\overline{Y}_2$ is in distance 2δ to the start point of u, since they are both in distance δ to the start point of X_2 and the same holds for the end points. This implies that $\mathsf{d}_{\mathcal{F}}(\mathsf{u}, \overline{Y}_2) \leq 2\delta$. Now, by the triangle inequality, $\mathsf{d}_{\mathcal{F}}(X_2, \overline{Y}_2) \leq \mathsf{d}_{\mathcal{F}}(X_2, \mathsf{u}) + \mathsf{d}_{\mathcal{F}}(\mathsf{u}, \overline{Y}_2) \leq 3\delta$. ∎

LEMMA 2.14. *Consider two polygonal curves X and Y and three points p, q, and r in their free space, and let $\delta' = \max(\mathsf{d}(\mathsf{p}), \mathsf{d}(\mathsf{q}), \mathsf{d}(\mathsf{r}))$. If $x(\mathsf{p}) \leq x(\mathsf{q}) \leq x(\mathsf{r})$, then* $\mathrm{prc}(\tau(\mathsf{q}, \mathsf{r})) \leq 3 \max(\delta', \mathrm{prc}(\tau(\mathsf{p}, \mathsf{r})))$.[1]

Proof. Let X_1 be the subcurve $X[x_{\mathsf{p}}, x_{\mathsf{r}}]$, and let $X_2 = X[x_{\mathsf{q}}, x_{\mathsf{r}}]$. Similarly, let $\overline{Y}_1$ be the shortcut $\overline{Y}[y_{\mathsf{p}}, y_{\mathsf{r}}]$ and let $\overline{Y}_2 = \overline{Y}[y_{\mathsf{q}}, y_{\mathsf{r}}]$. By Lemma 2.13 $\mathrm{prc}(\tau(\mathsf{q}, \mathsf{r})) \leq 3\delta'$ for $\delta' = \max(\mathsf{d}_{\mathcal{F}}(X_1, \overline{Y}_1), \delta)$. ∎

We will see in section 3.3.2 that the monotonicity property of Lemma 2.14 also enables a faster search over tunnel events. The property holds even if the tunnels under consideration are not valid. For example, if $x_{\mathsf{p}} < x_{\mathsf{r}}$ and $y_{\mathsf{p}} > y_{\mathsf{r}}$, then the tunnel $\tau(\mathsf{p}, \mathsf{r})$ is not a valid tunnel and it cannot be used by a valid solution. Nevertheless, $\tau(\mathsf{p}, \mathsf{r})$ has a well-defined price, and these prices have the required monotonicity property.

LEMMA 2.15. *For a parameter $\delta \geq 0$, let $\mathsf{p}_1, \ldots, \mathsf{p}_m$ be m points in the δ-free space in increasing order by their x-coordinates, and let $\psi_i = \mathrm{prc}(\tau(\mathsf{p}_i, \mathsf{p}_m))$ for any $1 \leq i \leq m$. Then we have the following:*

(A) *If $\psi_i \geq \delta$, then for all $j > i$, we have* $\mathrm{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)) \leq 3\psi_i$.

(B) *If $\psi_i > 3\delta$, then for all $j < i$, we have* $\mathrm{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)) \geq \psi_i/3$.

Proof. To see the first part of the claim, note that by Lemma 2.14, $\mathrm{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)) \leq 3 \max(\delta, \psi_i) \leq 3\psi_i$. As for the second part, we have by the same lemma that $\delta < \psi_i/3 \leq \max(\delta, \mathrm{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m)))$ and thus $\psi_i/3 \leq \mathrm{prc}(\tau(\mathsf{p}_j, \mathsf{p}_m))$. ∎

3. Approximating the shortcut Fréchet distance. We describe the algorithm to approximate the directed vertex-restricted shortcut Fréchet distance between two given polygonal curves X and Y where the number of shortcuts that can be used in a solution is unbounded. In section 4 we prove the correctness and analyze the complexity of this algorithm.

3.1. The tunnel procedure. A key element in the decision procedure is the **tunnel** procedure depicted in Figure 3.1. During the decision procedure, we will repeatedly invoke the **tunnel** procedure with a set of gates R, for which we already know that they are contained in the reachable free space $\mathcal{R}^{\infty}_{\leq\delta}(X, Y)$, and the left gate associated with a horizontal free space interval of $\mathcal{D}_{\leq\delta}(X, Y)$, in order to determine if and to which extent this interval is reachable.

Intuitively, this procedure receives as input a set of reachable points in the parametric space and a free space interval (in the form of the left gate) and we are asking if there exists an *affordable* tunnel connecting a reachable point to the interval. Here, affordable means that its price is less than δ. More precisely, the procedure receives a set of gates R and a gate p as input and returns the end point of an (approximately) affordable tunnel that starts at a gate of R and ends at either p or the leftmost point to the right of p in the same free space interval. If a tunnel between a gate in R and the free space interval of p exists which has price less than δ, then the algorithm

[1] Here, $\mathsf{d}(\mathsf{p})$ is the elevation of p (see Definition 2.2), and $\tau(\mathsf{q}, \mathsf{r})$ is the tunnel between q and r (see section 2.3.1).

tunnel$(R, \mathsf{p}, \varepsilon, \delta)$

1: Let $\mathsf{q} = (x_\mathsf{q}, y_\mathsf{q})$ point in R with max value of x_q,
   such that $x_\mathsf{q} \leq x_\mathsf{p}$ and $y_\mathsf{q} < y_\mathsf{p}$, where $\mathsf{p} = (x_\mathsf{p}, y_\mathsf{p})$.
2: $\phi =$ **price**(q, p), see Data Structure 2.11.
3: **if** $\phi \leq 3\delta$ **then**
4: Return p `// tunnel` $\tau(\mathsf{q}, \mathsf{p})$
5: Compute j such that $x_\mathsf{p} \in \mathcal{I}_{\text{edge}}(X, j) = [x_j, x_{j+1}]$
6: Let $\mathsf{q} = (x_\mathsf{q}, y_\mathsf{q})$ point in R with min value of x_q,
   such that $x_\mathsf{q} \in \mathcal{I}_{\text{edge}}(X, j)$, $x_\mathsf{q} \geq x_\mathsf{p}$, and $y_\mathsf{q} < y_\mathsf{p}$
7: **if** q does not exist **then**
8: Return null.
9: $\mathsf{v} = (x_\mathsf{q}, y_\mathsf{p})$
10: **if** $\mathsf{d}(\mathsf{v}) \leq \delta$ **then**
11: Return v `// vertical tunnel` $\tau(\mathsf{q}, \mathsf{v})$
12: **else**
13: Return null.

FIG. 3.1. *The* **tunnel** *procedure receives a set of gates R and a gate p in the parametric space and returns the end point of an affordable tunnel between R and p (or a point close to it) if it exists. The technical details of the range queries in Line* 1 *and Line* 6 *are described in the proof of Lemma* 4.2.

will return the end point of a tunnel of price less than or equal to $(1 + \varepsilon)3\delta$. If the algorithm returns null, then we know that no such tunnel of price less than δ exists.

The main idea of the **tunnel** procedure is the following. For a given tunnel, we can $(1 + \varepsilon)$-approximate its price, using a data structure which answers these queries in polylogarithmic time; see Data Structure 2.11. The desired tunnel could be a vertical tunnel which starts at a gate of R or could be a tunnel between a gate of R and p. Naively, one could test all tunnels that start from a gate in R and end in p; however, this takes time at least linear in the size of R. Since we are only interested in a constant factor approximation, it is sufficient, by Lemma 2.14, to test only the tunnel which corresponds to the shortest subcurve of X. The corresponding gates can be found in polylogarithmic time using a two-dimensional range tree, which is built on the set R and we assume is available to us. We can maintain the range tree during the decision procedure depicted in Figure 3.2. The technical details are described in the proof of Lemma 4.2. An alternative solution that uses a balanced binary search tree only is described in [22].

3.2. The decision algorithm. In the decision problem we want to know whether the shortcut Fréchet distance between two curves, X and Y, is smaller than or equal to a given value δ. The free space diagram $\mathcal{D}_{\leq\delta}(X, Y)$ may consist of a certain number of disconnected components and our task is to find a monotone path from $(0, 0)$ to $(1, 1)$ that traverses these components while using shortcuts between vertices of Y to "bridge" between points in different components or where there is no monotone path connecting them (see Figure 2.1). The decision algorithm exploits the monotonicity of the tunnel prices shown in Lemma 2.14 and is based on a breadth-first search in the free space diagram. (A similar idea was used in [24], but here the details are more involved.)

Given two curves X and Y and parameters δ and ε, the algorithm may output an answer equivalent to "yes" if there exists a shortcut curve Y' of Y such that

decider$(X, Y, \varepsilon, \delta)$
1: Assert that $\mathsf{d}(0,0) = \|X(0) - Y(0)\| \le \delta$ and $\mathsf{d}(1,1) \le \delta$
2: Let Q be a min-priority queue for nodes $v(i,j)$ with keys $(jn+i)$
3: Compute and enqueue the cells $C_{i,j}$ that have non-empty $I^h_{i,j}$ or $I^v_{i,j}$.
4: Let $R = \{(0,0)\}$.
5: **while** $Q \neq \emptyset$ **do**
6: Dequeue node $v(i,j)$ and its copies from Q
7: Let p be the left gate of $I^h_{i,j}$
8: $\mathsf{v} =$ **tunnel**$(R, \mathsf{p}, \varepsilon, \delta)$
9: Compute $R^h_{i,j}$ and $R^v_{i,j}$ from v, $R^v_{i-1,j}$, $R^h_{i,j-1}$, $I^v_{i,j}$ and $I^h_{i,j}$
10: **if** $R^v_{i,j} \neq \emptyset$ **then**
11: Enqueue $v(i+1,j)$ and insert edge between $v(i,j)$ and $v(i+1,j)$
12: **if** $R^h_{i,j} \neq \emptyset$ **then**
13: Enqueue $v(i,j+1)$ and insert edge between $v(i,j)$ and $v(i,j+1)$
14: Add gates of $R^h_{i,j}$ to R
15: **if** $(1,1) \in R$ **then**
16: Return "$\mathsf{d}_{\mathbb{S}}(X,Y) \le (1+\varepsilon)3\delta$"
17: **else**
18: Return "$\mathsf{d}_{\mathbb{S}}(X,Y) > \delta$"

FIG. 3.2. *The decision procedure* **decider** *for the shortcut Fréchet distance.*

$\mathsf{d}_{\mathbb{S}}(X,Y') \le \delta$ and an answer equivalent to "no" if there exists no shortcut curve such that $\mathsf{d}_{\mathbb{S}}(X,Y') \le (1+\varepsilon)3\delta$.

3.2.1. Detailed description of the decision procedure. The decision algorithm is depicted in Figure 3.3 (and Figure 3.2). The algorithm uses a directed graph G that has a node $v(i,j)$ for every free space cell $C_{i,j}$ whose boundary has a nonempty intersection with the free space $\mathcal{D}_{\le\delta}(X,Y)$. These intersections are defined as the free space intervals $I^h_{i,j}$, $I^v_{i,j}$, $I^h_{i-1,j}$, and $I^v_{i,j-1}$; see section 2.1. For any path along the edges of the graph G from $v(1,1)$ to $v(i,j)$, there exists a monotone path that traverses the corresponding cells of $\mathcal{D}_{\le\delta}(X,Y)$ while using zero or more affordable tunnels. A node $v(i,j)$ can have an incoming edge from another node $v(i',j')$ if $i' \le i$ and $j' \le j$ and either $v(i',j')$ is a neighboring node or the two cells can be connected by an affordable tunnel which starts at the lower boundary of the cell corresponding to $v(i',j')$ and ends at the upper boundary of the cell corresponding to $v(i,j)$. The idea of the algorithm is to propagate reachability intervals $R^v_{i,j} \subset I^v_{i,j}$ and $R^h_{i,j} \subset I^h_{i,j}$ while traversing a sufficiently large subgraph starting from $v(1,1)$ and computing the necessary parts of this subgraph on the fly. We store these intervals with the cell $v(i,j)$ that has them on the top (resp., right) boundary. The reachability intervals $R^v_{i,j}$ being computed satisfy

Decider$(\boldsymbol{X}, \boldsymbol{Y}, \varepsilon, \delta)$
1: Let $\varepsilon' = \varepsilon/10$
2: Compute $X = \text{simpl}(\boldsymbol{X}, \mu)$ and $Y = \text{simpl}(\boldsymbol{Y}, \mu)$ with $\mu = \varepsilon'\delta$
3: Call **decider**$(X, Y, \varepsilon', \delta')$ with $\delta' = (1+2\varepsilon')\delta$
4: Return either "$\mathsf{d}_{\mathbb{S}}(\boldsymbol{X}, \boldsymbol{Y}) \le (1+\varepsilon)3\delta$" or "$\mathsf{d}_{\mathbb{S}}(\boldsymbol{X}, \boldsymbol{Y}) > \delta$"

FIG. 3.3. *The resulting decision procedure* **Decider**. *A detailed description of the complete algorithm is given in section* 3.2.1.

$$\mathcal{R}^{\infty}_{\leq\delta}(X,Y)\cap I^{v}_{i,j}\subseteq R^{v}_{i,j}\subseteq\mathcal{R}^{\infty}_{\leq(1+\varepsilon)3\delta}(X,Y)\cap I^{v}_{i,j}, \tag{3.1}$$

and an analogous statement applies to $R^h_{i,j}$. The aim is to determine if either $(1,1)\in\mathcal{R}^{\infty}_{\leq(1+\varepsilon)3\delta}(X,Y)$ or $(1,1)\notin\mathcal{R}^{\infty}_{\leq\delta}(X,Y)$. Throughout the whole algorithm we also maintain a set of gates R, which represents the end points of the horizontal reachability intervals computed so far.

We will traverse the graph by handling the nodes in a row-by-row order, thereby handling any node $v(i,j)$ only after we handled the nodes $v(i',j')$, where $j'\leq j$, $i'\leq i$, and $(i'+j')<(i+j)$. To this end we keep the nodes in a min-priority queue where the node $v(i,j)$ has the key $(jn+i)$. The correctness of the computed reachability intervals will follow by induction on the order of these keys. Furthermore, it will ensure that we handle each node at most once and that we traverse at most three of the incoming edges to each node of the graph.

The queue is initialized with the entire node set at once. To compute this initial node set and the corresponding free space intervals we use Data Structure 2.12. The algorithm then proceeds by handling nodes in the order of extraction from this queue. When dequeuing nodes from the queue, the same node might appear three times (consecutively) in this queue. Once from each of its direct neighbors in the grid and once from the initial enqueuing.

In every iteration, the algorithm dequeues the one or more copies of the same node $v(i,j)$ and merges them into one node if necessary. Assume that $v(i,j)$ has an incoming edge that corresponds to an affordable tunnel. Let p be the left gate of $I^h_{i,j}$. We invoke $\textbf{tunnel}(R,\mathsf{p},\varepsilon,\delta)$ to test if this is the case. If the call returns null, then there is no such affordable tunnel. Otherwise, we know that the returned point v is contained in $R^h_{i,j}$. If there is more than one copy of this node in the queue, we also access the reachability intervals of the one or two neighboring vertices (i.e., $R^v_{i-1,j}$ and $R^h_{i,j-1}$). Using the reachability information from the at most three incoming edges obtained this way, we can determine if the cells $C_{i,j+1}$ and $C_{i+1,j}$ are reachable by computing the resulting reachability intervals $R^h_{i,j}$ at the top side and $R^v_{i,j}$ the right side of the cell $C_{i,j}$. Since the free space within a cell is convex and of constant complexity, this can be done in constant time.

Now, if $R^h_{i,j}\neq\emptyset$ we create a node $v(i,j+1)$, connect it to $v(i,j)$ by an edge, enqueue it, and add the gates of $R^h_{i,j}$ to R. If $R^v_{i,j}\neq\emptyset$ we create a node $v(i+1,j)$, connect it to $v(i,j)$ by an edge, and enqueue it. If we discover that the top right corner of the free space diagram is reachable this way, we output the equivalent to "yes" and the algorithm terminates. In this case we must have added $(1,1)$ as a gate to R. The algorithm may also terminate before this happens if there are no more nodes in the queue; in this case we output that no suitable shortcut curve exists.

3.3. The main algorithm. The given input is two curves X and Y. We want to use the approximate decision procedure **Decider**, described above, in a binary search fashion to compute the shortcut Fréchet distance. Conceptually, one can think of the decider as being exact. In particular, the algorithm would, for a given value of δ, call the decision procedure twice with parameters δ and $\delta'=\delta/4$ (using $\varepsilon=1/3$). If the two calls agree, then we can make an exact decision; if the two calls disagree, then we can output a $O(1)$-approximation of the shortcut Fréchet distance.

The challenge is how to choose the right subset of candidate values to guide this binary search. Some of the techniques used for this search have been introduced in previous papers. In particular, this holds for the search over vertex-vertex, vertex-edge, and monotonicity events which we describe as preliminary computations in

section 3.3.1. This stage of the algorithm eliminates the candidate values that also need to be considered for the approximation of the standard Fréchet distance and it is almost identical to the algorithm presented in [24].

As mentioned, a monotone path could also become usable by taking a tunnel. There are two types of events associated with a tunnel family: the first time such that any tunnel in this family is feasible, which is the *creation radius*. Fortunately, the creation radii of all tunnels are approximated by the set of vertex-vertex and vertex-edge event radii, and our first-stage search (see section 3.3.1) would thus take care of such events.

The second event we have to worry about is the first time that the feasible family of tunnels becomes usable via a tunnel (i.e., the price of some tunnel in this family is below the distance threshold δ). Luckily, it turns out that it is sufficient to search over the price of the canonical tunnel associated with such a family. The price of a specific tunnel can be approximated quickly using Data Structure 2.11. However, there are $\Theta(n^4)$ tunnel families, and potentially the algorithm has to consider all of them. Fortunately, because of c-packedness, only $O(n^2)$ of these events are relevant. A further reduction in running time is achieved by using a certain monotonicity property of the prices of these tunnels and our ability to represent them implicitly to search over them efficiently.

3.3.1. The algorithm: First stage. We are given two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ with total complexity n. We repeatedly compute sets of event values and perform binary searches on these values as follows.

We compute the set of vertices V of the two curves, and using well-separated pairs decomposition, we compute, in $O(n \log n)$ time, a set U of $O(n)$ distances that, up to a factor of two, represents any distance between any two vertices of V. Next, we use **Decider** (with fixed $\varepsilon = 1/3$) to perform a binary search for the atomic interval in U that contains the desired distance. Let $[\alpha, \beta]$ denote this interval. If $10\alpha \geq \beta/10$, then we are done, since we found a constant size interval that contains the Fréchet distance. Otherwise, we use the decision procedure to verify that the desired radius is not in the range $[\alpha, 10\alpha]$ and $[\beta/10, \beta]$. For $\alpha' = 3\alpha$, $\beta' = \beta/3$, let $\mathcal{I}' = [\alpha', \beta']$ denote the obtained interval.

We now continue the search using only **decider** and the simplified curves $X = \mathrm{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathrm{simpl}(\boldsymbol{Y}, \mu)$, where $\mu = \alpha'$. We extract the vertex-edge events of X and Y that are smaller than β'; see section 2.1. To this end, we compute all edges of X that are in distance at most β' of any vertex of Y and vice versa using Data Structure 2.12. Let U' be the set of resulting distances. We perform a binary search using **decider** to find the atomic interval $\mathcal{I}'' = [\alpha'', \beta'']$ of $U' \cap \mathcal{I}'$ that contains the shortcut Fréchet distance between X and Y.

Finally, we again search the margins of this interval, so that either we find the desired approximation or alternatively we output the interval $[10\alpha'', \beta''/10]$,

3.3.2. Second stage: Searching over tunnel prices. It remains to search over the canonical prices of tunnel families $\mathcal{T}(\mathsf{e}, \mathsf{e}', u, v)$, where $\mathsf{e} \neq \mathsf{e}'$[2]. After the first stage, we have an interval $[\alpha, \beta] = [10\alpha'', \beta''/10]$, and simplified curves X and Y of which the shortcut Fréchet distance is contained in $[\alpha, \beta]$ and approximates $\mathsf{d}_{\mathsf{S}}(\boldsymbol{X}, \boldsymbol{Y})$. By Lemma 2.10, the number of vertex-edge pairs in distance β is bounded by $O(cn/\varepsilon)$. The corresponding horizontal grid edges in the parametric space contain the canonical gates which are feasible for any value in $[\alpha, \beta]$. Let P denote the $m = O(cn/\varepsilon)$ points

[2]Since for the case where $\mathsf{e} = \mathsf{e}'$ the canonical price coincides with the creation event value.

in the parametric space that correspond to the canonical gates of these vertex-edge pairs; that is, for every feasible pair p (a vertex of Y) and e (an edge of X), we compute the closest point q on e to p and place the point corresponding to (q, p) in the free space into P.

It is sufficient to consider the tunnel families between these vertex-edge pairs, since all other families are not feasible in the remaining search interval. Thus, if we did not care about the running time, we could compute and search over the prices of the tunnels $\mathsf{P} \times \mathsf{P}$ using Data Structure 2.11. Naively, this would take roughly quadratic time. Instead, we use a more involved implicit representation of these tunnels to carry out this task.

Implicit search over tunnel prices. Consider the implicit matrix of tunnel prices $M = \mathsf{P} \times \mathsf{P}$ where the entry $M(i, j)$ is a $(1 + \varepsilon)$-approximation to the price of the canonical tunnel $\tau(\mathsf{p}_i, \mathsf{p}_j)$. By Lemma 2.15, the first j values of the jth row of this matrix are monotonically decreasing up to a constant factor, since they correspond to tunnels that share the same end point p_j and are ordered by their starting points p_i. (We ignore the values in this matrix above the diagonal.) Using Data Structure 2.11 we can $(1 + \varepsilon)$-approximate a value in the matrix in polylogarithmic time per entry. Similarly, the lower triangle of this matrix is sorted in increasing order in each column. As such, this matrix is sorted in both rows and columns and one can apply the algorithm of Frederickson and Johnson [26] to find the desired value. This requires $O(\log m)$ calls to **Decider** and the evaluation of $O(m)$ entries in the matrix and takes $O(m)$ time otherwise. Here, we are using **Decider** as an exact decision procedure. The algorithm will terminate this search with the desired constant factor approximation to the shortcut Fréchet distance.

4. Analysis.

4.1. Analysis of the tunnel procedure.

LEMMA 4.1. *Given the left gate* p *of a free space interval* $I^h_{i,j}$ *and a set of gates* R*, and parameters* $0 < \varepsilon \leq 1$ *and* $\delta > 0$*, the algorithm* **tunnel** *depicted in Figure* 3.1 *outputs one of the following:*

(i) *a point* $\mathsf{v} \in I^h_{i,j}$ *such that there exists a tunnel* $\tau(\mathsf{q}, \mathsf{v})$ *of price* $\operatorname{prc}(\tau(\mathsf{q}, \mathsf{v})) \leq (1 + \varepsilon)3\delta$ *from a gate* $\mathsf{q} \in R$*, or*

(ii) null*; in this case, there exists no tunnel of price less than or equal to* δ *between a gate of* R *and a point in* $I^h_{i,j}$*.*

Furthermore, in case (i)*, there exists no other point* $\mathsf{r} \in [\mathsf{p}, \mathsf{v}]$ *that is the end point of a tunnel from* R *with price less than or equal to* δ*.*

Proof. The correctness of this procedure follows from the monotonicity of the tunnel prices, which is affirmed by Lemma 2.14. Let ϕ be the $(1 + \varepsilon)$-approximation to the price of the tunnel, which we compute in line 2. This tunnel starts at a point in R and ends in p and it corresponds to the shortest subcurve $\widehat{X}$ of X over any such tunnel. Lemma 2.14 implies that if $\phi < 3\delta$, then there can be no other tunnel of price less than δ, which corresponds to a subcurve of X that contains $\widehat{X}$. Therefore, the price of any tunnel from a point $\mathsf{q} \in R$, which lies in the lower left quadrant of p, to a point that lies in the upper right quadrant of p has a price larger than δ. In particular, this holds for those tunnels that end to the right of p in the same free space interval. The only other possibility for a tunnel from R to $I^h_{i,j}$ is a vertical tunnel that lies to the right of p. Observe that a vertical tunnel which is feasible for δ always has price at most δ, since it corresponds to a subcurve of X that is equal to a point which is in distance δ to the shortcut edge. In line 5 and line 6 we compute the leftmost gate of R in the lower right quadrant of p which lies in the same column as p. If there exists

such a point with a vertical tunnel that ends in the free space interval $I^h_{i,j}$, then we return the end point of this tunnel. Otherwise we can safely output the equivalent to the answer that there exists no tunnel of price less than δ. ∎

4.2. Analysis of the decision procedure. Clearly, the priority queue operations take $O(N \log N)$ time and $O(N)$ space, where $N = \mathcal{N}_{\leq\delta}(X, Y)$ is the size of the node set, which corresponds to the complexity of the free space diagram. We invoke the **tunnel** procedure once for each node. Since we add at most a constant number of gates for every cell to R, the size of this set is also bounded by $O(N)$. Therefore, after the initialization the algorithm takes time near linear in the complexity of the free space diagram. We can reduce this complexity by first simplifying the input curves with $\mu = \Theta(\varepsilon\delta)$ before invoking the **decider** procedure, thereby paying another approximation factor. We denote the resulting wrapper algorithm with **Decider**; it is depicted in Figure 3.3. Now, the initial computation of the nodes takes near linear time by Data Structure 2.12 and therefore the overall running time is near linear. A more detailed analysis of the running time can be found in the following.

LEMMA 4.2. *Assume parameters $\delta > 0$ and $0 < \varepsilon \leq 1$ and two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^d$ of total complexity n. The algorithm* **Decider** *depicted in Figure* 3.3 *outputs one of the following:* (i) $\mathsf{d}_{\mathsf{S}}(\boldsymbol{X}, \boldsymbol{Y}) \leq (1+\varepsilon)3\delta$ *or* (ii) $\mathsf{d}_{\mathsf{S}}(\boldsymbol{X}, \boldsymbol{Y}) > \delta$. *In any case, the output returned is correct. The running time is $O(Cn \log^2 n)$, where $C = c^2\varepsilon^{-2d}\log(1/\varepsilon)$.*

Proof. The algorithm **Decider** computes the simplified curves $X = \mathrm{simpl}(\boldsymbol{X}, \mu)$ and $Y = \mathrm{simpl}(\boldsymbol{Y}, \mu)$ with $\mu = \Theta(\varepsilon\delta)$ before invoking the algorithm **decider** described in Figure 3.2 on these curves. By the correctness of the **tunnel** procedure (i.e., Lemma 4.1), one can argue by induction that the subsets of points of $\mathcal{R}^{\infty}_{\leq\delta}(X, Y)$ intersecting a grid edge are sufficiently approximated by the reachable intervals computed by **decider** (see (3.1)). By Lemma 2.9, this sufficiently approximates the decision with respect to the original curves.

It remains to analyze the running time. By Lemma 2.10, the size of the node set of the graph G is bounded by $N = O(cn/\varepsilon)$. This also bounds the size of the point set R and the number of calls to the **tunnel** procedure, as those are at most a constant number per node. During the **tunnel** procedure, which is depicted in Figure 3.1, we

(A) approximate the price of one tunnel in line 2 and

(B) invoke two orthogonal range queries on the set R in line 1 and line 6.

As for (A), building the data structure that supports this kind of query takes $T_1 = O(n\varepsilon^{-2d}\log^2(1/\varepsilon)\log^2 n)$ time by Data Structure 2.11. Since we perform $O(N)$ such queries, this takes $T_2 = O(N\varepsilon^{-3}\log n \log\log n) = O(cn\varepsilon^{-4}\log n\log\log n)$ time overall. As for (B), again, the set of gates R is a finite set of two-dimensional points and we can use two-dimensional range trees (with fractional cascading as described in [19]) to support the orthogonal range queries. We want to build this tree by adding $O(N)$ points throughout the algorithm execution. Since the range tree is a static data structure, we have to make it dynamic, but we need to support only insertions and no deletions. This can be easily done by using the logarithmic method if we allow an additional logarithmic factor to the running time; see also [7, 36]. In this method, the point set is distributed over $O(\log N)$ static range trees, which need to be queried independently and which are repeatedly rebuilt throughout the algorithm. Overall, maintaining this data structure and answering the orthogonal range queries takes $T_3 = O(N \log^2 N)$ time.

During the algorithm, we maintain a priority queue, where each node is added and extracted at most three times. As such, the priority queue operations take time

in $O(N \log N)$. The initial computation of the node set takes $T_4 = O(n \log n + c^2 n/\varepsilon)$ by Data Structure 2.12.

Therefore, the overall running time is $T_1 + T_2 + T_3 + T_4$, which is

$$O(n\varepsilon^{-2d} \log^2(1/\varepsilon) \log^2 n + cn\varepsilon^{-4} \log n \log\log n + cn \log^2 n + n \log n + c^2 n/\varepsilon) = O(Cn \log^2 n),$$

where $C = c^2 \varepsilon^{-2d} \log(1/\varepsilon)$. ∎

OBSERVATION 4.3. *It is easy to modify the* **decider** *algorithm such that it also outputs the respective shortcut curve and reparametrization which satisfies the Fréchet distance. We would modify the tunnel procedure such that it returns not only the end point but also the starting point of the computed tunnel. During the algorithm, we then insert an edge for each computed tunnel, thereby creating at most three incoming edges to each node. After the algorithm terminates, we can trace any path backward from* $(1,1)$ *to* $(0,0)$ *in the subgraph computed this way. This path encodes the shortcut curves as well as the reparametrizations.*

4.3. Analysis: Understanding tunnel events. The main algorithm uses the procedure **Decider** to perform a binary search for the minimum δ for which the decision procedure returns "yes." In the problem at hand we are allowed to use tunnels to traverse the free space diagram, and it is possible that a path becomes feasible by introducing a tunnel. The algorithm has to consider this new type of critical event.

Consider the first time (i.e., the minimal value of δ) that a decision procedure would try to use a tunnel of a certain family.

DEFINITION 4.4. *Given a tunnel family* $\mathcal{T}(\mathsf{e}_i, \mathsf{e}_j, u, v)$*, we call the minimal value of* δ *such that* $\mathcal{T}_{\leq\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ *is nonempty the* creation radius *of the tunnel family and we denote it with* $\mathrm{r}_{\mathrm{crt}}(\mathsf{e}_i, \mathsf{e}_j, u, v)$*. (Note that the price of a tunnel might be considerably larger than its creation radius.)*

LEMMA 4.5. *The creation radius* $\mathrm{r}_{\mathrm{crt}}(\mathsf{e}_i, \mathsf{e}_j, u, v) = \mathrm{r}_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v)$*; see Definition* 2.6*.*

Proof. Recall that the creation radius of the tunnel family is the minimal value of δ such that any tunnel in this family is feasible. Let u' be the closest point on e_i to u, and v' the closest point on e_j to v. If u' appears before v' on X, then the canonical tunnel is realized by $X(x_{\mathsf{q}}) = u'$ and $X(x_{\mathsf{q}}) = v'$ and the claim holds. In particular, this is the case if $i < j$.

Now, the only remaining possibility is that u' appears after v' on e. It must be that $i = j$; therefore let $\mathsf{e} = \mathsf{e}_i = \mathsf{e}_j$. Observe that in this case any tunnel in the family which is feasible for δ also has a price that is smaller than or equal to δ. Consider the point r realizing the quantity

$$\min_{\mathsf{r} \in \mathsf{e}} \max(\|\mathsf{r} - u\|, \|\mathsf{r} - v\|).$$

Note that r is the subcurve of X corresponding to the (vertical) canonical tunnel in this case. We claim that for any subsegment $\widehat{u}\widehat{v} \subseteq \mathsf{e}$ (agreeing with the orientation of e) we have that $\mathsf{d}_{\mathcal{F}}(\widehat{u}\widehat{v}, uv) \geq \mathsf{d}_{\mathcal{F}}(\mathsf{r}, uv)$. If $\widehat{u} = \widehat{v}$, then the claim trivially holds.

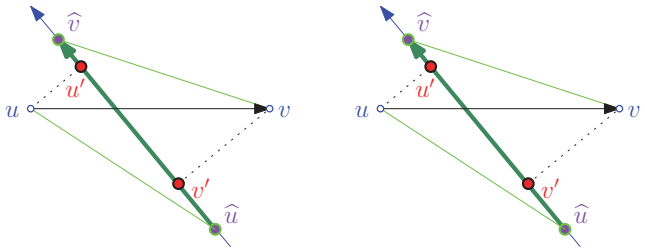


FIG. 4.1. *Two cases: v' appears either before or after $\widehat{u}$ along e, assuming that u' appears after v' on e.*

Assume that v' appears after $\widehat{u}$ along e (the case depicted in Figure 4.1). Since u' appears after v' along e, we have that $\|v'-u\| \le \|\widehat{u}-u\|$, as moving away from u' only increases the distance from u. Therefore,

$$\begin{aligned}\mathsf{d}_{\mathcal{F}}(\mathsf{r}, uv) &\le \mathsf{d}_{\mathcal{F}}(v', uv) = \max(\|v'-u\|, \|v'-v\|) \le \max(\|\widehat{u}-u\|, \|\widehat{v}-v\|)\\ &= \mathsf{d}_{\mathcal{F}}(\widehat{u}\widehat{v}, uv).\end{aligned}$$

Otherwise, if v' appears before $\widehat{u}$ along e, as depicted in Figure 4.1 on the right, then

$$\begin{aligned}\mathsf{d}_{\mathcal{F}}(\mathsf{r}, uv) &\le \mathsf{d}_{\mathcal{F}}(\widehat{u}, uv) = \max(\|\widehat{u}-u\|, \|\widehat{u}-v\|) \le \max(\|\widehat{u}-u\|, \|\widehat{v}-v\|)\\ &= \mathsf{d}_{\mathcal{F}}(\widehat{u}\widehat{v}, uv),\end{aligned}$$

since moving away from v' only increases the distance from v.

This implies that the minimum δ for a tunnel in $\mathcal{T}(\mathsf{e}_i, \mathsf{e}_i, u, v)$ to be feasible is at least $\mathsf{d}_{\mathcal{F}}(\mathsf{r}, uv) = \mathrm{r}_{\mathrm{crt}}(\mathsf{e}_i, \mathsf{e}_i, u, v)$. And r testifies that there is a tunnel in this family that is feasible for this value. ∎

The following lemma describes the behavior when δ rises above a tunnel price, such that the area in the free space that lies beyond this tunnel potentially becomes reachable by using this tunnel. More specifically, it implies that the first time (i.e., the minimal value of δ) that any tunnel of a family $\mathcal{T}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ is usable (i.e., its price is less than δ), any tunnel in the feasible set $\mathcal{T}_{\le\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ associated with this family will be usable.

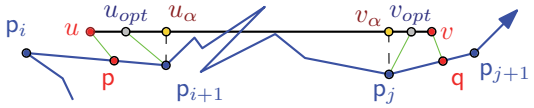
LEMMA 4.6. *Given a value $\delta \ge 0$, we have for any tunnel $\tau(\mathsf{f}, \mathsf{g})$ in the feasible subset of a given tunnel family $\mathcal{T}_{\le\delta}(\mathsf{e}_i, \mathsf{e}_j, u, v)$ that*

(i) *if $\delta \le \mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v))$, then $\mathrm{prc}(\tau(\mathsf{f}, \mathsf{g})) = \mathrm{prc}(\tau_{\min}(\mathsf{e}_i, \mathsf{e}_j, u, v))$,*

(ii) *otherwise, $\mathrm{prc}(\tau(\mathsf{f}, \mathsf{g})) \le \delta$.*

Proof. We first handle the case that $i \ne j$. Let $\mathsf{e}_i = \mathsf{p}_i\mathsf{p}_{i+1}$ and $\mathsf{e}_j = \mathsf{p}_j\mathsf{p}_{j+1}$.

Let $\mathsf{p} \in \mathsf{e}_i$ and $\mathsf{q} \in \mathsf{e}_j$ be some points on these edges that correspond to f and g, respectively. Observe that since this is a feasible tunnel in this family, we have that



$$\max(\|\mathsf{p}-u\|, \|\mathsf{q}-v\|) \le \delta.$$

Consider the optimal Fréchet matching of $X\langle \mathsf{p}, \mathsf{q}\rangle$ with uv, and let u_{opt} and v_{opt} be the points on uv that are matched to p_{i+1} and p_j by this optimal Fréchet matching. Let $\alpha = \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1}, \mathsf{p}_j\rangle, u_\alpha v_\alpha)$, where $u_\alpha v_\alpha$ is the subsegment of uv minimizing α.

We have by Observation 2.3 that

$$\begin{aligned}
\mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p},\mathsf{q}\rangle, uv) &= \max\Big(\mathsf{d}_{\mathcal{F}}(\mathsf{p}\mathsf{p}_{i+1}, uu_{\mathrm{opt}}), \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1},\mathsf{p}_j\rangle, u_{\mathrm{opt}}v_{\mathrm{opt}}), \mathsf{d}_{\mathcal{F}}(\mathsf{p}_j\mathsf{q}, v_{\mathrm{opt}}v),\Big),\\
&= \max\begin{pmatrix} \|\mathsf{p}-u\|,\\ \|\mathsf{p}_{i+1}-u_{\mathrm{opt}}\|,\\ \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1},\mathsf{p}_j\rangle, u_{\mathrm{opt}}v_{\mathrm{opt}}),\\ \|\mathsf{p}_j - v_{\mathrm{opt}}\|,\\ \|\mathsf{q}-v\|, \end{pmatrix},\\
&= \max\left(\|\mathsf{p}-u\|, \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1},\mathsf{p}_j\rangle, u_{\mathrm{opt}}v_{\mathrm{opt}}),\ \|\mathsf{q}-v\|\right)\\
&\geq \max(\|\mathsf{p}-u\|, \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1},\mathsf{p}_j\rangle, u_\alpha v_\alpha), \|\mathsf{q}-v\|)\\
&= \max\left(\mathsf{d}_{\mathcal{F}}(\mathsf{p}\mathsf{p}_{i+1}, uu_\alpha),\ \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1},\mathsf{p}_j\rangle, u_\alpha v_\alpha), \mathsf{d}_{\mathcal{F}}(\mathsf{p}_j\mathsf{q}, v_1 v)\right)\\
&\geq \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p},\mathsf{q}\rangle, uv).
\end{aligned}$$

For $\alpha = \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p}_{i+1},\mathsf{p}_j\rangle, u_\alpha v_\alpha)$, this implies that

$$\mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p},\mathsf{q}\rangle, uv) = \max(\|\mathsf{p}-u\|, \alpha, \|\mathsf{q}-v\|),$$

where $\alpha \leq \max(\alpha,\delta)$ is equal for all tunnels in the family. Now, if we have that $\delta \leq \mathrm{prc}(\tau_{\min}(\mathsf{e}_i,\mathsf{e}_j,u,v))$, then we have $\mathrm{prc}(\tau_{\min}(\mathsf{e}_i,\mathsf{e}_j,u,v)) = \alpha \geq \delta$ and

$$\mathrm{prc}(\tau(\mathsf{f},\mathsf{g})) = \mathsf{d}_{\mathcal{F}}(X\langle \mathsf{p},\mathsf{q}\rangle, uv) = \max(\|\mathsf{p}-u\|, \alpha, \|\mathsf{q}-v\|) \leq \max(\alpha,\delta) = \alpha.$$

This proves (i). Otherwise, we have $\mathrm{prc}(\tau_{\min}(\mathsf{e}_i,\mathsf{e}_j,u,v)) < \delta$, which implies that $\alpha < \delta$, but then $\mathrm{prc}(\tau(\mathsf{f},\mathsf{g})) \leq \delta$, implying (ii).

If $i = j$, then the Fréchet distance is between the shortcut segment and a subsegment of e_i. But this distance is the maximum distance between the corresponding end points, by Observation 2.3. As the distance between end points of shortcuts and subcurves corresponding to tunnels of $\mathcal{T}_{\leq\delta}(\mathsf{e}_i,\mathsf{e}_j,u,v)$ is at most δ, and by Lemma 4.5, the claim follows. □

Lemma 4.7 below implies that the set of creation radii of all tunnels is approximated by the set of vertex-vertex and vertex-edge event radii. A similar lemma was shown in [24] to prove this property for the monotonicity event values. Therefore, the algorithm eliminates these types of events in the first stage, in addition to eliminating the vertex-vertex and vertex-edge events.

LEMMA 4.7. *Consider an edge* $\mathsf{e} = \mathsf{pq}$ *of a curve* $\boldsymbol{X}$ *and two vertices* u *and* v *of a curve* $\boldsymbol{Y}$. *We have that* $x/2 \leq \mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) \leq 2x$, *where* x *is in the set* $\{\mathsf{d}(u,\mathsf{e}), \mathsf{d}(v,\mathsf{e}), \|u-v\|\}$.

Proof. First, observe that $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) \geq \|u-v\|/2$, as it is the maximum distance of some point on e from both u and v. In particular, if $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) \leq 2\|u-v\|$, then we are done.

As such, it must be that $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) > 2\|u-v\|$. Assume that u is closer to e than v, and let u' be the closest point on e to u. By the triangle inequality, the distance of v from u' is in the range $\mathcal{I} = [\|u-u'\|, \|u-u'\| + \|u-v\|]$. Observe that $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) \geq \|u-u'\|$ and $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) \leq \max(\|u-u'\|, \|v-u'\|)$. Thus, $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v) \in \mathcal{I}$. Note that if $\|u-u'\| \leq \|u-v\|$, then we are done, as this implies that $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v)$ is in the range $[\|u-v\|, 2\|u-v\|]$. Otherwise, $\mathrm{r}_{\mathrm{crt}}(\mathsf{e},\mathsf{e},u,v)$ is in the range $[\|u-u'\|, 2\|u-u'\|]$. In either case, the claim follows.

The case that v is closer to e than u follows by symmetry. □

4.4. Analysis of the main algorithm. The following lemma can be obtained using similar arguments as in the analysis of the main algorithm in [24]. We provide a simplified proof for the case here, where we are only interested in a constant factor approximation.

LEMMA 4.8. *Given two c-packed polygonal curves $\boldsymbol{X}$ and $\boldsymbol{Y}$ in $\mathbb{R}^d$ with total complexity n, the first stage of the algorithm (see section* 3.3.1*) outputs one of the following:*

(A) *an $O(1)$-approximation to the shortcut Fréchet distance between $\boldsymbol{X}$ and $\boldsymbol{Y}$;*

(B) *an interval $\widehat{\mathcal{I}}$ and curves X and Y with the following properties:*

(i) *$\mathsf{d}_{\mathbb{S}}(X,Y)$ is contained in $\widehat{\mathcal{I}}$ and $\mathsf{d}_{\mathbb{S}}(X,Y)/3 \le \mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) \le 3\mathsf{d}_{\mathbb{S}}(X,Y)$,*

(ii) *$\widehat{\mathcal{I}}$ contains no vertex-edge, vertex-vertex, or monotonicity event values and no tunnel creation radii (as defined in section* 4.3*) of X and Y.*

The running time is $O(c^2 n \log^3 n)$.

Proof. We first prove the correctness of the algorithm as stated in the claim. The set U approximates the vertex-vertex distances of the vertices of $\boldsymbol{X}$ and $\boldsymbol{Y}$ up to a factor of two. Therefore, the interval $\mathcal{I} = [\alpha, \beta]$, which we obtain from the first binary search, contains no vertex-vertex distance of $\boldsymbol{X}$ that is more than a factor of two away from its boundary. This implies that the simplification $X = \mathrm{simpl}(\boldsymbol{X}, \mu)$ results in the same curve for any $\mu \in [3\alpha, \beta/3]$. An analogous statement holds for $\boldsymbol{Y}$. Unless a constant factor approximation is found either in the interval $[\alpha, 10\alpha]$ or in the interval $[\beta/10, \beta]$, the algorithm continues the search using the procedure **decider** and the curves simplified with $\mu = 3\alpha$.

It is now sufficient to search for a constant factor approximation to $\mathsf{d}_{\mathbb{S}}(X,Y)$ in the interval $\mathcal{I}' = [3\alpha, \beta/3]$, since this will approximate the desired Fréchet distance by a constant factor. Indeed, by the result of the initial searches, we have that $3\mu \le 10\alpha \le \mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y})$. Lemma 2.9 implies that $\mathsf{d}_{\mathbb{S}}(X,Y) \le \mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) + 2\mu \le 3\mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y})$. On the other hand, the same lemma implies that $\mathsf{d}_{\mathbb{S}}(X,Y) \ge \mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) - 2\mu \ge \mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y})/3$. This implies that $\mathsf{d}_{\mathbb{S}}(X,Y) \in \mathcal{I}' = [3\alpha, \beta/3]$, since $\mathsf{d}_{\mathbb{S}}(\boldsymbol{X},\boldsymbol{Y}) \in [10\alpha, \beta/10]$. Note that this also proves the correctness of (i), since the returned interval is contained in $\mathcal{I}'$.

Observe that the set of vertex-vertex distances of X and Y is contained in the set of vertex-vertex distances of $\boldsymbol{X}$ and $\boldsymbol{Y}$. Clearly, $\mathcal{I}'$ cannot contain any vertex-vertex distances of X and Y. The algorithm therefore extracts the remaining vertex-edge events U' from the free space diagram and performs a binary search on them. We obtain the atomic interval $\mathcal{I}'' = [\alpha'', \beta'']$, which contains no vertex-edge events of X and Y. Note that by (2.1) and Lemma 4.5, the monotonicity event values, as described in section 2.1, coincide with the values of δ where a tunnel within a column of the parametric space becomes feasible, that is, with the quantity $\mathrm{r}_{\mathrm{crt}}(\mathsf{e}, \mathsf{e}, u, v)$. By Lemma 4.7, these event values would have to lie within a factor two of the boundaries of the interval $\mathcal{I}''$. Therefore, we again search the margins of this interval, so that we find the desired approximation or, alternatively, it must be in the interval $\mathcal{I}''' = [10\alpha'', \beta''/10]$, which now contains no vertex-vertex, vertex-edge, monotonicity, or tunnel creation events of X and Y. Since $\mathcal{I}'''$ is the interval that the algorithm returns, unless it finds a constant factor approximation to the desired Fréchet distance, the above argumentation implies (i) and (ii).

As for the running time, computing the set U using well-separated pairs decomposition can be done in $O(n \log n)$ time; see [24]. Computing the set U' takes time in $O(n \log n + c^2 n)$ by Data Structure 2.12 with $\mu = \beta/3$ and $\delta = \beta$. The algorithm invokes the decision procedure $O(\log n)$ times, and this dominates the overall running time; see Lemma 4.2. ∎

LEMMA 4.9. *Given two c-packed polygonal curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *in* $\mathbb{R}^d$ *of total complexity* n*, one can compute a constant factor approximation to* $\mathsf{d}_{\mathbb{S}}(\boldsymbol{X}, \boldsymbol{Y})$*. The running time is* $O(c^2 n \log^3 n)$*.*

Proof. First, the algorithm performs the preliminary computations as described in section 3.3.1. By Lemma 4.8, we either find a constant factor approximation or we obtain an interval $[\alpha, \beta]$ and simplified curves X and Y. Furthermore, the interval $[\alpha, \beta]$ does not contain any vertex-vertex, vertex-edge, monotonicity, or tunnel creation events of X and Y. Let P be the canonical gates that are feasible in the β-free space of X and Y. We have that $m = |\mathsf{P}| = O(n)$ and we can compute them using Data Structure 2.12 in $O(n \log n + c^2 n)$ time for $\varepsilon = 1/3$. Thus, the running time up to this stage is bounded by $O(c^2 n \log^3 n)$ by Lemma 4.8.

Now, we invoke the second stage of the algorithm described in section 3.3.2 on the matrix of implicit tunnel prices defined by P and return the output as our solution.

Consider a monotone path in the parametric space that corresponds to the optimal solution. If the price of this path is determined by either a vertex-vertex, a vertex-edge, or a monotonicity event, then we have found an approximation to the shortcut Fréchet distance already in the first stage of the search algorithm. If it is dominated by a tunnel price and this tunnel has both end points in the same column of the free space, then by Observation 2.3 it is a creation radius. By Lemma 4.5 this is equivalent to the minimum radius of the corresponding tunnel family. By Observation 2.7 the minimum radius corresponds to either a vertex-edge event or a monotonicity event. Thus, it lies outside the interval $[\alpha, \beta]$, since by Lemma 4.8 these critical values were eliminated in the first stage. Otherwise, this critical tunnel has to be between two columns. Let δ be the price of this tunnel (which is also the price of the whole solution).

Consider what happens to this path if we slightly decrease δ. Since δ is optimal, then either the critical tunnel ceases to be feasible or its price is not affordable anymore.

If the critical tunnel is no longer feasible, then one of its end points is also an end point of the free space interval it lies on. Consider the modified path in the free space, which uses the new end point of the free space interval. If the free space interval is empty, then this corresponds to a vertex edge event, and this is not possible inside the interval $[\alpha, \beta]$. The other possibility is that the path is no longer monotone. However, this corresponds to a monotonicity event, which again we already handled because of Lemma 4.8.

If the tunnel is still feasible, then it must be that the end points of this tunnel are contained in the interior of the free space interval and not on its boundary. Now Lemma 4.6(i) implies that the price of this tunnel is equal to the price of the canonical tunnel. As such, the price of the optimal solution is being approximated correctly in this case.

Observe that in the second stage we are searching over all tunnel events that lie in the remaining search interval (whether they are relevant in our case or not). Hence, the search would find the correct critical value, as it is one of the values considered in the search.

The running time of the second stage is bounded by the following:

(A) $O(n \log n \log\log n)$ time to compute the needed entries in the matrix using Data Structure 2.11;
(B) $O((c^2 n \log^2 n) \log n)$ time for the $O(\log n)$ calls to **Decider**;
(C) $O(n)$ for other computations.

Therefore, the overall running time of the algorithm is $O(c^2 n \log^3 n)$. ∎

4.5. Result. The following theorem states the main result for approximating the shortcut Fréchet distance.

THEOREM 4.10. *Given two c-packed polygonal curves* $\boldsymbol{X}$ *and* $\boldsymbol{Y}$ *in* $\mathbb{R}^d$ *with total complexity* n *and a parameter* $\varepsilon > 0$*, the main algorithm of* section 3 *computes a* $(3+\varepsilon)$*-approximation to the shortcut Fréchet distance between* X *and* Y*. The running time of the algorithm is* $O(c^2 n \log^2 n(\log n + \varepsilon^{-2d} \log(1/\varepsilon)))$ *time. The algorithm also outputs the shortcut curve of* $\boldsymbol{Y}$ *and the reparametrizations that realize the respective shortcut Fréchet distance.*

Proof. The result follows from Lemma 4.9. This yields an interval $\mathcal{I}$ that contains the value of the optimal solution. We can turn any constant factor approximation into a $(3+\varepsilon)$-approximation using **Decider** with $\varepsilon' = \varepsilon/3$ by invoking it over a constant number of subintervals of the form $[\alpha, \beta]$, where $\beta = (3+\varepsilon)\alpha$. These intervals are required to cover $\mathcal{I}$, and as such, **Decider** would return the desired approximation for one of them. (The running time of each call to **Decider** is stated in Lemma 4.2.)

It is easy to modify the algorithm such that it also outputs the shortcut curve and the reparametrizations realizing the approximate Fréchet distance; see Observation 4.3. ∎

Remark 4.11. One can extend the algorithm of Theorem 4.10 so that it approximates the Fréchet distance where only k shortcuts are allowed. The basic algorithm is similar, except that we keep track for the points of R how many shortcuts were used in computing them. The resulting algorithm has running time $O(c^2 kn \log^3 n)$ (for ε a constant). This version of the algorithm is described in the Driemel's thesis [22].

5. Data structures for Fréchet-distance queries. Given a polygonal curve Z in $\mathbb{R}^d$, we build a data structure that supports queries for the Fréchet distance of subcurves of Z to query segments pq. We describe the data structure in three stages. After establishing some basic facts in section 5.1, we first describe a data structure that achieves a constant factor approximation in section 5.2. We proceed by describing a data structure that answers queries for the Fréchet distance of the entire curve to a query segment up to an approximation factor of $(1+\varepsilon)$ in section 5.3. Finally, we describe how to combine these two results to obtain the final data structure for segment queries in section 5.4.

5.1. Useful lemmas for curves and segments.

DEFINITION 5.1. *For a curve* Z*, the segment connecting its end points is its* spine*, denoted by* $\mathrm{spine}(Z)$*.*

The following is a sequence of technical lemmas that we need later on. These lemmas affirm the following:

(A) The spine of a curve is, up to a factor of two, the closest segment to this curve with respect to the Fréchet distance; see Lemma 5.2.
(B) The Fréchet distance between a curve and its spine is monotone, up to a factor of two, with respect to subcurves; see Lemma 5.3.
(C) Shortcutting a curve cannot increase the Fréchet distance of the curve to a line segment; see Lemma 5.4 or [15].

LEMMA 5.2. *Let* pq *be a segment and* Z *be a curve. Then,* (i) $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \geq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, \mathrm{spine}(Z))$*, and* (ii) $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \geq \mathsf{d}_{\mathcal{F}}(\mathrm{spine}(Z), Z)/2$.

Proof. Let r and u be the end points of Z; that is, $\mathrm{spine}(Z) = \mathsf{ru}$.

(i) Since in any matching of pq with Z it must be that p is matched to r and q is matched to u, it follows that $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \geq \max(\|\mathsf{p} - u\|, \|\mathsf{q} - v\|) = \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, \mathsf{ru}) = \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, \mathrm{spine}(Z))$, by Observation 2.3.

(ii) By (i) and the triangle inequality, we have that

$$\mathsf{d}_{\mathcal{F}}(\text{spine}(Z), Z) \leq \mathsf{d}_{\mathcal{F}}(\text{spine}(Z), \mathsf{pq}) + \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \leq 2\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z),$$

which implies the claim. ∎

LEMMA 5.3. *Given two curves Z and $\widehat{Z}$ such that $\widehat{Z}$ is a subcurve of Z, then we have that* $\mathsf{d}_{\mathcal{F}}(\text{spine}(\widehat{Z}), \widehat{Z}) \leq 2\mathsf{d}_{\mathcal{F}}(\text{spine}(Z), Z)$.

Proof. Consider the matching that realizes the Fréchet distance between Z and spine(Z). It has to match the end points of $\widehat{Z}$ to points q and r on spine(Z). We have that $\mathsf{d}_{\mathcal{F}}(\widehat{Z}, \mathsf{qr}) \leq \mathsf{d}_{\mathcal{F}}(Z, \text{spine}(Z))$. By Lemma 5.2(i), we have $\mathsf{d}_{\mathcal{F}}(\text{spine}(\widehat{Z}), \mathsf{qr}) \leq \mathsf{d}_{\mathcal{F}}(\widehat{Z}, \mathsf{qr}) \leq \mathsf{d}_{\mathcal{F}}(Z, \text{spine}(Z))$. Now, by the triangle inequality, we have that

$$\mathsf{d}_{\mathcal{F}}(\widehat{Z}, \text{spine}(\widehat{Z})) \leq \mathsf{d}_{\mathcal{F}}(\widehat{Z}, \mathsf{qr}) + \mathsf{d}_{\mathcal{F}}(\mathsf{qr}, \text{spine}(\widehat{Z})) \leq 2\mathsf{d}_{\mathcal{F}}(Z, \text{spine}(Z)). \qquad \blacksquare$$

LEMMA 5.4. *Let $Z = u_1 u_2 \dots u_n$ be a polygonal curve, let pq be a segment, and let $i < j$ be any two indices. Then, for $Z' = Z\langle u_1, u_i\rangle \oplus u_i u_j \oplus Z\langle u_j, u_n\rangle$, we have* $\mathsf{d}_{\mathcal{F}}(Z', \mathsf{pq}) \leq \mathsf{d}_{\mathcal{F}}(Z, \mathsf{pq})$.

Proof. Consider the matching realizing $\mathsf{d}_{\mathcal{F}}(Z, \mathsf{pq})$, and break it into three portions:

- the portion matching $Z\langle u_1, u_i\rangle$ with a "prefix" $\mathsf{pp}' \subseteq \mathsf{pq}$,
- the portion matching $Z\langle u_i, u_j\rangle$ with a subsegment $\mathsf{p}'\mathsf{q}' \subseteq \mathsf{pq}$, and
- the portion matching $Z\langle u_j, u_n\rangle$ with a "suffix" $\mathsf{q}'\mathsf{q} \subseteq \mathsf{pq}$.

Now, by Lemma 5.2(i), we have that

$$\begin{aligned}
\mathsf{d}_{\mathcal{F}}(Z, \mathsf{pq}) &= \max(\mathsf{d}_{\mathcal{F}}(Z\langle u_1, u_i\rangle, \mathsf{pp}'),\ \mathsf{d}_{\mathcal{F}}(Z\langle u_i, u_j\rangle, \mathsf{p}'\mathsf{q}'),\ \mathsf{d}_{\mathcal{F}}(Z\langle u_j, u_n\rangle, \mathsf{q}'\mathsf{q})) \\
&\geq \max(\mathsf{d}_{\mathcal{F}}(Z\langle u_1, u_i\rangle, \mathsf{pp}'),\ \mathsf{d}_{\mathcal{F}}(u_i u_j, \mathsf{p}'\mathsf{q}'),\ \mathsf{d}_{\mathcal{F}}(Z\langle u_j, u_n\rangle, \mathsf{q}'\mathsf{q})) \\
&\geq \mathsf{d}_{\mathcal{F}}(Z', \mathsf{pq}). \qquad \blacksquare
\end{aligned}$$

5.2. Stage 1: Achieving a constant factor approximation. In this section we describe a data structure that preprocesses a curve Z to answer queries for the Fréchet distance of a subcurve of Z to a query segment up to a constant approximation factor. This data structure will be the basis for later extensions.

A query is specified by points u, v, p, and q. Here u and v are points on Z (and we are also given the edges of Z containing these two points), and the points p and q define the query segment. Our goal is to approximate $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.

5.2.1. The data structure.

Preprocessing. Build a balanced binary tree T on the *edges* of Z. Every internal node ν of T corresponds to a subcurve of Z, denoted by $\text{cr}(\nu)$. Let $\text{seg}(\nu)$ denote the spine of $\text{cr}(\nu)$ (Definition 5.1). For every node, we precompute its Fréchet distance of the curve $\text{cr}(\nu)$ to the segment $\text{seg}(\nu)$. Let d_ν denote this distance.

Answering a query. For the time being, assume that u and v are vertices of Z. In this case, one can compute, in $O(\log n)$ time, $k = O(\log n)$ nodes $\nu_1, \dots, \nu_k$ of T, such that $Z\langle u, v\rangle = \text{cr}(\nu_1) \oplus \text{cr}(\nu_2) \oplus \cdots \oplus \text{cr}(\nu_k)$. We compute the polygonal curve $Y = \text{seg}(\nu_1) \oplus \cdots \oplus \text{seg}(\nu_k)$ and compute its Fréchet distance from the segment pq. We denote this distance by $d = \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Y)$. We return

$$\Delta = d + \max_{i=1}^{k} \mathrm{d}_{\nu_i}$$

as the approximate distance between pq and the subcurve $Z\langle u, v\rangle$.

5.2.2. Analysis.

LEMMA 5.5. *Given a polygonal curve Z with n edges, one can preprocess it in $O(n \log^2 n)$ time such that for any pair u, v of vertices of Z and a segment pq, one can compute, in $O(\log n \log\log n)$ time, a 3-approximation to $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.*

Proof. The construction of the data structure and how to answer a query are described above. For the preprocessing time, observe that computing the Fréchet distance of a segment to a polygonal curve with k segments takes $O(k \log k)$ time [6]. Hence, the distance computations in each level of the tree T take $O(n \log n)$ time and $O(n \log^2 n)$ time overall.

As for the query time, computing Y takes $O(\log n)$ time, and computing its Fréchet distance from pq takes $O(\log n \log\log n)$ time [6].

Finally, observe that the returned distance Δ is a realizable Fréchet distance, as we can take the matching between pq and Y and chain it with the matching of every edge of Y with its corresponding subcurve of Z. Clearly, the resulting matching has width at most Δ.

Let t be the index realizing $\max_{i=1}^{k} \mathrm{d}_{\nu_i}$. Then, by a repeated application of Lemma 5.4, we have that $d = \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Y) \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$. Thus,

$$\begin{aligned}\Delta = d + \max_{i=1}^{k} \mathrm{d}_{\nu_i} &= \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Y) + \mathrm{d}_{\nu_t} \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) + \mathsf{d}_{\mathcal{F}}(\mathrm{seg}(v_t), \mathrm{cr}(v_t)) \\ &\leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) + 2 \min_{\mathsf{p'q'} \subseteq \mathsf{pq}} \mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathrm{cr}(v_t)) \leq 3\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z).\end{aligned}$$

To see the last step, consider the matching realizing $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$, and consider the subsegment $\mathsf{p'q'}$ of pq that is being matched to $\mathrm{cr}(v_t) \subseteq Z$. Clearly, $\mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathrm{cr}(v_t)) \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$. ∎

THEOREM 5.6. *Given a polygonal curve Z with n edges, one can preprocess it in $O(n \log^2 n)$ time and using $O(n)$ space such that, given a query specified by*

(i) *a pair of points u and v on the curve Z,*
(ii) *the edges containing these two points, and*
(iii) *a pair of points p and q,*

one can compute, in $O(\log n \log\log n)$ time, a 3-approximation to $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.

Proof. This follows by a relatively minor modification of the above algorithm and analysis. Indeed, given u and v (and the edges containing them), the data structure computes the two vertices u', v' that are end points of these edges that lie between u and v on the curve. The data structure then concatenates the segments uu' and $v'v$ to the approximation Y (here Y is computed for the vertices u' and v'). The remaining details are as described above. ∎

5.3. Stage 2: A segment query to the entire curve. In this section we describe a data structure that preprocesses a curve to answer queries for the Fréchet distance of the entire curve to a query segment up to an approximation factor of $(1 + \varepsilon)$. We will use this data structure as a component in our later extensions.

5.3.1. The data structure. We need the following relatively easy construction of an exponential grid. Figure 5.1 illustrates the idea. The details can be found in [22].

LEMMA 5.7 (see [22]). *Given a point $u \in \mathbb{R}^d$, a parameter $0 < \varepsilon \leq 1$, and an interval $[\alpha, \beta] \subseteq \mathbb{R}$ one can compute in $O(\varepsilon^{-d} \log(\beta/\alpha))$ time and space an exponential grid of points $G(u)$ such that for any point $\mathsf{p} \in \mathbb{R}^d$ with $\|\mathsf{p} - u\| \in [\alpha, \beta]$, one can compute in constant time a grid point $\mathsf{p'} \in G(u)$ with $\|\mathsf{p} - \mathsf{p'}\| \leq (\varepsilon/2)\|\mathsf{p} - u\|$.*

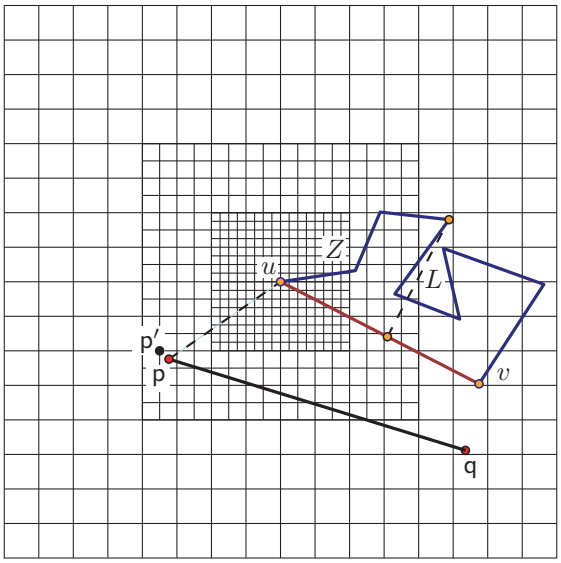


FIG. 5.1. *We build an exponential grid around each end point of the curve such that for any point* p*, which has distance to the end point in the range* $[\varepsilon L/4, L/\varepsilon]$*, there exists a grid point* p' *which is relatively close by.*

Preprocessing. We are given a polygonal curve Z in $\mathbb{R}^d$ with n segments, and we would like to preprocess it for $(1+\varepsilon)$-approximate Fréchet distance queries against a query segment. To this end, let $L = \mathsf{d}_{\mathcal{F}}(uv, Z)$, where uv is the spine of Z. We construct an exponential grid $G(u)$ of points around u with the range $[\alpha, \beta] = [\varepsilon L/4, L/\varepsilon]$ as described in Lemma 5.7 and illustrated in Figure 5.1. We construct the same grid $G(v)$ around the vertex v.

Now, for every pair of points $(\mathsf{p}', \mathsf{q}') \in G(u) \times G(v)$ we compute the Fréchet distance $D[\mathsf{p}', \mathsf{q}'] = \mathsf{d}_{\mathcal{F}}(\mathsf{p}'\mathsf{q}', Z)$ and store it. Thus, we take $O(\chi^2 n \log n)$ time to build a data structure that requires $O(\chi^2)$ space, where $\chi = \varepsilon^{-d}\log(1/\varepsilon)$.

Answering a query. Given a query segment pq, we compute the distance

$$r = \max(\|\mathsf{p} - u\|, \|\mathsf{q} - v\|).$$

If $r \leq \varepsilon L/4$, then we return $L - r$ as the approximation to the distance $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$. If $r \geq L/\varepsilon$, then we return r as the approximation. Otherwise, let p' (resp., q') be the nearest neighbor to p in $G(u)$ (resp., $G(v)$). We return the distance

$$\Delta = D[\mathsf{p}', \mathsf{q}'] - \max(\|\mathsf{p} - \mathsf{p}'\|, \|\mathsf{q} - \mathsf{q}'\|)$$

as the approximation.

5.3.2. Analysis.

LEMMA 5.8. *Given a polygonal curve* Z *with* n *vertices in* $\mathbb{R}^d$*, one can build a data structure, in* $O(\chi^2 n \log n))$ *time, that uses* $O(\chi^2)$ *space, such that given a query segment* pq *one can* $(1+\varepsilon)$*-approximate* $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$ *in* $O(1)$ *time, where* $\chi = \varepsilon^{-d}\log(1/\varepsilon)$.

Proof. The data structure is described above. Given pq we compute the distance of the end points of this segment from the end points of Z. If they are too close, or if one of them is too far away, then we are done since in this case the Fréchet distance

is dominated either by these distances or by the precomputed value L. Otherwise, we find the two cells in the exponential grid that contain p and q (that is, the indices of the grid points that are close to them) as described above. Using the indices of the grid points, we can directly look up the approximation of the Fréchet distance in constant time.

Now, we argue about the quality of the approximation using the notation which is also used above. There are three cases: either (i) $r \leq \varepsilon L/4$, or (ii) $L \leq \varepsilon r$, or (iii) $\varepsilon L/4 \leq r \leq L/\varepsilon$. Let Δ be the returned value. We claim that in all three cases, it holds that

$$\Delta \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \leq (1+\varepsilon)\Delta. \tag{5.1}$$

First note that by the triangle inequality,

$$L - r \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \leq L + r. \tag{5.2}$$

Now, in case (i) above, L dominates the distance value and we return $\Delta = L - r$. Thus, (5.1) follows from (5.2).

In case (ii), r dominates the distance value and we return $\Delta = r$. Since r is at most $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$, again (5.1) follows from (5.2).

In case (iii), the precomputed Fréchet distance of $\mathsf{p'q'}$ to Z dominates the distance. Recall that we return $\Delta = \mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, Z) - \mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq})$ in this case. Again, by the triangle inequality, it holds that

$$\mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, Z) - \mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}) \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \leq \mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, Z) + \mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}). \tag{5.3}$$

Since r is at least $\varepsilon L/4$ and by Observation 2.3, Lemma 5.7 implies that

$$\mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}) \leq \max(\|\mathsf{p} - \mathsf{p'}\|, \|\mathsf{q} - \mathsf{q'}\|) \leq (\varepsilon/2)r;$$

thus, since also r is at most $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z)$ it follows by (5.3) that

$$\Delta \leq \mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z) \leq \Delta + 2\mathsf{d}_{\mathcal{F}}(\mathsf{p'q'}, \mathsf{pq}) \leq \Delta + \varepsilon r \leq (1+\varepsilon)\Delta.$$

This implies the claim. ∎

5.4. Stage 3: A segment query to a subcurve. In this section we describe a data structure that preprocesses a curve Z to answer queries for the Fréchet distance of a subcurve of Z to a query segment up to an approximation factor of $(1+\varepsilon)$. For this we combine the data structures developed in the previous sections.

As in section 5.2, a query is defined by two points u and v on Z and a segment with end points p and q. The goal is now a $(1+\varepsilon)$-approximation to $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$.

5.4.1. The data structure.

Preprocessing. Let Z be a given polygonal curve with n vertices. We build the data structure of Theorem 5.6. Next, for each node of the resulting tree T, we build for its subcurve the data structure of Lemma 5.8 using $\varepsilon' = \varepsilon/3$.

Answering a query. Using the data structure of Theorem 5.6 we first compute a 3-approximation r to $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$; that is, $\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle) \leq r \leq 3\mathsf{d}_{\mathcal{F}}(\mathsf{pq}, Z\langle u, v\rangle)$. This query also results in a decomposition of $Z\langle u, v\rangle$ into $m = O(\log n)$ subcurves. Let $u = v_0, v_1, \ldots, v_{m-1}, v_m = v$ be the vertices of these subcurves, where v_0v_1 and $v_{m-1}v_m$ are subsegments of Z.

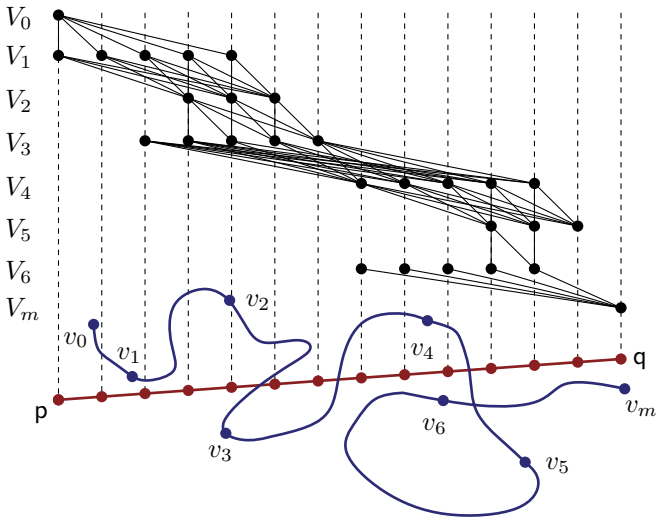


FIG. 5.2. *Schematic illustration of the graph* G *on the vertex set* $\bigcup V_i$.

We want to find points on pq that can be matched to $v_1, \ldots, v_{m-1}$ under a $(1+\varepsilon)$-approximate Fréchet matching. To this end, we uniformly partition the segment pq into segments of length at most $\varepsilon r/c_1$, where c_1 is a sufficiently large constant which we define later. Let Π be the set of vertices of this implicit partition. For each vertex v_i, for $i = 1, \ldots, m-1$, we compute its nearest point on pq, and let $V_i \subseteq \Pi$ be the set of all vertices in Π that are in distance at most $2r$ from v_i. The set V_i is the set of candidate points to match v_i in the matching that realizes the Fréchet distance.

Now, we build a graph G where $\bigcup_i V_i$ is the multiset of vertices. Two points $x \in V_i$ and $y \in V_{i+1}$ are connected by a direct edge in this graph if and only if y is after x in the oriented segment pq. See Figure 5.2 for a schematic illustration. The price of such an edge $x \to y$ is a $(1+\varepsilon/4)$-approximation to the Fréchet distance between $Z\langle v_i, v_{i+1}\rangle$ and xy. The portion $Z\langle v_i, v_{i+1}\rangle$ of the curve corresponds to a node in T, and this node has an associated data structure that can answer such queries in constant time (see Lemma 5.8). For any point $x \in V_1$, we directly compute the Fréchet distance v_0v_1 with px. Similarly, we compute, for each $y \in V_{m-1}$, the Fréchet distance of the segment $v_{m-1}v_m$ to the segment yq. We add the corresponding edges to G together with the vertices p and q.

Using a variant of Dijkstra's algorithm for bottleneck shortest paths, we now compute a path in this graph which minimizes the maximum cost of any single edge visited by the path, connecting p with q. The cost of this path is returned as the approximation to the Fréchet distance between $Z\langle u, v\rangle$ and pq. Intuitively, this path corresponds to the cheapest matching of $Z\langle u, v\rangle$ (broken into subcurves by the vertices $v_0, \ldots, v_m$) with $V_0 \times V_1 \times \cdots V_{m-1} \times V_m$, where $V_0 = \{\mathsf{p}\}$, $V_m = \{\mathsf{q}\}$, and every subcurve $Z\langle v_i, v_{i+1}\rangle$ is matched with two points in the corresponding sets V_i and V_{i+1}.

5.4.2. Analysis.

Query time. Computing the set of vertices $v_0, v_1, \ldots, v_m$ takes $O(m) = O(\log n)$ time. The graph G has $N = O(m/\varepsilon)$ vertices and they can be computed in $O(m/\varepsilon)$ time. In particular, the number of vertices in V_i is bounded by $O(1/\varepsilon)$, since they are spread apart on a line segment by $\varepsilon r/c_1$ and contained inside a ball of radius $2r$. Thus, the graph has $O((1/\varepsilon)^{-2})$ edges connecting V_i with V_{i+1} and $M = O(m/\varepsilon^2)$

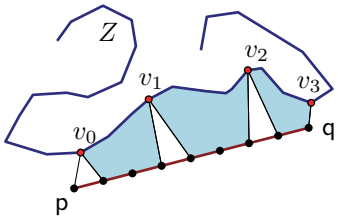


FIG. 5.3. *Illustration of the error introduced by snapping.*

edges in total. The cost of each edge can be computed in constant time; see Lemma 5.8. Computing the cheapest path between p and q in G can be done in $O(N \log N + M) = O((m/\varepsilon)\log(m/\varepsilon) + m/\varepsilon^2)$ time, using Dijkstra's algorithm for bottleneck shortest paths. Overall, the query time is

$$O(m + (m/\varepsilon)\log(m/\varepsilon) + m/\varepsilon^2) = O(\varepsilon^{-2}\log n \log\log n).$$

Quality of approximation. Consider the matching that realizes the Fréchet distance between the query segment pq and the subcurve $Z\langle u, v\rangle$, and break it at the vertices of $v_0, \ldots, v_m$. Now, snap the matching such that the end points of $Z\langle v_i, v_{i+1}\rangle$ are mapped to their closest vertices in V_i and V_{i+1}, respectively, for all i. This introduces an error of at most $\varepsilon r/c_1 \leq (\varepsilon/3)\mathsf{d}_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq})$ if we choose $c_1 \geq 9$; see Figure 5.3 for an illustration. We get another factor of $(1+\varepsilon') = (1+\varepsilon/3)$ error since we are approximating the price of these portions using Lemma 5.8. Therefore, the approximation has price at most

$$(1+\varepsilon/3)(1+\varepsilon/3)\cdot \mathsf{d}_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq}) \leq (1+\varepsilon)\cdot \mathsf{d}_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq}).$$

Preprocessing time and space. Building the data structure described in Theorem 5.6 takes $O(n\log^2 n)$ time. For each node v of this tree, building the data structure of Lemma 5.8 takes $O(\chi^2 \mathrm{l}(v)\log \mathrm{l}(v))$ time per node, where $\mathrm{l}(v)$ is the number of vertices of the curve stored in the subtree of v. As such, overall, the preprocessing time is $O(\chi^2 n\log^2 n)$. For each node, this data structure requires $O(\chi^2)$ space and thus the overall space usage is $O(\chi^2 n)$, where $\chi = \varepsilon^{-d}\log(1/\varepsilon)$.

Putting the above together, we get the following result.

THEOREM 5.9. *Given a polygonal curve Z with n vertices in $\mathbb{R}^d$, one can build a data structure, in $O(\chi^2 n\log^2 n)$ time, that uses $O(n\chi^2)$ space such that for a query segment pq, and any two points u and v on the curve (and the segments of the curve that contain them), one can $(1+\varepsilon)$-approximate the distance $\mathsf{d}_{\mathcal{F}}(Z\langle u, v\rangle, \mathsf{pq})$ in $O(\varepsilon^{-2}\log n\log\log n)$ time, and $\chi = \varepsilon^{-d}\log(1/\varepsilon)$.*

We emphasize that the result of Theorem 5.9 assumed nothing on the input curve Z. In particular, the curve Z is not necessarily c-packed.

6. Universal vertex permutation and its applications. We would like to extend the data structure described in section 5.2 to support queries with curves of more than one segment. For this, we first introduce a new method to represent a polygonal curve in a way such that we can extract a simplification with a small number of segments quickly. We describe this method in section 6.1 and we describe the extension of the data structure in section 6.2.

6.1. Universal vertex permutation. We use the data structure described in section 5.4 to preprocess Z such that, given a number of vertices $k \in \mathbb{N}$, we can quickly return a simplification of Z which has

(i) $2k-1$ vertices of the original curve and
(ii) minimal Fréchet distance to Z, up to a constant factor, compared to any simplification of Z with only k vertices.

The idea is to compute a permutation of the vertices such that the curve formed by the first k vertices in this permutation is a good approximation to the optimal simplification of a curve using (roughly) k vertices.

DEFINITION 6.1. *Let Z be a polygonal curve with vertices $V(Z)$. Let $V \subseteq V(Z)$ be a subset of the vertices that contains the end points of Z. We call the polygonal curve obtained by connecting the vertices in V in their order along Z a* spine curve *of Z and we denote it with Z_V. Additionally we may call Z_V a* **k-**spine curve *of Z if it has k vertices.*

DEFINITION 6.2. *Given a polygonal curve Z and a permutation $\Phi = \langle v_1, \ldots, v_n \rangle$ of the vertices of Z, where v_1 and v_2 are the end points of Z, let V_i be the subset $\{v_j \mid 1 \leq j \leq i\}$ of the vertices for any $2 \leq i \leq n$. We call Φ a* universal vertex permutation *if it holds that*

(i) $c_1 \mathsf{d}_{\mathcal{F}}(Z_{V_i}, Z) \geq \mathsf{d}_{\mathcal{F}}(Z_{V_{i+1}}, Z)$ *for any $2 \leq i < n$, and*
(ii) $\mathsf{d}_{\mathcal{F}}(Z_{V_i}, Z) \leq c_2 \mathsf{d}_{\mathcal{F}}(Y, Z)$ *for any polygonal curve Y with $\lceil i/c_3 \rceil$ vertices,*

where c_1, c_2, and c_3 are constants larger than one which do not depend on n.

6.1.1. Construction of the permutation. We compute a universal vertex permutation of Z. The idea of the algorithm is to estimate for each vertex the error introduced by removing it and repeatedly remove the vertex with the lowest error in a greedy fashion.

Specifically, for each vertex v that is not an end point of Z, let v^- be its predecessor on Z and let v^+ be its successor on Z. Let ϕ_v be a (11/10)-approximation of $\mathsf{d}_{\mathcal{F}}(Z\langle v^-, v^+ \rangle, v^- v^+)$. Insert the vertex v with weight ϕ_v into a min-heap $\mathcal{H}$. Repeat this for all the internal vertices of Z.

At each step, the algorithm extracts the vertex v from the heap $\mathcal{H}$ having minimum weight. Let $u = v^-(Z_{\mathcal{H}})$ and $w = v^+(Z_{\mathcal{H}})$ be the predecessor and successor of v in the curve $Z_{\mathcal{H}}$, respectively, where $\mathcal{H}$ denotes the set of vertices currently in the heap with the addition of the two end points of Z.

The algorithm removes v from $\mathcal{H}$ and updates the weight of u and w in $\mathcal{H}$. (If the vertex being updated is an end point of Z its weight is $+\infty$ and its weight is not being updated.) Updating the weight of a vertex u is done by computing its predecessor and successor vertices in the current curve $Z_{\mathcal{H}}$ (i.e., $u^- = u^-(Z_{\mathcal{H}})$ and $u^+ = u^+(Z_{\mathcal{H}})$) and approximating the Fréchet distance of the subcurve of (the *original* curve) Z between these two vertices and the segment $u^- u^+$. Formally, the updated weight of u is ϕ_u, which is a (11/10)-approximation to

$$\mathsf{d}_{\mathcal{F}}(Z\langle u^-, u^+ \rangle, u^- u^+).$$

The updated weight of w is computed in a similar fashion.

The algorithm stops when $\mathcal{H}$ is empty. Reversing the order of the handled vertices results in a permutation $\langle v_1, \ldots, v_n \rangle$, where v_1 and v_2 are the two end points of Z.

Implementation details. Using Theorem 5.9, the initialization takes $O(n \log^2 n)$ time overall, using $\varepsilon = 1/10$. In addition, the algorithm keeps the current set of vertices of $\mathcal{H}$ in a doubly linked list in the order in which the vertices appear along the original curve Z. In each iteration, the algorithm performs one extract-min from the min-heap $\mathcal{H}$ and calls the data structure of Theorem 5.9 twice to update the weight of the two neighbors of the extracted vertex. As such, overall, the running time of this algorithm is $O(n \log^2 n)$.

Extracting a spine curve quickly. Given a parameter K, we would like to be able to quickly compute the spine curve Z_{V_K}, where $V_K = \{v_1, \ldots, v_K\}$. To this end, we compute for $i = 1, \ldots \lfloor \log_2 n \rfloor$ the spine curve $Z_{V_{2^i}}$ by removing the unused vertices from $Z_{V_{2^{i+1}}}$. Naturally, we also store the original curve Z. Clearly, one can store these $O(\log n)$ curves in $O(n)$ space and compute them in linear time. Now, given K, one can find the first curve in this collection that has more vertices than K, copy it, and remove from it all the unused vertices. Clearly, this query can be answered in $O(K)$ time.

6.1.2. Analysis.

LEMMA 6.3. *Let $\langle v_1, \ldots, v_n \rangle$ be the permutation computed above. Consider a value k, and let $V_k = \{u_1, \ldots, u_k\}$ be an ordering of the vertices of $v_1, \ldots, v_k$ by their order along Z. Then, the following holds:* $\mathsf{d}_{\mathcal{F}}(Z, Z_{V_k}) \leq \max_{1 \leq i \leq k-1} \mathsf{d}_{\mathcal{F}}(Z\langle u_i, u_{i+1} \rangle, u_i u_{i+1})$.

Proof. This is immediate as one can combine for $i = 1, \ldots, k-1$ the matchings realizing $\mathsf{d}_{\mathcal{F}}(Z\langle u_i, u_{i+1} \rangle, v_i u_{i+1})$ to obtain matchings of Z_{V_k} and Z and such that the Fréchet distance is the maximum used in any of these matchings. ∎

Let $v_1, \ldots, v_n$ be the permutation of the vertices of Z as computed in the preprocessing stage, and let $\phi(v_i)$ denote weight of vertex v_i at the time of its extraction. We have the following three lemmas to prove that the computed permutation is universal.

LEMMA 6.4. *For any $1 \leq i \leq n$, it holds that* $\max_{i \leq j \leq n} \phi(v_j) \leq 4\phi(v_i)$.

Proof. We show that the weight of a vertex at the time of extraction is at most four times smaller than the final weight of any of the vertices extracted before this vertex. Let v_i be a vertex and let $\phi_j(v_i)$ be the weight of this vertex at the time of extraction of some other vertex v_j with $j > i$. Clearly, $\phi(v_j) = \phi_j(v_j) \leq \phi_j(v_i)$, since the algorithm extracted v_j with the minimum weight at the time. If $\phi(v_i) = \phi_i(v_i) \geq \phi_j(v_i)$, then the claim holds.

Otherwise, if $\phi(v_i) = \phi_i(v_i) < \phi_j(v_i)$, then there must be a vertex which caused the weight of v_i to be updated. Let k be the minimum index such that $j \geq k > i$ and $\phi_j(v_i) = \phi_k(v_i)$. We have that $\phi(v_i)$ is a $\frac{11}{10}$-approximation of the Fréchet distance $\mathsf{d}_{\mathcal{F}}(u^i w^i, Z\langle u^i, w^i \rangle)$ for two vertices u^i and w^i. Similarly, we have that $\phi_k(v_i)$ is a $\frac{11}{10}$-approximation of the Fréchet distance $\mathsf{d}_{\mathcal{F}}(u^k w^k, Z\langle u^k, w^k \rangle)$ for two vertices u^k and w^k. Observe that since the extraction of v_k caused the weight of v_i to be updated, it must be that $Z\langle u^k, w^k \rangle$ is a subcurve of $Z\langle u^i, w^i \rangle$. Hence, by Lemma 2.13, we have that

$$\frac{10}{11} \cdot \phi_k(v_i) \leq \mathsf{d}_{\mathcal{F}}(u^k w^k, Z\langle u^k, w^k \rangle) \leq 3\mathsf{d}_{\mathcal{F}}(u^i w^i, Z\langle u^i, w^i \rangle) \leq 3 \cdot \frac{11}{10} \cdot \phi(v_i).$$

Now it follows that $\phi(v_j) \leq \phi_j(v_i) = \phi_k(v_i) \leq 4\phi(v_i)$, which proves the claim. ∎

LEMMA 6.5. *For any $3 \leq i \leq n$ it holds that* $\mathsf{d}_{\mathcal{F}}(Z_{V_i}, Z) \leq 5\phi(v_{i+1})$.

Proof. Let $u_1, \ldots, u_i$ be the vertices in V_i in the order in which they appear on Z_{V_i}. Consider the mapping between Z and this spine curve, which associates every edge $u_j u_{j+1}$ of Z_{V_i} with the subcurve $Z\langle u_j, u_{j+1} \rangle$. Clearly, it holds that

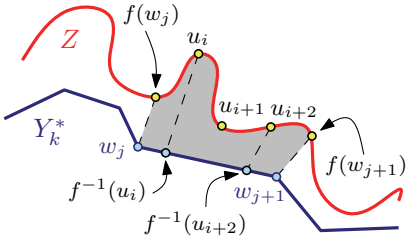
$$\mathsf{d}_{\mathcal{F}}(Z, Z_{V_i}) \leq \max_{1 \leq j < i} \mathsf{d}_{\mathcal{F}}(Z\langle u_j, u_{j+1} \rangle, u_j u_{j+1}) \leq \frac{11}{10} \max_{i < j \leq n} \phi(v_j).$$

Indeed, if u_{j+1} is the successor of u_j on Z; then $\mathsf{d}_{\mathcal{F}}(Z\langle u_j, u_{j+1} \rangle, u_j u_{j+1}) = 0$; otherwise, there must be a vertex which appears on Z between u_j and u_{j+1}, which is contained in $V_n \setminus V_i$, and the weight of this vertex is the approximation of this distance at the time of extraction. Now it follows by Lemma 6.4 that $\mathsf{d}_{\mathcal{F}}(Z, Z_{V_i}) \leq 5\phi(v_{i+1})$. ∎

LEMMA 6.6. *For any $2 \leq k \leq n/2 - 1$, let Y_k^* be the curve with the smallest Fréchet distance from Z with k vertices. (Note that Y_k^* is not restricted to having its vertices lying on Z.) We have that $\mathsf{d}_{\mathcal{F}}(Z, Y_k^*) \geq (5/11)\phi(v_{K+1})$, where $K = 2k - 1$.*

Proof. Let $f : Y_k^* \to Z$ be the mapping realizing the Fréchet distance between Y_k^* and Z. Let $V_i = \langle v_1, \ldots, v_i \rangle$ for $i = 1, \ldots, n$.

Since Y_k^* has only k vertices, it breaks Z into $k - 1$ subcurves. Since, $K \geq 2(k-1)+1$, there must be three consecutive vertices u_i, u_{i+1}, u_{i+2} on Z_{V_K} and two vertices w_j, w_{j+1} of Y_k^*, such that the vertices u_i, u_{i+1}, u_{i+2} appear on the subcurve $Z' = Z\langle f(w_j), f(w_{j+1})\rangle$:



Now, $f^{-1}(u_i)f^{-1}(u_{i+2}) \subseteq w_j w_{j+1}$ and by Lemma 5.2, we have

$$\begin{aligned}
\mathsf{d}_{\mathcal{F}}(Z, Y_k^*) &\geq \mathsf{d}_{\mathcal{F}}(Z\langle f(w_j), f(w_{j+1})\rangle, w_j w_{j+1}) \geq \mathsf{d}_{\mathcal{F}}(Z\langle u_i, u_{i+2}\rangle, f^{-1}(u_i)f^{-1}(u_{i+2})) \\
&\geq \mathsf{d}_{\mathcal{F}}(Z\langle u_i, u_{i+2}\rangle, f^{-1}(u_i)f^{-1}(u_{i+2})) \\
&\geq \frac{1}{2}\mathsf{d}_{\mathcal{F}}(Z\langle u_i, u_{i+2}\rangle, \operatorname{spine}(Z\langle u_i, u_{i+2}\rangle)) \\
&\geq \frac{1}{2} \cdot \frac{10}{11}\phi_{K+1}(u_{i+1}) \geq \frac{5}{11}\phi_{K+1}(v_{K+1}) = \frac{5}{11}\phi(v_{K+1}),
\end{aligned}$$

as the simplification algorithm removed the minimum weight vertex at time $K + 1$ (i.e., v_{K+1}). ∎

6.1.3. The result.

THEOREM 6.7. *Given a polygonal curve Z with n edges, we can preprocess it using $O(n)$ space and $O(n \log^2 n)$ time such that, given a parameter $k \in \mathbb{N}$, we can output in $O(k)$ time a $(2k-1)$-spine curve Z' of Z and a value δ, such that*

(i) $\delta/11 \leq \mathsf{d}_{\mathcal{F}}(Y_k^*, Z)$ *and*

(ii) $\mathsf{d}_{\mathcal{F}}(Z', Z) \leq \delta$,

where Y_k^* is the polygonal curve with k vertices with minimal Fréchet distance from Z. (For $k \geq n/2$ we output Z and $\delta = 0$.)

Proof. The algorithm computing the universal vertex permutation and its associated data structure is described above for $K = 2k - 1$. Specifically, it returns the spine curve $Z' = Z_{V_K}$ as the required approximation with the value $\delta = 5\phi(v_{K+1})$. Computing Z' takes $O(k)$ time. By Lemmas 6.5 and 6.6, we have that Z' and δ satisfy the claim.

Building the data structure takes $O(n \log^2 n)$ time, and it uses $O(n)$ space using $\varepsilon = 1/10$. Each query to this data structure takes $O(\log n \log\log n)$ time. We perform a constant number of these queries to the data structure per extraction from the heap, thus getting the claimed preprocessing time. ∎

6.2. Extending the data structure for Fréchet-distance queries. We use the universal vertex permutation described in the previous section to extend our data structure of section 5.2 to support queries with more than one segment.

6.2.1. The data structure. The input is a polygonal curve $Z \in \mathbb{R}^d$ with n vertices.

Preprocessing. Similar to the algorithm of section 5.2, build a balanced binary tree T on Z. For every internal node ν of T construct the data structure of Theorem 6.7 for $\mathrm{cr}(\nu)$, denoted by $\mathcal{D}_\nu$, and store it at ν.

Answering a query. Given any two vertices u and v of Z, and a query polygonal curve Q with k segments, the task is to approximate $\mathsf{d}_{\mathcal{F}}(\mathsf{Q}, Z\langle u, v\rangle)$. We initially proceed as in section 5.2, computing in $O(\log n)$ time $m = O(\log n)$ nodes $\nu_1, \ldots, \nu_m$ of T such that $Z\langle u, v\rangle = \mathrm{cr}(\nu_1) \oplus \mathrm{cr}(\nu_2) \oplus \cdots \oplus \mathrm{cr}(\nu_k)$. Now, extract a simplified curve with K vertices from $\mathcal{D}_{\nu_i}$, denoted by $\mathrm{simpl}_{\mathrm{K}}(\nu_i)$, for $i = 1, \ldots, m$, where $K = 2k-1$. For $i = 1, \ldots, m$, let δ_i denote the simplification error (as returned by $\mathcal{D}_{\nu_i}$), where $\mathsf{d}_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i), \mathrm{cr}(\nu_i)) \leq \delta_i$ and $\delta_i/11$ is a lower bound to the Fréchet distance of *any* curve with at most k vertices from $\mathrm{cr}(\nu_i)$ for $i = 1, \ldots, m$ (see Theorem 6.7).

Next, compute the polygonal curve $\mathsf{S} = \mathrm{simpl}_{\mathrm{K}}(\nu_1) \oplus \cdots \oplus \mathrm{simpl}_{\mathrm{K}}(\nu_m)$ and its Fréchet distance from Q; that is, $d = \mathsf{d}_{\mathcal{F}}(\mathsf{S}, \mathsf{Q})$. We return

$$\Delta = d + \max_{1 \leq i}^{m} \delta_i \tag{6.1}$$

as the approximate distance between Q and $Z\langle u, v\rangle$.

6.2.2. Analysis.

Query time. Extracting the $m = O(\log n)$ relevant nodes takes $O(\log n)$ time. Querying these m data structures for the simplification of the respective subcurves, takes $O(km)$ overall by Theorem 6.7. Computing the Fréchet distance between the resulting simplification S of $Z\langle u, v\rangle$, which has $O(mk)$ edges, and Q takes time $O(k^2 m \log(k^2 m))$ [6]. Thus the overall time used for answering a query is

$$O(m + km + k^2 m \log(k^2 m)) = O(k^2 m \log(km)) = O(k^2 \log n \log(k \log n)).$$

Preprocessing time and space. Building the initial tree T takes $O(n)$ time and it requires $O(n)$ space. Let $\mathrm{l}(\nu)$ denote the number of vertices of $\mathrm{cr}(\nu)$. For each node ν, computing the additional information and storing it requires $O(\mathrm{l}(\nu))$ space and $O(\mathrm{l}(\nu) \log^2 \mathrm{l}(\nu))$ time. Recall that T is a balanced binary tree and for the nodes $\nu_1, \ldots, \nu_t$ contained in one level of the tree it holds that $\sum_{1 \leq i}^{t} \mathrm{l}(\nu_i) = n$. Thus, computing and storing the additional information takes an additional $O(n \log^3 n)$ time and $O(n \log n)$ space by Theorem 6.7.

Quality of approximation. By the following lemma the data structure achieves a constant factor approximation.

LEMMA 6.8. *Given a polygonal curve Z and a query curve Q with k segments, the value Δ (see (6.1)) returned by the above data structure is a constant factor approximation to $\mathsf{d}_{\mathcal{F}}(\mathsf{Q}, Z\langle u, v\rangle)$.*

Proof. Clearly, Δ bounds the required distance from above, as one can extract a matching of Q and $Z\langle u, v\rangle$ realizing Δ. As such, we need to prove that $\Delta = O(r)$, where $r = \mathsf{d}_{\mathcal{F}}(\mathsf{Q}, Z\langle u, v\rangle)$.

So, let $f : \mathsf{Q} \to Z\langle u, v\rangle$ be the mapping realizing $r = \mathsf{d}_{\mathcal{F}}(\mathsf{Q}, Z\langle u, v\rangle)$, and let $\mathsf{Q}_i = f^{-1}(\mathrm{cr}(\nu_i))$ for $i = 1, \ldots, m$. Clearly, $r = \max_i \mathsf{d}_{\mathcal{F}}(\mathsf{Q}_i, \mathrm{cr}(\nu_i))$. Since Q_i has at most k vertices, by Theorem 6.7 we have

$$\frac{\delta_i}{11} \leq \mathsf{d}_{\mathcal{F}}(\mathsf{Q}_i, \mathrm{cr}(\nu_i)) \leq r \qquad \text{and} \qquad \mathsf{d}_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i), \mathrm{cr}(\nu_i)) \leq \delta_i \tag{6.2}$$

for $i = 1, \ldots, m$. In particular, we have $\delta_i \leq 11r$. Now, by the triangle inequality, we have that

$$\mathsf{d}_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i), \mathsf{Q}_i) \leq \mathsf{d}_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i), \mathrm{cr}(\nu_i)) + \mathsf{d}_{\mathcal{F}}(\mathrm{cr}(\nu_i), \mathsf{Q}_i) \leq \delta_i + r \leq 12r.$$

As such, $d = \mathsf{d}_{\mathcal{F}}(\mathsf{S}, \mathsf{Q}) \leq \max_i \mathsf{d}_{\mathcal{F}}(\mathrm{simpl}_{\mathrm{K}}(\nu_i), \mathsf{Q}_i) \leq 12r$. Now, $\Delta = d + \max_i \delta_i \leq 12r + 11r = 23r$. ∎

The result. Putting the above together, we get the following result. We emphasize that k is being specified together with the query curve, and the data structure works for any value of k.

THEOREM 6.9. *Given a polygonal curve Z with n edges, we can preprocess it in $O(n \log^3 n)$ time and $O(n \log n)$ space such that, given a query specified by*

(i) *a pair of points u and v on the curve Z,*
(ii) *the edges containing these two points, and*
(iii) *a query curve Q with k segments,*

one can approximate $\mathsf{d}_{\mathcal{F}}(\mathsf{Q}, Z\langle u, v\rangle)$ up to a constant factor in $O(k^2 \log n \log(k \log n))$ time.

Proof. The preprocessing is described and analyzed above. The query procedure needs to be modified slightly since the u and v are not necessarily vertices of Z. However, this can be done the same way as for the initial data structure in Theorem 5.6. Let u', v' be the first and last vertices of Z contained in $Z\langle u, v\rangle$. We now extract the $m = O(\log n)$ nodes $\nu_1, \ldots, \nu_m$ of T such that

$$X = uu' \oplus \mathrm{cr}(\nu_1) \oplus \ldots \oplus \mathrm{cr}(\nu_m) \oplus v'v = Z\langle u, v\rangle .$$

We continue with the procedure as described above using this node set. The analysis of Lemma 6.8 applies with minor modifications. ∎

7. Conclusions. In this paper, we presented algorithms for approximating the Fréchet distance when one is allowed to perform shortcuts on the original curves. More specifically the presented algorithms approximate the directed vertex-restricted shortcut Fréchet distance. Surprisingly, for c-packed curves it is possible to compute a constant factor approximation in a running time which is near linear in the complexity of the input curves.

We also presented a way to compute an ordering of the vertices of the curve such that any prefix of this ordering serves as a good approximation to the curve in the Fréchet distance, and it is optimal up to constant factors. We used this universal vertex permutation to develop a data structure that can quickly approximate (up to a constant factor) the (regular) Fréchet distance between a query curve and the input curve. Surprisingly, the query time is logarithmic in the complexity of the original curve (and near quadratic in the complexity of the query curve).

There are many open questions for further research. The most immediate questions are how to extend our result to the other definitions of a shortcut Fréchet distance mentioned in the introduction and how to improve the approximation factor. The work in this paper is a step toward solving these more difficult questions.

As for exact computations, it is easy to see that one can obtain polynomial-time algorithms by modifying the algorithms presented in this paper even for general polygonal curves; see also [22]. Surprisingly, a more recent result shows that if the requirement that shortcuts have to start and end at input vertices is dropped, the problem of computing the shortcut Fréchet distance becomes NP-hard [16, 22].

Acknowledgments. The authors thank Mark de Berg, Marc van Kreveld, Benjamin Raichel, Jessica Sherette, and Carola Wenk for insightful discussions on the problems studied in this paper and related problems. The authors also thank the anonymous referees for their detailed and insightful comments.

REFERENCES

[1] M. Abam, M. de Berg, P. Hachenberger, and A. Zarei, *Streaming algorithms for line simplification*, Discrete Comput. Geom., 43 (2010), pp. 497–515.

[2] P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir, *Computing the discrete Fréchet distance in subquadratic time*, in Proceedings of the 24rd ACM-SIAM Symposium on Discrete Algorithms, 2013, pp. 156–167.

[3] P. K. Agarwal, S. Har-Peled, N. Mustafa, and Y. Wang, *Near-linear time approximation algorithms for curve simplification in two and three dimensions*, Algorithmica, 42 (2005), pp. 203–219.

[4] C. Aggarwal and S. Yu, *An effective and efficient algorithm for high-dimensional outlier detection*, Int. J. Very Large Data Bases, 14 (2005), pp. 211–221.

[5] H. Alt, A. Efrat, G. Rote, and C. Wenk, *Matching planar maps*, J. Algorithms, 49 (2003), pp. 262–283.

[6] H. Alt and M. Godau, *Computing the Fréchet distance between two polygonal curves*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 75–91.

[7] J. L. Bentley and J. B. Saxe, *Decomposable searching problems* I*: Static-to-dynamic transformation*, J. Algorithms, 1 (1980), pp. 301–358.

[8] S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu, *Simplifying* 3*d polygonal chains under the discrete fréchet distance*, in Proceedings of the 8th Latin American Symposium on Theoretical Informatics, 2008, pp. 630–641.

[9] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, *On map-matching vehicle tracking data*, in Proceedings of the 31st VLDB Conference, 2005, pp. 853–864.

[10] K. Buchin, M. Buchin, and J. Gudmundsson, *Detecting single file movement*, in Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2008, pp. 288–297.

[11] K. Buchin, M. Buchin, J. Gudmundsson, Maarten L., and J. Luo, *Detecting commuting patterns by clustering subtrajectories*, in Proceedings of the 19th Annual International Symposium on Algorithms and Computation, 2008, pp. 644–655.

[12] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk, *How difficult is it to walk the dog?*, in Proceedings of the 23rd European Workshop on Computational Geometry, 2007, pp. 170–173.

[13] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer, *Four Soviets Walk the Dog—With an Application to Alt's Conjecture*, arXiv:1209.4403, 2012.

[14] K. Buchin, M. Buchin, and Y. Wang, *Exact algorithms for partial curve matching via the Fréchet distance*, in Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms, 2009, pp. 645–654.

[15] K. Buchin, M. Buchin, and C. Wenk, *Computing the Fréchet distance between simple polygons*, Comput. Geom. Theory Appl., 41 (2008), pp. 2–20.

[16] M. Buchin, A. Driemel, and B. Speckmann, *Computing the Fréchet distance with shortcuts is NP-hard*, in Proceedings of the 29th European Workshop on Computational Geometry, 2013, pp. 43–46.

[17] D. Chen, A. Driemel, L. Guibas, A. Nguyen, and C. Wenk, *Approximate map matching with respect to the Fréchet distance*, in Proceedings of the 13th Workshop on Algorithm Engineering Experience, 2011.

[18] M. de Berg, *Improved bounds on the union complexity of fat objects*, Discrete Comput. Geom., 40 (2008), pp. 127–140.

[19] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed., Springer, Berlin, 2008.

[20] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels, *Realistic input models for geometric algorithms*, Algorithmica, 34 (2002), pp. 81–97.

[21] M. de Berg and M. Streppel, *Approximate range searching using binary space partitions*, Comput. Geom. Theory Appl., 33 (2006), pp. 139–151.

[22] A. Driemel, *Realistic Analysis for Algorithmic Problems on Geographical Data*, Ph.D. thesis, Utrecht University, 2013.

[23] A. Driemel and S. Har-Peled, *Jaywalking Your Dog—Computing the Fréchet Distance with Shortcuts*, arXiv:1107.1720, 2011.
[24] A. Driemel, S. Har-Peled, and C. Wenk, *Approximating the Fréchet distance for realistic curves in near linear time*, Discrete Comput. Geom., 48 (2012), pp. 94–127.
[25] A. Efrat, *The complexity of the union of* (α, β)*-covered objects*, SIAM J. Comput., 34 (2005), pp. 775–787.
[26] G. N. Frederickson and D. B. Johnson, *Generalized selection and ranking: Sorted matrices*, SIAM J. Comput., 13 (1984), pp. 14–30.
[27] L. J. Guibas, J. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink, *Approximating polygons and subdivisions with minimum link paths*, Internat. J. Comput. Geom. Appl., 3 (1993), pp. 383–415.
[28] S. Har-Peled and B. Raichel, *The Fréchet distance revisited and extended*, in Proceedings of the 27th Annual ACM Symposium on Computational Geometry, 2011, pp. 448–457; also available online from http://www.cs.uiuc.edu/~sariel/papers/10/frechet3d/.
[29] S. Har-Peled and B. Raichel, *The Fréchet distance revisited and extended*, ACM Trans. Algorithms, to appear.
[30] E. J. Keogh and M. J. Pazzani, *Scaling up dynamic time warping to massive dataset*, in Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery, 1999, pp. 1–11.
[31] M. S. Kim, S. W. Kim, and M. Shin, *Optimization of subsequence matching under time warping in time-series databases*, in Proceedings of the ACM Symposium on Applied Computing, 2005, pp. 581–586.
[32] J. Gudmundsson, M. de Berg, and A. Cook IV, *Fast Fréchet queries*, in Proceedings of the 22nd Annual International Symposium on Algorithms and Computation, 2011.
[33] A. Maheshwari, J.-R. Sack, K. Shahbaz, and H. Zarrabi-Zadeh, *Fréchet distance with speed limits*, Comput. Geom. Theory Appl., 44 (2011), pp. 110–120.
[34] R. Maronna, D. Martin, and V. Yohai, *Robust Statistics: Theory and Methods*, Wiley, New York, 2006.
[35] A. Mascret, T. Devogele, I. Le Berre, and A. Hénaff, *Coastline matching process based on the discrete Fréchet distance*, in Proceedings of the 12th International Symposium on Spatial Data Handling, 2006, pp. 383–400.
[36] M. H. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Comput. Sci. 156, Springer, Berlin, 1983.
[37] J. Serrà, E. Gómez, P. Herrera, and X. Serra, *Chroma binary similarity and local alignment applied to cover song identification*, IEEE Trans. Audio Speech Language Process., 16 (2008), pp. 1138–1151.
[38] C. Wenk, R. Salas, and D. Pfoser, *Addressing the need for map-matching speed: Localizing global curve-matching algorithms*, in Proceedings of the 18th International Conference Scientific and Statistical Database Management, 2006, pp. 879–888.