A Dimension-Reducing Fréchet Simplification Oracle

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

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

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

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

Abstract

Let P be a polygonal curve with n vertices in the plane. We construct a data structure of size $O(n \log n)$ suited for simplification queries of the following kind. Given a query line ℓ and an integer $k \geq 1$, find a curve Q on ℓ with at most k vertices that minimizes the discrete Fréchet distance to P, among all such curves. Using our data structure, a query can be handled in $O(k^2 \log^3 n + k \log^4 n)$ time.

More generally, a geometric tree T on n vertices in the plane can be preprocessed into a near-linear-size structure so that, given a pair u, v of its vertices, a line ℓ, and an integer $k \geq 1$, one can find a curve Q on ℓ with at most k vertices that minimizes the discrete Fréchet distance to the path from u to v in T, in time $O(k^2 \operatorname{polylog} n)$.

For the general *dimension-reduction* problem, where P is a curve in $\mathbb{R}^d$ ($d \geq 3$), $0 < \varepsilon_0 < 1$ is a real parameter, and a query specifies a g-flat h ($1 \leq g \leq d-1$) and an integer $k \geq 1$, we construct a data structure of size $O(n \log n + f(\varepsilon_0)n)$, where $f(\varepsilon_0) = (1 + 1/\varepsilon_0)^{(d-1)/2}$, that allows us to find a curve Q on h with at most k vertices, whose discrete Fréchet distance to P is at most $1 + \varepsilon_0$ times the distance of Q^* to P, where Q^* is such a curve that minimizes the distance to P. The query handling time is $O(f(\varepsilon_0)k^2 \log^2 n)$.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Computational geometry, discrete Fréchet distance, curve simplification oracle, restricted minimum enclosing disk queries

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2025.6

Funding *Boris Aronov*: Work partially supported by NSF Grant CCF-20-08551.
Matthew J. Katz: Work partially supported by Grant 2019715/CCF-20-08551 from the US-Israel Binational Science Foundation/US National Science Foundation.
Indu Ramesh: Work supported by NSF Grant CCF-20-08551.

1 Introduction

Curve simplification is an important and extensively studied problem in computational geometry and related fields. Here, we restrict our attention to curve simplification with the quality of the simplification measured using the discrete Fréchet distance between the original curve and its simplification. See below for a comprehensive survey of research in this area.

In this paper, we introduce a novel class of curve simplification problems, in which the simplified curve is required to lie within a specified subspace or region. More formally, let P be a polygonal curve in $\mathbb{R}^d$. We are interested in problems where, in addition to standard

© Boris Aronov, Tsuri Farhana, Matthew J. Katz, and Indu Ramesh;
licensed under Creative Commons License CC-BY 4.0
36th International Symposium on Algorithms and Computation (ISAAC 2025).
Editors: Ho-Lin Chen, Wing-Kai Hon, and Meng-Tsung Tsai; Article No. 6; pp. 6:1–6:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameters such as a threshold on the distance between P and the desired simplification Q or a bound on the length (i.e., number of vertices) of Q, Q must also reside in a designated subspace or region. Moreover, since it is often desirable to find the optimal simplification of P for multiple subspaces or regions, we concentrate on the multi-shot formulation of these problems.

These questions can also be viewed as *dimension-reduction* problems. That is, we wish to replace an input d-dimensional curve by a curve in some lower-dimensional space, while minimizing the Fréchet distance to the input curve.

Specifically, we focus on the following problem. Let P be a polygonal curve in $\mathbb{R}^d$. Preprocess P into a compact data structure to efficiently support queries that specify a g-flat[1] h $(1 \leq g \leq d-1)$ and an integer $k \geq 1$, and request a curve Q on h of length at most k that minimizes the discrete Fréchet distance to P. For $d = 2$, the g-flat h is simply a line ℓ in the plane, and we present an efficient solution to the problem, as well as to the more general problem in which the input is a geometric tree T rather than a curve P. In the latter problem, a query also specifies the path in the tree to which the query applies. For $d > 2$, we present an efficient solution that provides approximate answers to queries, that is, given h and k, it returns a curve Q on h with at most k-vertices which is approximately the best such curve on h, in terms of its discrete Fréchet distance to P.

Basic definitions. In this paper we view a finite sequence of points $P = (p_1, \ldots, p_n)$ in $\mathbb{R}^d$ as a polygonal curve; we refer to n as the *size* or *length* of P and the points as its *vertices.*

Let $P = (p_1, \ldots, p_n)$ and $Q = (q_1, \ldots, q_m)$ be polygonal curves in $\mathbb{R}^d$. A *legal walk* of P and Q (or just *walk*, for short) is a monotonically increasing sequence of pairs $(r_1, \ldots, r_l)$, where $r_1 = (p_1, q_1)$, $r_l = (p_n, q_m)$, and, in general, when advancing from $r_k = (p_i, q_j)$, r_{k+1} is one of the following: (p_{i+1}, q_j) (provided $i < n$), (p_i, q_{j+1}) (provided $j < m$), or (p_{i+1}, q_{j+1}) (provided $i < n$ and $j < m$). The *cost* of a walk is the maximum distance among the distances between the vertices of each pair $r_k = (p_i, q_j)$ in the walk. The *discrete Fréchet distance* $d_{\mathrm{dF}}(P, Q)$ between P and Q is the cost of the minimum-cost walk of P and Q.

Given a g-flat h, $1 \leq g \leq d-1$, we say that Q is a *k-vertex curve on h* if $Q = (q_1, \ldots, q_k)$ is a sequence of k points on h. An *at-most-k-vertex curve on h* is a k'-vertex curve on h with $1 \leq k' \leq k$.

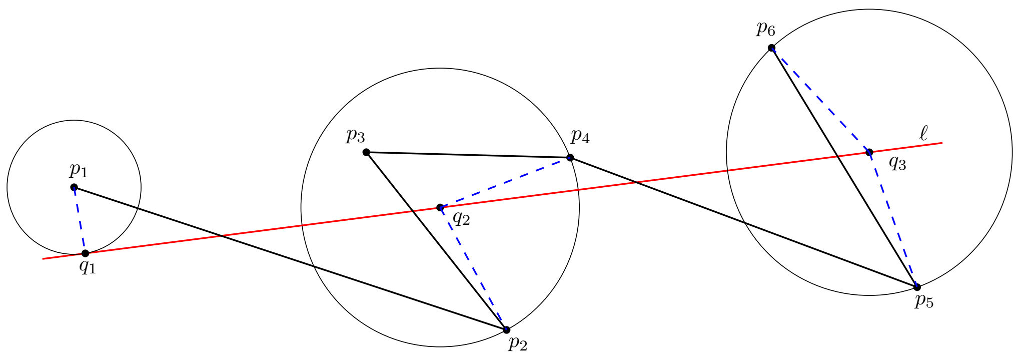

Figure 1 A polygonal curve $P = (p_1, \ldots, p_6)$ and a query line ℓ. In this example, $k = 3$ and $Q = (q_1, q_2, q_3)$ is the returned curve.

[1] That is, a g-dimensional flat – a g-dimensional affine subspace of $\mathbb{R}^d$.

Main problem statement. In this paper we study the following problem. Given a curve P in $\mathbb{R}^d$ as above, preprocess it into a near-linear size data structure to efficiently support queries of the following type: "Given a g-flat h, $1 \le g \le d-1$, and integer $k \ge 1$, find an at-most-k-vertex curve Q on h minimizing $d_{\mathrm{dF}}(P, Q)$." We first focus on the planar version, where P is a curve in the plane and queries are lines. See Figure 1 for an example.

We also consider the more general setting, still in the plane, where the input is a geometric tree T rather than a curve P, and the queries are of the form: "Given two nodes p, q of the tree, a line ℓ, and integer k, find an at-most-k-vertex curve Q on ℓ minimizing the discrete Fréchet distance between Q and the path from p to q in T."

In other words, we first study problems in which we need to preprocess a given curve P (or tree T) in the plane, so that one can efficiently handle *simplification* queries with respect to a query line. The maximum length k of the desired simplification is also a parameter of the query.

Next, we further generalize the problem so that a query may specify an arbitrary polygonal region in which Q must reside; query time will of course depend on the complexity of the region.

Finally, we consider the problem in three and higher dimensions, where P is a curve in $\mathbb{R}^d$, $d \ge 3$, and queries are g-flats, $1 \le g \le d-1$. Since we are interested in a near-linear size data structure, we resort to approximate queries (see discussion in Section 6), with a prespecified error parameter $\varepsilon_0 > 0$. That is, the answer to a query is an at-most-k-vertex curve Q on h, such that $d_{\mathrm{dF}}(P, Q) \le (1+\varepsilon_0) d_{\mathrm{dF}}(P, Q^*)$, where Q^* is such a curve minimizing the distance to P.

Our results. We present efficient solutions to these problems. Specifically, given P or T in the plane, we construct a data structure of size $O(n \log n)$ that allows us to answer a query in $O(k^2 \operatorname{polylog} n)$ time; refer to Theorem 7 in Section 3 and Theorem 8 in Section 4 for the exact statements. As an intermediate result, we present an algorithm for the corresponding decision problem (with respect to a curve P or tree T). For example, given a line ℓ and distance r, we can determine in $O(k \log^2 n)$ time whether there exists an at-most-k-vertex curve on ℓ within discrete Fréchet distance at most r from P, and if so, return such a curve Q with fewest vertices. In Section 5 we extend the data structure so that a query may specify a polygonal region in the plane that restricts the location of the curve Q.

In higher dimensions, given P in $\mathbb{R}^d$ and $0 < \varepsilon_0 < 1$, we construct a data structure of size $O(n \log n + \frac{n}{\varepsilon^{(d-1)/2}})$, where $\varepsilon = \frac{\varepsilon_0}{1+\varepsilon_0}$, that allows us to answer a query approximately in $O(\frac{k^2 \log^2 n}{\varepsilon^{(d-1)/2}})$ time, that is, the discrete Fréchet distance between P and the curve that we return is at most $1+\varepsilon_0$ times the distance between P and the optimal curve; refer to Theorem 12 in Section 6 for the exact statement. Our solution uses coresets.

Related work on curve simplification. We focus here on (global) simplification under the Fréchet distance in the plane, where the quality of a simplification is the Fréchet distance between the original curve P and its simplification Q.

There are two main versions of the simplification problem. In the first, we are given a threshold δ and the goal is to compute a curve Q of minimum length, such that the distance between P and Q is at most δ. In the second, we are given an integer $k \ge 1$ and the goal is to compute a curve Q of length at most k that minimizes the distance to P. Moreover, each of the two main versions gives rise to four standard variants, depending on whether we are using the continuous or discrete Fréchet distance, and whether Q is *restricted* (i.e., a subsequence of P) or *unrestricted* (i.e., any sequence of points); Van de Kerkhof et al. [18] also consider an intermediate *curve-restricted* option.

The following results are for the first version under the continuous Fréchet distance. Bringmann and Chaudhury [7] presented an algorithm with running time $O(n^3)$ for restricted simplification and established a conditional cubic lower bound. An algorithm with the same running time was also proposed by Van de Kerkhof et al. [18] (see also Van Kreveld et al. [19]). For unrestricted simplification, Guibas et al. [12] provide an algorithm with running time $O(n^2 \log^2 n)$, and Agarwal et al. [2] provide an $O(n \log n)$ algorithm, which returns a simplification Q' of length at most the length of an optimal simplification with threshold $\delta/8$. The latter two statements can be deduced from the corresponding papers, as observed by Van de Kerkhof et al. [18].

Bereg et al. [4] studied both restricted and unrestricted simplification for each of the two main versions, under the discrete Fréchet distance in three-dimensional space. For the first version, they compute an optimal unrestricted simplification in $O(n \log n)$ time and an optimal restricted simplification in $O(n^2)$ time. For the second version, they compute an optimal unrestricted simplification in $O(kn \log n \log(n/k))$ time, where k is the length of the computed simplification, and an optimal restricted simplification in $O(n^3)$ time. Also under the discrete Fréchet distance, Fan et al. [10] considered the so called *chain-pair simplification problem* in the plane, where the goal is to simultaneously compute restricted simplifications of length k for two input curves, such that the distance between the simplifications themselves is also below some threshold.

Finally, under the continuous Fréchet distance, Driemel and Har-Peled [8] presented a near-linear time algorithm that computes a permutation of the vertices of P, such that any prefix of $2k-1$ vertices of this permutation is an approximation (up to a constant factor) of P compared to any polygonal curve with k vertices. Subsequently, Filtser [11] slightly improved the approximation factor under the discrete Fréchet distance.

More related work. A series of papers [16, 15, 5] culminates in an algorithm that preprocesses an n-point set S in the plane in linear space and $O(n \log n)$ time to support $O(\log n)$ time queries of the form "What is the smallest circle centered on the query line and enclosing S?"

There is also a substantial amount of work on so-called *range-aggregate* queries. Given an n-point set S in the plane, we want to preprocess it for queries of the following type: compute some aggregate quantity about the points of S contained in an axis-aligned query rectangle. While range counting, reporting, and more general semigroup queries have been widely studied [1], some types of queries are not readily decomposable (in the sense of the answer being easily synthesized from the answer for smaller queries). For example, Brass et al. [6] discuss computing the area (or perimeter) of the convex hull, width, and smallest enclosing disk of the points of S in the query rectangle. They describe algorithms for preprocessing S into a data structure of near-linear size with polylogarithmic-time queries, for all of the above queries except for the width; the width data structure is more expensive. Gupta et al. [13] explore several variants of closest-pair-in-range problems. For example, they show a near-linear-space structure for computing the closest pair of points within the query rectangle in two dimensions with polylogarithmic query time. Our observation in Remark 4 is a further generalization in this direction: Given a point set in the plane, we can preprocess it so that, given a query rectangle *and* a query line, we can determine the smallest enclosing circle of the points in the rectangle, among all circles centered on the line.

Overview and organization. We first consider the planar version of the dimension-reduction problem. To answer a *decision* query of the form "given a line ℓ, an integer k, and a distance $r > 0$, determine whether there exists an at-most-k-vertex curve on ℓ within Fréchet distance at most r from P", we need to repeatedly perform the primitive operation $\texttt{prefix}_\ell(P[i,n],r)$ for different values of i: Find the longest (contiguous) subcurve of P, beginning at p_i, that fits into a disk of radius r centered at a point of ℓ. In Section 2, we describe our data structure for the planar version and then use it to implement $\texttt{prefix}_\ell(P[i,n],r)$ in $O(\log^2 n)$ time. Our data structure is obtained by combining known data structures in a non-trivial manner. In Section 3, we obtain our first main result, i.e., the solution for the planar version of the dimension-reduction problem. Using the data structure and primitive operations developed in Section 2, we present efficient algorithms for handling both decision queries and "regular" queries, where the algorithm for regular queries is based on the one for decision queries and on the primitive operation $\mathrm{radius}_\ell(P[i,j])$ (developed in Section 2) which, given a (contiguous) subcurve of P, returns the radius of the smallest disk centered at a point of ℓ that contains the subcurve. In Section 6, we obtain our second main result, i.e., the solution for the d-dimensional version, $d \geq 3$, of the dimension-reduction problem. The algorithms in this general version are similar to those in the planar version, however, we need to replace the exact primitive operations above by their approximate d-dimensional counterparts. These latter operations are implemented using coresets. Finally, in Sections 4 and 5, we extend our result for the planar version in two directions. In Section 4, the input is a geometric tree T (rather than a curve P), and queries specify, in addition to ℓ and k, a path in T, by providing its start and end vertices. In Section 5, queries specify a polygonal region rather than a line ℓ, to contain the simplified curve.

2 Facts and tools

2.1 Facts

For two points p, q in the plane, let $d(p,q)$ be the Euclidean distance between them; many of our observations generalize to other norms, but in this version we focus on the Euclidean metric, for simplicity of presentation.

For a point set Q, let $\mathrm{radius}(Q)$ be the radius of the *minimum enclosing circle* (or *mec*, for short) of Q, and let $\mathrm{radius}(Q,p)$ be the radius of the mec of Q centered at p. Moreover, let $\mathrm{radius}_\ell(Q)$ be the radius of the mec of Q centered at a point of line ℓ, and let $\mathrm{radius}_\ell(Q,x)$ be the radius of the mec of Q centered at $x \in \ell$, as a function of x. We give the proof of the following standard fact below, for completeness.

▶ **Fact 1.** *Both* $\mathrm{radius}(Q,x)$ *and* $\mathrm{radius}_\ell(Q,x)$ *are convex and have a unique minimum.*

Proof. Since $\mathrm{radius}(Q,x) = \max_{q \in Q} d(q,x)$, and $d(q,x)$ is a convex function of x, $\mathrm{radius}(Q,x)$ is convex. Therefore $\mathrm{radius}_\ell(Q,x)$ is also convex, as a restriction of the convex function $\mathrm{radius}(Q,x)$ to a line.

As for uniqueness of minimum, we prove it for $\mathrm{radius}_\ell(Q,x)$; it also proves the claim for $\mathrm{radius}(Q,x)$.

For a contradiction, suppose p and $q \neq p$ are both minima of $\mathrm{radius}_\ell(Q,x)$. Then the disks D_p, D_q of radius $r := \mathrm{radius}_\ell(Q,p) = \mathrm{radius}_\ell(Q,q)$ centered at p and q, respectively, each cover Q. Therefore Q is contained in the lens $D_p \cap D_q$. Let s be a point on the open segment $pq \subset \ell$. Then the disk centered at s and just covering the lens $D_p \cap D_q$ contains Q and has a smaller radius, contradicting the assumption that p and q were minima of $\mathrm{radius}_\ell(Q,x)$. ◀

2.2 Primitives in the plane

We now list the operations that will be useful in solving the two-dimensional problems addressed in this paper. Each of the operations below depends on the query line ℓ.

$\texttt{disk}_\ell(Q)$: The smallest-radius disk centered at a point of ℓ and containing a set Q of points; let $\text{radius}_\ell(Q)$ and $\text{center}_\ell(Q)$ denote its radius and center, respectively.

$\texttt{feasible}_\ell(Q, r)$: The interval of ℓ (which may be empty) that is the locus of centers of disks of radius r containing Q.

$\texttt{fits?}_\ell(Q, r)$: Is there a disk of radius r centered at a point of ℓ and containing a set Q of points? Equivalent to $\texttt{feasible}_\ell(Q, r) \neq \emptyset$ or to $\text{radius}_\ell(Q) \leq r$; the former version is sometimes more efficient.

$\texttt{prefix}_\ell(S, r)$: For a sequence S of points, its longest prefix (which may be empty or all of S) that fits into a disk of radius r centered at a point of ℓ. In practice, $\texttt{prefix}_\ell(S, r)$ returns the length of the prefix, so 0 if it is empty.

2.3 Data structures in the plane

We now describe the data structures we employ to implement the above operations. Consider a polygonal curve P of length $n > 1$. For clarity of presentation we assume that n is a power of two; the general case requires minimal straightforward modifications. We denote by $P[i, j]$ the (contiguous) subcurve $(p_i, p_{i+1}, \ldots, p_j)$, for $1 \leq i \leq j \leq n$.

The hierarchical decomposition. The *hierarchical decomposition* of P is a binary tree, in which the root corresponds to $P = P[1, n]$, its left and right children to $P[1, n/2]$ and $P[n/2 + 1, n]$, respectively, and so forth. The leaves correspond to the single-vertex curves. Each node of the tree corresponds to a *canonical subcurve* of P (we will use the term "canonical set" when the ordering of the vertices in the subcurve is immaterial). A general subcurve $P[i, j]$ corresponds to the concatenation of $O(\log n)$ canonical subcurves, which can be identified from the value of the indices i and j in $O(\log n)$ time.

We often build data structures based on the hierarchical decomposition of a curve, where at each node we store an additional structure tailored to its corresponding canonical subcurve.

Farthest-point Voronoi diagram as a map. For a given set of points Q in the plane, the *farthest-point Voronoi diagram* $\text{FVD}(Q)$ is a planar map decomposing the plane into convex regions $R(p)$, one for each $p \in Q$, where $R(p)$ is the locus of all points in the plane for which p is the farthest point in Q. $\text{FVD}(Q)$ is a planar map with $O(|Q|)$ vertices, edges, and faces. Each face is a convex unbounded polygon. We preprocess the planar map for point location, so that, given a query point q, one can find the region $R(p)$ containing q, and thereby the farthest point of q in Q, in $O(\log |Q|)$ time.

Centroid decomposition. Consider a free tree T of degree at most three. A *centroid edge* is an edge of T whose removal splits T into two trees of almost equal size, with between $1/3$ and $2/3$ of the vertices of T. It is known to exist in any degree-at-most-three tree. The existence of a centroid edge naturally induces a *centroid decomposition* of T: the root is a centroid edge e. Each subtree left and right of the root is, recursively, a centroid decomposition of the two trees formed by the removal of e from T. Each internal node corresponds to an edge of T. Each leaf corresponds to either a single edge or to two edges of T.

Farthest-point Voronoi diagram as a tree. Given a set Q in the plane, the 1-skeleton of FVD(Q) (i.e., the collection of its vertices and edges, viewed as a graph with virtual vertices at the "ends" of infinite ray edges) is a tree, which, for Q in general position, has degree three. For simplicity of exposition, we will assume that the points of Q are in general position; refer to [5] for how to handle degeneracies in Q when constructing the centroid decomposition.

We now describe what is essentially the centroid-decomposition-based data structure from [5] for computing the smallest disk enclosing a given set Q of points, with the center on the query line ℓ, i.e., $\mathtt{disk}_\ell(Q)$. For a fixed set Q, the centroid decomposition of FVD(Q) requires linear space and $O(|Q|\log|Q|)$ time to build. In our data structure, for a path P, we build the centroid decomposition for all canonical subsets of P, which requires $O(n \log n)$ overall space and $O(n \log^2 n)$ time.

We store the centroid decomposition of FVD(Q) together with some additional geometric information, as follows: Each internal node x corresponds to an edge e_x of FVD(Q), which is a line segment[2] separating regions $R(v_x)$ and $R(w_x)$. Let $m(e_x)$ be the midpoint of e_x. There exist two infinite rays $\rho(v_x)$ and $\rho(w_x)$[5] emanating from $m(e_x)$, one fully contained in $R(v_x)$ and one in $R(w_x)$, respectively. Store these two rays together with the centroid edge e_x and points v_x and v_w at the current node x of the centroid decomposition tree. Notice that the polygonal line $\rho(v_x) \cup \rho(w_x)$ splits the plane into two regions: one contains the subtree corresponding to the left child of x and the other – the subtree of its right child.

The full structure T_{FVD}. Let $T_{\mathrm{FVD}} = T_{\mathrm{FVD}}(P)$ be the data structure constructed as follows: We start with the hierarchical decomposition of P. At each node of the decomposition, which corresponds to a canonical subcurve $P[i,j]$ of P, we store FVD($P[i,j]$) both in the planar map and centroid decomposition form.

2.4 Implementations

We now describe how to efficiently implement the primitives and analyze their running times.

2.4.1 Implementation of $\mathtt{disk}_\ell(Q)$

If Q is a canonical set of size n. The algorithm in [5] preprocesses Q in linear space and $O(n \log n)$ time to support such an operation in $O(\log n)$ time; see Section 2.3 for a description of the centroid decomposition data structure. Roughly speaking, the search proceeds by descending the centroid decomposition of FVD(Q), narrowing down the portion of the query line ℓ containing the disk center, at a constant cost per level. At a leaf of the decomposition, at most three farthest neighbors in Q remain, which allows to identify the center and radius directly. We use a more elaborate version below to solve the two-set version of the problem.

If Q is a union of two canonical sets. Let Q_1 and Q_2 be two canonical sets of total size at most n, and let ℓ be the query line. Put $Q \coloneqq Q_1 \cup Q_2$. We wish to compute $\mathtt{disk}_\ell(Q)$, the smallest disk with center on ℓ enclosing Q; let x^* denote its center, to be computed. We will need the following information about the sets Q_1 and Q_2 (already stored in T_{FVD}): FVD(Q_1) and FVD(Q_2) viewed both as planar maps, preprocessed for point location, and as planar trees, stored as centroid decompositions, see Section 2.3.

[2] Unbounded edges have to be handled slightly differently; we omit the details.

For clarity of explanation, identify ℓ with the x-axis and let x denote a generic point of ℓ. The center of the desired disk is the point $x = x^*$ minimizing $\mathrm{radius}_\ell(Q, x) = \max(\mathrm{radius}_\ell(Q_1, x), \mathrm{radius}_\ell(Q_2, x))$. Each of the three functions are convex on ℓ, with a unique minimum, by Fact 1.

Note that the following *sidedness test* can be performed in $O(\log n)$ time: Given a point $x \in \ell$, we can determine whether x^* lies before, after, or at x along ℓ. Indeed, after querying with x in $\mathrm{FVD}(Q_1)$ and $\mathrm{FVD}(Q_2)$, we can determine the distance from x to the farthest points of Q (possibly more than one) and therefore the direction in which this distance decreases along ℓ – that's the direction towards x^*; if no such direction exists, then $x = x^*$. The bottleneck of this operation is the lookup of x in the planar map $\mathrm{FVD}(Q_1)$ and $\mathrm{FVD}(Q_2)$, at a cost of $O(\log n)$.

Finally, we describe the computation of $\mathtt{disk}_\ell(Q)$. Throughout the search we maintain the current segment $s \subset \ell$ that is guaranteed to contain x^*. We initially set $s := \ell$.

We now descend the centroid decompositions of $\mathrm{FVD}(Q_1)$ and $\mathrm{FVD}(Q_2)$. Without loss of generality, suppose we are currently descending the decomposition of $\mathrm{FVD}(Q_1)$, with the current node corresponding to edge e and its two rays ρ_1, ρ_2 splitting the plane into wedges W_1 and W_2; s intersects $\rho_1 \cup \rho_2$ at most twice. At each of the intersection points we perform our sidedness test and eliminate part of s as the possible position of x^*. The result is an, in general, smaller updated segment s, contained fully in W_1 or in W_2 and containing x^*. We descend the decomposition of $\mathrm{FVD}(Q_1)$ to the corresponding child of e.

Before terminating the current step, we check if the updated s crosses e. If it does, we split s at the intersection point, check what side contains x^* by another sidedness test, and shrink s further. Notice that the resulting segment s is guaranteed not to meet any of the edges of the discarded child of e in the centroid decomposition, nor e itself. Thus we inductively maintain the invariant that the current segment s can only intersect the edges of $\mathrm{FVD}(Q_i)$ corresponding to the current subtree in its centroid decomposition.

We continue in this manner, until we reach the bottom of both hierarchies. At a single-edge leaf (a two-edge leaf is handled similarly) of a centroid decomposition of, say, $\mathrm{FVD}(Q_1)$, one edge remains, and we apply the same trick to shrink the segment s to only one side of the edge, belonging to, say the Voronoi region of $q_1 \in Q_1$. Similarly, we narrow things down to one site $q_2 \in Q_2$. By construction, now $x^* \in s$, and s intersects no edges of $\mathrm{FVD}(Q_1)$ nor of $\mathrm{FVD}(Q_2)$. In particular, for all points on s, q_1 is the furthest point of Q_1 and q_2 – of Q_2. We check one of three possibilities: (a) q_1 could be the only point determining $\mathtt{disk}_\ell(Q)$: we project q_1 orthogonally to ℓ and check if its projection q_1' lies in s. If so, check that Q is contained in the disk of radius $d(q_1, q_1')$ centered at q_1' (i.e., that $d(q_2, q_1') \le d(q_1, q_1')$) – then this is $\mathtt{disk}_\ell(Q)$. (b) We check if q_2 is the only point defining $\mathtt{disk}_\ell(Q)$ similarly. Finally, (c) we construct the intersection of the bisector $b(q_1, q_2)$ with s (or, equivalently, with ℓ). This point q is the center of $\mathtt{disk}_\ell(Q)$ and its radius is $d(q, q_1) = d(q, q_2)$. It is easily checked that the processing at the bottom of the two hierarchies takes $O(1)$ work once the identity of q_1 and q_2 is known.

Since the descent of the two logarithmic-height hierarchies takes $O(\log n)$ time for one step (the bottleneck in the sidedness test is consulting a point with known position in one of the FVDs for its furthest point in the other set: in each sidedness test we already know where we are in one diagram, but not in the other), we reach the bottom and the solution in $O(\log^2 n)$ time. We conclude that

▶ **Lemma 2.** *The operation $\mathtt{disk}_\ell(Q_1 \cup Q_2)$, where Q_1, Q_2 are two canonical sets of total size n, can be performed in $O(\log^2 n)$ time.*

If Q is a union of $m \geq 2$ canonical sets.

▶ **Lemma 3.** *Let $Q_1, \ldots, Q_m$ be $m \geq 2$ canonical subsets of P of total size at most n, let their union be Q, and let ℓ be a line. We can compute $\mathtt{disk}_\ell(Q)$ in $O(m^2 \log^2 n)$ time.*

Proof. For each pair (i, j), with $1 \leq i < j \leq m$, we compute $\mathtt{disk}_\ell(Q_i \cup Q_j)$ by an application of the two-set procedure above and return the largest disk. The runtime bound is immediate, so we focus on correctness.

Since $\mathtt{disk}_\ell(Q)$ is determined by at most two points of Q (refer to [16, Observation 1]), it follows that the desired disk is indeed $\mathtt{disk}_\ell(Q_k \cup Q_l)$, if one of the defining points comes from Q_k and the other from Q_l with $l \neq k$; if the two points come from the same set Q_k, then for every $l \neq k$, $\mathtt{disk}_\ell(Q_k \cup Q_l) = \mathtt{disk}_\ell(Q)$. If $m = 2$, we are done. Otherwise, notice that only the largest of the disks $\mathtt{disk}_\ell(Q_i \cup Q_j)$, $i < j$, can be $\mathtt{disk}_\ell(Q)$, as any smaller disk would not enclose $Q_k \cup Q_l$, by definition of $\mathtt{disk}_\ell(Q_k \cup Q_l)$ and Fact 1. ◀

▶ Remark 4. Lemma 3 is interesting in its own right, since it allows us to solve problems such as the following. Let S be a set of n points in the plane. Construct a data structure of near-linear size to support, in $O(\text{polylog}\, n)$ time, queries of the form: Given an axis-aligned rectangle R and a line ℓ, return the smallest enclosing circle of $S \cap R$ with center on ℓ.

2.4.2 Implementation of $\mathtt{feasible}_\ell(Q, r)$

If Q is a single canonical set. Let $s := \mathtt{feasible}_\ell(Q, r)$ be the desired locus of candidate centers $x \in \ell$ such that $\text{radius}_\ell(Q, x) \leq r$; it is a contiguous interval on ℓ, if non-empty, since $\text{radius}_\ell(Q, x)$ is convex; see Fact 1. This interval is empty if $r < \text{radius}_\ell(Q)$, consists of a single point if $r = \text{radius}_\ell(Q)$, and is a positive-length interval with two endpoints if $r > \text{radius}_\ell(Q)$. Since $\text{radius}_\ell(Q, x)$ is convex and monotone along each of the two half-lines into which l is split by $\text{center}_\ell(Q)$, we can binary search along each half-line to determine the endpoints of s. Using the same data structure as in [5] allows us to conduct the binary search in $O(\log n)$ time. We omit the easy details.

If Q is the union of m canonical sets $Q_1, \ldots, Q_m$. Compute $s_i := \mathtt{feasible}_\ell(Q_i, r)$, for $i = 1, \ldots, m$, and return $\bigcap s_i$. Running time is dominated by the cost of computing the m feasible intervals, so it is $O(m \log n)$.

If Q is the union of a dynamic collection of canonical sets. The goal is to maintain the feasibility interval subject to addition and removal of canonical sets. For each set, store its feasible interval and, using two heaps H_L and H_R, maintain the rightmost left endpoint and leftmost right endpoint of the intervals. (i) On insertion of a new set Q', compute $[a_i, b_i] := \mathtt{feasible}_\ell(Q', r)$, add a_i to H_L, and b_i to H_R. (ii) On deletion of Q', remove the endpoints of the stored interval $\mathtt{feasible}_\ell(Q', r)$ from H_L and H_R. (iii) The current feasible interval is always $[\max(H_L), \min(H_R)]$, unless $\max(H_L) > \min(H_R)$, in which case it is empty.

If we use a standard binary heap to implement H_L and H_R, insertion costs $O(\log n + \log m) = O(\log n)$, deletion $O(\log m)$, and current feasible region is available in time $O(1)$, where m is the current number of sets in Q and n is the total number of points involved.

2.4.3 Implementation of $\texttt{fits?}_\ell(Q, r)$

This test is logically equivalent to $\texttt{feasible}_\ell(Q, r) \neq \emptyset$ or to $\text{radius}_\ell(Q) \leq r$. The former is more efficient unless Q is a single canonical set (in which case the latter is more straightforward, but equally efficient asymptotically).

2.4.4 Implementation of $\texttt{prefix}_\ell(S, r)$

This operation can be implemented as a binary search along S, using $\texttt{fits?}(S', r)$ as a blackbox for candidate prefixes S' as the decision procedure, at the cost of a multiplicative logarithmic factor in running time. However, sometimes we can do better, as detailed below.

If S is a canonical subcurve. At any moment during a binary search in the hierarchical decomposition of S (which itself is a canonical subcurve of P), there are $O(\log n)$ canonical sets $S_1, \ldots, S_i$ comprising the current prefix being tested. As binary search progresses down the hierarchy, either a new canonical set S_{i+1} is added to the current prefix (if the prefix is too short), or S_i is replaced by a new, smaller S_i' (if the prefix is too long). We use the dynamic structure for feasible interval maintenance from Section 2.4.2 to keep track of this information and check if the current feasible interval is empty. Since the total number of insertions into and deletions from our collection of canonical sets is $O(\log n)$, the overall cost is $O(\log^2 n)$.

If S is a subcurve of P. Using the hierarchical decomposition of P, we express S as a concatenation $S_1 \cdot \ldots \cdot S_m$ of $m = O(\log n)$ canonical subcurves $S_1, \ldots, S_m$. In $O(m \log n)$ time, we compute the feasible intervals $s_1, ..., s_m \subset \ell$ for all subcurves S_i. Then in time $O(m)$, we compute the largest index j such that $S_1 \cdot \ldots \cdot S_j$ fits into a disk of radius r, but $S_1 \cdot \ldots \cdot S_{j+1}$ does not, by computing the largest $j \leq m$ such that $\bigcap_{1 \leq i \leq j} s_i \neq \emptyset$ (if $j = m$, $\texttt{prefix}_\ell(S, r)$ returns all of S). We then binary search within S_{j+1} to identify the longest prefix of S_{j+1} that can be concatenated to $S_i \cdot \ldots \cdot S_j$ and still fit into a disk of radius r in $O(\log^2 n)$ time, using a slight modification of the above single-canonical-subcurve algorithm. The overall time complexity is $O(m \log n + m + \log^2 n) = O(\log^2 n)$.

3 Discrete Fréchet distance simplification on a line

In this section we present an algorithm for answering queries, which refer to the already preprocessed input curve $P = (p_1, \ldots, p_n)$, of the following type: Given a line ℓ and an integer $k \geq 1$ find an at-most-k-vertex curve Q on ℓ minimizing $d_{\text{dF}}(P, Q)$; that is, find a curve $Q = (q_1, \ldots, q_{k'})$, where $k' \leq k$ and $q_1, \ldots, q_{k'} \in \ell$, that realizes the expression

$$\min_{\substack{k' \in [1,k] \\ q_1, \ldots, q_{k'} \in \ell}} \{d_{\text{dF}}(P, (q_1, \ldots, q_{k'}))\}.$$

We assume that $k < n$, since otherwise the curve $Q = (\overline{p}_1, \ldots, \overline{p}_n)$ clearly achieves the minimum, where $\overline{p}_i$ is the orthogonal projection of p_i on ℓ.

Recall that we write $P[i, j]$, for $1 \leq i \leq j \leq n$, to denote the (contiguous) subcurve $(p_i, \ldots, p_j)$ of P. For a point q in the plane, the distance from q to the vertex of $P[i, j]$ farthest from (nearest to) it, is denoted by $d_{\max}(P[i, j], q)$ ($d_{\min}(P[i, j], q)$). Notice that

$$\min_{q \in \ell}\{d_{\text{dF}}(P[i, j], (q))\} = \min_{q \in \ell}\{d_{\max}(P[i, j], q)\} = \text{radius}_\ell(P[i, j]),$$

where $\text{radius}_\ell(P[i, j])$ is the radius of $\texttt{disk}_\ell(P[i, j])$ (see Section 2.2).

Let $Q = (q_1, \ldots, q_{k'})$ be the desired curve and let W be a walk of P and Q of cost $d_{\mathrm{dF}}(P,Q)$. We may assume that W does not match two consecutive points in Q to the same point in P. That is, W does not contain two consecutive pairs of the form $(p_i, q_j), (p_i, q_{j+1})$, since, if it does, we may delete the pair (p_i, q_{j+1}) from W (and the point q_{j+1} from Q, if it was only matched to p_i) to obtain a legal walk W' (and a curve Q') of the same cost.

This assumption implies that, in order to compute $d_{\mathrm{dF}}(P,Q)$, we need to find an optimal partition of P into k' subcurves. That is,

$$d_{\mathrm{dF}}(P,(q_1,\ldots,q_{k'})) = \min_{1 \le i_1 < i_2 < \cdots < i_{k'-1} < n} \max \left\{ \begin{array}{c} d_{\max}(P[1,i_1], q_1) \\ \vdots \\ d_{\max}(P[i_{k'-2}+1, i_{k'-1}], q_{k'-1}) \\ d_{\max}(P[i_{k'-1}+1, n], q_{k'}) \end{array} \right\}.$$

We conclude that, given ℓ and k, our task is to find $k' \le k$ and a sequence of indices $1 \le i_1 < i_2 < \cdots < i_{k'-1} < n$ that minimize the expression

$$\max\{\mathrm{radius}_\ell(P[1,i_1]), \ldots, \mathrm{radius}_\ell(P[i_{k'-2}+1, i_{k'-1}]), \mathrm{radius}_\ell(P[i_{k'-1}+1, n])\}.$$

The desired sequence is then $Q = (\mathrm{center}_\ell(P[1,i_1]), \ldots, \mathrm{center}_\ell(P[i_{k'-1}+1,n]))$, where $\mathrm{center}_\ell(P[i,j])$ is the center of $\mathtt{disk}_\ell(P[i,j])$.

3.1 The Algorithm

Given the supporting data structures and set of primitive operations, the query algorithm is pretty simple. We first present an algorithm for the decision problem, i.e., given ℓ, k, and r, determine if there exists an at-most-k-vertex curve Q on ℓ such that $d_{\mathrm{dF}}(P,Q) \le r$.

The decision algorithm. The function Decision in Algorithm 1 uses a simple greedy recursive approach for solving the decision problem.

Algorithm 1 The decision algorithm: Given a distance $r > 0$, determine if there exists a sequence of at most k points on line ℓ at discrete Fréchet distance at most r from $P[i,n]$.

```
function DECISION(i, ℓ, k, r)
    if i = n + 1 then
        return TRUE
    if k = 0 then
        return FALSE
    l ← prefix_ℓ(P[i, n], r)
    if l = 0 then                                   ▷ p_i is too far from ℓ
        return FALSE
    return DECISION(i + l, ℓ, k − 1, r)
```

▶ **Lemma 5.** *Let ℓ be a line in the plane, $k \ge 1$ an integer, and $r > 0$. The call* Decision$(1,\ell,k,r)$ *returns* true *if and only if there exist $k' \le k$ points on ℓ, $Q = (q_1, \ldots, q_{k'})$, such that $d_{\mathrm{dF}}(P,Q) \le r$. The running time is k' times the cost of a call to* $\mathtt{prefix}_\ell$ *with a subcurve of P, i.e., $O(k \log^2 n)$.*

Proof. The time bound is obvious. Moreover, it is clear that, if the call Decision$(1,\ell,k,r)$ returns true, then there exist $k' \le k$ such points – namely, the centers of the disks corresponding to the computed prefixes. We now prove the other direction by induction on k.

If there exists a point q_1 on ℓ such that $d_{\mathrm{dF}}(P,(q_1)) \leq r$, then the call $\texttt{prefix}_\ell(P[1,n],r)$ at the top level of the recursion will return n and the algorithm will return TRUE just at the beginning of the next level (since $i = n+1$).

Assume now that for any preprocessed curve P', if there exist fewer than k points on ℓ, $Q' = (q'_1,\dots,q'_{k-1})$, such that $d_{\mathrm{dF}}(P',Q') \leq r$, then the call $\textsc{Decision}(1,\ell,k-1,r)$ (applied to P') returns TRUE. Moreover, assume there exist k points on ℓ, $Q = (q_1,\dots,q_k)$, such that $d_{\mathrm{dF}}(P,Q) \leq r$. We need to show that the call $\textsc{Decision}(1,\ell,k,r)$ returns TRUE. Indeed, consider a walk W of P and Q of cost $d_{\mathrm{dF}}(P,Q)$, and let l' be the largest index such that W matches q_1 to $p_{l'}$. By definition, the index l returned by the call $\texttt{prefix}_\ell(P[1,n],r)$ at the top level of the recursion is at least as large as l'. Now, since the cost of W is $d_{\mathrm{dF}}(P,Q)$ and W matches q_1 to the prefix $P[1,l']$, we know that $d_{\mathrm{dF}}(P[l'+1,n],(q_2,\dots,q_k)) \leq r$ and therefore there exists a sequence Q' of at most $k-1$ points on ℓ, which is $(q_2,\dots,q_k)$ or a suffix of it, such that $d_{\mathrm{dF}}(P[l+1,n],Q') \leq r$. Therefore, by the induction hypothesis, the call $\textsc{Decision}(l+i,\ell,k-1,r)$ at the top level of the recursion (where $i=1$) returns TRUE, and thus $\textsc{Decision}(1,\ell,k,r)$ returns TRUE. ◀

▶ Note. A minor modification of the $\textsc{Decision}$ function, with the same preprocessing of P, can solve the problem of finding, given line ℓ and distance r, the shortest sequence of points Q on ℓ lying within discrete Fréchet distance r of P. (No such set exists if r is smaller than the maximum distance from a point of P to ℓ; this can be efficiently checked with no asymptotic slowdown.) The query can be answered in time $O(k \log^2 n)$, where n is the length of P and k is the length of the shortest such curve Q. Performing the same query on a subcurve $P[i,j]$ of P can be done at the same asymptotic cost.

The optimization algorithm. We now present the query algorithm. That is, given ℓ and k, find an at-most-k-vertex curve Q on ℓ minimizing $d_{\mathrm{dF}}(P,Q)$. The function $\textsc{Optimization}$ in Algorithm 2 actually returns the distance $d_{\mathrm{dF}}(P,Q)$, but it can be easily modified (within the same bounds) to return Q as well.

Algorithm 2 The optimization algorithm: Find the smallest distance r, for which there exists a sequence Q of at most k points on ℓ such that $d_{\mathrm{dF}}(P[i,n],Q) \leq r$.

```
function OPTIMIZATION(i, ℓ, k)
    if k = 0 then return ∞
    Binary search for the smallest index j, i ≤ j ≤ n, such that
        DECISION(i, ℓ, k, radius_ℓ(P[i, j])) returns TRUE
    d_1 ← radius_ℓ(P[i, j])
    if j > i then
        d_2 ← OPTIMIZATION(j, ℓ, k − 1)
    else
        d_2 ← ∞
    return min{d_1, d_2}
```

▶ **Lemma 6.** *Let ℓ be a line in the plane, and let $k \geq 1$. The call $\textsc{Optimization}(i,\ell,k)$ returns the smallest r, for which there exists a sequence Q of at most k points on ℓ such that $d_{\mathrm{dF}}(P,Q) \leq r$. The running time is $O(k^2 \log^3 n + k \log^4 n)$.*

Proof. If $k=1$, then the algorithm returns $\min_{q_1 \in \ell} d_{\mathrm{dF}}(P[i,n],(q_1)) = \mathrm{radius}_\ell(P[i,n])$. Indeed, let j be the smallest index such that $\textsc{Decision}(i,\ell,1,\mathrm{radius}_\ell(P[i,j]))$ returns TRUE, so the algorithm sets $d_1 \leftarrow \mathrm{radius}_\ell(P[i,j]) \leq \mathrm{radius}_\ell(P[i,n])$. But since $k=1$, the latter inequality must be an equality, so $d_1 = \text{radius}_\ell(P[i,n])$. As for d_2, it is set to ∞ either by the recursive call with $k = 0$ (if $j > i$) or by the **else** clause (if $j = i$). Hence, the algorithm returns $d_1 = \min_{q_1 \in \ell} d_{\text{dF}}(P[i,n],(q_1))$ as claimed.

Assume now that $k > 1$ and that the algorithm returns the correct value for $k-1$. Let

$$d := \min_{\substack{k' \in [1,k] \\ q_1,\ldots,q_{k'} \in \ell}} d_{\text{dF}}(P[i,n],(q_1,\ldots,q_{k'})),$$

and let $Q = (q_1,\ldots,q_{k'})$ be a sequence of k' points on ℓ such that $d_{\text{dF}}(P[i,n],Q) = d$. Finally, let W be a minimum-cost walk of P and Q (i.e., a walk of cost d). Let $P[i,j]$ be the shortest prefix for which $\textsc{Decision}(i,\ell,k,\text{radius}_\ell(P[i,j]))$ returns TRUE. Then $d_1 = \text{radius}_\ell(P[i,j])$ and by definition $d_1 \geq d$. Notice that the inductive assumption implies that the value d_2 returned by the recursive call (if such a call is needed) is correct. We consider the cases $j = i$ and $j > i$ separately.

If $j = i$, $\text{radius}_\ell((p_i))$ is feasible. This radius is equal to the distance from p_i to ℓ. Notice that, for each point p of $P[i,n]$, the distance between p and ℓ is a lower bound on $d = \text{radius}_\ell(P[i,n])$, so we must have $d_1 = d$. Moreover, since $j \not> i$, the algorithm sets d_2 to ∞ and therefore returns the correct value.

Assume therefore that $j > i$ and let $d' := \text{radius}_\ell(P[i,j-1])$. We know that the call $\textsc{Decision}(i,\ell,k,\text{radius}_\ell(P[i,j-1]))$ returned FALSE, so $d' < d \leq d_1$. We distinguish between two subcases.

- If $d_1 = d$, we claim that the value d_2 returned by the recursive call must be greater or equal than d_1, and therefore the algorithm returns the correct value, d_1. Indeed, if $d_2 < d_1$, then we found $k' \leq k$ points $q'_1,\ldots,q'_{k'}$, where $q'_1 = \text{center}_\ell(P[i,j-1])$ and $q'_2,\ldots,q'_{k'}$ are the centers induced by the recursive call, such that $d_{\text{dF}}(P[i,n],(q'_1,\ldots,q'_{k'})) \leq \max\{d',d_2\} < d_1$, contradicting the assumption that $d_1 = d = \min_{\substack{k' \in [1,k] \\ q_1,\ldots,q_{k'} \in \ell}} d_{\text{dF}}(P[i,n],(q_1,\ldots,q_{k'}))$.
- If $d_1 > d$, then let $P[i,j']$ be the prefix assigned to q_1 by the minimum-cost walk W. Notice that $j' \leq j-1$, since otherwise d would be greater or equal than d_1. So d (which as we recall is greater than d') is the smallest distance between $P[j'+1,n]$ and a curve on ℓ of length at most $k-1$, while d_2 is the smallest distance of either $P[j'+1,n]$ (if $j' = j-1$) or a proper suffix of $P[j'+1,n]$ (if $j' < j-1$) and such a curve on ℓ. Thus, $d \geq d_2$. But, since $\max\{d',d_2\} = d_2$ is feasible, we conclude that $d = d_2$ and the algorithm returns the correct value (since $d_2 = d < d_1$).

As for the running time, at each level of the recursion we issue $O(\log n)$ calls to the decision algorithm, where each of them is preceded by a call to $\texttt{disk}_\ell(P[i,j])$, with the appropriate index j, to obtain $\text{radius}_\ell(P[i,j])$. Since the running time of the decision algorithm is $O(k\log^2 n)$ and the running time of $\texttt{disk}_\ell(P[i,j])$ is $O(\log^4 n)$, and the number of levels is $O(k)$, we conclude that the total running time is $O(k^2\log^3 n + k\log^5 n)$.

This bound can be slightly improved by splitting the search into two stages: First find the canonical subcurve S_t that contains the point p_j that we are looking for, by adding subcurves one by one. Then, look for p_j within S_t by binary search using the tree of the canonical subcurve S_t. The search consists of $\log n$ steps, but in each step the set of canonical subcurves to which we need to apply $\texttt{disk}_\ell$ either grows by one, or we replace the last subcurve with a shorter one. Hence, $\texttt{disk}_\ell$ does not need to recompute the smallest disk for all $O(m^2)$ pairs of subcurves, but only for the $O(m)$ pairs involving the new curve. This effectively speeds up $\texttt{disk}_\ell$ by a logarithmic factor, resulting in total query time of $O(k^2\log^3 n + k\log^4 n)$. ◀

The following theorem summarizes our main result.

▶ **Theorem 7.** *Let P be a polygonal curve with n vertices in the plane. One can preprocess P in time $O(n \log^2 n)$ into a data structure of size $O(n \log n)$, so that given a line ℓ and a positive integer k, an at-most-k-vertex curve on ℓ minimizing $d_{dF}(P, Q)$ (and the corresponding distance) can be found in $O(k^2 \log^3 n + k \log^4 n)$ time.*

4 Extension to geometric trees

A geometric tree is a tree whose vertices are points in the plane and whose edges are line segments connecting the corresponding points. In this section, we expand the data structure to support preprocessing a geometric tree T so that, given two vertices u, v of the tree, an arbitrary line ℓ, and an integer $k > 0$, one can find an at-most-k-vertex curve Q on ℓ that minimizes the Fréchet distance to the path Π_{uv} from u to v in T. We use ideas similar to those in [3], which in turn use heavy-path decomposition of T [17].

More specifically, the heavy-path decomposition of T is a collection $\{\pi_i\}$ of vertex-disjoint paths in T which collectively cover all the vertices of T. Any path Π_{uv} in T is a concatenation of $O(\log n)$ subpaths of (some of) the paths π_i and, this information can be extracted from the heavy-path decomposition data structure in time $O(\log^2 n)$ [17] (in fact, a faster implementation is possible, but would not help us, as it is not the bottleneck here). We then construct the data structure $T_{\mathrm{FVD}}(\pi_i)$, for each curve π_i, as above. Since the paths π_i are disjoint, the total size of the data structures involved is $O(n \log n)$, where n is the number of vertices in T.

Given a decomposition of Π_{uv} as above, we simply run Algorithm 2 (which, in turn, invokes Algorithm 1), on a concatenation of $l = O(\log n)$ subpaths $P_1, \ldots, P_l$ of preprocessed heavy paths. Decision algorithm requires an implementation of $\texttt{prefix}_\ell$ for such a concatenation. Since we can partition the subpaths further, each into $O(\log n)$ canonical subpaths, we are effectively running the $\texttt{prefix}_\ell$ algorithm for a subpath, as in Section 2.4.4, but with $m = l \cdot O(\log n) = O(\log^2 n)$ canonical subpaths, resulting in running time of $O(\log^3 n)$ for $\texttt{prefix}_\ell$ and $O(k \log^3 n)$ for Decision.

As for Optimization, conceptually we run Algorithm 2 with minor modifications to speed it up. There are at most k levels of recursion. We will describe and analyze one level and multiply the runtime by k.

At a fixed level of recursion, we need to find the shortest prefix of the concatenation of $O(\log n)$ subpaths $P_1, P_2, \ldots, P_l$ that satisfies the Decision condition. We first find j so that $P_1 \cdot \ldots \cdot P_j$ satisfies it, but $P_1 \cdot \ldots \cdot P_{j-1}$ does not (or $j = 1$, in which case we are done), adding one subpath at a time, sequentially. This takes at most l invocations of Decision, for a total of $O(k \log^4 n)$ time. Now we subdivide P_j into a logarithmic number of canonical subcurves and apply the same logic to find the canonical subcurve P' that contains the (still unknown) point p_j we want. This involves $O(\log n)$ more invocations of Decision, for a total of $O(k \log^4 n)$ time.

However, we did not account for the work of the radius_ℓ algorithm. It is easily checked that during these first two steps, over all the calls to radius_ℓ, it is enough to compute the minimum enclosing radius with center on ℓ for at most $\binom{m}{2} = O(\log^4 n)$ pairs of canonical subpaths constituting the subpaths $P_1, \ldots, P_l$, at a cost of $O(\log^2 n)$ per pair; so the total cost of all the radius_ℓ runs at the current level of recursion up to now is at most $O(\log^6 n)$.

Finally, we use the hierarchical decomposition of P' to find desired point p_j, essentially as in the proof of Theorem 7. We observe that descending the hierarchy in binary search, we always either add the last canonical subpath or replace the last canonical subpath, so the number of pairs of subpaths that need to be recomputed by the radius_ℓ algorithm on each descent step is only $O(m) = O(\log^2 n)$ (rather than $O(m^2)$) and therefore each radius_ℓ call takes $O(m \log^2 n) = O(\log^4 n)$ time, for a total of $O(\log^5 n)$ over the entire binary search.

To summarize, each level of recursion in this implementation of Optimization takes $O(k\log^4 n + \log^6 n)$ time. We conclude that that the full running time of Optimization is $O(k^2\log^4 n + k\log^6 n)$, which finishes the proof of the following theorem:

▶ **Theorem 8.** *Let T be a geometric tree on n vertices. One can preprocess T in time $O(n\log^2 n)$ into a data structure of size $O(n\log n)$, so that given vertices u and v in T, a line ℓ, and a positive integer k, the at-most-k-vertex curve Q on ℓ minimizing the discrete Fréchet distance between Q and the path Π_{uv} between u and v in T, can be found in $O(k^2\log^4 n + k\log^6 n)$ time.*

5 Extension to polygonal regions in the plane

Let R be a (topologically closed) polygonal region in the plane bounded by m edges and P be an n-vertex curve. Given an integer $k > 0$, we show how to find an at-most-k-vertex curve $Q \subseteq R$ that minimizes the discrete Fréchet distance $d_{dF}(P, Q)$. That is, we find a curve $Q = (q_1, \ldots, q_{k'})$, with $k' \leq k$ and $q_1, \ldots, q_{k'} \in R$, that realizes the expression

$$\min_{\substack{k' \in [1,k] \\ q_1,\ldots,q_{k'} \in R}} \{d_{\mathrm{dF}}(P, (q_1, \ldots, q_{k'}))\}.$$

Note that Q need not be unique.

Observe that we only need to implement one new primitive, which we refer to as $\texttt{disk}_R(S)$, that computes the smallest-radius disk centered at a point of R for a subcurve S of P (and its corresponding radius and center). Using $\texttt{disk}_R(S)$ one can perform binary search over P to implement $\texttt{prefix}_R(S, r)$ at the cost of a logarithmic factor in the running time. The decision and optimization functions (Algorithms 1 and 2) will use the new primitive and otherwise remain the same.

We now sketch how to implement $\texttt{disk}_R(S)$ for a subcurve $S = P[i, j]$. Let c be the center of the mec containing the subcurve $P[i, j]$ and centered at a point of R.

First we find the center of the unconstrained mec of $P[i, j]$, using the data structure in [9] for the mec of a planar point set. We partition $P[i, j]$ into $t = O(\log n)$ canonical subcurves (as in Section 2.3) and preprocess each subcurve as in [9]. Using the algorithm from Section 3 of [9], we can compute the mec of the union of the subcurves, that is, of $P[i, j]$ in deterministic time $O(t^3\log^3 n) = O(\log^6 n)$ (refer to the discussion after Lemma 2 in that paper). Alternatively, Theorem 2 in [9] provides an algorithm for finding the mec in expected time $O(t\log t\log n + \sqrt{t}\log t\log^3 n) = O(\log^{3.5} n\log\log n)$.[3] If the center of the resulting mec lies in R, we are done: we found $\texttt{disk}_R(S)$.

Otherwise, since $d_{\max}(P[i, j], x)$ is a convex function of x, the center c lies on the boundary of R. For each bounding segment s of R, we compute $\texttt{disk}_\ell(P[i, j])$ for the supporting line ℓ of s in time $O(t^2\log^2 n) = O(\log^4 n)$; refer to Lemma 3. Once again, if the center $c(s) := \mathrm{center}_\ell(P[i, j])$ lies in s, this gives a candidate mec, otherwise the endpoint of s closest to $c(s)$ provides such a candidate. After repeating the process for each boundary edge of R, we return (the center of) the smallest disk found. The total cost is thus $O(m\log^4 n)$.

To summarize, $\texttt{disk}_R(P[i, j])$ can be computed in deterministic time $O(\log^6 n + m\log^4 n)$ or in expected time $O(\log^{3.5} n\log\log n + m\log^4 n) = O(m\log^4 n)$, assuming $m \neq 0$.

[3] The statements are phrased in terms of solving an LP over an intersection of polyhedra in $\mathbb{R}^3$, but a later discussion points out that the same logic, with minor modifications, applies to finding the mec of a discrete point set in the plane as well [9].

We now implement $\texttt{prefix}_R$ by binary search over the prefix length using $\texttt{disk}_R(S)$ in expected time $O(m \log^5 n)$. Expected running time of DECISION will be $O(km \log^5 n)$ and that of OPTIMIZATION will be $O(k^2 m \log^6 n)$ (see the time analysis of Lemma 6 for details).

We did not optimize the logarithmic factors.

▶ **Theorem 9.** *Let P be a polygonal curve with n vertices in the plane. One can preprocess P in time $O(n \log^2 n)$ into a data structure of size $O(n \log n)$, so that given a polygonal domain R bounded by m segments, an at-most-k-vertex curve $Q \subseteq R$ that minimizes the discrete Fréchet distance $d_{dF}(P, Q)$ and the corresponding distance can be found in expected time $O(k^2 m \log^6 n)$. Alternatively, it can be found in deterministic time $O(k^2 m \log^6 n + k^2 \log^8 n)$.*

6 Discrete Fréchet distance simplification on a g-flat

In this section we address the general version of the dimension-reduction problem mentioned in the introduction. That is, P is a polygonal curve in $\mathbb{R}^d$, for some constant $d > 2$, and we need to preprocess P into a compact data structure to efficiently support queries that specify a g-flat h ($1 \le g \le d-1$) and an integer $k \ge 1$, and request a curve Q on h of length at most k that minimizes the discrete Fréchet distance to P.

It is unlikely that there exists a solution to this problem with bounds similar to those that we obtained in Section 3, since it seems that, on the one hand, a primitive analogous to $\texttt{disk}_\ell(Q)$, where "disk" is replaced by "ball" and ℓ is replaced by h, is essential, but on the other hand, it is probably impossible to implement such a primitive efficiently in higher dimensions, i.e., in polylogarithmic time after near-linear time preprocessing of Q.

Therefore, since we are interested in a solution to the general dimension-reduction problem with near-linear time preprocessing and polylogarithmic time query, we resort to approximate queries. Specifically, we present such a solution (for a prespecified parameter $0 < \varepsilon_0 < 1$) that given h returns a curve Q on h of length at most k that minimizes the discrete Fréchet distance to P up to a factor of $1 + \varepsilon_0$ (i.e., $d_{\mathrm{dF}}(P, Q) \le (1 + \varepsilon_0) d_{\mathrm{dF}}(P, Q^*)$, where Q^* is such a curve that minimizes the discrete Fréchet distance to P).

Set $\varepsilon := \frac{\varepsilon_0}{1+\varepsilon_0}$; then, $0 \le \varepsilon \le 1/2$ and $\frac{1}{1-\varepsilon} = 1 + \varepsilon_0$. We first define the primitive $\texttt{ball}_h^\varepsilon(Q)$, which returns a $(1-\varepsilon)$-approximation of the smallest ball centered at a point of h and containing a set Q of points. That is, let $\mathrm{radius}_h^\varepsilon(Q)$ denote the radius of the returned ball, then $(1-\varepsilon)\,\mathrm{radius}_h(Q) \le \mathrm{radius}_h^\varepsilon(Q) \le \mathrm{radius}_h(Q)$, where $\mathrm{radius}_h(Q)$ is the radius of the smallest ball with center on h.

We use *coresets* to implement $\texttt{ball}_h^\varepsilon(Q)$, assuming Q is the set of points corresponding to a subsequence of P. More precisely, we preprocess P so that given a subsequence $P[i, j]$, for $1 \le i \le j \le n$, one can compute $\texttt{ball}_h^\varepsilon(P[i, j])$ in $O(c_\varepsilon \log n)$ time, where $c_\varepsilon = c_\varepsilon(\varepsilon)$ is a constant, see below. Before presenting the details, we provide a brief overview of the coreset-related definitions and results that we use.

Coresets. Let S be a compact set in $\mathbb{R}^d$. Let $\theta \in \mathbb{S}^{d-1}$ be a direction. The *directional width* $\mathrm{width}(S, \theta)$ of S in direction θ is the minimum distance between a pair of hyperplanes with normal θ containing S between them. Fix a number ε, $0 < \varepsilon < 1$. A subset C of S is an *ε-coreset* for S *with respect to directional width*, if it has the property that $\mathrm{width}(C, \theta) \ge (1-\varepsilon)\,\mathrm{width}(S, \theta)$, for all directions θ; since $C \subset S$, $\mathrm{width}(C, \theta) \le \mathrm{width}(S, \theta)$. We say that a subset C of S is an *ε-coreset* for S for *farthest neighbors* if, for any point q, $d_{\max}(C, q) \ge (1-\varepsilon) d_{\max}(S, q)$. It follows that if B is any ball containing C, then the ball $\frac{B}{1-\varepsilon}$, obtained by scaling B around its center by a factor of $1/(1-\varepsilon)$, contains S.

The following facts are known (refer for example to [14, Section 23.1]):

- A compact set in $\mathbb{R}^d$ has an ε-coreset for directional width of size $c_\varepsilon \coloneqq O(1/\varepsilon^{(d-1)/2})$.
- Such an ε-coreset for a set of n points can be constructed in time $O(n + c_\varepsilon^3)$.
- An $(\varepsilon/4)$-coreset for directional width is an ε-coreset for farthest neighbors.
- If $C_1, C_2, \ldots, C_t$ are ε-coresets for $S_1, S_2, \ldots, S_t$ for width or farthest neighbors, then $\bigcup C_i$ is an ε-coreset for $\bigcup S_i$ for width or farthest neighbors.

As we are interested in coresets for farthest neighbors, with a slight abuse of terminology, we will use unqualified "coreset" to denote them hereafter.

Smallest ball with a center on a flat. We recall the classical randomized LP-type algorithm of Welzl [20] for finding the smallest enclosing ball of an n-point set S in $\mathbb{R}^d$ in $O_d(n)$ expected time (with the implied constant depending on d). An inspection of the algorithm reveals that it can be used essentially unmodified to compute the smallest enclosing ball of S whose center is constrained to lie in a given g-flat; such a ball will be defined by at most $g+1$ points.

Implementation of $\texttt{ball}_h^\varepsilon(P[i,j])$. We now explain how to implement $\texttt{ball}_h^\varepsilon(P[i,j])$, which is the basic building block of the approximation algorithm described above. As in Section 2.3 we once again construct a hierarchical decomposition of P, storing P of length n, then $P[1, n/2]$ and $P[n/2+1, n]$ as its children and so forth, thereby creating a hierarchy of canonical subcurves. With each canonical subcurve $P[i,j]$ we associate its coreset C_{ij} for farthest neighbors, of size at most c_ε. An arbitrary subcurve P' of P is the union of $O(\log n)$ canonical subcurves. If we collect the points of the corresponding coresets, we obtain a coreset C' for P' of size $c_\varepsilon \log n$. Running the exact minimum-enclosing-ball with center on a g-flat algorithm on the set C', by the above discussion, gives an ε-approximation of corresponding ball for P'. Indeed, we obtain the smallest possible ball B with the center constrained to the flat and enclosing C'. By the coreset property $\frac{B}{1-\varepsilon}$ contains P'. There cannot be a substantially smaller ball with center on the flat and containing P', as such a ball would also contain C' and would contradict minimality of B.

The expected running time is dominated by that of the ball-finding routine, which is $O_d(c_\varepsilon \log n) = O_d(\log n/\varepsilon^{(d-1)/2})$, concluding our description of an implementation of $\texttt{ball}_h^\varepsilon$.

The decision algorithm. We now wish to devise a decision algorithm, similar to the one in Section 3. Given $r > 0$, our decision algorithm will return TRUE if there exists a sequence Q of at most k points on h such that $d_{\mathrm{dF}}(P,Q) \le \frac{r}{1-\varepsilon}$. It will return FALSE otherwise, in which case we may only conclude that for any sequence Q of at most k points on h, $d_{\mathrm{dF}}(P,Q) > r$.

For this purpose, we define the primitive $\texttt{prefix}_h^\epsilon(S,r)$ for a sequence S of points and a radius r. It returns a prefix $S[1,k]$ of S, such that $\texttt{ball}_h^\varepsilon(S[1,k])$ returns a radius that is at most r where as $\texttt{ball}_h^\varepsilon(S[1,k+1])$ returns a radius that is greater than r. In practice, $\texttt{prefix}_h^\varepsilon(S,r)$ returns the length of the prefix, so 0 if it is empty. Thus, if $\texttt{prefix}_h^\varepsilon(S,r) = k$, $0 < k < n$, then $\mathrm{radius}_h(S[1,k]) \le \frac{r}{1-\varepsilon}$ and $\mathrm{radius}_h(S[1,k+1]) > r$. (If $\texttt{prefix}_h^\varepsilon(S,r) = 0$, then $\mathrm{radius}_h(S[1,1]) > r$, and if $\texttt{prefix}_h^\varepsilon(S,r) = n$, then $\mathrm{radius}_h(S[1,n]) \le \frac{r}{1-\varepsilon}$.)

▶ **Observation 10.** *If $\texttt{prefix}_h(S,r)$ returns the largest k' for which $\mathrm{radius}_h(S[1,k']) \le r$, then $\texttt{prefix}_h(S,r) \le \texttt{prefix}_h^\varepsilon(S,r) \le \texttt{prefix}_h(S, \frac{r}{1-\varepsilon})$.*

To implement $\texttt{prefix}_h^\epsilon(S,r)$ we perform a binary search. That is, we set $i = \lfloor \frac{|S|}{2} \rfloor$ and call $\texttt{ball}_h^\varepsilon(S[1,i])$. Now, if $\mathrm{radius}_h^\varepsilon(S[1,i]) \le r$, then we increase i, and otherwise we decrease i, etc. Thus, the cost of a call to $\texttt{prefix}_h^\epsilon$ with a subcurve of P is $O(c_\varepsilon \log^2 n)$ time.

We now present the decision algorithm (see Algorithm 3).

Algorithm 3 The decision algorithm: Given a distance $r > 0$, determine if there exists a sequence of at most k points on h at discrete Fréchet distance at most $\frac{r}{1-\varepsilon}$ from $P[i, n]$.

```
function ApproxDecision(i, h, k, r)
    if i = n + 1 then
        return TRUE
    if k = 0 then
        return FALSE
    l ← prefix_h^ε(P[i, n], r)
    if l = 0 then                                ▷ p_i is too far from h
        return FALSE
    return ApproxDecision(i + l, h, k − 1, r)
```

The following lemma follows from Observation 10.

▶ **Lemma 11.** *Let h be a g-dimensional flat in $\mathbb{R}^d$, $k \geq 1$ an integer, and $r > 0$. If the call* AprоxDecision$(1, h, k, r)$ *returns* TRUE*, then there exists a sequence Q of $k' \leq k$ points on h, such that $d_{\mathrm{dF}}(P, Q) \leq \dfrac{r}{1-\varepsilon}$. If it returns* FALSE*, then for any sequence Q of $k' \leq k$ points on h, $d_{\mathrm{dF}}(P, Q) > r$. The running time is k' times the cost of a call to* $\texttt{prefix}_h^{\varepsilon}$ *with a subcurve of P.*

Finally, we run the optimization algorithm in Section 3, using the decision algorithm above (and replacing radius_ℓ by $\text{radius}_h^{\varepsilon}$). The following theorem summarizes the section's main result.

▶ **Theorem 12.** *Let P be a polygonal curve with n vertices in $\mathbb{R}^d$. Let $0 < \varepsilon_0 < 1$, and set $\varepsilon = \frac{\varepsilon_0}{1+\varepsilon_0}$. One can preprocess P in time $O(n \log n + \frac{n}{\varepsilon^{3(d-1)/2}})$ into a data structure of size $O(n \log n + \frac{n}{\varepsilon^{(d-1)/2}})$, so that given a g-dimensional flat h and a positive integer k, an at-most-k-vertex curve Q on h such that $d_{\mathrm{dF}}(P, Q) \leq \frac{d_{\mathrm{dF}}(P,Q^*)}{1-\varepsilon} = (1+\varepsilon_0) d_{\mathrm{dF}}(P, Q^*)$ (and the corresponding distance) can be found in $O(\frac{k^2 \log^2 n}{\varepsilon^{(d-1)/2}})$ time, where Q^* is such a curve that minimizes the discrete Fréchet distance to P.*

7 Discussion and open problems

We mention several open problems:

- We did not strive to optimize the query time. How far can it be reduced, without substantially increasing the space and preprocessing time, in Theorems 7, 8, 9, and 12?
- Can the factor k^2 in the query time in these theorems be made linear in k? Or at least subquadratic?

References

1 Pankaj K Agarwal. Range searching. In *Handbook of Discrete and Computational Geometry*, pages 1057–1092. Chapman and Hall/CRC, 2017.

2 Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3-4):203–219, 2005. doi:10.1007/S00453-005-1165-Y.

3 Boris Aronov, Tsuri Farhana, Matthew J. Katz, and Indu Ramesh. Discrete Fréchet distance oracles. In Wolfgang Mulzer and Jeff M. Phillips, editors, *40th International Symposium on Computational Geometry (SoCG 2024)*, pages 10:1–10:14, 2024. `doi:10.4230/LIPIcs.SoCG.2024.10`.

4 S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *LATIN 2008: Theoretical Informatics, 8th Latin American Symposium, Búzios, Brazil, April 7-11, 2008, Proceedings*, volume 4957 of *Lecture Notes in Computer Science*, pages 630–641. Springer, 2008. `doi:10.1007/978-3-540-78773-0_54`.

5 Prosenjit Bose, Stefan Langerman, and Sasanka Roy. Smallest enclosing circle centered on a query line segment. In *Proceedings of the 20th Annual Canadian Conference on Computational Geometry, Montréal, Canada, August 13-15, 2008*, 2008.

6 Peter Brass, Christian Knauer, Chan-Su Shin, Michiel H. M. Smid, and Ivo Vigan. Range-aggregate queries for geometric extent problems. In Anthony Wirth, editor, *Nineteenth Computing: The Australasian Theory Symposium, CATS 2013, Adelaide, Australia, February 2013*, volume 141 of *CRPIT*, pages 3–10. Australian Computer Society, 2013. URL: `http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV141Brass.html`.

7 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. *J. Comput. Geom.*, 11(2):94–130, 2020. `doi:10.20382/JOCG.V11I2A5`.

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

9 David Eppstein. Dynamic three-dimensional linear programming. *ORSA Journal on Computing*, 4(4):360–368, 1992. `doi:10.1287/IJOC.4.4.360`.

10 Chenglin Fan, Omrit Filtser, Matthew J. Katz, Tim Wylie, and Binhai Zhu. On the chain pair simplification problem. In Frank Dehne, Jörg-Rüdiger Sack, and Ulrike Stege, editors, *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, volume 9214 of *Lecture Notes in Computer Science*, pages 351–362. Springer, 2015. `doi:10.1007/978-3-319-21840-3_29`.

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

12 Leonidas J. Guibas, John Hershberger, Joseph S. B. Mitchell, and Jack Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geom. Appl.*, 3(4):383–415, 1993. `doi:10.1142/S0218195993000257`.

13 Prosenjit Gupta, Ravi Janardan, Yokesh Kumar, and Michiel H. M. Smid. Data structures for range-aggregate extent queries. *Comput. Geom.*, 47(2):329–347, 2014. `doi:10.1016/J.COMGEO.2009.08.001`.

14 Sariel Har-Peled. *Geometric approximation algorithms.* Number 173 in Mathematical Surveys and Monographs. American Mathematical Soc., 2011.

15 Arindam Karmakar, Sasanka Roy, and Sandip Das. Fast computation of smallest enclosing circle with center on a query line segment. *Inf. Process. Lett.*, 108(6):343–346, 2008. `doi:10.1016/J.IPL.2008.07.002`.

16 Sasanka Roy, Arindam Karmakar, Sandip Das, and Subhas C. Nandy. Constrained minimum enclosing circle with center on a query line segment. *Comput. Geom.*, 42(6-7):632–638, 2009. `doi:10.1016/J.COMGEO.2009.01.002`.

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

18 Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global curve simplification. In Michael A. Bender, Ola Svensson, and Grzegorz Herman, editors, *27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany*, volume 144 of *LIPIcs*, pages 67:1–67:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPICS.ESA.2019.67`.

19 Marc J. van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *J. Comput. Geom.*, 11(1):1–25, 2020. doi:10.20382/JOCG.V11I1A1.

20 Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In Hermann A. Maurer, editor, *New Results and New Trends in Computer Science, Graz, Austria, June 20-21, 1991, Proceedings [on occasion of H. Maurer's 50th birthday]*, volume 555 of *Lecture Notes in Computer Science*, pages 359–370. Springer, 1991. doi:10.1007/BFB0038202.