Static and Streaming Data Structures for Fréchet Distance Queries

ARNOLD FILTSER, Bar-Ilan University, Israel
OMRIT FILTSER, The Open University of Israel, Israel

Given a curve P with points in $\mathbb{R}^d$ in a streaming fashion, and parameters $\varepsilon > 0$ and k, we construct a distance oracle that uses $O(\frac{1}{\varepsilon})^{kd} \log \varepsilon^{-1}$ space, and given a query curve Q with k points in $\mathbb{R}^d$ returns in $\tilde{O}(kd)$ time a $1 + \varepsilon$ approximation of the discrete Fréchet distance between Q and P.

In addition, we construct simplifications in the streaming model, oracle for distance queries to a sub-curve (in the static setting), and introduce the zoom-in problem. Our algorithms work in any dimension d, and therefore we generalize some useful tools and algorithms for curves under the discrete Fréchet distance to work efficiently in high dimensions.

CCS Concepts: • **Theory of computation** → **Computational geometry**; *Streaming, sublinear and near linear time algorithms*; *Data structures design and analysis;*

Additional Key Words and Phrases: Fréchet distance, distance oracle, streaming algorithm, simplification, high dimension, the “zoom in” problem

ACM Reference format:
Arnold Filtser and Omrit Filtser. 2023. Static and Streaming Data Structures for Fréchet Distance Queries. *ACM Trans. Algor.* 19, 4, Article 39 (October 2023), 36 pages.
https://doi.org/10.1145/3610227

1 INTRODUCTION

Measuring the similarity of two curves or trajectories is an important task that arises in various applications. The Fréchet distance and its variants became quite popular in past few decades and were widely investigated in the literature. Algorithms for various tasks regarding curves under the Fréchet distance were implemented, and some were successfully applied to real datasets in applications of computational biology [36, 44], coastline matching [39], analysis of a football match [32], and more (see also GIS Cup SIGSPATIAL’17 [43]).

The Fréchet distance between two curves P and Q is often described by the man-dog analogy, in which a man is walking along P, holding a leash connected to its dog who walks along Q, and the goal is to minimize the length of the leash that allows them to fully traverse their curves without backtracking. In the discrete Fréchet distance, only distances between vertices are taken into consideration. Eiter and Mannila [27] presented an $O(nm)$-time simple dynamic programming algorithm to compute the discrete Fréchet distance of two curves P and Q with n and m vertices. A polylog improvement exists (see [3]); however, there is a sequence of papers [9, 11, 13] showing

Authors’ addresses: A. Filtser, Bar-Ilan university, Ramat Gan, 5290002, Israel, The department of computer science, building 503, room 307, Israel; e-mail: arnold.filtser@biu.ac.il; O. Filtser, The Open University of Israel, Derekh ha-Universita 1, Ra’anana, 4353701, Israel, Department of Mathematics and Computer Science; e-mail: omrit.filtser@gmail.com.

1549-6325/2023/10-ART39 $15.00
https://doi.org/10.1145/3610227

that under SETH, there are no strongly subquadratic algorithms for both continuous and discrete versions, even if the solution may be approximated up to a factor of 3.

In applications where there is a need to compute the distance to a single curve many times, or when the input curve is extremely large and quadratic running time is infeasible, a natural solution is to construct a data structure that allows fast distance queries. In this article, we are mainly interested in the following problem under the discrete Fréchet distance. Given $P \in \mathbb{R}^{d\times m}$ (a d-dimensional polygonal curve of length m), preprocess it into a data structure that given a query curve $Q \in \mathbb{R}^{d\times k}$ quickly returns a $(1+\varepsilon)$-approximation of $d_{dF}(P,Q)$, where d_{dF} is the discrete Fréchet distance. Such a data structure is called $(1+\varepsilon)$-*distance oracle for* P.

Recently, Driemel et al. [24] showed how to construct a $(1+\varepsilon)$-distance oracle under the discrete Fréchet distance, with query time that does not depend on m, the size of the input curve. Their data structure uses $k^k \cdot O(\frac{1}{\varepsilon})^{kd} \cdot \log^k \frac{1}{\varepsilon}$ space and has $O(k^2 d + \log \frac{1}{\varepsilon})$ query time.[1] They also consider the streaming scenario, where the curve is given as a stream and its length is not known in advance. Their streaming algorithm can answer queries at any point in the stream in $O(k^4 d \cdot \log^2 \frac{m}{\varepsilon})$ time, and it uses $\log^2 m \cdot k^k \cdot O(\frac{\log m}{\varepsilon})^{kd} \cdot \log^k(\frac{\log m}{\varepsilon})$ space. In addition, Driemel et al. [24] provide a one-pass algorithm that constructs a $(1+\varepsilon)$-distance oracle in $O(m^2 \cdot k^{k+3} k\varepsilon^{-dk}(\log \frac{1}{\varepsilon})^k)$ time, where the space required for the resulted distance oracle is $O(k^k \varepsilon^{-dk}(\log \frac{1}{\varepsilon})^k)$, and its query time is $O(k^2 + \log \frac{1}{\varepsilon})$. Their techniques in the streaming case include a merge-and-reduce framework, which leads to the high query time.

To achieve a query time that does not depend on m (in the static case), Driemel et al. [24] first compute an (approximation of) optimal k-simplification of the input curve P. An optimal k-simplification of a curve P is a curve Π of length at most k that minimizes $d_{dF}(P,\Pi)$ over all other curves of length at most k. Note that as the triangle inequality applies for d_{dF}, a trivial 3-distance oracle is just computing an optimal k-simplification Π of P, and for a query Q returning $d_{dF}(P,\Pi) + d_{dF}(\Pi,Q)$ (see Observation 2.1). Specifically, Driemel et al. [24] present a streaming algorithm that maintains an 8-approximation for an optimal k-simplification of the input curve and uses $O(kd)$ space. Abam et al. [1] show a streaming algorithm that maintains a simplification under the continuous Fréchet distance. Their algorithm maintains a $2k$-simplification that is $(4\sqrt{2}+\varepsilon)$-approximation compared to an optimal k-simplification, using $O(k\varepsilon^{-0.5}\log^2 \frac{1}{\varepsilon})$ space. In the static scenario, Bereg et. al. [8] show how to compute an optimal k-simplification of a curve $P \in \mathbb{R}^{3\times m}$ in $O(mk \log m \log(m/k))$ time.

For the (continuous) Fréchet distance, Driemel and Har-Peled [21] presented a $(1+\varepsilon)$-distance oracle for the special case of $k=2$ (queries are segments). Their data structure uses $O(\frac{1}{\varepsilon})^{2d} \cdot \log^2 \frac{1}{\varepsilon}$ space and has $O(d)$ query time. In addition, they show how to use the preceding data structure to construct a distance oracle for segment queries to a subcurve (again only for queries of length $k=2$). This data structure uses $m \cdot O(\frac{1}{\varepsilon})^{2d} \cdot \log^2 \frac{1}{\varepsilon}$ space and can answer $(1+\varepsilon)$-approximated distance queries to any subcurve of P in $O(\varepsilon^{-2} \log m \log\log m)$ time. In previous work [30], the second author showed how to apply their techniques to the discrete Fréchet distance and achieve the same space bound with $O(\log m)$ query time. For general k, Driemel and Har-Peled [21] provided a constant factor distance oracle that uses $O(md \log m)$ space, it and can answer distance queries between any subcurve of P and query Q of length k in $O(k^2 d \log m \log(k \log m))$ time.

For the special case where the queries are horizontal segments, De Berg et al. [20] constructed a data structure that uses $O(m^2)$ space and can answer exact distance queries (under the continuous Fréchet distance) in $O(\log^2 m)$ time. Buchin et al. [14] extended their analysis and proved that their space is bounded by $O(n^{3/2})$.

[1]Driemel et al. [24] also considered the more general case where the curves are from a metric space with bounded doubling dimension. Here, we present only their results for Euclidean space.

The best known approximation algorithm for the discrete Fréchet distance between two curves $P, Q \in \mathbb{R}^{d\times m}$ is an f-approximation that runs in $O(m \log m + m^2/f^2)$ time for constant d, presented by Chan and Rahmati [17] (improving over the work of Bringmann and Mulzer [11]). The situation is better when considering restricted (realistic) families of curves such as c-packed, κ-bounded, and backbone curves, for which there exist small factor approximation algorithms in near linear time (e.g., see [5, 22, 31]).

Other related problems include the approximate nearest neighbor problem for curves, where the input is a set of curves that needs to be preprocessed to answer (approximated) nearest neighbor queries (see [6, 10, 23–25, 28, 29, 35]), and range searching for curves, where the input is a set of curves and the query algorithm has to return all curves that are within some given distance from the query curve (see [2, 7, 12, 18, 19, 26, 29]). We refer to previous work [29] for a more detailed survey of these problems.

Our Results. We consider distance oracles under the discrete Fréchet distance in both the static and streaming scenarios. Table 1 provides a summary of new and previous results.

In the static case, given an input curve $P \in \mathbb{R}^{d\times m}$, we construct a $(1 + \varepsilon)$-distance oracle with $O(\frac{1}{\varepsilon})^{kd} \cdot \log \frac{1}{\varepsilon}$ storage space and $\tilde{O}(kd)$ query time (Theorem 1). Notice that our bounds in both storage space and query time do not depend on m, and they are significantly smaller than the bounds of Driemel et al. [24]. Interestingly, for the streaming setting, we manage to achieve the exact same bounds as for the static case (Theorem 3), thus providing a polynomial improvement (from k^4 to k) in the query time compared to Driemel et al. [24].

As in the work of Driemel et al. [24], we use simplifications to get bounds that do not depend on m. Therefore, in the static case, we present an algorithm that computes in $\tilde{O}(\frac{md}{\varepsilon^{4.5}})$ time a $(1 + \varepsilon)$-approximation for an optimal k-simplification of a curve $P \in \mathbb{R}^{d\times m}$ (Theorem 7). Note that the algorithm of Bereg et al. [8] returns an optimal k simplification; however, it works only for constant dimension d and has quadratic running time for the case $k = \Omega(m)$. For the streaming setting, we present a streaming algorithm that uses $O(\varepsilon)^{-\frac{d+1}{2}} \log^2 \frac{1}{\varepsilon} + O(kd \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ space and computes a $(1+\varepsilon)$-approximation for an optimal k-simplification of the input curve (Corollary 5.7). In addition, we present a streaming algorithm that uses $O(kd \cdot \frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ space and computes a $(1.22 + \varepsilon)$-approximation for an optimal k-simplification of the input curve (Corollary 5.5).

We also consider the problem of distance queries to a subcurve, as in other works [21, 30]. Here, given a curve $P \in \mathbb{R}^{d\times m}$ (in the static setting), we construct a data structure that uses $m \log m \cdot O(\frac{1}{\varepsilon})^{kd} \cdot \log \frac{1}{\varepsilon}$ space, and given a query curve $Q \in \mathbb{R}^{d\times k}$ and two indexes $1 \leq i \leq j \leq m$ returns in $\tilde{O}(k^2 d)$ time a $(1 + \varepsilon)$-approximation of $d_{dF}(P[i, j], Q)$, where $P[i, j]$ is the subcurve of P from index i to j (Theorem 5). Notice that in this problem, the space bound must be $\Omega(m)$, as given such a data structure, one can (essentially) recover the curve P.

Related to both the subcurve distance oracle and simplifications, we present a new problem called the *zoom-in problem*. In this problem, given a curve $P \in \mathbb{R}^{d\times m}$ and a parameter $k < m$, our goal is to construct a data structure that given two indexes $1 \leq i < j \leq m$ returns an (approximation of) optimal k-simplification for $P[i, j]$. This problem is motivated by applications that require visualization of a large curve without displaying all of its details; in addition, it enables "zoom-in" operations, where only a specific part of the curve needs to be displayed. For example, if the curve represents the historical prices of a stock, one might wish to examine the rates during a specific period of time. In such cases, a new simplification needs to be calculated. We present a data structure with $O(mkd \log \frac{m}{k})$ space such that given a pair of indices $1 \leq i < j \leq m$ returns in $O(kd)$ time a $2k$-simplification that is a $(1 + \varepsilon)$-approximation compared to an optimal k simplification of $P[i, j]$.

Finally, our algorithms work and are analyzed for any dimension d. Unfortunately, many tools and algorithms developed for curves under the discrete Fréchet distance considered only constant

Table 1. Old and New Results Under the Discrete Fréchet Distance

	Space	Time	Comments
Static $(1+\varepsilon)$-distance oracle	$O(kd)$	$O(k^2 d)$	$(3+\varepsilon)$-approximation, Observation 2.1
	$O(\frac{1}{\varepsilon})^{2d} \cdot \log^2 \frac{1}{\varepsilon}$	$O(d)$	$k=2$, continuous [21]
	$k^k \cdot O(\frac{1}{\varepsilon})^{kd} \cdot \log^k \frac{1}{\varepsilon}$	$O(k^2 d + \log \frac{1}{\varepsilon})$	[24]
	$O(\frac{1}{\varepsilon})^{kd} \cdot \log \frac{1}{\varepsilon}$	$\tilde{O}(kd)$	Theorem 1
Streaming $(1+\varepsilon)$-distance oracle	$\log^2 m \cdot k^k \cdot O(\frac{\log m}{\varepsilon})^{kd} \cdot \log^k(\frac{\log m}{\varepsilon})$	$O(k^4 d \cdot \log^2 \frac{m}{\varepsilon})$	[24]
	$O(\frac{1}{\varepsilon})^{kd} \cdot \log \frac{1}{\varepsilon}$	$\tilde{O}(kd)$	Theorem 3
$(1+\varepsilon)$-distance oracle with subcurve queries	$m \cdot O(\frac{1}{\varepsilon})^{2d} \cdot \log^2 \frac{1}{\varepsilon}$	$O(\frac{\log m \log\log m}{\varepsilon^2})$	$k=2$, continuous [21]
	$m \cdot O(\frac{1}{\varepsilon})^{2d} \cdot \log^2 \frac{1}{\varepsilon}$	$O(\log m)$	$k=2$ [30]
	$m \log m \cdot O(\frac{1}{\varepsilon})^{kd} \cdot \log \frac{1}{\varepsilon}$	$\tilde{O}(k^2 d)$ [2]	Theorem 5

	Space	Approx.	Comments
Simplification in streaming	$O(kd \cdot \varepsilon^{-0.5} \log^2 \frac{1}{\varepsilon})$	$4\sqrt{2}+\varepsilon$	$2k$ vertices, continuous [1]
	$O(kd)$	8	[24]
	$kd \cdot O(\frac{\log \varepsilon^{-1}}{\varepsilon})$	$1.22+\varepsilon$	Corollary 5.5
	$k \log^2 \frac{1}{\varepsilon} \cdot O(\frac{1}{\varepsilon})^{\frac{d+1}{2}}$	$1+\varepsilon$	Corollary 5.7

We do not state the preprocessing time, as typically it is just an m factor times the space bound.

or low dimensions, and they have exponential running time in high dimensions (this phenomenon is usually referred to as "the curse of dimensionality"). Therefore, we present a simple technique (Lemma 8.1) that allows us to achieve efficient approximation algorithms in high dimensions. Specifically, we use the technique in Theorem 7 to compute an approximation for an optimal simplification in arbitrary dimension d, and to remove the exponential factor from the approximation algorithm of Chan and Rahmati [17] (see Theorem 6).

Lower Bound. Driemel and Psarros [23] proved a cell probe lower bound for a decision distance oracle, providing evidence that our Theorem 1 might be tight. In the cell probe model, one constructs a data structure that is divided into cells of size w. Given a query, one can probe some cells of the data structure and perform unbounded local computation. The complexity of a cell probe data structure is measured with respect to the maximum number of probes performed during a query, as well as the size w of the cells. Theorem 1 works in this regime, where the number of probes is $O(1)$ and $w = O(kd)$. Fix any constants $\gamma, \lambda \in (0,1)$. Consider a cell probe distance oracle $\mathcal{O}$ for curves in $\mathbb{R}^d$ where $d = \Theta(\log m)$, which has word size $w < m^\lambda$, and provide answers for queries of length $k < m^\gamma$, with approximation factor $< \sqrt{3/2}$, while using only a constant number of probes. Driemel and Psarros [23] showed that $\mathcal{O}$ must use space $2^{\Omega(kd)}$.

2 PRELIMINARIES

For two points $x, y \in \mathbb{R}^d$, denote by $\|x-y\|$ the Euclidean norm. Let $P = (p_1, \ldots, p_m) \in \mathbb{R}^{d \times m}$ be a polygonal curve of length m with points in $\mathbb{R}^d$. For $1 \le i \le j \le m$, denote by $P[i,j]$ the subcurve $(p_i, \ldots, p_j)$, and let $P[i] = p_i$. We use $\circ$ to denote the concatenation of two curves or points into a new curve—for example, $P \circ P[1] = (p_1, \ldots, p_m, p_1)$. Let $[m]$ denote the set $\{1, \ldots, m\}$.

[2] Note that additional $O(\log m)$ bit operations are required to read the input and search the data structure.

Our main goal is to solve the following problem.

Problem 1 ($(1+\varepsilon)$-Distance Oracle). *Given a curve $P \in \mathbb{R}^{d\times m}$, preprocess P into a data structure that given a query curve $Q \in \mathbb{R}^{d\times k}$ for some $k \geq 1$ returns a $(1+\varepsilon)$ approximation of $d_{dF}(P,Q)$.*

We assume throughout the article that $\varepsilon \in (0, \frac{1}{4})$. Note that the more natural framework for Problem 1 is when $k \leq m$; however, our solution will hold for general k.

We consider distance oracles in both the static and streaming settings. In the streaming model, the input curve $P \in \mathbb{R}^{d\times m}$ is presented as a data stream of a sequence of points in $\mathbb{R}^d$. The length m of the curve is unlimited and unknown in advance, and the streaming algorithm may use some limited space S, which is independent of m. The algorithm maintains a data structure that can answer queries with respect to the curve seen so far. In each step, a new point is revealed, and it can update the data structure accordingly. It is impossible to access previously revealed points, and the algorithm may only access the current point and the data structure.

The Discrete Fréchet Distance. To simplify the presentation, in this article we follow the definition of Eiter and Mannila [27] and Bereg et al. [8] for the discrete Fréchet distance (which is equivalent to the more commonly used definition, e.g., see [29]).

Consider two curves $P \in \mathbb{R}^{d\times m_1}$ and $Q \in \mathbb{R}^{d\times m_2}$. A ***paired walk*** along P and Q is a sequence of pairs $\omega = \{(\mathcal{P}_i, \mathcal{Q}_i)\}_{i=1}^t$ such that $\mathcal{P}_1, \ldots, \mathcal{P}_t$ and $\mathcal{Q}_1, \ldots, \mathcal{Q}_t$ partition P and Q, respectively, into (disjoint) nonempty subcurves, and for any i it holds that $|\mathcal{P}_i| = 1$ or $|\mathcal{Q}_i| = 1$.

A paired walk ω along P and Q is ***one-to-many*** if $|\mathcal{P}_i| = 1$ for all $1 \leq i \leq |P|$. We say that ω ***matches*** the pair $p \in P$ and $q \in Q$ if there exists i such that $p \in \mathcal{P}_i$ and $q \in \mathcal{Q}_i$.

The ***cost*** of a paired walk $\omega = \{\mathcal{P}_i, \mathcal{Q}_i\}_{i=1}^t$ along P and Q is $\max_i d(\mathcal{P}_i, \mathcal{Q}_i)$, where $d(\mathcal{P}_i, \mathcal{Q}_i) = \max_{(p,q)\in\mathcal{P}_i\times\mathcal{Q}_i} \|p-q\|_2$. In other words, it is the maximum distance over all matched pairs.

The ***discrete Fréchet distance*** is defined over the set $\mathcal{W}$ of all paired walks as

$$d_{dF}(P,Q) = \min_{\omega\in\mathcal{W}} \max_{(\mathcal{P}_i,\mathcal{Q}_i)\in\omega} d(\mathcal{P}_i, \mathcal{Q}_i).$$

A paired walk ω is called an ***optimal walk*** along P and Q if the cost of ω is exactly $d_{dF}(P,Q)$.

Simplifications. An ***optimal k-simplification*** of a curve P is a curve Π of length at most k such that $d_{dF}(P,\Pi) \leq d_{dF}(P,\Pi')$ for any other curve Π' of length at most k.

An ***optimal δ-simplification*** of a curve P is a curve Π with minimum number of vertices such that $d_{dF}(P,\Pi) \leq \delta$. Notice that for an optimal δ-simplification Π of a curve P, there always exists an optimal walk along Π and P that is one-to-many (otherwise, we can remove vertices from Π without increasing the distance). We will use this observation throughout the article.

The vertices of a simplification may be arbitrary or otherwise restricted to some bounded set. A simplification Π of P is ***vertex restricted*** if its set of vertices is a subset of the vertices of P, in the same order as they appear in P.

In some cases, when we want to achieve reasonable space and query bounds while having a small approximation factor, we use a bi-criteria simplification. An ***(α, k, γ)-simplification*** of a curve P is a curve Π of length at most $\alpha \cdot k$ such that for any curve Π' of length at most k it holds that $d_{dF}(P,\Pi) \leq \gamma \cdot d_{dF}(P,\Pi')$. When $\alpha = 1$, we might abbreviate the notation and write (k,γ)-simplification.

In our construction, we use $(k, 1+\varepsilon)$-simplifications to reduce the space bounds of our data structure. However, using simplification in a trivial manner leads to a constant approximation distance oracle, as follows. Given a curve $P \in \mathbb{R}^{d\times m}$, compute and store a $(k, 1+\frac{\varepsilon}{2})$-simplification Π of P, and for a query Q, compute $d_{dF}(\Pi, Q)$ in $O(k^2 d)$ time and return $d_{dF}(Q,\Pi) + d_{dF}(\Pi, P)$. By the triangle inequality, $d_{dF}(Q,P) \leq d_{dF}(Q,\Pi) + d_{dF}(\Pi,P)$. Since Π is a $(k, 1+\frac{\varepsilon}{2})$-simplification of

P, we have $d_{dF}(\Pi, P) \leq (1+\frac{\varepsilon}{2})d_{dF}(Q,P)$, and by the triangle inequality, $d_{dF}(Q,\Pi) + d_{dF}(\Pi,P) \leq d_{dF}(Q,P) + d_{dF}(P,\Pi) + d_{dF}(\Pi,P) \leq (3+\varepsilon)d_{dF}(Q,P)$.

Observation 2.1. *Given a curve $P \in \mathbb{R}^{d\times m}$, there exists data structure with $O(kd)$ space such that given a query $Q \in \mathbb{R}^{d\times k}$ returns a $(3+\varepsilon)$-approximation of $d_{dF}(P,Q)$ in $O(k^2 d)$ time.*

Cover of a Curve. To construct an efficient distance oracle, we introduce the notion of curve cover. A ***(k, r, ε)-cover*** of a curve $P \in \mathbb{R}^{d\times m}$ is a set C of curves of length k such that $d_{dF}(P,W) \leq (1+\varepsilon)r$ for every $W \in C$, and for any curve $Q \in \mathbb{R}^{d\times k}$ with $d_{dF}(P,Q) \leq r$, there exists some curve $W \in C$ with $d_{dF}(Q,W) \leq \varepsilon r$.

Notice that a $(k, r, \frac{\varepsilon}{4})$-cover C of a curve P can be used to construct the following decision version of a distance oracle.

Problem 2 ((k, r, ε)-Decision Distance Oracle). *Given a curve $P \in \mathbb{R}^{d\times m}$ and parameters $r \in \mathbb{R}_+$, $\varepsilon \in (0, \frac{1}{2})$, and $k \in [m]$, create a data structure that given a query curve $Q \in \mathbb{R}^{d\times k}$, if $d_{dF}(P,Q) \leq r$, returns a value Δ such that $d_{dF}(P,Q) \leq \Delta \leq d_{dF}(P,Q) + \frac{\varepsilon}{2}r$, and if $d_{dF}(P,Q) > (1+\varepsilon)r$, it returns NO. (In the case that $r < d_{dF}(P,Q) < (1+\varepsilon)r$, the data structure returns either NO or a value Δ such that $d_{dF}(P,Q) \leq \Delta \leq d_{dF}(P,Q) + \frac{\varepsilon}{2}r$.)*

The idea is that given a query curve Q, if $d_{dF}(P,Q) \leq r$, then there exists some $W \in C$ such that $d_{dF}(Q,W) \leq \frac{\varepsilon}{4}r$ and $d_{dF}(P,W) \leq (1+\frac{\varepsilon}{4})r$. By the triangle inequality,

$$d_{dF}(P,Q) \leq d_{dF}(P,W) + d_{dF}(Q,W) \leq d_{dF}(P,Q) + 2d_{dF}(Q,W) \leq d_{dF}(P,Q) + \frac{\varepsilon}{2}r.$$

However, if $d_{dF}(P,Q) > (1+\varepsilon)r$, then for any $W \in C$ we have

$$d_{dF}(Q,W) \geq d_{dF}(P,Q) - d_{dF}(P,W) > (1+\varepsilon)r - \left(1+\frac{\varepsilon}{4}\right)r > \frac{\varepsilon}{4}r\,.$$

Therefore, we have the following observation.

Observation 2.2. *Assume that there exists a data structure that stores a (k, r, ε)-cover C for P of size S such that given a curve $Q \in \mathbb{R}^{d\times k}$ with $d_{dF}(Q,P) \leq r$, return in time T a curve $W \in C$ with $d_{dF}(Q,W) \leq \varepsilon r$ and the value $\mathrm{dist(W)} = \mathrm{d_{dF}(P,W)}$. Then there exists a (k, r, ε)-decision distance oracle for P with the same space and query time.*

Note that sometimes we abuse the notation and relate to C as the data structure from the preceding observation.

Uniform Grids. Consider the infinite d-dimensional grid with edge length $\frac{\varepsilon}{\sqrt{d}}r$, with a point at the origin. For a point $x \in \mathbb{R}^d$, denote by $G_{\varepsilon,r}(x,R)$ the set of grid points that are contained in $B_2^d(x,R)$, the d-dimensional ball of radius R centered at x. The following claim is a generalization of Corollary 7 from previous work [29]. The proof can be found in Appendix A.1.

Claim 2.3. $|G_{\varepsilon,r}(x, cr)| = O(\frac{c}{\varepsilon})^d$.

3 ARTICLE OVERVIEW

Distance Oracle: The Static Case

Given a curve $P \in \mathbb{R}^{d\times m}$, we first consider a more basic version of the $(1+\varepsilon)$-distance oracle, namely a *decision distance oracle*. Here, in addition to P, we are given a distance threshold r. For a query curve $Q \in \mathbb{R}^{d\times k}$, the decision distance oracle either returns a value $\Delta \in [d_{dF}(P,Q), d_{dF}(P,Q)+\varepsilon r]$ or declares that $d_{dF}(P,Q) \geq (1+\varepsilon)r$. We construct a decision distance oracle by discretizing the space of query curves (using a uniform grid). In other words, we simply store the answers to the set of all grid curves at a distance at most $(1+\varepsilon)r$ from P in a hash table. The query algorithm then "snaps" the points of Q to the grid, to obtain the closest grid curve, and returns the precomputed answer from the hash table. Clearly, we have a linear $O(kd)$ query time. As was shown by the authors and Katz [29], the number of grid curves that we need to store is $O(\frac{1}{\varepsilon})^{kd}$, which is also a bound on the size of the distance oracle (see Lemma 4.1).

Next, we consider a generalized version that we call a *bounded range distance oracle*. Here, in addition to P, we are given a range of distances $[\alpha, \beta] \subset \mathbb{R}$. For a query $Q \in \mathbb{R}^{d\times k}$, the distance oracle is guaranteed to return a $(1+\varepsilon)$-approximation of $d_{dF}(P,Q)$ only if $d_{dF}(P,Q) \in [\alpha, \beta]$. Such an oracle is constructed using $\log \frac{\beta}{\alpha}$ decision distance oracles for exponentially growing scales, and given a query we perform a binary search among them. Thus, in total, compared to the decision version, we have an overhead of $\log \frac{\beta}{\alpha}$ in the space and $\log\log \frac{\beta}{\alpha}$ in the query time (see Lemma 4.2).

The main goal is to construct a general distance oracle that will succeed on all queries. To achieve space and query bounds independent of m, our first step is to precompute a $(k, 1+\varepsilon)$-simplification Π of P. Note that following Observation 2.1, given a query $Q \in \mathbb{R}^{d\times k}$ we can simply return $\Delta = d_{dF}(Q,\Pi) + d_{dF}(\Pi, P)$, which is a constant approximation for $d_{dF}(P,Q)$ computed in $O(k^2 d)$ time. However, we can achieve a $1+\varepsilon$ approximation as follows:

- If $d_{dF}(Q,\Pi) = \Omega(\frac{1}{\varepsilon}) \cdot d_{dF}(\Pi,P)$, then $d_{dF}(Q,\Pi)$ is a $(1+\varepsilon)$-approximation for $d_{dF}(P,Q)$.
- If $d_{dF}(Q,\Pi) = O(\varepsilon) \cdot d_{dF}(\Pi,P)$, then $d_{dF}(P,\Pi)$ is a $(1+\varepsilon)$-approximation for $d_{dF}(P,Q)$.
- Else, we have $d_{dF}(Q,P) \in [\Omega(\varepsilon), O(\frac{1}{\varepsilon})] \cdot d_{dF}(\Pi,P)$. This is a bounded range for which we can precompute a bounded range distance oracle for P.

The only caveat is that computing $d_{dF}(Q,\Pi)$ takes $O(k^2 d)$ time. Our solution is to construct a distance oracle for Π. At first glance, it seems that we are back to the same problem. However, in this case, Π and Q have the same length! Thus, our entire construction boils down to computing a *symmetric distance oracle*—that is, a distance oracle for the special case of $m = k$.

To achieve a near linear query time, our symmetric distance oracle first computes a coarse approximation of $d_{dF}(P,Q)$ in near linear time (using Theorem 6). Roughly speaking, if the approximated distance $\tilde{\Delta}$ is very large or very small, we show that a $(1+\varepsilon)$-approximation can be computed directly in linear time. Else, to reduce the approximation factor, we maintain a polynomial number of ranges $[\alpha, \beta]$, for which we construct a bounded range distance oracle. We show that if $d_{dF}(P,Q)$ does not fall in any of the precomputed ranges, then (an approximation of) the distance can be computed in linear time.

We elaborate on the different cases. The decision whether $\tilde{\Delta}$ is very large or very small and the construction of bounded range distance oracles are done with respect to the lengths of the edges of the input curve. First, observe that if the distance between two curves X and Y is smaller than half the length of the shortest edge of X, then $d_{dF}(X,Y)$ can be computed in linear time. Next, using Theorem 6, we get a value $\tilde{\Delta}$ such that $d_{dF}(P,Q) \in [\frac{\tilde{\Delta}}{md}, \tilde{\Delta}]$. Let $l_1 \le l_2 \le \cdots \le l_{m-1}$ be a sorted list of the lengths of edges of P. We have four cases:

- If $\tilde{\Delta} < \frac{l_1}{2}$, then the distance between P and Q is smaller than half of the shortest edge in P, and thus by the preceding observation we can compute $d_{dF}(P,Q)$ exactly in linear time.
- If $\tilde{\Delta} > \frac{dm^2}{\varepsilon} l_{m-1}$, then $d_{dF}(P[1],Q) = \max_{1\le i\le m} \|P[1] - Q[i]\|$ is a good enough approximation of $d_{dF}(P,Q)$, because $d_{dF}(P,P[1]) \le m \cdot l_{m-1} < \varepsilon \frac{\tilde{\Delta}}{md} \le \varepsilon \cdot d_{dF}(P,Q)$.

Else, we precompute the bounded range distance oracle for the ranges $[\frac{1}{\mathrm{poly}(\frac{md}{\varepsilon})}, \mathrm{poly}(\frac{md}{\varepsilon})] \cdot l_i$ for each i:

- If $\tilde{\Delta}$ falls in one of the preceding ranges, we simply use the appropriate distance oracle to return an answer.

- Else, there is some i such that $\text{poly}(\frac{md}{\varepsilon}) \cdot l_i < \tilde{\Delta} < \frac{1}{\text{poly}(\frac{md}{\varepsilon})} \cdot l_{i+1}$. Thus, l_{i+1} is much larger than l_i. Let P' be the curve obtain from P by "contracting" all edges of length at most l_i. It holds that $d_{dF}(P', P) \leq m \cdot l_i \ll \frac{\varepsilon}{md} \cdot \tilde{\Delta} \leq \varepsilon \cdot d_{dF}(P, Q)$, thus $d_{dF}(P', Q)$ is a $1+\varepsilon$ approximation of $d_{dF}(P, Q)$. However, the shortest edge of P' has length at least $\frac{1}{2} l_{i+1}$, which is much larger than $d_{dF}(P', Q)$. Hence, $d_{dF}(P', Q)$ can be computed in linear time.

To remove the logarithmic dependency on $\frac{1}{\varepsilon}$ in the query time, we subdivide our ranges into smaller overlapping ranges and obtain the following theorem.

Theorem 1. *Given a curve $P \in \mathbb{R}^{d \times m}$ and parameters $\varepsilon \in (0, \frac{1}{4})$ and integer $k \geq 1$, there exists a distance oracle with $O(\frac{1}{\varepsilon})^{dk} \cdot \log \varepsilon^{-1}$ storage space, $m \log \frac{1}{\varepsilon} \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d \log m))$ expected preprocessing time, and $\tilde{O}(kd)$ query time.*

Curve Simplification in the Stream

Given a curve P as a stream, our goal is to maintain a $(k, 1+\varepsilon)$-simplification of P. For a curve $P \in \mathbb{R}^{d \times m}$ in the static model, an optimal δ-simplification of P can be computed using a greedy algorithm, which simply finds the largest index i such that $P[1, i]$ can be enclosed by a ball of radius δ and then recurse for $P[i+1, m]$. This greedy simplification algorithm was presented by Bereg et al. [8] for constant dimension, and was generalized to arbitrary dimension d by the authors and Katz [29] (see Lemma 8.2). In the static model, a $(k, 1+\varepsilon)$-simplification can be computed by searching over all possible values of δ with the greedy δ-simplification algorithm as the decision procedure.

Denote by γ-MEB a streaming algorithm for computing a γ-approximation of the minimum enclosing ball. Given a curve P in a streaming fashion, and a γ-MEB algorithm as a black box, we first implement a streaming version of the greedy simplification algorithm called `GreedyStreamSimp` (see Algorithm 2). This algorithm gets as an input a parameter δ and acts in the same manner as the greedy simplification, where instead of a static minimum enclosing ball algorithm, it uses the γ-MEB black box. The resulting simplification Π will be the sequence of centers of balls of radius δ constructed by γ-MEB. Note that Π is at a distance at most δ from P, and every curve at distance δ/γ from P has length at least $|\Pi|$ (see Claim 5.1). However, the length of Π is essentially unbounded, and our goal is to construct a simplification of length k.

If we knew in advance the distance δ^* between P and an optimal k-simplification of P, we could execute `GreedyStreamSimp` with the parameter $\delta = \gamma\delta^*$ and obtain a (k, γ)-simplification. Since δ^* is not known in advance, our `LeapingStreamSimp` algorithm tries to guess it. The `LeapingStreamSimp` algorithm (see Algorithm 3) gets as an input the desired length k, as well as two additional parameters, init and inc. It sets the initial estimation of δ to be init. Then, it simply simulates `GreedyStreamSimp` (with parameter δ) as long as the simplification Π at hand is of length at most k. Once this condition is violated, `LeapingStreamSimp` performs a leaping step as follows. Suppose that after reading $P[m]$ the length condition is violated—that is, $|\Pi| = k+1$. In this case, `LeapingStreamSimp` will increase its guess of δ^* by setting $\delta \leftarrow \delta \cdot \text{inc}$. Then, `LeapingStreamSimp` starts a new simulation of `GreedyStreamSimp`, with the new guess δ, and the previous simplification Π as input (instead of $P[1, m]$). Now, `LeapingStreamSimp` continues processing the stream points $P[m+1, \ldots]$ as if nothing happened. Such a leaping step will be performed each time the length condition is violated. As a result, eventually `LeapingStreamSimp` will hold an estimate δ and a simplification Π such that Π is an actual simplification constructed by the `GreedyStreamSimp` with parameter δ. Alas, Π was not constructed with respect to the observed curve P but rather with respect to some other curve P' such that $d_{dF}(P, P') \leq \frac{2}{\text{inc}}\delta$ (see Claim 5.2). Furthermore, the estimate δ will be bounded by the distance to the optimal simplification δ^* multiplied by a factor of $\approx \gamma \cdot \text{inc}$.

To obtain a $1+\varepsilon$ approximation of δ^*, we run $\approx \frac{1}{\varepsilon}$ instances of `LeapingStreamSimp`, with different initial guess parameter init. Then, at each point in time, for the instance with the minimum estimation δ, it holds that $\delta < (1+\varepsilon)\delta^*$. We thus prove the following theorem.

THEOREM 2. *Suppose that we are given a black box streaming algorithm γ-MEB for $\gamma \in [1,2]$ that uses storage space $S(d,\gamma)$. Then for every parameter $\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{\log \varepsilon^{-1}}{\varepsilon} \cdot (S(d,\gamma) + kd))$ space, and given a curve P in $\mathbb{R}^d$ in a streaming fashion computes a $(k, \gamma(1+\varepsilon))$-simplification Π of P and a value L such that $d_{dF}(\Pi, P) \leq L \leq \gamma(1+\varepsilon)\delta^*$.*

Plugging existing γ-MEB algorithms, we obtain the following (see Corollaries 5.5 and 5.7):

- $(k, 1+\varepsilon)$-simplification in streaming using $O(\varepsilon)^{-\frac{d+1}{2}} \log^2 \varepsilon^{-1} + O(kd\varepsilon^{-1} \log \varepsilon^{-1})$ space.
- $(k, 1.22+\varepsilon)$-simplification in streaming using $O(\frac{\log \varepsilon^{-1}}{\varepsilon} \cdot kd)$ space.

We note that our `LeapingStreamSimp` algorithm is a generalization of the algorithm of Driemel et al. [24] for computing a $(k, 8)$-simplification. Specifically, one can view the algorithm from Driemel et al. [24] as a specific instance of `LeapingStreamSimp`, where fixing the parameters init $= 1$ and inc $= 2$, and using a simple 2-MEB algorithm.

Distance Oracle: The Streaming Case

Our basic approach here is to imitate our static distance oracle from Theorem 1. We maintain a $(k, 1+\varepsilon)$-simplification Π of P (using Corollary 5.7). As we have the simplification explicitly, we can also construct a symmetric distance oracle for Π. Thus, when a query Q for P arrives, we can estimate $d_{dF}(\Pi, Q)$ quickly. If either $d_{dF}(\Pi, Q) > \frac{1}{\varepsilon} \cdot d_{dF}(\Pi, P)$ or $d_{dF}(\Pi, Q) < \varepsilon \cdot d_{dF}(\Pi, P)$, as discussed previously, we can answer immediately. Else, we have $d_{dF}(Q, P) \in [\Omega(\varepsilon), O(\frac{1}{\varepsilon})] \cdot d_{dF}(\Pi, P)$, which is a bounded range. In the static case, we simply prepared answers to all of the possible queries in this range ahead of time. However, in the streaming case, this range is constantly changing and is unknown in advance. How can we be prepared for the unknown?

The first key observation is that given a parameter r, one can maintain a decision distance oracle[3] in a stream. Specifically, given a decision distance oracle for a curve $P[1, m]$ with storage space independent of m (as in Lemma 6.1), and a new point $P[m+1]$, we show how to construct a decision distance oracle for $P[1, m+1]$. However, the scales r for which we construct the decision distance oracles are unknown in advance, and we need a way to update r on demand.

Our solution is similar in spirit to the maintenance of simplification in the stream. In other words, we will create a leaping version of the decision oracle—that is, a data structure that receives as input a pair of parameters init and inc. Initially, it sets the scale parameter r to init. As long as the distance oracle is not empty (i.e., there is at least one curve at distance r from P), it continues to simulate the streaming algorithm that constructs a decision distance oracle for fixed r. If it becomes empty after reading the point $P[m]$ for the stream, then instead of despairing, the oracle updates its scale parameter to $r \cdot$ inc, chooses an arbitrary curve W from the distance oracle of $P[1, m-1]$, and initializes a new distance oracle for $W \circ P[m]$ using the new parameter r. From here on, the oracle continues to simulate the construction of a decision distance oracle as before while performing a leaping step each time it becomes empty.

As a result, at each step, we have an actual decision distance oracle for some parameter r. However, the oracle returns answers not with respect to the observed curve P but rather with respect to some other curve P' such that $d_{dF}(P, P') \leq \frac{2}{\mathrm{inc}} r$ (see Lemma 6.2). We show that if we maintain $O(\log \frac{1}{\varepsilon})$ such leaping distance oracles for different values of init, we will always be able to answer queries for curves Q such that $d_{dF}(Q, P) \in [\Omega(\varepsilon), O(\frac{1}{\varepsilon})] \cdot d_{dF}(\Pi, P)$.

[3]Here, by *decision distance oracle* we mean cover (see Observation 2.2).

THEOREM 3. *Given parameters $\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{1}{\varepsilon})^{kd} \log \varepsilon^{-1}$ space, and given a curve P with points in $\mathbb{R}^d$ constructs a $(1+\varepsilon)$-distance oracle with $\tilde{O}(kd)$ query time.*

Distance Oracle to a Subcurve and the "Zoom-In" Problem

Following Driemel and Har-Peled [21] and previous work [30], we consider a generalization of the distance oracle problem, where the query algorithm gets as an input two indices $1 \leq i \leq j \leq m$ in addition to a query curve $Q \in \mathbb{R}^{d\times k}$ and returns $(1+\varepsilon)$-approximation of $d_{dF}(P[i,j], Q)$. Note that a trivial solution is storing $O(m^2)$ distance oracles: for any $1 \leq i \leq j \leq m$, store a $(1+\varepsilon)$-distance oracle for $P[i,j]$. However, when m is large, one might wish to reduce the quadratic storage space at the cost of increasing the query time or approximation factor.

Before presenting our solution to the preceding problem, we introduce a closely related problem that we call the *zoom-in problem*. Given a curve $P \in \mathbb{R}^{d\times m}$ and an integer $1 \leq k < m$, our goal is to preprocess P into a data structure that given $1 \leq i < j \leq m$ returns an (α, k, γ)-simplification of $P[i,j]$. Our solutions to the zoom-in problem and the distance oracle to a subcurve problem have a similar basic structure, which consists of hierarchically partitioning the input curve P. Given a query, a solution is constructed by basically concatenating two precomputed solutions. Each level represents a subcurve $P[a,b]$, and we precompute solutions to all subcurves of the form $P[i,x]$ and $P[x+1,j]$, where $x = \lfloor \frac{a+b}{2} \rfloor$ is the middle point. Given a subcurve query $P[i,j]$, we simply merge the solutions for $P[i,x]$ and $P[x+1,j]$. We obtain the following theorems.

THEOREM 4. *Given a curve P consisting of m points and parameters $k \in [m]$ and $\varepsilon \in (0, \frac{1}{2})$, there exists a data structure with $O(mkd \log \frac{m}{k})$ space such that given a pair of indices $1 \leq i < j \leq m$ returns in $O(kd)$ time a $(2, k, 1+\varepsilon)$-simplification of $P[i,j]$. The prepossessing time for general d is $\tilde{O}(m^2 d\varepsilon^{-4.5})$ and for fixed d is $\tilde{O}(m^2\varepsilon^{-1})$.*

THEOREM 5. *Given a curve $P \in \mathbb{R}^{d\times m}$ and parameter $\varepsilon > 0$, there exists a data structure that given a query curve $Q \in \mathbb{R}^{d\times k}$, and two indexes $1 \leq i \leq j \leq m$, returns an $(1+\varepsilon)$-approximation of $d_{dF}(P[i,j], Q)$. The data structure has $m \log m \cdot O(\frac{1}{\varepsilon})^{dk} \cdot \log \varepsilon^{-1}$ storage space, $m^2 \log \frac{1}{\varepsilon} \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d \log m))$ expected preprocessing time, and $\tilde{O}(k^2 d)$ query time.*

High-Dimensional Discrete Fréchet Algorithms

In Lemma 8.1, we present a simple technique that is useful when one wants to get an approximated distance over a set of points in any dimension d. More precisely, given a set of points V in $\mathbb{R}^d$, we compute a near linear number of distance values that approximate all possible distances between points in V. The idea is using d **Well-Separated Pair Decompositions (WSPDs)**, one for each projection of the points to an axis. Each WSPD is then one-dimensional and thus has near linear size. By arbitrarily picking a pair of (one-dimensional) points from each pair of subsets in the WSPDs, we get a set of distances M such that for any two points $x, y \in V$ there exists a distance $\alpha \in M$ that is close enough to $\|x-y\|$. Note that in the work of Bringmann and Mulzer [11] (Theorem 5.5), the authors use WSPD for a similar purpose; however, since they use one d-dimensional WSPD, they obtain a set of size exponential in d. We use it to remove the exponential factor from the approximation algorithm of Chan and Rahmati [17] and to generalize the algorithm of Bereg et al. [8] for computing a $(1+\varepsilon)$-approximation of the optimal k simplification in any dimension. Note that the algorithm of Bereg et al. [8] has running time $\tilde{O}(mk)$ for a curve in $P \in \mathbb{R}^{d\times m}$, and by considering $(k, 1+\varepsilon)$-simplifications instead of optimal k-simplifications, we manage to reduce the running time to $\tilde{O}(\frac{md}{\varepsilon^{4.5}})$. We obtain the following theorems.

Theorem 6. *Given two curves P and Q in $\mathbb{R}^{d\times m}$, and a value $f \geq 1$, there is an algorithm that returns a value $\tilde{\Delta}$ such that $d_{dF}(P,Q) \leq \tilde{\Delta} \leq f \cdot d_{dF}(P,Q)$ in $O(md\log(md)\log d + (md/f)^2 d\log(md)) = \tilde{O}(md + (md/f)^2 d)$ time.*

Theorem 7. *Given a curve $P \in \mathbb{R}^{m\times d}$ and parameters $k \in [m]$, $\varepsilon \in (0, \frac{1}{2})$, there is an $\tilde{O}(\frac{md}{\varepsilon^{4.5}})$-time algorithm that computes a $(k, 1+\varepsilon)$-simplification Π of P. In addition, the algorithm returns a value δ such that $d_{dF}(P,\Pi) \leq \delta \leq (1+\varepsilon)\delta^*$, where δ^* is the distance between P to an optimal k-simplification.*
Furthermore, if d is fixed, the algorithm can be executed in $m \cdot O(\frac{1}{\varepsilon} + \log\frac{m}{\varepsilon}\log m)$ time.

4 DISTANCE ORACLE: THE STATIC CASE

We begin by constructing a $(1+\varepsilon)$-distance oracle for the static case, where a curve $P \in \mathbb{R}^{d\times m}$ is given in the preprocessing stage. To achieve a $(1+\varepsilon)$ approximation for the distance between P and a query $Q \in \mathbb{R}^{d\times k}$ in near linear time, our distance oracle first computes a very rough estimation of this distance. Then, to reduce the approximation factor, we maintain a polynomial number of ranges $[\alpha,\beta]$, for which we store a distance oracle that can answer queries only when the answer is in the range $[\alpha,\beta]$. This structure uses a set of (k,r,ε)-covers, where r grows exponentially in the given range $[\alpha,\beta]$.

We describe the ingredients of our distance oracle from the bottom up, starting with the basic construction of a curve cover, then present the bounded range distance oracle, describe a solution for the case where $k = m$ (the symmetric case), and finally show how to combine all ingredients and construct a $(1+\varepsilon)$-distance oracle for P with near linear query time and $O(\frac{1}{\varepsilon})^{kd}\log\varepsilon^{-1}$ storage space.

4.1 Cover of a Curve

Given input curve $P \in \mathbb{R}^{d\times m}$, in this section we show how to construct a data structure that stores a (k,r,ε)-cover C of size $O(\frac{1}{\varepsilon})^{kd}$ for P and has a linear look-up time.

Our data structure is based on the ANN data structure presented in previous work [29]. For a single curve P, this data structure essentially solves a decision version of the distance oracle: given parameters r and ε, the $(r, 1+\varepsilon)$-ANN data structure uses $O(\frac{1}{\varepsilon})^{kd}$ storage space, and given a query curve $Q \in \mathbb{R}^{d\times k}$ returns YES if $d_{dF}(P,Q) \leq r$ and NO if $d_{dF}(P,Q) > (1+\varepsilon)r$ (if $r < d_{dF}(P,Q) \leq (1+\varepsilon)r$ it can return either YES or NO).

Using the same technique from previous work [29] with a slight adaptation, one can construct a (k,r,ε)-cover with the same space and look-up bounds. We include the basic details here for completeness.

Consider the infinite d-dimensional grid with edge length $\frac{\varepsilon}{\sqrt{d}}r$, and let

$$\mathcal{G} = \bigcup_{1\leq i\leq m} G_{\varepsilon,r}(P[i], (1+\varepsilon)r).$$

Let C be the set of all curves W with k points from $\mathcal{G}$ such that $d_{dF}(P,W) \leq (1+\varepsilon)r$. Previous work [29] showed that $|C| = O(\frac{1}{\varepsilon})^{kd}$, and that it can be computed in $m \cdot O(\frac{1}{\varepsilon})^{kd}$ time.

The Data Structure. We insert the curves of C into the dictionary $\mathcal{D}$ as follows. For each curve $W \in C$, if $W \notin \mathcal{D}$, insert W into $\mathcal{D}$ and set $\mathrm{dist(W)} \leftarrow \mathrm{d_{dF}(P,W)}$.

Previous work [29] showed that $\mathcal{D}$ can be implemented using cuckoo hashing [41] so that given a query curve Q, one can find Q in $\mathcal{D}$ (if it exists) in $O(kd)$ time, the storage space required for $\mathcal{D}$ is $O(\frac{1}{\varepsilon})^{kd}$, and it can be constructed in $m \cdot (O(\frac{1}{\varepsilon})^{kd} + d\log m)$ expected time.

The Query Algorithm. Let $Q \in \mathbb{R}^{d\times k}$ be the query curve. The query algorithm is as follows. For each $1 \le i \le k$, find the grid point x_i (not necessarily from $\mathcal{G}$) closest to $Q[i]$. This can be done in $O(kd)$ time by rounding. Then, search for the curve $W' = (x_1, \ldots, x_k)$ in the dictionary $\mathcal{D}$. If W' is in $\mathcal{D}$, return W' and dist(W′); otherwise, return NO. The total query time is then $O(kd)$.

Correctness. First, by the construction, for any $W \in C$ we have $d_{dF}(P, W) \le (1+\varepsilon)r$. Second, let $Q \in \mathbb{R}^{d\times k}$ be a query curve such that $d_{dF}(P, Q) \le r$. Notice that $\|Q[i] - x_i\|_2 \le \frac{\varepsilon}{2\sqrt{d}}r$ because the length of the grid edges is $\frac{\varepsilon}{\sqrt{d}}r$, and thus $d_{dF}(Q, W') \le \frac{\varepsilon}{2}r$. By the triangle inequality, $d_{dF}(P, W') \le d_{dF}(P, Q) + d_{dF}(Q, W') \le (1+\varepsilon)r$, and therefore W' is in C.

By Observation 2.2, we obtain the following lemma.

Lemma 4.1. *Given a curve $P \in \mathbb{R}^{d\times m}$ and parameters $r \in \mathbb{R}_+$, $\varepsilon \in (0, \frac{1}{4})$, and $k \ge 1$, there is an algorithm that constructs a (k, r, ε)-decision distance oracle with $O(\frac{1}{\varepsilon})^{kd}$ storage space, $m \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d \log m))$ expected preprocessing time, and $O(kd)$ query time.*

4.2 Bounded Range Distance Oracle

We now show how to use (k, r, ε)-covers to solve the following problem.

Problem 3 (Bounded Range Distance Oracle). *Given a curve $P \in \mathbb{R}^{d\times m}$, a range $[\alpha, \beta]$ (where $\beta \ge 4\alpha$), and parameters $\varepsilon \in (0, \frac{1}{4})$ and $k \ge 1$, preprocess P into a data structure that given a query curve $Q \in \mathbb{R}^{d\times k}$ with $d_{dF}(P, Q) \in [\alpha, \beta]$ returns a $(1+\varepsilon)$ approximation of $d_{dF}(P, Q)$.*

The Data Structure. For every $0 \le i \le \lceil \log \beta/\alpha \rceil$, we construct a decision distance oracle $\mathcal{D}_i$ with parameter $r_i = \alpha \cdot 2^i$, and $\varepsilon' = \frac{\varepsilon}{4}$. The total storage space is therefore $O(\log \beta/\alpha) \cdot O(\frac{1}{\varepsilon})^{kd}$, whereas the preprocessing time is $m \log \beta/\alpha \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d \log m))$ in expectation.

The Query Algorithm. Given a query curve $Q \in \mathbb{R}^{d\times k}$ such that $d_{dF}(P, Q) \in [\alpha, \beta]$, perform a binary search on the values $r_i = \alpha \cdot 2^i$ for $0 \le i \le \lceil \log \beta/\alpha \rceil$ using the decision distance oracles as follows. Let $[s', t']$ be the current range. We describe a recursive binary search where the invariant is that $\alpha \cdot 2^{s'} \le d_{dF}(P, Q) \le \alpha \cdot 2^{t'}$. If $t' \le s' + 2$, return the answer of $\mathcal{D}_{t'}$. Else, let $x = \lfloor \frac{t'-s'}{2} \rfloor$ and query $\mathcal{D}_x$ with Q. If it returns a distance, then set the current range to $[s', x+1]$. Otherwise, set the current range to $[x, t']$.

The number of decision queries is $O(\log \log \beta/\alpha)$. By Lemma 4.1, the total query time is therefore $O(kd \log \log \beta/\alpha)$.

Correctness. Assume that $\alpha \cdot 2^{s'} \le d_{dF}(P, Q) \le \alpha \cdot 2^{t'}$. If $t' \le s'+2$, then $\mathcal{D}_{t'}$ returns a value Δ such that $d_{dF}(P, Q) \le \Delta \le d_{dF}(P, Q) + \varepsilon' r_{t'} = d_{dF}(P, Q) + \frac{\varepsilon}{4}\alpha \cdot 2^{t'} = d_{dF}(P, Q) + \varepsilon\alpha \cdot 2^{s'} \le (1+\varepsilon) d_{dF}(P, Q)$.

Else, if $\mathcal{D}_x$ returns a distance value, then $d_{dF}(P, Q) \le (1+\varepsilon)r_x = (1+\varepsilon)\alpha \cdot 2^x \le \alpha \cdot 2^{x+1}$ (for $\varepsilon < 1$), so $\alpha \cdot 2^{s'} \le d_{dF}(P, Q) \le \alpha \cdot 2^{x+1}$ and the invariant still hold. In addition, notice that $x+1 < t'$ because $t' - s' > 1$. If $\mathcal{D}_x$ returns NO, then $d_{dF}(P, Q) > r_x = \alpha \cdot 2^x$, so $\alpha \cdot 2^x \le d_{dF}(P, Q) \le \alpha \cdot 2^{t'}$ and the invariant still hold.

We obtain the following lemma.

Lemma 4.2. *Given a curve $P \in \mathbb{R}^{d\times m}$, a range $[\alpha, \beta]$ where $\beta \ge 4\alpha$, and parameters $\varepsilon \in (0, \frac{1}{4})$, $k \ge 1$, there exists a bounded range distance oracle with $O(\frac{1}{\varepsilon})^{kd} \cdot \log \beta/\alpha$ storage space, $m \log \beta/\alpha \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d \log m))$ expected preprocessing time, and $O(kd \log \log \beta/\alpha)$ query time.*

4.3 Symmetric Distance Oracle

We construct a $(1+\varepsilon)$-distance oracle for the symmetric case of $k = m$. This time, the query algorithm of our distance oracle does not simply return a precomputed value but actually performs a smart case analysis that allows both fast query time and a relatively small storage space complexity. The main idea is to first perform a fast computation of a very rough approximation of the distance between the query Q and the input P. If the approximated distance is very large or very small, we show that a $(1+\varepsilon)$-approximation can be returned right away. For the other cases, we use a (precomputed) set of bounded range distance oracles, as described in the previous section.

The decision whether the approximated distance is very large or very small depends on the length of the smallest and largest edges of P. Denote $\lambda(P) = \frac{1}{2}\min_{1\le i\le m-1}\{\|P[i]-P[i+1]\|\}$—that is, $\lambda(P)$ is half the length of the shortest edge in P. Let $l_1 \le l_2 \le \cdots \le l_{m-1}$ be a sorted list of the lengths of P's edges, and set $l_0 = \lambda(P) = \frac{l_1}{2}$ and $l_m = \frac{dm^2}{\varepsilon} l_{m-1}$.

Let $X \in \mathbb{R}^{d\times m_1}$ and $Y \in \mathbb{R}^{d\times m_2}$ be two curves. Notice that there are no $i \in [m_1 - 1]$ and $j \in [m_2]$ such that $\|Y[j]-X[i]\| < \lambda(X)$ and $\|Y[j]-X[i+1]\| < \lambda(X)$, because otherwise $\|X[i]-X[i+1]\| < 2\lambda(X) = \min_{1\le i\le m_1-1}\{\|X[i]-X[i+1]\|\}$. Therefore, if $d_{dF}(X,Y) < \lambda(X)$, then there exists a single paired walk ω along X and Y with cost $d_{dF}(X,Y)$, and ω is one-to-many.

ALGORITHM 1: SmallDistance(X, Y)

input : Curves $X \in \mathbb{R}^{d\times m_1}$ and $Y \in \mathbb{R}^{d\times m_2}$
output : Either $d_{dF}(X,Y)$ or NO

Compute $\lambda(X)$
Set $j \leftarrow 1$, $\Delta \leftarrow 0$
for $1 \le i \le m_1$ **do**
  if $j > m_2$ *OR* $\|X[i]-Y[j]\| \ge \lambda(X)$ **then**
    return NO
  while $\|X[i]-Y[j]\| < \lambda(X)$ **do**
    $\Delta \leftarrow \max\{\Delta, \|X[i]-Y[j]\|\}$
    $j \leftarrow j+1$
return Δ

Consider Algorithm 1, which essentially attempts to compute the one-to-many paired walk ω along X and Y with respect to $\lambda(X)$, in a greedy fashion. If the algorithm fails to do so, then $d_{dF}(X,Y) \ge \lambda(X)$. It is easy to see that the running time of Algorithm 1 is $O(m_1 + m_2)$. Therefore, we obtain the following claim.

Claim 4.3. *Algorithm 1 runs in linear time, and if $d_{dF}(X,Y) < \lambda(X)$, then it returns $d_{dF}(X,Y)$; else, it returns NO.*

Our query algorithm first computes an m-approximation $\tilde{\Delta}$ of $d_{dF}(P,Q)$ using Theorem 6. The query algorithm contains four basic cases, depending on the value $\tilde{\Delta}$. Cases 1 through 3 do not require any precomputed values and compute the returned approximated distance in linear time. For case 4, we store a set of $O(m \cdot \lceil \log_m \frac{1}{\varepsilon} \rceil))$ bounded range distance oracles as follows.

First, consider the following ranges of distances: for $1 \le i \le m-1$, set $[\alpha_i, \beta_i] = [\frac{1}{5dm} l_i, \frac{dm^2}{\varepsilon} l_i]$. Notice that $\frac{\beta_i}{\alpha_i} = \frac{5d^2m^3}{\varepsilon}$. Next, for each of the preceding ranges, we construct a set of overlapping subranges, each with ratio $(dm)^2$. More precisely, for every $0 \le i \le m-1$ and $\lfloor \log_{dm} \frac{1}{5dm} \rfloor \le j \le \lfloor \log_{dm} \frac{m}{\epsilon} \rfloor$, set $[\alpha_i^j, \beta_i^j] = [l_i \cdot (dm)^j, l_i \cdot (dm)^{j+2}]$ and construct a bounded range distance oracle $\mathcal{D}_i^j$ with the range $[\alpha_i^j, \beta_i^j]$ using Lemma 4.2.[4]

[4]Note that initially we could use Lemma 4.2 directly on the range $[\beta_i, \alpha_i]$. However, the further subdivision to subranges of size polynomial in m saves an $\log\log\frac{1}{\varepsilon}$ factor from the query time.

The Query Algorithm. Given a query curve $Q \in \mathbb{R}^{d\times m}$, compute in $O(md\log(md)\log d)$ time a value $\tilde{\Delta}$ such that $d_{dF}(P,Q) \le \tilde{\Delta} \le md \cdot d_{dF}(P,Q)$ (using the algorithm from Theorem 6):

Case 1: $\tilde{\Delta} < \frac{l_1}{2}$. Return SmallDistance$(P,Q)$.

Case 2: $\tilde{\Delta} > \frac{dm^2}{\varepsilon} \cdot l_{m-1}$. Return $d_{dF}(P[1],Q)$.

If both cases 1 and 2 do not hold, then we have $l_0 = \frac{l_1}{2} \le \tilde{\Delta} \le \frac{dm^2}{\varepsilon} \cdot l_{m-1} = l_m$. There must be an index $i \in [1, m-1]$ such that one of the following two cases hold:

Case 3: $\frac{dm^2}{\varepsilon} l_i \le \tilde{\Delta} \le \frac{1}{5} l_{i+1}$. Let $S = \{P[j] \mid \|P[j]-P[j+1]\| \le l_i\}$, and let P' be the curve obtained by removing the points of S from P. Return SmallDistance(P',Q).

Case 4: $\frac{1}{5} l_i \le \tilde{\Delta} \le \frac{dm^2}{\varepsilon} l_i$. We have $d_{dF}(P,Q) \in [\frac{\tilde{\Delta}}{dm}, \tilde{\Delta}]$, so let j be an index such that $[\frac{\tilde{\Delta}}{dm}, \tilde{\Delta}] \subseteq [\alpha_i^j, \beta_i^j]$, query $\mathcal{D}_i^j$ with Q, and return the answer.

Correctness. We show that the query algorithm returns a value Δ^* such that $(1-\varepsilon)d_{dF}(P,Q) \le \Delta^* \le (1+\varepsilon)d_{dF}(P,Q)$.

First, we claim that the four cases in our query algorithm are disjoint, and that $\tilde{\Delta}$ falls in one of them. The reason is that if cases 1 and 2 do not hold, then $l_0 \le \tilde{\Delta} \le l_m$, so there must exist an index i such that $l_i \le \tilde{\Delta} \le l_{i+1}$. If $\frac{m^2}{\varepsilon} l_i \le \tilde{\Delta} \le \frac{1}{5} l_{i+1}$ then we are in case 3, and otherwise, $l_i \le \tilde{\Delta} < \frac{dm^2}{\varepsilon} l_i$ or $\frac{1}{5} l_{i+1} < \tilde{\Delta} \le l_{i+1}$ and we are in case 4.

We proceed by case analysis:

Case 1: $\tilde{\Delta} < \frac{l_1}{2}$. Then $d_{dF}(P,Q) \le \tilde{\Delta} \le \lambda(P)$, and by Claim 4.3 we return $d_{dF}(P,Q)$.

Case 2: $\tilde{\Delta} > \frac{dm^2}{\varepsilon} \cdot l_{m-1}$. Then $d_{dF}(P,Q) \ge \frac{1}{md} \cdot \tilde{\Delta} > \frac{m}{\varepsilon} \cdot l_{m-1}$. Notice that $d_{dF}(P[1],P) \le \sum_{i=1}^{m-1} l_i \le m \cdot l_{m-1} < \varepsilon \cdot d_{dF}(P,Q)$. By the triangle inequality, we have

$$d_{dF}(P[1],Q) \le d_{dF}(P,Q) + d_{dF}(P[1],P) < (1+\varepsilon)d_{dF}(P,Q)$$

and

$$d_{dF}(P[1],Q) \ge d_{dF}(P,Q) - d_{dF}(P[1],P) > (1-\varepsilon)d_{dF}(P,Q).$$

Case 3: $\frac{dm^2}{\varepsilon} l_i \le \tilde{\Delta} \le \frac{1}{5} l_{i+1}$. Thus,

$$\frac{m}{\varepsilon} l_i \le \frac{\tilde{\Delta}}{dm} \le d_{dF}(P,Q) \le \tilde{\Delta} \le \frac{1}{5} l_{i+1}\ . \tag{1}$$

Denote $P' = (P[j_1], P[j_2], \ldots, P[j_u])$. We argue that $\lambda(P') > \frac{1}{4} l_{i+1}$—that is, for every $s \in [1, u-1]$, $\|P[j_s] - P[j_{s+1}]\| > \frac{1}{2} l_{i+1}$. Fix such an index s. If $j_s = j_{s+1} - 1$, then as $P[j_s] \notin S$, $\|P[j_s] - P[j_{s+1}]\| \ge l_{i+1}$. Otherwise, as $P[j_s] \notin S$ and $\{P[j_s+1], \ldots, P[j_{s+1}-1]\} \subseteq S$, by the triangle inequality,

$$\begin{aligned}\|P[j_s] - P[j_{s+1}]\| &\ge \|P[j_s] - P[j_s+1]\| - \sum_{t=1}^{j_{s+1}-j_s-1} \|P[j_s+t] - P[j_s+t+1]\| \\ &\ge l_{i+1} - m \cdot l_i \overset{(1)}{\ge} l_{i+1} - \frac{\varepsilon}{5} \cdot l_{i+1} > \frac{1}{2} l_{i+1}\ .\end{aligned}$$

Notice that $d_{dF}(P,P') \le m \cdot l_i \overset{(1)}{\le} \varepsilon \cdot d_{dF}(P,Q)$. This is as the cost of the paired walk $\omega = \{(P[1,j_1], P'[1])\} \cup \{(P[j_{s-1}+1, j_s], P'[s]) \mid 2 \le s \le u\}$ is at most $m \cdot l_i$ (again using triangle inequality). Thus, $d_{dF}(P',Q) \le d_{dF}(P,Q) + d_{dF}(P,P') \le (1+\varepsilon)d_{dF}(P,Q)$ and $d_{dF}(P',Q) \ge d_{dF}(P,Q) - d_{dF}(P,P') \ge (1-\varepsilon)d_{dF}(P,Q)$.

Finally, as $d_{dF}(P',Q) \le (1+\varepsilon)d_{dF}(P,Q) \overset{(1)}{\le} \frac{1+\varepsilon}{5} l_{i+1} < \frac{1}{4} l_{i+1} < \lambda(P')$, by Claim 4.3 we can compute $d_{dF}(P',Q)$, which is a $(1+\varepsilon)$-approximation of $d_{dF}(P,Q)$.

Case 4: $\frac{1}{5}l_i \le \tilde{\Delta} \le \frac{dm^2}{\varepsilon}l_i$. Then $\frac{1}{5dm}l_i \le \frac{\tilde{\Delta}}{dm} \le d_{dF}(P,Q) \le \tilde{\Delta} \le \frac{dm^2}{\varepsilon}l_i$. Let $\lfloor \log_{dm} \frac{1}{5dm} \rfloor \le j \le \lfloor \log_{dm} \frac{m}{\varepsilon} \rfloor$ be an index such that $[\frac{\tilde{\Delta}}{dm}, \tilde{\Delta}] \subseteq [l_i \cdot (dm)^j, l_i \cdot (dm)^{j+2}]$. For example, the maximal j such that $l_i \cdot (dm)^j \le \frac{\tilde{\Delta}}{dm}$ will do. By Lemma 4.2, as $d_{dF}(P,Q)$ is in the range, using the bounded range distance oracle $\mathcal{D}_i^j$ we will return an $(1+\varepsilon)$-approximation of $d_{dF}(P,Q)$.

Running Time and Storage Space. Computing $\tilde{\Delta}$ takes $O(md\log(md)\log d)$ time according to Theorem 6. Deciding which case is relevant for our query takes $O(m)$ time. Case 1 takes $O(md)$ time according to Claim 4.3. Case 2 can be computed in $O(md)$ time, as it is simply finding the maximum among m distances, each computed in $O(d)$ time. For case 3, we can compute P' sequentially in $O(md)$ time, then compute $d_{dF}(P',Q)$ in $O(md)$ time using Claim 4.3. For case 4, according to Lemma 4.2, the query time is $O(md\log\log md)$ time. Thus, $O(md\log(md)\log d)$ in total.

Cases 1 through 3 do not require any precomputed values, whereas in case 4 each $\mathcal{D}_i^j$ uses $O(\frac{1}{\varepsilon})^{md} \cdot \log md = O(\frac{1}{\varepsilon})^{md}$ space and $m\log md \cdot (O(\frac{1}{\varepsilon})^{md} + O(d\log m)) = O(\frac{1}{\varepsilon})^{md}$ expected preprocessing time. Thus, the total storage space and running time is $m \cdot \log_{dm}(\frac{5dm^2}{\varepsilon}) \cdot O(\frac{1}{\varepsilon})^{md} = O(\frac{1}{\varepsilon})^{md} \cdot \log \varepsilon^{-1}$. We conclude the following.

Theorem 8. *Given a curve $P \in \mathbb{R}^{d\times m}$ and parameter $\varepsilon \in (0,\frac{1}{4})$, there exists a $(1+\varepsilon)$-distance oracle with $O(\frac{1}{\varepsilon})^{dm} \cdot \log \varepsilon^{-1}$ storage space and preprocessing time, and $\tilde{O}(md)$ query time.*

Remark 4.4. Interestingly, previous work [29] constructed a near neighbor data structure with query time $O(md)$. Combined with the standard reduction [34], they obtain a nearest neighbor data structure with query time $O(md\log n)$ (given that the number of input curves is n). The case of distance oracle is similar with $n = 1$. However, we cannot use the NNS data structure as a black box, as it actually returns a neighbor and not a distance. Obtaining a truly linear query time ($O(md)$) in Theorem 8 is an intriguing open question.

4.4 $(1+\varepsilon)$-Distance Oracle for Any k

Let Π be a $(k, 1+\varepsilon)$-simplification of P, which can be computed in $\tilde{O}(\frac{md}{\varepsilon^{4.5}})$ time using Theorem 7.[5] In addition, we obtain from Theorem 7 an estimate L such that $d_{dF}(P,\Pi) \le L \le (1+\varepsilon)d_{dF}(P,\Pi^*)$, where Π^* is the optimal k-simplification.

We construct a symmetric distance oracle $\mathcal{O}_\Pi$ for Π using Theorem 8. In addition, for every index $i \in [0, \lceil \log \frac{1}{\varepsilon} \rceil]$, we construct an asymmetric bounded range distance oracle $\mathcal{O}_i$ for P, with the range $[2^{i-1} \cdot L, 2^{i+3} \cdot L]$ using Lemma 4.2.[6]

The Query Algorithm. Given a query curve $Q \in \mathbb{R}^{d\times k}$, we query $\mathcal{O}_\Pi$ and get a value Δ such that $d_{dF}(Q,\Pi) \le \Delta \le (1+\varepsilon)d_{dF}(Q,\Pi)$. We continue by case analysis:

- If $\Delta \ge \frac{1}{\varepsilon} \cdot L$, return $(1+\varepsilon)\Delta$.
- Else, if $\Delta \le 7L$, return the answer of $\mathcal{O}_0$ for Q.
- Else, let i be the maximal index such that $2^i \cdot L \le \frac{\Delta}{2}$, and return the answer of $\mathcal{O}_i$ for Q.

Correctness. We show that the query algorithm always returns a value $\tilde{\Delta}$ such that $d_{dF}(P,Q) \le \tilde{\Delta} \le (1+6\varepsilon)d_{dF}(P,Q)$. Afterward, the ε parameter can be adjusted accordingly. By the triangle

[5] If $d \le 4$, then the running time will be $\tilde{O}(\frac{m}{\varepsilon})$. In any case, the contribution of this step to the preprocessing time is insignificant.
[6] Actually, as the aspect ratio is constant, we could equivalently use Lemma 4.1 here with parameters $2^{i+3} \cdot L$ and $\frac{\varepsilon}{16}$ instead of Lemma 4.2.

inequality, it holds that

$$\frac{\Delta}{1+\epsilon} - L \le d_{dF}(Q,\Pi) - d_{dF}(P,\Pi) \le d_{dF}(P,Q) \le d_{dF}(Q,\Pi) + d_{dF}(P,\Pi) \le \Delta + L\,.$$

- If $\Delta \ge \frac{1}{\varepsilon} \cdot L$, then $d_{dF}(P,Q) \le (1+\varepsilon)\Delta$. Furthermore, it holds that $d_{dF}(P,Q) \ge \frac{1}{1+\varepsilon}\Delta - \varepsilon\Delta \ge (1-2\varepsilon)\Delta$, which implies $(1+\varepsilon)\Delta \le \frac{1+\varepsilon}{1-2\varepsilon} \cdot d_{dF}(P,Q) < (1+6\varepsilon) \cdot d_{dF}(P,Q)$.
- Else, if $\Delta \le 7L$, then $d_{dF}(P,Q) \le 8L$. Since Q is a k point curve, we have that $d_{dF}(P,Q) \ge d_{dF}(P,\Pi^*) \ge \frac{L}{1+\varepsilon} > \frac{L}{2}$ (here, Π^* is the optimal k-simplification). Hence, $d_{dF}(P,Q) \in [\frac{L}{2}, 8L]$, and O_0 returns a $(1+\varepsilon)$-approximation for $d_{dF}(P,Q)$.
- Else, $L < \frac{1}{7}\Delta$, hence $\frac{\Delta}{2} \le \frac{\Delta}{1+\epsilon} - L \le d_{dF}(P,Q) \le \Delta + L \le 2\Delta$. Recall that i is chosen to be the maximal index such that $2^i \cdot L \le \frac{\Delta}{2}$. By maximality of i, we have $2\Delta \le 2^{i+3} \cdot L$. Thus, $d_{dF}(P,Q) \in [\frac{\Delta}{2}, 2\Delta] \subseteq [2^i \cdot L, 2^{i+3} \cdot L]$. As $\Delta < \frac{1}{\varepsilon} \cdot L$, it follows that $i \le \log\frac{1}{2\varepsilon}$. In particular, we constructed an asymmetric bounded range distance oracle O_i for P, with the range $[2^{i-1} \cdot L, 2^{i+3} \cdot L]$. It follows that O_i returns a $(1+\varepsilon)$-approximation for $d_{dF}(P,Q)$.

To bound the space, note that we constructed a single distance oracle using Theorem 8 and $\log\frac{1}{\varepsilon}$ distance oracles using Lemma 4.2 (each with constant ratio). Thus, $O(\frac{1}{\varepsilon})^{dk} \cdot \log\varepsilon^{-1}$ in total. Similarly, the expected preprocessing time is bounded by $m\log\frac{1}{\varepsilon} \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d\log m))$. The query time is bounded by $O(kd\log(kd)\log d)$.

Theorem 1. *Given a curve $P \in \mathbb{R}^{d\times m}$ and parameters $\varepsilon \in (0, \frac{1}{4})$ and integer $k \ge 1$, there exists a distance oracle with $O(\frac{1}{\varepsilon})^{dk} \cdot \log\varepsilon^{-1}$ storage space, $m\log\frac{1}{\varepsilon} \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d\log m))$ expected preprocessing time, and $\tilde{O}(kd)$ query time.*

5 CURVE SIMPLIFICATION IN THE STREAM

Given a curve P as a stream, our goal in this section is to maintain a $(k, 1+\varepsilon)$-simplification of P, with space bounds that depend only on k and ε. A main ingredient is to maintain a δ-simplification. For the static model, Bereg et al. [8] presented an algorithm that for fixed dimension computed an optimal δ-simplification in $O(m\log m)$ time. This algorithm was generalized to arbitrary dimension d by the authors and Katz [29] (see Lemma 8.2). The algorithm is greedy: it finds the largest index i such that $P[1,i]$ can be enclosed by a ball of radius $(1+\varepsilon)\delta$ and then recurse for $P[i+1,m]$. The result is a sequence of balls, each with a radius at most $(1+\varepsilon)\delta$. The sequence of centers of the balls is a simplification of P with distance $(1+\varepsilon)\delta$. If the number of balls is larger than k, then $d_{dF}(P,\Pi) > \delta$ for every curve Π with at most k points. The greedy algorithm essentially constructs a one-to-many paired walk along the resulted simplification and P.

Let γ-MEB denote a streaming algorithm that maintains a γ-approximation of the minimum enclosing ball of a set of points. In other words, in each point of time, the algorithm has a center γ-MEB.c $\in \mathbb{R}^{\mathrm{d}}$ and a radius γ-MEB.r $\in \mathbb{R}$ such that all points observed by this time are contained in $B_{\mathbb{R}^d}(\gamma\text{-MEB.c}, \gamma\text{-MEB.r})$, and the minimum enclosing ball of the observed set of points has a radius at least γ-MEB.r$/\gamma$. In Section 5.1, we discuss several streaming MEB algorithms, all for $\gamma \in (1,2]$. For now, we will simply assume that we have such an algorithm as a black box.

In Algorithm 2, we describe a key subprocedure of our algorithm called `GreedyStreamSimp`, which finds a greedy simplification while using γ-MEB as a black box. `GreedyStreamSimp` receives as input a parameter δ and a curve P in a streaming fashion, and returns a simplification Π computed in a greedy manner. Specifically, it looks for the longest prefix of P such that γ-MEB.r $\le \delta$—that is, the radius returned by γ-MEB is at most δ. Then, it continues recursively on the remaining points. Note that there is no bound on the size of the simplification Π, which can have any size between 1 and $|P|$. Nevertheless, we obtain a simplification Π at a distance at most δ from P such that every curve at distance δ/γ from P has length at least Π.

ALGORITHM 2: `GreedyStreamSimp`(P, δ)

input : A curve P, parameter $\delta > 0$, a black box algorithm γ-MEB
output : Simplification Π of P, and a γ-MEB structure for the suffix of P matched to the last point of Π.

Initialize γ-MEB with $\{P[1]\}$
Set $\Pi \leftarrow \gamma$-MEB.c
for $i = 2$ *to* $|P|$ **do**
  Add $P[i]$ to γ-MEB
  if γ-MEB.r $\leq \delta$ **then**
    Change the last point of Π to γ-MEB(c)
  else
    Initialize γ-MEB with $P[i]$
    Set $\Pi \leftarrow \Pi \circ \gamma$-MEB.c
return $(\Pi, \gamma$-MEB$)$

Claim 5.1. *Let* Π *be a simplification computed by* `GreedyStreamSimp` *with parameters* P, δ. *Then* $d_{dF}(P, \Pi) \leq \delta$, *and for any other simplification* Π' *of* P, *if* $d_{dF}(P, \Pi') \leq \frac{\delta}{\gamma}$, *then* $|\Pi| \leq |\Pi'|$.

Proof. First, notice that $d_{dF}(P, \Pi) \leq \delta$ is straightforward, because `GreedyStreamSimp` constructs a one-to-many paired walk ω along Π and P such that for each pair $(\Pi[i], \mathcal{P}_i) \in \omega$, $\mathcal{P}_i$ is contained in a ball with a radius at most δ.

Let Π' be a simplification of P with $d_{dF}(P, \Pi') \leq \frac{\delta}{\gamma}$, and consider an optimal walk ω along Π' and P. If ω is not one-to-many, we remove vertices from Π' until we get a simplification Π'' with $d_{dF}(P, \Pi'') \leq \frac{\delta}{\gamma}$ and an optimal one-to-many walk. Denote by Π''_i the subcurve of Π'' that ω matches to $P[1, i]$ and by A''_i the subsequence of points from $P[1, i]$ matched to the last point of Π''_i. In addition, denote by Π_i and γ-MEB$_\text{i}$ the state of these objects right after we finish processing $P[i]$, and let A_i denote the subset of points from P inserted to γ-MEB$_\text{i}$.

We show by induction on the iteration number, i, that either $|\Pi''_i| > |\Pi_i|$, or $|\Pi''_i| = |\Pi_i|$ and $|A''_i| \geq |A_i|$. For $i = 1$, the claim is trivial. We assume that the claim is true for iteration $i \geq 1$ and prove that it also holds in iteration $i + 1$.

If in iteration i we had $|\Pi''_i| > |\Pi_i|$, then in iteration $i + 1$ either $|\Pi_{i+1}| = |\Pi_i|$ or $|\Pi_{i+1}| = |\Pi_i| + 1$ and $|A_{i+1}| = 1$, so the claim holds.

Thus, assume that in iteration i we had $|\Pi''_i| = |\Pi_i|$ and $|A''_i| \geq |A_i|$. If after adding $P[i+1]$ to A_i we have γ-MEB.r $> \delta$, then the minimum enclosing ball of $A_i \cup P[i+1]$ has a radius larger than $\frac{\delta}{\gamma}$. This means that the minimum enclosing ball of $A''_i \cup P[i+1]$ also has a radius larger than $\frac{\delta}{\gamma}$, and thus the length of both Π''_i and Π_i increase by 1, so $|\Pi''_{i+1}| = |\Pi_{i+1}|$ and $|A''_{i+1}| \geq |A_{i+1}| = 1$.

Else, if γ-MEB.r $\leq \delta$, then the minimum enclosing ball of $A_i \cup P[i+1]$ has a radius at most δ, and $A_{i+1} = A_i \cup P[i+1]$. If $|\Pi''_{i+1}| > |\Pi''_i|$, then we are done because $|\Pi''_i| = |\Pi_i| = |\Pi_{i+1}|$. Else, if $|\Pi''_{i+1}| = |\Pi''_i|$, then $P[i+1]$ is added to both A''_i and A_i, and we get $|\Pi''_{i+1}| = |\Pi_{i+1}|$ and $|A''_{i+1}| \geq |A_{i+1}|$. □

In Algorithm 3, we present our main procedure for the streaming simplification algorithm called `LeapingStreamSimp`. Essentially, this algorithm tries to imitate the `GreedyStreamSimp` algorithm. Indeed, if we would know in advance the distance between P and an optimal simplification Π^* of length k, we could find such a simplification by applying `GreedyStreamSimp` with parameter $\gamma \cdot \delta^*$. However, as $d_{dF}(P, \Pi^*)$ is unknown in advance, `LeapingStreamSimp` tries to guess it. In addition to k and γ-MEB, `LeapingStreamSimp` gets as input the parameters init ≥ 1 and inc ≥ 2. The init is used for the initial guess of $d_{dF}(P, \Pi^*)$, whereas inc is used to update the current guess,

ALGORITHM 3: `LeapingStreamSimp`$(k, \gamma$-MEB, init, inc$)$

input : A curve P in a streaming fashion, parameters $k \in \mathbb{N}$ and init ≥ 1, inc ≥ 2, a black box algorithm γ-MEB
output : Simplification Π of P with at most k points

```
Read P[1, k + 1]                                   // Ignore one of any two equal consecutive points
Set δ ← init · ½ min_{i∈[k]} ||P[i] − P[i + 1]||
Set (Π, γ-MEB) ← GreedyStreamSimp(P, δ)
for i ≥ k + 2 to m do
    Read P[i] and add it to γ-MEB
    if γ-MEB.r ≤ δ then
        Change the last point in Π to γ-MEB(c)
    else
        Initialize γ-MEB with P[i]
        Set Π ← Π ∘ γ-MEB.c
        while |Π| = k + 1 do
            δ ← δ · inc
            Set (Π, γ-MEB) ← GreedyStreamSimp(Π, δ)
return Π
```

when the previous guess turns out to be too small. In more detail, `LeapingStreamSimp` starts by reading the first $k + 1$ points (as up to this point, our simplification is simply the observed curve). At this stage, the optimal simplification of length k is at distance $\frac{1}{2}\min_{i\in[k]}\|P[i] - P[i+1]\|$. The algorithm updates its current guess δ of $d_{dF}(P, \Pi^*)$ to init $\cdot \frac{1}{2}\min_{\mathrm{i}\in[\mathrm{k}]}\|\mathrm{P}[\mathrm{i}] - \mathrm{P}[\mathrm{i}+1]\|$ and executes `GreedyStreamSimp` on the $k + 1$ observed points with parameter δ. Now the `LeapingStreamSimp` algorithm simply imitates `GreedyStreamSimp` with parameter δ as long as the simplification contains at most k points. Once this condition is violated (i.e., our guess turned out to be too small), the guess δ is multiplied by inc. Now we compute a greedy simplification of the current simplification Π using the new parameter δ. This process is continued until we obtain a simplification of length at most k. At this point, we simply turn back to the previous imitation of the greedy simplification. As a result, eventually `LeapingStreamSimp` will hold an estimate δ and a simplification Π such that Π is an actual simplification constructed by `GreedyStreamSimp` with parameter δ. Alas, Π was not constructed with respect to the observed curve P but rather with respect to some other curve P', where $d_{dF}(P, P') < \frac{2}{\mathrm{inc}}\delta$ (Claim 5.2). Furthermore, the estimate δ will be bounded by the distance to the optimal simplification δ^*, times $\approx \gamma \cdot$ inc (Lemma 5.3).

In the analysis of the algorithm, by Π_i and δ_i we refer to the state of the algorithm right after we finish processing $P[i]$.

Claim 5.2. *After reading m points, $d_{dF}(\Pi_m, P[1,m]) \leq (1 + \frac{2}{\mathrm{inc}})\delta_m$. Moreover, there exists a curve P' such that Π_m is the simplification returned by* `GreedyStreamSimp` *for the curve P' and parameter δ_m, and $d_{dF}(P', P[1,m]) \leq \frac{2\delta_m}{\mathrm{inc}}$.*

Proof. The first part of the claim is a corollary that follows from the second part. Indeed, by Claim 5.1, we have $d_{dF}(\Pi_m, P') \leq \delta_m$, and by the triangle inequality,

$$d_{dF}(\Pi_m, P[1,m]) \leq d_{dF}(\Pi_m, P') + d_{dF}(P', P[1,m]) \leq \left(1 + \frac{2}{\mathrm{inc}}\right)\delta_m.$$

We prove the second part by induction on m. For $m = k + 1$, the claim is clearly true for $P' = P[1, k + 1]$. Assume that the claim is true for $m - 1 \geq k + 1$, so by the induction hypothesis there exists a curve P' such that $d_{dF}(P', P[1, m-1]) \leq \frac{2\delta_{m-1}}{\mathrm{inc}}$, and Π_{m-1} is the simplification returned by `GreedyStreamSimp` for the curve P' and parameter δ_{m-1}.

If there is no leap step, then $\delta_m = \delta_{m-1}$. Let $P'' = P' \circ P[m]$, then $d_{dF}(P'', P[1, m]) \leq \frac{2\delta_{m-1}}{\mathrm{inc}} = \frac{2\delta_m}{\mathrm{inc}}$. Since in this case `LeapingStreamSimp` imitates the steps of `GreedyStreamSimp`, we get that Π_m will be exactly the simplification returned by `GreedyStreamSimp` for the curve P'' and parameter δ_m.

Else, if a leap step is taken, then $\delta_m = \delta_{m-1} \cdot \mathrm{inc}^{\mathrm{h}}$ for some $h \geq 1$, so $\delta_{m-1} = \frac{\delta_m}{\mathrm{inc}^{\mathrm{h}}} \leq \frac{\delta_m}{\mathrm{inc}}$. By the first part of the claim, we have $d_{dF}(\Pi_{m-1}, P[1, m-1]) \leq (1 + \frac{2}{\mathrm{inc}})\delta_{m-1}$.

Let $P'' = \Pi_{m-1} \circ P[m]$, then for $\mathrm{inc} \geq 2$,

$$d_{dF}(P'', P[1, m]) \leq d_{dF}(\Pi_{m-1}, P[1, m-1]) \leq \left(1 + \frac{2}{\mathrm{inc}}\right)\delta_{m-1} \leq \left(1 + \frac{2}{\mathrm{inc}}\right)\frac{\delta_m}{\mathrm{inc}} \leq \frac{2\delta_m}{\mathrm{inc}}.$$

The claim follows as the algorithm sets Π_m to be the simplification returned by `GreedyStreamSimp` on the curve P'' and parameter δ_m. ◻

Consider an optimal k-simplification Π^*_m of $P[1, m]$, and denote $\delta^*_m = d_{dF}(\Pi^*_m, P[1, m])$. We will assume that $\mathrm{inc} > 2\gamma$. Set $\eta = \frac{\gamma\,\mathrm{inc}}{\mathrm{inc} - 2\gamma}$.

Lemma 5.3. *Let h be the minimal such that $\eta \cdot \delta^*_m \leq \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$. Then, $\delta_m \leq \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$.*

Proof. Assume by contradiction that $\delta_m > \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$, and let i be the minimum index such that $\delta_i > \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$. Then, when reading the ith point, the algorithm performs a leap step (otherwise, $\delta_i = \delta_{i-1} \leq \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$).

By Claim 5.2, there exists a curve P' such that Π_{i-1} is the simplification returned by `GreedyStreamSimp` for the curve P' and parameter δ_{i-1}, and $d_{dF}(P', P[1, i-1]) \leq \frac{2\delta_{i-1}}{\mathrm{inc}}$.

Consider the time when the algorithm sets $\delta \leftarrow \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$. Since $\delta_i > \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$, the algorithm calls `GreedyStreamSimp` with the curve $P' \circ P[i]$ and parameter $\delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$, and gets a simplification of length $k+1$ (otherwise, $\delta_i \leq \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$).

Consider an optimal k-simplification $\tilde{\Pi}$ of the curve $P[1, i]$ with distance $\delta^*_m = d_{dF}(\tilde{\Pi}, P[1, i])$. By the triangle inequality,

$$d_{dF}(\tilde{\Pi}, P' \circ P[i]) \leq d_{dF}(\tilde{\Pi}, P[1, i]) + d_{dF}(P[1, i], P' \circ P[i]) \leq \delta^*_m + \frac{2\delta_{i-1}}{\mathrm{inc}}.$$

Therefore, by Claim 5.1, the simplification returned by `GreedyStreamSimp` for the curve $P' \circ P[i]$ with parameter $\gamma(\delta^*_m + \frac{2\delta_{i-1}}{\mathrm{inc}})$ has length at most k. But by the minimality of i,

$$\begin{aligned}\gamma\left(\delta^*_m + \frac{2\delta_{i-1}}{\mathrm{inc}}\right) &\leq \frac{\gamma}{\eta}\delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}} + \frac{2\gamma}{\mathrm{inc}}\delta_{\mathrm{k}+1} \cdot \mathrm{inc}^{\mathrm{h}} \\ &= \left(\frac{\mathrm{inc} - 2\gamma}{\mathrm{inc}} + \frac{2\gamma}{\mathrm{inc}}\right) \cdot \delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}} = \delta_{\mathrm{k}+1} \cdot \mathrm{inc}^{\mathrm{h}}.\end{aligned}$$

This contradicts the fact that `GreedyStreamSimp` returns a simplification of length $k+1$ when δ is set to $\delta_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$. ◻

We are now ready to prove the main theorem.

Theorem 2. *Suppose that we are given a black box streaming algorithm γ-MEB for $\gamma \in [1, 2]$ that uses storage space $S(d, \gamma)$. Then for every parameter $\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{\log \varepsilon^{-1}}{\varepsilon} \cdot (S(d, \gamma) + kd))$ space, and given a curve P in $\mathbb{R}^d$ in a streaming fashion computes a $(k, \gamma(1+\varepsilon))$-simplification Π of P and a value L such that $d_{dF}(\Pi, P) \leq L \leq \gamma(1+\varepsilon)\delta^*$.*

Proof. The algorithm is quite simple: for every $i \in [1, \lceil \log_{(1+\varepsilon)} \frac{1}{\varepsilon} \rceil]$, run the algorithm LeapingStreamSimp$(k, \gamma\text{-MEB}, (1+\varepsilon)^{\mathrm{i}}, \frac{1}{\varepsilon})$—that is, LeapingStreamSimp with parameters init $= (1+\varepsilon)^{\mathrm{i}}$ and inc $= \frac{1}{\varepsilon}$. After observing the curve $P[1,m]$, the ith instance of the algorithm will hold a simplification $\Pi_{m,i}$ and distance estimation $\delta_{m,i}$. The algorithm finds the index $i_{\min}$ for which $\delta_{m,i}$ is minimized and returns $\Pi_{m,i}$ with $L = (1 + \frac{2}{\mathrm{inc}})\delta_{m,i}$.

Note that the space required for each copy of LeapingStreamSimp is $S(d,\gamma) + O(kd)$, as in each iteration it simply holds a single version of γ-MEB and at most k points of the current simplification. Thus, the space guarantee holds. Recall that the optimal simplification is denoted by Π^*_m, where $d_{dF}(P[1,m], \Pi^*_m) = \delta^*_m$. We will argue that $d_{dF}(P[1,m], \Pi_{m,i_{\min}}) = \gamma(1 + O(\varepsilon))\delta^*_m$. Afterward, the ε parameter can be adjusted accordingly.

We can assume that $m > k$, as otherwise we can simply return the observed curve P (and $L = 0$). Set $\delta_{\min} = \lambda[1, k+1] = \delta^*_{k+1}$.[7] First, note that as Π^*_m contains at most k points, it follows that $\delta^*_m \geq \delta_{\min}$. Hence, there are indices $1 \leq j \leq \lceil \log_{(1+\varepsilon)} \frac{1}{\varepsilon} \rceil$ and $h \geq 0$ such that

$$(1+\varepsilon)^{j-1} \left(\frac{1}{\varepsilon}\right)^h \delta_{\min} < \eta \cdot \delta^*_m \leq (1+\varepsilon)^j \left(\frac{1}{\varepsilon}\right)^h \delta_{\min} .$$

Note that for this particular j, we have that $(1+\varepsilon)^j (\frac{1}{\varepsilon})^h \delta_{\min} = \delta^j_{k+1} \cdot \mathrm{inc}^{\mathrm{h}}$. Hence, by Lemma 5.3, we have that $\delta_{m,j} \leq (1+\varepsilon)^j (\frac{1}{\varepsilon})^h \delta_{\min} \leq (1+\varepsilon) \cdot \eta \cdot \delta^*_m$. Using Claim 5.2, we have that

$$d_{dF}(P[1,m], \Pi_{m,i_{\min}}) \leq \left(1 + \frac{2}{\mathrm{inc}}\right) \delta_{m,i_{\min}} \leq (1+2\varepsilon)\delta^j_m \leq (1+4\varepsilon) \cdot \eta \cdot \delta^*_m = (1+O(\varepsilon)) \cdot \gamma \cdot \delta^*_m , \quad (2)$$

where the last step follows as $\eta = \frac{\gamma \mathrm{inc}}{\mathrm{inc} - 2\gamma} = \gamma \cdot \frac{1}{1-2\gamma\varepsilon} = (1+8\varepsilon) \cdot \gamma$ for[8] $\gamma \leq 2$ and $\varepsilon \leq \frac{1}{8}$. □

5.1 Approximating the Minimum Enclosing Ball

Computing the minimum enclosing ball of a set of points in Euclidean space is a fundamental problem in computational geometry. In the static setting, Megiddo [40] showed how to compute an MEB in $O(n \log n)$ time (for fixed dimension d), whereas Kumar et al. [38] provided a static $(1+\varepsilon)$-MEB algorithm running in $O(\frac{nd}{\varepsilon} + \varepsilon^{-4.5} \log \frac{1}{\varepsilon})$ time.

A very simple 2-MEB data structure in the streaming setting can be constructed as follows. Let x be the first observed point, and set 2-MEB.c $\leftarrow$ x and 2-MEB.r $\leftarrow$ 0. For each point y in the remainder of the stream, set 2-MEB.r $\leftarrow \max\{$2-MEB.r, $\|\mathrm{x} - \mathrm{y}\|\}$. In other words, this algorithm simply computes the distance from x to its farthest point from the set. The approximation factor is 2 because any ball that encloses x and its farthest point y has a radius at least $\|x - y\|/2$. The space used by this algorithm is clearly $O(d)$. In addition, notice that the center of the ball in this algorithm is a point from the stream. This means that when using this 2-MEB in our streaming simplification algorithm, we obtain a simplification Π with points from P. This is sometimes a desirable property (like in applications from computational biology, e.g., see [8]). Moreover, using Theorem 2, we obtain a $(2+\varepsilon)$ approximation factor, which is close to optimal because a vertex-restricted k-simplification is a 2-approximation for an optimal (nonrestricted) k-simplification of a given curve.

Corollary 5.4. *For every parameter $\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{\log \varepsilon^{-1}}{\varepsilon} \cdot kd)$ space, and given a curve P in $\mathbb{R}^d$ in a streaming fashion computes a vertex-restricted $(k, 2+\varepsilon)$-simplification Π of P.*

[7] Recall that $\lambda[1, k+1] = \frac{1}{2} \min_{i \in [k]} \|P[i] - P[i+1]\|$.
[8] For $\gamma = 1 + \varepsilon$ and $\varepsilon \leq \frac{1}{4}$, we will obtain $\eta \leq (1+6\varepsilon)\gamma$.

For a better approximation factor (using a simplification with arbitrary vertices), we can use the following γ-MEB algorithms. Chan and Pathak [16] (improving over Agarwal and Sharathkumar [4]) constructed a γ-MEB algorithm for $\gamma = 1.22$, also using $O(d)$ space. We conclude the following.

Corollary 5.5. *For every parameter $\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{\log \varepsilon^{-1}}{\varepsilon} \cdot kd)$ space, and given a curve P in $\mathbb{R}^d$ in a streaming fashion computes an $(k, 1.22 + \varepsilon)$-simplification Π of P.*

Finally, as was observed by Chan and Pathak [16], using streaming techniques for ε-kernels [45], for every $\varepsilon \in (0, \frac{1}{2})$ there is a $(1 + \varepsilon)$-MEB algorithm that uses $O(\varepsilon^{-\frac{d-1}{2}} \log \frac{1}{\varepsilon})$ space.

Lemma 5.6. *For every parameter $\varepsilon \in (0, \frac{1}{2})$, there is a $(1 + \varepsilon)$-MEB algorithm that uses $O(\varepsilon)^{-\frac{d-1}{2}} \log \frac{1}{\varepsilon}$ space.*

As we rely heavily on Lemma 5.6 and are not aware of a published proof, we attach a proof sketch in Appendix A.2.

Corollary 5.7. *For every parameter$\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\varepsilon)^{-\frac{d+1}{2}} \log^2 \varepsilon^{-1} + O(kd\varepsilon^{-1} \log \varepsilon^{-1})$ space, and given a curve P in $\mathbb{R}^d$ in a streaming fashion computes an $(k, 1 + \varepsilon)$-simplification Π of P.*

6 DISTANCE ORACLE: THE STREAMING CASE

Similarly to our static distance oracle, we first describe a construction (this time in the streaming model) of a data structure that stores a (k, r, ε)-cover C of size $O(\frac{1}{\varepsilon})^{kd}$ for P and has a linear look-up time. Then, we show how to combine several of those structures (together with a streaming simplification) to produce a streaming distance oracle.

6.1 Cover of a Curve

This entire subsection is dedicated to proving the following lemma.

Lemma 6.1. *Given parameters $r \in \mathbb{R}_+$, $\varepsilon \in (0, \frac{1}{4})$, and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{1}{\varepsilon})^{kd}$ space, and given a curve P in $\mathbb{R}^d$ constructs a decision distance oracle with $O(kd)$ query time.*

The cover that we describe in the following is an extended version of the cover described in Section 4.1, which also contains grid curves of length smaller than k. Those smaller curves will allow us to update the cover when a new point of P is discovered, when all we have is the new point and the previous cover. More precisely, we store a set of covers $C_i = \{C_{i,k'}\}_{1 \le k' \le k}$ for $P[1, i]$ such that $C_{i,k'}$ is a (k', r, ε)-cover for $P[1, i]$ with grid curves, exactly as we constructed for the static case. In other words, it contains exactly the set of all curves W with k' points from $\mathcal{G}_i = \bigcup_{1 \le t \le i} G_{\varepsilon,r}(P[t], (1 + \varepsilon)r)$ and $d_{dF}(P[1, i], W) \le (1 + \varepsilon)r$. We call such a cover a *(k', r, ε)-grid-cover.*

Algorithm 5 (`ExtendCover`) is a subroutine that constructs the set of covers C_i for $P[1, i]$, given only the set C_{i-1} (for $i \ge 2$) and the new point $P[i]$. Algorithm 4 (`StreamCover`) is the streaming algorithm that first reads $P[1]$ and constructs a set $C_1 = \{C_{1,k'}\}_{1 \le k' \le k}$ such that $C_{1,k'}$ is a (k', r, ε)-grid-cover for $P[1]$,[9] then calls `ExtendCover` for each new observed point.

Assume that $C' = \{C_{i-1,j}\}_{1 \le k' \le k}$ such that $C_{i-1,k'}$ is a (k', r, ε)-grid-cover for $P[1, i-1]$. We show that given the point $P[i]$ and C', Algorithm 5 outputs a set $C = \{C_{i,k'}\}_{1 \le k' \le k}$ such that $C_{i,k'}$ is a (k', r, ε)-grid-cover for $P[1, i]$.

[9]This is the set of all curves with at most k points from $G_{\varepsilon,r}(P[1], (1 + \varepsilon)r)$, with their distance to $P[1]$.

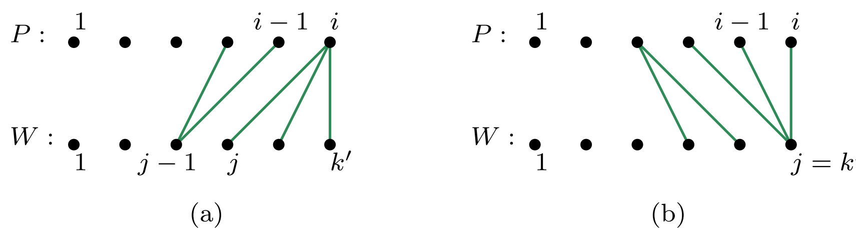


Fig. 1. Constructing $C_{i,k'}$ from $\{C_{i-1,j}\}_{1\le j\le k}$.

ALGORITHM 4: `StreamCover`(P, k, r, ε)

input : A curve P, parameters $r > 0, k \in \mathbb{N}, \varepsilon \in (0, \frac{1}{4})$
output: A (k, r, ε)-cover C for P

Read $P[1]$.
Construct a set $C_1 = \{C_{1,k'}\}_{1\le k'\le k}$ such that $C_{1,k'}$ is a (k', r, ε)-grid-cover for $P[1]$.[9]
Set $i \leftarrow 2$
while *read* $P[i]$ **do**
    Set $C_i \leftarrow$`ExtendCover`$(C_{i-1}, P[i], k, r, \varepsilon)$
    Set $i \leftarrow i + 1$ and delete C_{i-1}
Return C_i

ALGORITHM 5: `ExtendCover`$(C', p, k, r, \varepsilon)$

input : A (k, r, ε)-cover C' for some (nonempty) curve P', a point p, parameters $r > 0, k \in \mathbb{N}, \varepsilon \in (0, \frac{1}{4})$
output: A (k, r, ε)-cover C for $P' \circ p$

Set $C \leftarrow \emptyset$ and $\tilde{C} \leftarrow \emptyset$
Construct the set $\tilde{C}$ of all curves with at most k points from $G_{\varepsilon,r}(p, (1+\varepsilon)r)$.
for *each* $W \in C'$ *of length* $j < k$ **do**
    for *each* $X \in \tilde{C}$ *of length* $k' \le k - j$ **do**
        Set dist $\leftarrow \max\{C'.\mathrm{dist}(W), \max_{1\le t\le k'}\{\|X[t] - p\|\}\}$
        insert $(W \circ X, \text{dist})$ into C `// if` $W \circ X \in C$`, keep the entry with minimal dist`
for *each* $W \in C'$ *of length* k' ***such that*** $W[k'] \in G_{\varepsilon,r}(p, (1+\varepsilon)r)$ **do**
    Set dist $\leftarrow \max\{C'.\mathrm{dist}(W), \|W[k'] - p\|\}$
    insert (W, dist) into C `// if` $W \in C$`, keep the entry with minimal dist`
Return C

Let W be a curve with points from $\mathcal{G}_i = \bigcup_{1\le t\le i} G_{\varepsilon,r}(P[t], (1+\varepsilon)r)$ such that $d_{dF}(P[1,i], W) \le (1+\varepsilon)r$. Consider an optimal walk along W and $P[1,i]$, and let $j \le k'$ be the smallest index such that $P[i]$ is matched to $W[j]$. Notice that $W[j,k']$ is contained in $G_{\varepsilon,r}(P[i], (1+\varepsilon)r)$, and thus $W[j,k']$ is in $\tilde{C}$.

If $P[i-1]$ is matched to $W[j-1]$ (Figure 1(a)), then $d_{dF}(P[1,i-1], W[1,j-1]) \le (1+\varepsilon)r$, and by the induction hypothesis, $C_{i-1,j-1}$ contains $W[1,j-1]$, with the value $C'.\mathrm{dist}(W[1,j-1]) = \mathrm{d_{dF}}(P[1,i-1], W[1,j-1])$. Moreover, $d_{dF}(P[1,i], W) = \max\{d_{dF}(P[1,i-1], W[1,j-1]), d_{dF}(P[i], W[j,k'])\}$, and indeed the algorithm inserts $W = W[1,j-1] \circ W[j,k']$ to C with the distance $\max\{C'.\mathrm{dist}(W'[1,j-1]), \mathrm{d_{dF}}(P[i], W[j,k'])\}$.

Else, if $P[i-1]$ is matched to $W[j]$ (see Figure 1(b)). Note that in this case, it must be that $j = k'$ because $P[i]$ is also matched to $W[j]$. Then, $d_{dF}(P[1, i-1], W[1, k']) \leq (1+\varepsilon)r$, and by the induction hypothesis, $C_{i-1,k'}$ contains W, with the value $C'.\mathrm{dist}(\mathrm{W}) = \mathrm{d_{dF}(P[1, i-1], W)}$. This time, $d_{dF}(P[1, i], W) = \max\{d_{dF}(P[1, i-1], W), \|W[k'] - P[i]\|\}$, and indeed the algorithm inserts W to C with the distance $\max\{C'.\mathrm{dist(W')}, \mathrm{\|W[k'] - P[i]\|}\}$.

The other direction (showing that if W is in C, then W is a grid curve with at most k' points and $d_{dF}(P[1, i], W) \leq (1+\varepsilon)r$) can be proven by reversing the arguments.

The space required for our algorithm is bounded by size of the set $\{C_{i,k'}\}_{1\leq k'\leq k}$. Since each $C_{i,k'}$ is exactly a (k', r, ε)-grid-cover as described in Section 4.1, we have $|C_{i,k'}| = O(\frac{1}{\varepsilon})^{k'd}$, and the total space is bounded by $\sum_{k'=1}^{k} |C_{i,k'}| \leq k \cdot O(\frac{1}{\varepsilon})^{kd} = O(\frac{1}{\varepsilon})^{kd}$.

The query algorithm remains the same. Given a query $Q \in \mathbb{R}^k \times d$, we first compute a rounded curve W' as in Section 4.1 (where $W'[i]$ is the closest grid point to $Q[i]$). If W' is in the cover, we return W' and $\mathrm{dist(W')}$; otherwise, we return NO. The query time is thus $O(kd)$.

6.2 Cover with Growing Values of *r*

In the static scenario (Section 4.4), we used a set of bounded range distance oracles, each containing a set of covers (decision distance oracles) of the input curve P. However, we chose the ranges with respect to the distance L between P and a $(k, 1+\varepsilon)$-simplification Π of P, and in the streaming scenario this distance can increase in each round.

The `StreamCover` algorithm presented in the previous subsection maintains a (k, r, ε)-cover of P for some given initial value r that does not change. However, as more points of P are read, it might be that the cover becomes empty (which also means that L becomes larger than r). Therefore, Algorithm 6 (`LeapingStreamCover`), presented in this subsection, simulates `StreamCover` until the cover becomes empty. Then, it increases r by some given factor (similarly to the leap step in Algorithm 3), recomputes the cover for the new value r, and continues simulating `StreamCover` for the new cover and r. Note that as in Algorithm 3, a leaping step can occur several times before the algorithm moves on to the next point of P. Nevertheless, by first computing a simplification, we can actually compute how many leap steps are required without performing them all.

We start by reading the first $k+1$ points and assume that there are no two consecutive identical points in $P[1, k+1]$ (otherwise, ignore the duplicate and continue reading until observing $k+1$ points without counting consecutive duplicates). Up until this point, we can simply compute a cover as we did in the static case.

Following the notation in the previous section, denote by Π_m^* an optimal k-simplification of $P[1, m]$, and let $\delta_m^* = d_{dF}(P[1, m], \Pi_m^*)$. Denote by r_m the value of r at the end of round m (i.e., when $P[m]$ is the last point read by `StreamCover`, right before reading $P[m+1]$).

In addition to ε and k, the input for Algorithm 6 contains two parameters: $\mathrm{init} > 0$ and $\mathrm{inc} \geq 2$. The parameter init is the initial value of r, and inc is the leaping factor by which we multiply r when the cover becomes empty.

Our goal is to maintain a set of covers with r_m values that are not too far from δ_m^*. For this, we run our algorithm with $\mathrm{inc} = 2^{\mathrm{t}}$ for the minimum integer t such that $2^t \geq \frac{25}{\varepsilon}$, and $\mathrm{init} = 2^{\mathrm{i}}$ for some $i \in [0, t-1]$. Notice that $t = \log\frac{1}{\varepsilon} + O(1)$. The intuition is that to get a good estimation for the true δ_m^*, we will run t instances of our algorithm, with initial r values growing exponentially between $\delta_{k+1}^* = \lambda(P[1, k+1])$ and $\mathrm{inc} \cdot \delta_{\mathrm{k+1}}^*$. Once an instance fails (i.e., its cover becomes empty), the r value is multiplied by inc until the cover becomes nonempty. Roughly speaking, if h is the number of times we had to multiply the initial value init by inc so that the cover is nonempty, then $\mathrm{inc^{h-1}} \cdot \mathrm{init} \cdot \delta_{\mathrm{k+1}}^* \lesssim \delta_{\mathrm{m}}^* \lesssim \mathrm{inc^h} \cdot \mathrm{init} \cdot \delta_{\mathrm{k+1}}^*$ (because otherwise Π_m^* is evidence that the cover is not empty after $h-1$ multiplications), and thus $\mathrm{inc^h} \cdot \mathrm{init} \cdot \delta_{\mathrm{k+1}}^* \in \delta_{\mathrm{m}}^* \cdot [\Theta(1), \mathrm{O}(\frac{1}{\varepsilon})]$.

ALGORITHM 6: LeapingStreamCover$(P, k, \varepsilon, \text{init}, \text{inc})$

input : A curve P, parameters $k \in \mathbb{N}, \varepsilon \in (0, \frac{1}{2})$, init > 0, inc ≥ 2
output: A (k, r, ε)-cover C for P for some $r \in \delta^* \cdot [\frac{1}{1+2\varepsilon}, (1+2\varepsilon) \cdot \text{inc}]$

Read $P[1, k+1]$.
Set $r \leftarrow \text{init} \cdot \lambda(\mathrm{P}[1, \mathrm{k}+1])$
Construct the set C_{k+1} of (k', r, ε)-grid-covers for $P[1, k+1]$, for $1 \leq k' \leq k$.
Set $i \leftarrow k+2$
while *read* $P[i]$ **do**
  Set $C_i \leftarrow$ExtendCover$(C_{i-1}, P[i], k, r, \varepsilon)$
  while $C_i = \emptyset$ **do**
    Let $W \in C_{i-1}$ be an arbitrary curve
    Set $r \leftarrow r \cdot \text{inc}$
    Set $C_i \leftarrow$StreamCover$(W \circ P[i], k, r, \varepsilon)$
  Set $i \leftarrow i+1$ and delete C_{i-1}
Return $C = C_i$

Lemma 6.2. *At the end of round m, $r_m \in \delta^*_m \cdot [\frac{1}{1+2\varepsilon}, (1+2\varepsilon) \cdot \text{inc}]$. Moreover, there exists a curve P' such that C_m is a (k, r_m, ε)-grid-cover for P' and $d_{dF}(P', P[1, m]) \leq \frac{2}{\text{inc}} \cdot r_m$.*

Proof. The proof is by induction on m. For the base case, $m = k+1$; note that after reading $P[1, k+1]$, we have $r_{k+1} = \text{init} \cdot \delta^*_{\mathrm{k}+1} \in \delta^*_{\mathrm{k}+1} \cdot [1, \frac{\text{inc}}{2}] \subseteq \delta^*_{\mathrm{m}} \cdot [\frac{1}{1+2\varepsilon}, (1+2\varepsilon) \cdot \text{inc}]$, and C_{k+1} is a $(k, r_{k+1}, \varepsilon)$-grid-cover for $P' = P[1, k+1]$.

For the induction step, suppose that C_{m-1} is a $(k, r_{m-1}, \varepsilon)$-grid-cover for a curve P' where $r_{m-1} \in \delta^*_{m-1} \cdot [\frac{1}{1+2\varepsilon}, (1+2\varepsilon) \cdot \text{inc}]$ and $d_{dF}(P', P[1, m-1]) \leq \frac{2}{\text{inc}} \cdot r_{m-1}$.

If there is no leap step in round m, then $r_m = r_{m-1}$, and as shown in the previous subsection, ExtendCover returns a (k, r_m, ε)-cover C_m for the curve $P' \circ P[m]$. We claim that the induction hypothesis holds with respect to $P' \circ P[m]$. Clearly, $d_{dF}(P' \circ P[m], P[1, m]) \leq d_{dF}(P', P[1, m-1]) \leq \frac{2}{\text{inc}} \cdot r_{m-1} = \frac{2}{\text{inc}} \cdot r_m$. Next, note that $r_m = r_{m-1} \leq (1+2\varepsilon) \cdot \text{inc} \cdot \delta^*_{\mathrm{m}-1} \leq (1+2\varepsilon) \cdot \text{inc} \cdot \delta^*_{\mathrm{m}}$, because $\delta^*_{m-1} \leq \delta^*_m$. Finally, as there was no leap step, there is some curve $W \in C_m$ such that $d_{dF}(W, P' \circ P[m]) \leq (1+\varepsilon) r_m$. By the triangle inequality,

$$\begin{aligned}\delta^*_m \leq d_{dF}(W, P[1, m]) &\leq d_{dF}(W, P' \circ P[m]) + d_{dF}(P' \circ P[m], P[1, m]) \\ &\leq (1+\varepsilon) r_m + \frac{2}{\text{inc}} \cdot r_m \leq (1+2\varepsilon) r_m.\end{aligned}$$

We conclude that $r_m \in \delta^*_m \cdot [\frac{1}{1+2\varepsilon}, (1+2\varepsilon) \cdot \text{inc}]$.

Next, we consider the case where round m is a leap step. As we performed a leap step, $C_m = \emptyset$, so there is no grid curve at distance at most $(1+\varepsilon) r_{m-1}$ from $P' \circ P[m]$. Therefore, it follows from the triangle inequality that there is no curve at distance r_{m-1} from $P' \circ P[m]$, as by rounding any curve we get a grid curve within distance εr_{m-1} from it. In particular,

$$\begin{aligned}\delta^*_m = d_{dF}(\Pi^*_m, P[1, m]) &\geq d_{dF}(\Pi^*_m, P' \circ P[m]) - d_{dF}(P' \circ P[m], P[1, m]) \\ &\geq r_{m-1} - \frac{2}{\text{inc}} r_{m-1} = \left(1 - \frac{2}{\text{inc}}\right) \cdot r_{m-1}. \quad (3)\end{aligned}$$

Let W_{m-1} be an arbitrary grid curve at distance $(1+\varepsilon) r_{m-1}$ from P' chosen by the algorithm. The algorithm chooses $r_m = r_{m-1} \cdot \text{inc}^{\text{h}}$ such that $h \geq 1$ is the minimal integer so that there is a grid curve W_m of length k at distance $(1+\varepsilon) r_m$ from $W_{m-1} \circ P[m]$. The algorithm then constructs a (k, r_m, ε)-grid-cover for $W_{m-1} \circ P[m]$. It holds that

$$d_{dF}(W_{m-1} \circ P[m], P[1,m]) \leq d_{dF}(W_{m-1}, P') + d_{dF}(P', P[1, m-1])$$
$$\leq (1+\varepsilon) r_{m-1} + \frac{2}{\text{inc}} \cdot r_{m-1} \leq 2 \cdot r_{m-1} \leq \frac{2}{\text{inc}} r_m . \tag{4}$$

It remains to prove that r_m is in $\delta^*_m \cdot [\frac{1}{1+2\varepsilon}, (1+2\varepsilon) \cdot \text{inc}]$. First,

$$\begin{aligned}(1+\varepsilon) r_m &\geq d_{dF}(W_m, W_{m-1} \circ P[m]) \\ &\geq d_{dF}(W_m, P[1,m]) - d_{dF}(P[1,m], P' \circ P[m]) - d_{dF}(P' \circ P[m], W_{m-1} \circ P[m]) \\ &\geq \delta^*_m - \frac{2}{\text{inc}} \cdot r_{m-1} - (1+\varepsilon) \cdot r_{m-1}.\end{aligned}$$

The third inequality holds by the induction hypothesis, and the fact that every length k curve is at distance at least δ^*_m from $P[1,m]$. It follows that $\delta^*_m \leq \left(1 + \varepsilon + \frac{3+\varepsilon}{\text{inc}}\right) \cdot r_m \leq (1+2\varepsilon) \cdot r_m$.

For the second bound, we continue by case analysis. If $h = 1$, then $r_m = \text{inc} \cdot \mathrm{r_{m-1}}$, and by Equation (3), $\delta^*_m \geq (1 - \frac{2}{\text{inc}}) \cdot r_{m-1} \geq \frac{1-\varepsilon}{\text{inc}} \cdot r_m$. Thus, $r_m \leq (1+2\varepsilon) \cdot \text{inc} \cdot \delta^*_{\mathrm{m}}$. Else, $h \geq 2$ and thus $r_{m-1} \cdot \text{inc}^2 \leq \mathrm{r_m}$. It follows that there is no grid curve of length k at distance $(1+\varepsilon) \cdot \frac{r_m}{\text{inc}}$ from $W_{m-1} \circ P[m]$. In particular, there is no length k curve at distance $\frac{r_m}{\text{inc}}$ from $W_{m-1} \circ P[m]$. Hence,

$$\delta^*_m = d_{dF}(\Pi^*_m, P[1,m]) \geq d_{dF}(\Pi^*_m, W_{m-1} \circ P[m]) - d_{dF}(W_{m-1} \circ P[m], P[1,m])$$
$$\overset{(4)}{\geq} \frac{r_m}{\text{inc}} - 2 \cdot r_{m-1} \geq \frac{r_m}{\text{inc}} - \frac{2}{\text{inc}^2} \cdot r_m \geq (1-\varepsilon) \cdot \frac{r_m}{\text{inc}} ,$$

implying $r_m \leq (1+2\varepsilon) \cdot \text{inc} \cdot \delta^*_{\mathrm{m}}$. The lemma now follows. □

6.3 General Distance Oracle

The high-level approach that we use here is similar to that in Section 4.4, except the simplification and the oracles have to be computed in a streaming fashion. The main challenge therefore is that the value $L \approx d_{dF}(P, \Pi)$, on which the entire construction of Section 4.4 relys upon, is unknown in advance.

The objects stored by our streaming algorithm in round m are as follows: an approximated k-simplification Π_m of the observed input curve $P[1,m]$, with a value L_m (from Theorem 2), a distance oracle $\mathcal{O}_{\Pi_m}$ for Π_m (from Theorem 8), and a set of $O(\log \frac{1}{\varepsilon})$ covers of $P[1,m]$ computed by the `LeapingStreamCover` algorithm.

For the sake of simplicity, we assume that there are no two equal consecutive points among the first $k+1$ points in the data stream (as we can just ignore such redundant points).

The Algorithm. First, using Corollary 5.7, we maintain a $(k, 1+\varepsilon)$-simplification Π_m of $P[1,m]$ with a value L_m such that

$$\delta^*_m \leq d_{dF}(P[1,m], \Pi_m) \leq L_m \leq (1+\varepsilon)\delta^*_m \leq (1+\varepsilon) d_{dF}(P[1,m], \Pi_m) , \tag{5}$$

where the first and last inequalities follow as δ^*_m is the minimal distance from $P[1,m]$ to any curve of length k. In addition, at the end of each round, using Theorem 8, we will compute a static $(1+\varepsilon)$-distance oracle $\mathcal{O}_{\Pi_m}$ for Π_m.

Second, let t be the minimum integer such that $2^t \geq \frac{25}{\varepsilon}$. As in the previous subsection, we set $\text{inc} = 2^{\mathrm{t}}$ and $\text{init}_{\mathrm{i}} = 2^{\mathrm{i}}$ for $i \in [0, t-1]$. Then we run t instances of `LeapingStreamingDecision` simultaneously: for every $i \in [0, t-1]$, we run `LeapingStreamingDecision`$(P, k, \varepsilon, \text{init}_{\mathrm{i}}, \text{inc})$.

Denote by $C_{i,m}$ the $(k, r_{i,m}, \varepsilon)$-cover created by the execution of `LeapingStreamingDecision`$(P, k, \varepsilon, \text{init}_{\mathrm{i}}, \text{inc})$ at the end of round m, where $r_{i,m}$ is the distance parameter of the cover $C_{i,m}$. Note that $r_{i,m} = 2^i \cdot \text{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}}$ for some index $j \geq 0$. By

Observation 2.2 and Lemma 6.2, at the end of round m we have a $(k, 2\varepsilon, r_{i,m})$-decision distance oracle $\mathcal{O}_{i,m}$ for some curve P' such that $d_{dF}(P[1,m], P') \le \frac{2}{\text{inc}} r_{i,m}$. By Lemma 6.2,

$$r_{i,m} = 2^i \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}} \in \left[\frac{1}{1+2\varepsilon}, (1+2\varepsilon)\cdot \mathrm{inc}\right] \cdot \delta^*_{\mathrm{m}} \overset{(5)}{\subseteq} \left[\frac{1}{1+4\varepsilon}, (1+2\varepsilon)\cdot \mathrm{inc}\right] \cdot \mathrm{L_m}\,. \tag{6}$$

The query algorithm follows the lines in Section 4.4. Given a query curve $Q \in \mathbb{R}^{d\times m}$, we query $\mathcal{O}_{\Pi_m}$ and get a value Δ such that $d_{dF}(Q, \Pi_m) \le \Delta \le (1+\varepsilon) d_{dF}(Q, \Pi_m)$:

- If $\Delta \ge \frac{1}{\varepsilon} \cdot L_m$, return $(1+\varepsilon)\Delta$.
- Else, if $\Delta \le 3L_m$, let $j \ge 0$ and $i \in [0, t-1]$ be the unique indices such that $2^i \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}} \le 10 \cdot \mathrm{L_m} < 2^{\mathrm{i+1}} \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}}$.[10] Return $\mathcal{O}_{i,m}(Q) + \frac{2}{\text{inc}} r_{i,m}$.
- Else, set $\alpha = \lceil \frac{\Delta}{L_m} \rceil \in [4, \lceil \frac{1}{\varepsilon} \rceil]$, and let $j \ge 0$ and $i \in [0, t-1]$ be the unique indices such that $2^i \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}} \le 10 \cdot \alpha \mathrm{L_m} < 2^{\mathrm{i+1}} \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}}$.[10] Return $\mathcal{O}_{i,m}(Q) + \frac{2}{\text{inc}} r_{i,m}$.

Correctness. We show that in each of the preceding cases, the query algorithm returns a value in $[1, 1+O(\varepsilon)] \cdot d_{dF}(P, Q)$. Afterward, the ε parameter can be adjusted accordingly. By the triangle inequality, it holds that

$$d_{dF}(P[1,m], Q) \le d_{dF}(Q, \Pi_m) + d_{dF}(P[1,m], \Pi_m) \le \Delta + L_m \tag{7}$$

and

$$d_{dF}(P[1,m], Q) \ge d_{dF}(Q, \Pi_m) - d_{dF}(P[1,m], \Pi_m) \ge \frac{\Delta}{1+\epsilon} - L_m\,. \tag{8}$$

- If $\Delta \ge \frac{1}{\varepsilon} \cdot L_m$, then $d_{dF}(P[1,m], Q) \overset{(7)}{\le} (1+\varepsilon)\Delta$, and $d_{dF}(P[1,m], Q) \overset{(8)}{\ge} \frac{1}{1+\varepsilon}\Delta - \varepsilon\Delta \ge (1-2\varepsilon)\Delta$. It follows that $(1+\varepsilon)\Delta \le \frac{1+\varepsilon}{1-2\varepsilon} \cdot d_{dF}(P[1,m], Q) = (1+O(\varepsilon)) \cdot d_{dF}(P[1,m], Q)$.

For the next two cases, we first show that there exists a value ϕ such that $d_{dF}(P[1,m], Q) \in [\frac{1}{4}, 4] \cdot \phi$ (each case has a different ϕ value). Then, the rest of the analysis for both cases continues at ♣ (as it is identical given ϕ):

- If $\Delta \le 3L_m$, then $d_{dF}(P[1,m], Q) \overset{(7)}{\le} 4L_m$. We also have that $d_{dF}(P[1,m], Q) \ge d_{dF}(P[1,m], \Pi^*_m) = \delta^*_m \overset{(5)}{\ge} \frac{L_m}{1+\varepsilon}$. Hence, $d_{dF}(P[1,m], Q) \in [\frac{1}{1+\varepsilon}, 4] \cdot L_m \subseteq [\frac{1}{4}, 4] \cdot L_m$. Set $\phi = L_m$.
- Else, $L_m < \frac{1}{3}\Delta$, and hence $d_{dF}(P[1,m], Q) \overset{(7)}{\le} \frac{4}{3}\Delta$ and $d_{dF}(P[1,m], Q) \overset{(8)}{\ge} \frac{1}{1+\varepsilon}\Delta - \frac{1}{3}\Delta > \frac{1}{3}\Delta$. By the definition of α, it holds that $\Delta \le \alpha \cdot L_m \le (\frac{\Delta}{L_m} + 1) \cdot L_m \le \frac{4}{3}\Delta$, and hence $\Delta \in [\frac{3}{4}, 1] \cdot \alpha L_m$. Thus, $d_{dF}(P[1,m], Q) \in [\frac{1}{3}\Delta, \frac{4}{3}\Delta] \subseteq [\frac{1}{4}, \frac{4}{3}] \cdot \alpha L_m \subseteq [\frac{1}{4}, 4] \cdot \alpha L_m$. Set $\phi = \alpha L_m$.

♣ We have $d_{dF}(P[1,m], Q) \in [\frac{1}{4}, 4] \cdot \phi$, and let $j \ge 0$ and $i \in [0, s-1]$ be the unique indices such that $2^i \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}} \le 10 \cdot \phi < 2^{\mathrm{i+1}} \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}}$.[10] Consider the decision distance oracle $\mathcal{O}_{i,m}$. If $r_{i,m} \ne 2^i \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}}$, then one of the following hold in contradiction to Equation (6):

$$r^i_m \ge 2^i \cdot \mathrm{inc}^{\mathrm{j+1}} \cdot \delta^*_{\mathrm{k+1}} \ge \frac{\mathrm{inc}}{2} \cdot 2^{\mathrm{i+1}} \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}} \ge \frac{\mathrm{inc}}{2} \cdot 10 \cdot \phi > (1+2\varepsilon) \cdot \mathrm{inc} \cdot \mathrm{L_m}\,,$$

$$r^i_m \le 2^i \cdot \mathrm{inc}^{\mathrm{j-1}} \cdot \delta^*_{\mathrm{k+1}} \le \frac{1}{\mathrm{inc}} \cdot 2^{\mathrm{i}} \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}} \le \frac{1}{\mathrm{inc}} \cdot 10 \cdot \phi \le \frac{\alpha}{\mathrm{inc}} \cdot 10 \cdot \mathrm{L_m} \overset{(*)}{<} \frac{1}{1+4\varepsilon} \cdot \mathrm{L_m}\,,$$

where the inequality $^{(*)}$ follows as $\frac{\alpha}{\mathrm{inc}} \cdot 10 \le \lceil \frac{1}{\varepsilon} \rceil \cdot \frac{\varepsilon}{25} \cdot 10 < \frac{1}{2} < \frac{1}{1+4\varepsilon}$. We conclude that $r_{i,m} = 2^i \cdot \mathrm{inc}^{\mathrm{j}} \cdot \delta^*_{\mathrm{k+1}}$, hence $r_{i,m} \le 10 \cdot \phi < 2 \cdot r_{i,m}$.

[10] Such indexes i, j exist as $L_m \ge \delta^*_m \ge \delta^*_{k+1}$, and they are unique because $inc = 2^t$.

Following Lemma 6.2, let P' be the curve for which $C_{i,m}$ is a $(k, r_{i,m}, \varepsilon)$ cover. Then

$$\begin{aligned} d_{dF}(P', Q) &\leq d_{dF}(P', P[1,m]) + d_{dF}(P[1,m], Q) \\ &\leq \frac{2}{\text{inc}} \cdot r_{i,m} + 4\phi \leq \left(\frac{2}{\text{inc}} + \frac{4}{5}\right) r_{i,m} \leq r_{i,m}. \end{aligned}$$

Thus, $\mathcal{O}_{i,m}$ will return some value. Recall that our algorithm returns $\mathcal{O}_{i,m}(Q) + \frac{2}{\text{inc}} r_{i,m}$. It holds that

$$\mathcal{O}_{i,m}(Q) \geq d_{dF}(P', Q) \geq d_{dF}(P[1,m], Q) - d_{dF}(P', P[1,m]) \geq d_{dF}(P[1,m], Q) - \frac{2}{\text{inc}} r_{i,m} .$$

In addition,

$$\begin{aligned} \mathcal{O}_{i,m}(Q) \leq (1+\varepsilon) d_{dF}(P', Q) \leq d_{dF}(P', Q) + \varepsilon r_{i,m} &\leq d_{dF}(P[1,m], Q) + d_{dF}(P', P[1,m]) + \varepsilon r_{i,m} \\ &\leq d_{dF}(P[1,m], Q) + \frac{2}{\text{inc}} r_{i,m} + \varepsilon r_{i,m} . \end{aligned}$$

It follows that the returned value is bounded by $d_{dF}(P[1,m], Q) + (\frac{4}{\text{inc}} + \varepsilon) r_{i,m} \leq d_{dF}(P[1,m], Q) + (\frac{4}{\text{inc}} + \varepsilon) 10 \cdot \phi = (1 + O(\varepsilon)) \cdot d_{dF}(P[1,m], Q)$.

Space and Query Time. To maintain a simplification Π_m, according to Corollary 5.7 we used $O(\varepsilon)^{-\frac{d+1}{2}} \log^2 \varepsilon^{-1} + O(kd\varepsilon^{-1} \log \varepsilon^{-1})$ space. The distance oracle $\mathcal{O}_{\Pi_m}$ requires $O(\frac{1}{\varepsilon})^{dk} \cdot \log \varepsilon^{-1}$ space by Theorem 8. Finally, we use $O(\log \frac{1}{\varepsilon})$ decision distance oracles, with covers constructed by the `LeapingStreamCover` algorithm. As this algorithm simulates `StreamCover`, by Lemma 6.1 the space consumption is $O(\frac{1}{\varepsilon})^{kd}$. We conclude that the total space used by our streaming algorithm is $O(\frac{1}{\varepsilon})^{dk} \cdot \log \varepsilon^{-1}$.

Regarding query time, we first query Π_m, which takes $\tilde{O}(kd)$ time (Theorem 8). Afterward, we might perform another query in $O(kd)$ time (Lemma 6.1). All other computations take $O(kd)$ time. The theorem follows.

Theorem 3. *Given parameters $\varepsilon \in (0, \frac{1}{4})$ and $k \in \mathbb{N}$, there is a streaming algorithm that uses $O(\frac{1}{\varepsilon})^{kd} \log \varepsilon^{-1}$ space, and given a curve P with points in $\mathbb{R}^d$ constructs a $(1+\varepsilon)$-distance oracle with $\tilde{O}(kd)$ query time.*

7 DISTANCE ORACLE TO A SUBCURVE AND THE "ZOOM-IN" PROBLEM

In this section, we consider the following generalization of the distance oracle problem.

Problem 4. *Given a curve $P \in \mathbb{R}^{d \times m}$ and parameter $\varepsilon > 0$, preprocess P into a data structure that given a query curve $Q \in \mathbb{R}^{d \times k}$, and two indexes $1 \leq i \leq j \leq m$, returns an $(1+\varepsilon)$-approximation of $d_{dF}(P[i,j], Q)$.*

A trivial solution is to store for any $1 \leq i \leq j \leq m$ a distance oracle for $P[i,j]$, then the storage space increases by a factor of m^2. In cases where m is large, one might wish to reduce the storage space at the cost of increasing the query time or approximation factor.

We begin by introducing the *zoom-in problem*, which is closely related to the preceding problem. Our solution to the "zoom-in" problem will be used as a skeleton for a distance oracle to a subcurve.

7.1 The "Zoom-In" Problem

When one needs to visualize a large curve, it is sometimes impossible to display all of its details, and displaying a simplified curve is a natural solution. In some visualization applications, the user wants to "zoom-in" and see a part of the curve with the same level of detail. For example, if the curve represents the historical prices of a stock, one might wish to examine the rates during a specific period of time. In such cases, a new simplification needs to be calculated. In the following problem, we wish to construct a data structure that allows a quick zoom-in (or zoom-out) operation.

Problem 5 (Zoom-in to a Curve). *Given a curve $P \in \mathbb{R}^{d\times m}$ and an integer $1 \leq k < m$, preprocess P into a data structure that given $1 \leq i < j \leq m$ returns an optimal k-simplification of $P[i,j]$.*

To make the space and preprocessing time reasonable, we introduce a bi-criteria approximation version of the zoom-in problem: instead of returning an optimal k-simplification of $P[i,j]$, the data structure will return an (α,k,γ)-simplification of $P[i,j]$ (i.e., a curve $\Pi \in \mathbb{R}^{d\times\alpha k}$ such that $d_{dF}(P[i,j],\Pi) \leq \gamma d_{dF}(P[i,j],\Pi')$ for any $\Pi' \in \mathbb{R}^{d\times k}$).

We will use the two following observations.

Observation 7.1. *Let $\{P_i\}_{i=1}^s$, $\{Q_i\}_{i=1}^s$ be curves. Then $d_{dF}(P_1 \circ P_2 \circ \cdots \circ P_s, Q_1 \circ Q_2 \circ \cdots \circ Q_s) \leq \max_i\{d_{dF}(P_i,Q_i)\}$.*

Observation 7.2. *Let P be a curve and P' a subcurve of P. Let Π' be a (k,γ)-simplification of P'. Then for any $\Pi \in \mathbb{R}^{d\times k}$, it holds that $d_{dF}(P',\Pi') \leq \gamma d_{dF}(P,\Pi)$.*

Proof. Consider a paired walk ω along P and Π, and let Π'' be a subcurve of Π that contains all points of Π that were matched by ω to the points of P'. Then clearly $d_{dF}(P',\Pi'') \leq d_{dF}(P,\Pi)$, and by the definition of (k,γ)-simplification, $d_{dF}(P',\Pi') \leq \gamma d_{dF}(P',\Pi'') \leq \gamma d_{dF}(P,\Pi)$. □

In the following, we present a data structure for the zoom-in problem with $O(mk\log\frac{m}{k})$ space, which returns $(k, 1+\varepsilon, 2)$-simplifications in $O(kd)$ time.

For simplicity of the presentation, we will assume that m is a power of 2 (otherwise, add to the curve P, $2^{\lceil\log m\rceil} - m$ copies of the point $P[m]$). Construct a recursive structure with $\log\frac{m}{k}$ levels as follows. The first level contains a $(k, 1+\varepsilon, 1)$-simplification of $P[i,\frac{m}{2}]$ for any $1 \leq i \leq \frac{m}{2}$, and $P[\frac{m}{2}+1, j]$ for any $\frac{m}{2}+1 \leq j \leq m$. In the second level, we recurs on the sub curves $P[1,\frac{m}{2}]$ and $P[\frac{m}{2}+1, m]$. The ith level corresponds to 2^i sets of simplifications, and each set corresponds to a subcurve of length $\frac{m}{2^i}$. In the last level, the length of the corresponding subcurves is at most k. The total space of the data structure is $O(mkd\log\frac{m}{k})$, this is because each point $P[i]$ is responsible for a single simplification (a curve in $\mathbb{R}^{d\times k}$) in $\log\frac{m}{k}$ different levels. As all curves at the i level of the recursion have length $\frac{m}{2^i}$, using Theorem 7 the preprocessing time is

$$\sum_{i=1}^{\log\frac{m}{k}} m \cdot \tilde{O}\left(\frac{m}{2^i}\cdot\frac{d}{\varepsilon^{4.5}}\right) = \tilde{O}(m^2 d\varepsilon^{-4.5}).$$

If d is fixed, then according to Theorem 7 the preprocessing time will be

$$\sum_{i=1}^{\log\frac{m}{k}} m \cdot O\left(\frac{m}{2^i}\cdot\left(\frac{1}{\varepsilon} + \log\frac{m}{2^i\varepsilon}\log\frac{m}{2^i}\right)\right) = O\left(m^2\cdot\left(\frac{1}{\varepsilon}+\log\frac{m}{\varepsilon}\log m\right)\right) = \tilde{O}\left(\frac{m^2}{\varepsilon}\right).$$

Given two indexes $1 \leq i < j \leq m$, if $j - i \leq k$, simply return $P[i,j]$. Else, let t be the smallest integer such that $i \leq x\cdot\frac{m}{2^t} < j$ for some $x \in [2^{t-1}]$. Let Π_1 and Π_2 be the simplifications of $P[i, x\cdot\frac{m}{2^t}]$ and $P[x\cdot\frac{m}{2^t}+1, j]$, respectively. Return the concatenation $\Pi_1 \circ \Pi_2$.

We argue that $\Pi_1 \circ \Pi_2$ is a $(2,k,1+\varepsilon)$-simplification of $P[i,j]$. Indeed, let $\Pi \in \mathbb{R}^{d\times k}$ be an arbitrary length k curve. By Observation 7.2, we have $d_{dF}(P[i, x\cdot\frac{m}{2^t}], \Pi_1) \leq (1+\varepsilon)d_{dF}(P[i,j],\Pi)$ and $d_{dF}(P[x\cdot\frac{m}{2^t}+1, j], \Pi_2) \leq (1+\varepsilon)d_{dF}(P[i,j],\Pi)$. By Observation 7.1, we conclude that $d_{dF}(P[i,j], \Pi_1\circ\Pi_2) \leq (1+\varepsilon)d_{dF}(P[i,j],\Pi)$.

Theorem 4. *Given a curve P consisting of m points and parameters $k \in [m]$ and $\varepsilon \in (0, \frac{1}{2})$, there exists a data structure with $O(mkd \log \frac{m}{k})$ space such that given a pair of indices $1 \leq i < j \leq m$ returns in $O(kd)$ time a $(2, k, 1 + \varepsilon)$-simplification of $P[i, j]$. The prepossessing time for general d is $\tilde{O}(m^2 d\varepsilon^{-4.5})$ and for fixed d is $\tilde{O}(m^2\varepsilon^{-1})$.*

Remark 7.3. In the preceding algorithm, we essentially constructed a digraph G over $\{1, 2, \ldots, n\}$, such that the edge (i, j) corresponds to a $(1, k, 1 + \varepsilon)$-simplification of $P[i, j]$. The preceding graph contains $O(n \log n)$ edges and has "diameter" 2. In other words, for every pair $i < j$, there is a point $l \in [i, j]$ such that $(i, l), (l, j)$ are edges of G. In general, such a graph with s edges and diameter h will translate into a data structure returning $(h, k, 1 + \varepsilon)$-simplification with $O(skd)$ space.

It follows from Solomon [42] that for every $h \geq 2$, one can construct such a digraph with diameter h and $O(m \cdot \alpha_h(m))$ edges. Here the function α_h is the inverse of a certain Ackermann function at the $\lfloor \frac{k}{2} \rfloor$th level of the primitive recursive hierarchy, where $\alpha_2(m) = \lceil \log m \rceil$, $\alpha_3(m) = \lceil \log \log m \rceil$, $\alpha_4(m) = \log^* m$, and roughly α_k is close to $\lfloor \frac{k-2}{2} \rfloor$-iterated log-star. We refer to the work of Solomon [42] for further details. Furthermore, Kahalon et al. [37] constructed a data structure of size $O(m \cdot \alpha_h(m))$ that given $i < j$, in $O(h)$ time it returns a path from i to j in G with at most h edges.

We conclude that for every $h \geq 2$, one can construct a data structure with $O(kdm\alpha_h(m))$ space such that given a pair of indices $1 \leq i < j \leq m$ returns in $O(kdh)$ time an $(h, k, 1 + \varepsilon)$-simplification of $P[i, j]$.

7.2 (1+ε)-Factor Distance Oracle to a Subcurve

Note that as described in Observation 2.1, by the triangle inequality, a solution to the zooming problem can be used to answer approximate distance queries to a subcurve in $O(k^2 d)$ time. However, the approximation factor will be constant.

Our simplification for $P[i, j]$ is obtained by finding a partition of $P[i, j]$ into two disjoint subcurves, for which we precomputed $(k, 1 + \varepsilon)$-simplifications. To achieve a $(1 + \varepsilon)$ approximation factor, instead of storing $(k, 1 + \varepsilon)$-simplifications, we will store distance oracles that will be associated with the same set of subcurves, and then find an optimal matching between the query Q and a partition of $P[i, j]$.

Let $\mathcal{O}$ be a $(1 + \varepsilon)$-distance oracle with storage space $S(m, k, d)$, query time $T(m, k, d)$, and $PT(m, k, d)$ expected preprocessing time. Using Theorem 1, we can obtain $S(m, k, d) = O(\frac{1}{\varepsilon})^{dk} \cdot \log \varepsilon^{-1}$, $T(m, k, d) = \tilde{O}(kd)$, and $PT(m, k, d) = m \log \frac{1}{\varepsilon} \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d \log m))$.

Given two indexes $1 \leq i < j \leq m$, if $j - i \leq k$, simply compute and return $d_{dF}(P[i, j], Q)$ in $O(k^2 d)$ time. Else, let t and x be the integers as in the previous subsection, set $y = x \cdot \frac{m}{2^t}$, and let $\mathcal{O}_1$ and $\mathcal{O}_2$ be distance oracles for $P[i, y]$ and $P[y + 1, j]$, respectively. Return

$$\tilde{\Delta} = \min\Big\{ \min_{1 \leq q \leq k} \max\{\mathcal{O}_1(Q[1, q]), \mathcal{O}_2(Q[q, k])\}, \min_{1 \leq q \leq k-1} \max\{\mathcal{O}_1(Q[1, q]), \mathcal{O}_2(Q[q + 1, k])\}\Big\}.$$

The query time is therefore $O(k^2 d + k \cdot T(m, k, d)) = \tilde{O}(k^2 d)$. The storage space is $m \log \frac{m}{k} \cdot S(m, k, d) = m \log m \cdot O(\frac{1}{\varepsilon})^{dk} \cdot \log \frac{1}{\varepsilon}$ because we construct $m \log \frac{m}{k}$ distance oracles (instead of simplifications). The expected preprocessing time is

$$\sum_{i=1}^{\log \frac{m}{k}} m \cdot PT\left(\frac{m}{2^i}, k, d\right) = \sum_{i=1}^{\log \frac{m}{k}} m \cdot \frac{m}{2^i} \log \frac{1}{\varepsilon} \cdot \left(O\left(\frac{1}{\varepsilon}\right)^{kd} + O\left(d \log \frac{m}{2^i}\right)\right) = m^2 \log \frac{1}{\varepsilon} \cdot \left(O\left(\frac{1}{\varepsilon}\right)^{kd} + O(d \log m)\right).$$

Correctness. We argue that $d_{dF}(P[i,j],Q) \le \tilde{\Delta} \le (1+\varepsilon)d_{dF}(P[i,j],Q)$.
If $\tilde{\Delta} = \min_{1\le q\le k} \max\{\mathcal{O}_1(Q[1,q]), \mathcal{O}_2(Q[q,k])\}$, then

$$\tilde{\Delta} \ge \max\{d_{dF}(P[i,y],Q[1,q]), d_{dF}(P[y+1,j],Q[q,k])\} \ge d_{dF}(P[i,j],Q).$$

Similarly, if $\tilde{\Delta} = \min_{1\le q\le k-1} \max\{\mathcal{O}_1(Q[1,q]), \mathcal{O}_2(Q[q+1,k])\}$, then

$$\tilde{\Delta} \ge \min_{1\le q\le k-1} \max\{\mathcal{O}_1(Q[1,q]), \mathcal{O}_2(Q[q+1,k])\} \ge d_{dF}(P[i,j],Q).$$

Consider an optimal paired walk ω along $P[i,j]$ and Q, and let $1 \le q \le k$ be the maximum index such that ω matches $P[y]$ and $Q[q]$. If ω matches $P[y+1]$ and $Q[q]$, then

$$d_{dF}(P[i,j],Q) = \max\{d_{dF}(P[i,y],Q[1,q]), d_{dF}(P[y+1,j],Q[q,k])\}$$

and therefore $\max\{\mathcal{O}_1(Q[1,q]), \mathcal{O}_2(Q[q,k])\} \le (1+\varepsilon)d_{dF}(P[i,j],Q)$. Else, it must be that ω matches $P[y+1]$ and $Q[q+1]$ (due to the maximality of q) and then

$$d_{dF}(P[i,j],Q) = \max\{d_{dF}(P[i,y],Q[1,q]), d_{dF}(P[y+1,j],Q[q+1,k])\}$$

and therefore $\max\{\mathcal{O}_1(Q[1,q]), \mathcal{O}_2(Q[q+1,k])\} \le (1+\varepsilon)d_{dF}(P[i,j],Q)$. We conclude that $\tilde{\Delta} \le (1+\varepsilon)d_{dF}(P[i,j],Q)$.

Theorem 5. *Given a curve $P \in \mathbb{R}^{d\times m}$ and parameter $\varepsilon > 0$, there exists a data structure that given a query curve $Q \in \mathbb{R}^{d\times k}$, and two indexes $1 \le i \le j \le m$, returns an $(1+\varepsilon)$-approximation of $d_{dF}(P[i,j],Q)$. The data structure has $m\log m \cdot O(\frac{1}{\varepsilon})^{dk} \cdot \log\varepsilon^{-1}$ storage space, $m^2\log\frac{1}{\varepsilon} \cdot (O(\frac{1}{\varepsilon})^{kd} + O(d\log m))$ expected preprocessing time, and $\tilde{O}(k^2 d)$ query time.*

8 HIGH-DIMENSIONAL DISCRETE FRÉCHET ALGORITHMS

Most of the algorithms for curves under the (discrete) Fréchet distance that were proposed in the literature were only presented for constant or low dimension. The reason being is that it is often the case that the running time scales exponentially with the dimension, usually referred to as "the curse of dimensionality."

In this section, we provide a basic tool for finding a small set of critical values in d-dimensional space and show how to apply it for tasks concerning approximation algorithms for curves under the discrete Fréchet distance. Although algorithms for those tasks exist in low dimensions, their generalization to high-dimensional Euclidean space either do not exist or suffer from exponential dependence on the dimension.

Chan and Rahmati [17] (improving on the work of Bringmann and Mulzer [11]) presented an algorithm that given two curves P and Q in $\mathbb{R}^{d\times m}$, and a value $1 \le f \le m$, finds a value $\tilde{\Delta}$ such that $d_{dF}(P,Q) \le \tilde{\Delta} \le f d_{dF}(P,Q)$, in time $O(m\log m + m^2/f^2)\cdot\exp(d)$. Actually, their algorithm consists of two parts: decision and optimization. Fortunately, the decision algorithm is only polynomial in d.

Theorem 9 ([17]). *Given two curves P and Q in $\mathbb{R}^{d\times m}$, there exists an algorithm with running time $O(md + (md/f)^2 d)$ that returns YES if $d_{dF}(P,Q) \le 1$ and NO if $d_{dF}(P,Q) \ge f$; if $1 \le d_{dF}(P,Q) \le f$, the algorithm may return either YES or NO.*

The optimization procedure is the one presented by Bringmann and Mulzer [11], which adds an $O(m\log m)$ additive factor to the running time (for constant d). However, the running time of the optimization procedure depends exponentially on d. In Theorem 6, we show that the exponential factor in the running time can be removed without affecting the approximation factor.

Theorem 6. *Given two curves P and Q in $\mathbb{R}^{d\times m}$, and a value $f \ge 1$, there is an algorithm that returns a value $\tilde{\Delta}$ such that $d_{dF}(P,Q) \le \tilde{\Delta} \le f \cdot d_{dF}(P,Q)$ in $O(md\log(md)\log d + (md/f)^2 d\log(md)) = \tilde{O}(md + (md/f)^2 d)$ time.*

Bereg et al. [8] presented an algorithm that computes in $O(mk \log m \log(m/k))$ time an optimal k-simplification of a curve $P \in \mathbb{R}^{3\times m}$. In Theorem 7, we improve the running time and generalize this result to the arbitrary high dimension d while allowing a $1+\varepsilon$ approximation. Note that the method of Bereg et al. [8] works only for dimension $d \leq 3$, and for $k = \Omega(m)$ has quadratic running time, whereas our algorithm runs essentially in linear time (up to a polynomial dependence in ε), for the arbitrary large dimension d.

Theorem 7. *Given a curve $P \in \mathbb{R}^{m\times d}$ and parameters $k \in [m]$, $\varepsilon \in (0, \frac{1}{2})$, there is an $\tilde{O}(\frac{md}{\varepsilon^{4.5}})$-time algorithm that computes a $(k, 1+\varepsilon)$-simplification Π of P. In addition, the algorithm returns a value δ such that $d_{dF}(P, \Pi) \leq \delta \leq (1+\varepsilon)\delta^*$, where δ^* is the distance between P to an optimal k-simplification.*
Furthermore, if d is fixed, the algorithm can be executed in $m \cdot O(\frac{1}{\varepsilon} + \log \frac{m}{\varepsilon} \log m)$ time.

The algorithms for both Theorem 6 and Theorem 7 use the following lemma.

Lemma 8.1. *Consider a set V of n points in $\mathbb{R}^d$ and an interval $[a, b] \subset \mathbb{R}_+$. Then for every parameter $O(nd \cdot (\log n + \frac{1}{\varepsilon} \log(\frac{b}{a} d)))$-time algorithm, which returns a set $M \subset \mathbb{R}_+$ of $O(\frac{n}{\epsilon} d \log(d\frac{b}{a}))$ numbers such that for every pair of points $x, y \in V$ and a real number $\beta \in [a, b]$, there is a number $\alpha \in M$ such that $\alpha \leq \beta \cdot \|x - y\|_2 \leq (1+\varepsilon)\cdot\alpha$.*

Proof. For every $i \in [d]$, denote by x_i the ith coordinate of a point x, and let $V_i = \{x_i \mid x \in V\} \subset \mathbb{R}$. Set $\delta = \frac{1}{2}$. We construct a $\frac{1}{\delta}$-WSPD $\mathcal{W}_i$ for V_i. Specifically, $\mathcal{W}_i = \{\{A_1, B_1\}, \ldots, \{A_s, B_s\}\}$ is a set of $s \leq \frac{n}{\delta}$ pairs of sets $A_j, B_j \subseteq V_i$ such that for every $x, y \in V_i$, there is a pair $\{A_j, B_j\} \in \mathcal{W}_i$ such that $x \in A_j$ and $y \in B_j$ (or vice versa), and for every $j \in [s]$, $\max\{\mathrm{diam(A_j), diam(B_j)}\} \leq \delta \cdot \mathrm{d(A_j, B_j)}$, where $d(A_j, B_j) = \min_{x\in A_j, y\in B_j} |x - y|$. Such a WSPD exists, and it can be constructed in $O(n \log n + \frac{n}{\delta})$ time (e.g., see [33], Theorem 3.10).

Observe that by the definition of WSPD, and the triangle inequality, for any $\{A, B\} \in \mathcal{W}_i$ and two points $p \in A$ and $q \in B$, it holds that

$$d(A, B) \leq |p - q| \leq d(A, B) + \mathrm{diam(A)} + \mathrm{diam(B)} \leq (1 + 2\delta) \cdot \mathrm{d(A, B)} = 2 \cdot \mathrm{d(A, B)}. \tag{9}$$

For each set $\mathcal{W}_i$ and pair $\{A, B\} \in \mathcal{W}_i$, pick some arbitrary points $x' \in A$ and $y' \in B$, and set $\delta_i = |x' - y'|$. By Equation 9, $d(A, B) \leq \delta_i \leq 2 \cdot d(A, B)$.

Now for each $1 \leq i \leq d$, set

$$M_i = \left\{\delta_i \cdot (1+\varepsilon)^q \mid \{A, B\} \in \mathcal{W}_i,\ q \in \left[\left\lfloor \log_{1+\varepsilon} \frac{a}{2} \right\rfloor, \left\lfloor \log_{1+\varepsilon}(2b\sqrt{d}) \right\rfloor\right]\right\},$$

and let $M = \bigcup_{i=1}^d M_i$. We argue that the set M satisfies the condition of the lemma. Note that indeed $|M| \leq d \cdot \frac{n}{\delta} \cdot \log_{1+\epsilon}(\frac{4b}{a}\sqrt{d}) = O(\frac{n}{\epsilon} d \log(d \cdot \frac{b}{a}))$. Further, the construction time is

$$O\left(n \log n + \frac{n}{\delta}\right) \cdot d + O\left(d \cdot \frac{n}{\delta} \cdot \log_{1+\epsilon}\left(\frac{4b}{a}\sqrt{d}\right)\right) = O\left(nd \cdot \left(\log n + \frac{1}{\varepsilon} \log\left(\frac{b}{a} d\right)\right)\right).$$

Consider some pair $x, y \in V$, and let i be the coordinate where $|x_i - y_i|$ is maximized. Then $|x_i - y_i| = \|x - y\|_\infty \leq \|x - y\|_2 \leq \sqrt{d} \cdot \|x - y\|_\infty$. Let $\{A, B\} \in \mathcal{W}_i$ be a pair such that $x_i \in A$ and $y_i \in B$. By Equation 9, $d(A, B) \leq |x_i - y_i| \leq 2 \cdot d(A, B)$, and therefore $\frac{1}{2}\delta_i \leq |x_i - y_i| \leq 2\delta_i$.

We conclude that $\frac{1}{2}\delta_i \leq \|x - y\|_2 \leq 2\sqrt{d} \cdot \delta_i$. It follows that for every real parameter $\beta \in [a, b]$, there is a unique integer $q \in [\lfloor \log_{1+\varepsilon} \frac{a}{2} \rfloor, \lfloor \log_{1+\varepsilon}(2b\sqrt{d}) \rfloor]$ such that

$$(1+\epsilon)^q \cdot \delta_i \leq \beta \|x - y\|_2 \leq (1+\epsilon)^{q+1} \cdot \delta_i.$$

As $(1+\epsilon)^q \cdot \delta_i \in M$, the lemma follows. □

8.1 Approximation Algorithm: Proof of Theorem 6

First, if $f \leq 2$, we compute $d_{dF}(P,Q)$ exactly in $O(m^2 d)$ time. Otherwise, we set $f' = f/2$.

Next, we apply the algorithm from Lemma 8.1 for the points of $P \cup Q$ with parameter $\varepsilon = \frac{1}{2}$ and interval $[1,1]$ to obtain a set M of $O(md \log d)$ scalars that is constructed in $O(md \log(md))$ time. Notice that there exist two points $x \in P$ and $y \in Q$ such that $d_{dF}(P,Q) = \|x-y\|$. Therefore, there exists $\alpha^* \in M$ such that $\alpha^* \leq d_{dF}(P,Q) \leq \frac{3}{2}\alpha^* < 2\alpha^*$.

Then, we sort the numbers in M (in $O(md \log(md) \log d)$ time), and let $\alpha_1, \ldots, \alpha_{|M|}$ be the sorted list of scalars. We call α_i a YES-entry if the algorithm from Theorem 9 returns YES on the input f' and P, Q scaled by $2\alpha_i$, and a NO-entry if it returns NO on this input. Notice that any α_i must be a NO-entry if $2\alpha_i \cdot f' \leq d_{dF}(P,Q)$, and any α_i must be a YES-entry if $2\alpha_i \geq d_{dF}(P,Q)$. Moreover, $\alpha_{|M|}$ must be a YES-entry because $d_{dF}(P,Q) \leq 2\alpha^* \leq 2\alpha_{|M|}$.

If α_1 is a YES-query, then $d_{dF}(P,Q) \leq 2\alpha_1 f'$. We return $\tilde{\Delta} = 2\alpha_1 f'$, and as $\alpha_1 \leq \alpha^* \leq d_{dF}(P,Q)$, we get that $d_{dF}(P,Q) \leq \tilde{\Delta} = 2\alpha_1 f' \leq 2f' d_{dF}(P,Q) = f d_{dF}(P,Q)$.

Else, using binary search on $\alpha_1, \ldots, \alpha_{|M|}$, we find some i such that α_i is a YES-entry and α_{i-1} is a NO-entry, and return $\tilde{\Delta} = 2\alpha_1 f'$. Since α_i is a YES-query, we have $d_{dF}(P,Q) \leq 2\alpha_i f'$. As α_{i-1} is a NO-entry, we have $d_{dF}(P,Q) > 2\alpha_{i-1}$, and thus $\alpha^* > \alpha_{i-1}$ and $\alpha^* \geq \alpha_i \geq \alpha_{i-1}$, so $\alpha_i \leq \alpha^* \leq d_{dF}(P,Q)$, and we get that $d_{dF}(P,Q) \leq \tilde{\Delta} = 2\alpha_i f' \leq 2f' d_{dF}(P,Q) = f d_{dF}(P,Q)$. This search takes $\log(md) \cdot O(md + (md/f)^2 d)$ time. □

8.2 Computing a $(k, 1+\varepsilon)$-Simplification: Proof of Theorem 7

Bereg et al. [8] presented an algorithm that for constant d computes an optimal δ-simplification (this is a simple greedy simplification using the linear time minimum enclosing ball algorithm of Megiddo [40]). The authors and Katz [29] generalized this algorithm to high-dimension d by providing an algorithm, which given a scalar δ computes an approximation to the optimal δ-simplification.

Lemma 8.2 ([29]). *Let C be a curve consisting of m points in $\mathbb{R}^d$. Given parameters $r > 0$ and $\varepsilon \in (0,1]$, there exists an algorithm that runs in $O(\frac{d \cdot m \log m}{\varepsilon} + m \cdot \varepsilon^{-4.5} \log \frac{1}{\varepsilon})$ time and returns a curve Π such that $d_{dF}(C,\Pi) \leq (1+\varepsilon)r$. Furthermore, for every curve Π' with $|\Pi'| < |\Pi|$, it holds that $d_{dF}(C,\Pi') > r$.*

We begin with the following observation.

Observation 8.3. *Consider a curve $P \in \mathbb{R}^{m \times d}$ and an optimal k-simplification Π of P. Then there exists a pair of points $x, y \in P$ and a scalar $\beta \in [\frac{1}{2}, \frac{1}{\sqrt{2}}]$ such that $d_{dF}(P,\Pi) = \beta \cdot \|x-y\|$.*

Proof. Let ω be a one-to-many paired walk along Π and P. If no such a walk exists, then we can simply remove vertices from Π without increasing the distance to P.

Notice that there must exist an index $1 \leq j \leq k$ such that $(\Pi[j], P[i_1, i_2]) \in \omega$, and the minimum enclosing ball B of $P[i_1, i_2]$ has radius $d_{dF}(P,\Pi)$. Otherwise, for each j, we can move $\Pi[j]$ to the center of the appropriate minimum enclosing ball B, and decrease the distance between $\Pi[j]$ and $P[i_1, i_2]$, which will decrease $d_{dF}(P,\Pi)$, in contradiction to the optimality of Π.

Let $x, y \in P$ be the points such that $\|x-y\| = \max_{i_1 \leq p,q \leq i_2} \|P[p]-P[q]\|$ (the diameter of $P[i_1, i_2]$). Then the radius of B is at least $\frac{1}{2}\|x-y\|$, and by Jung's theorem, it is at most $\sqrt{\frac{d}{2(d+1)}} \cdot \|x-y\| < \sqrt{\frac{1}{2}} \cdot \|x-y\|$. □

Proof of Theorem 7. We first present the algorithm for general dimension d. Afterward, we will reduce the running time for the case where d is fixed.

Fix $\varepsilon' = \frac{\varepsilon}{3}$. Using Lemma 8.1 for the points of P with parameter ε' and interval $[\frac{1}{2}, \frac{1}{\sqrt{2}}]$, we obtain a set M of $O(\frac{m}{\epsilon} d \log d)$ scalars, which is constructed in $O(md \cdot (\log m + \frac{1}{\varepsilon} \log d))$ time. We sort the numbers in M, and using binary search we find the minimal $\alpha \in M$ such that the algorithm of Lemma 8.2 with parameter $(1+\varepsilon')\alpha$ returns a simplification Π_α of length at most k such that $d_{dF}(P, \Pi_\alpha) \leq (1+\varepsilon')^2\alpha$. We return the curve Π_α with parameter $(1+\varepsilon')^2\alpha$.

Let δ^* be the distance between P and an optimal k-simplification of P. We argue that $d_{dF}(P, \Pi_\alpha) \leq (1+\varepsilon)\delta^*$. First note that by Observation 8.3, there exist a pair of points $x, y \in P$ and a scalar $\beta \in [\frac{1}{2}, \frac{1}{\sqrt{2}}]$ such that $\delta^* = \beta \cdot \|x - y\|$. It follows from Lemma 8.1 that there is some $\alpha^* \in M$ such that $\alpha^* \leq \delta^* \leq (1+\varepsilon') \cdot \alpha^*$. In particular, the algorithm from Lemma 8.2 with the parameter $(1+\varepsilon')\alpha^*$ would return a curve Π_{α^*} of length at most k such that $d_{dF}(P, \Pi_{\alpha^*}) \leq (1+\varepsilon')^2\alpha^*$. Hence, our algorithm will find some $\alpha \in M$, so that $\alpha \leq \alpha^*$, and will return the curve Π_α. It holds that

$$d_{dF}(P, \Pi_\alpha) \leq (1+\varepsilon')^2\alpha \leq (1+\varepsilon')^2\alpha^* \leq (1+\varepsilon')^2\delta^* < (1+\varepsilon)\delta^* \ .$$

Sorting M takes $O(|M| \log |M|) = O(\frac{md}{\epsilon} \log \frac{md}{\epsilon} \cdot \log d)$ time. Finally, we have at most $\log |M|$ executions of the algorithm of Lemma 8.2, which take us $O(\log \frac{md}{\epsilon}) \cdot O(\frac{d \cdot m \log m}{\varepsilon} + m \cdot \varepsilon^{-4.5} \log \frac{1}{\varepsilon})$ time. The overall running time is

$$O\left(\frac{md}{\epsilon} \log \frac{md}{\epsilon} \cdot \log md + m \log \frac{md}{\epsilon} \cdot \varepsilon^{-4.5} \log \frac{1}{\varepsilon}\right) = \tilde{O}\left(\frac{md}{\varepsilon^{4.5}}\right).$$

For the case where the dimension d is fixed, instead of using Lemma 8.2, we will simply find an optimal $((1+\varepsilon')\alpha)$-simplification in $O(m \log m)$ time using the work of Bereg et al. [8]. The reset of the algorithm will remain the same. The correctness proof follows the exact same lines. The running time will be $O(m \cdot (\log m + \frac{1}{\varepsilon})) + \log(\frac{m}{\varepsilon}) \cdot O(m \log m) = m \cdot O(\frac{1}{\varepsilon} + \log \frac{m}{\varepsilon} \log m)$. □

APPENDIX

A MISSING PROOFS

A.1 Proof of Claim 2.3

Claim 2.3. $|G_{\varepsilon,r}(x, cr)| = O(\frac{c}{\varepsilon})^d$.

Proof. We scale our grid so that the edge length is 1, hence we are looking for the number of lattice points in $B_2^d(x, \frac{c\sqrt{d}}{\varepsilon})$. By Lemma 5 from previous work by the authors and Katz [29], we get that this number is bounded by the volume of the d-dimensional ball of radius $\frac{c\sqrt{d}}{\varepsilon} + \sqrt{d} \leq \frac{(c+1)\sqrt{d}}{\varepsilon}$.

Using Stirling's formula, we conclude that the volume of this ball is

$$V_2^d\left(\frac{(c+1)\sqrt{d}}{\varepsilon}\right) = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}\left(\frac{(c+1)\sqrt{d}}{\varepsilon}\right)^d = O\left(\frac{c}{\varepsilon}\right)^d.$$

□

A.2 (1+ε)-MEB: Proof of Lemma 5.6

We begin with a definition of ε-kernel of a set of points.

Definition A.1 (ε-kernel). For a set of points $X \subseteq \mathbb{R}^d$, and a direction $\vec{u} \in \mathbb{S}^{d-1}$, the directional width of X along u is defined by $W(X, \vec{u}) = \max_{\vec{p}, \vec{q} \in X} \langle \vec{p} - \vec{q}, \vec{u} \rangle$ (here, $\langle \cdot, \cdot \rangle$ denotes the inner product). A subset $Y \subseteq X$ of points is called an *ε-kernel* of X if for every direction $\vec{u} \in \mathbb{S}^{d-1}$,

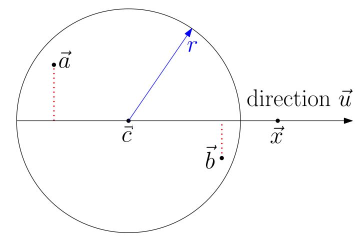
$$W(Y, \vec{u}) \geq (1-\varepsilon) W(X, \vec{u}) \ .$$

It was shown by Chan [15] that every set $X \subseteq \mathbb{R}^d$ has an ε-kernel of size $O(\varepsilon^{-\frac{d-1}{2}})$. Zarrabi-Zadeh [45] showed how to efficiently maintain an ε-kernel in the streaming model.

Theorem 10 ([45]). *Given a stream of points $X \subseteq \mathbb{R}^d$, an ε-kernel of X can be maintained using $O(\varepsilon)^{-\frac{d-1}{2}} \cdot \log \frac{1}{\varepsilon}$ space.*[11]

We make the following observation.

Claim A.2. *Consider a set $X \subseteq \mathbb{R}^d$, and let Y be an ε-kernel for $\varepsilon \in (0, \frac{1}{4})$. Consider a ball $B(\vec{c}, r)$ containing Y. Then $X \subseteq B(\vec{x}, (1 + 3\varepsilon)r)$.*



Proof. We will assume that X is a finite set, and the proof can be generalize to infinite sets using standard compactness arguments. Assume for contradiction that there is a point $\vec{x} \in X$ such that $\vec{x} \notin B(x, (1 + 3\varepsilon)r)$. Set $\vec{u} = \frac{\vec{x}-\vec{c}}{\|\vec{x}-\vec{c}\|}$, and let $\vec{a} = \operatorname{argmin}_{\vec{a} \in Y}\langle \vec{a}, \vec{u}\rangle$ and $\vec{b} = \operatorname{argmax}_{\vec{p} \in Y}\langle \vec{p}, \vec{u}\rangle$. Refer to the illustration on the right. Set $x = \langle \vec{x}, \vec{u}\rangle$, $a = \langle \vec{a}, \vec{u}\rangle$, and $b = \langle \vec{b}, \vec{u}\rangle$. As $\vec{x} \notin B(\vec{c}, (1+3\varepsilon)r)$, the distance between the projection of $\vec{x}$ in direction $\vec{u}$ to the projection of every point in $B(\vec{c}, (1+3\varepsilon)r)$ is greater than $3\varepsilon r$, thus $x - b > 3\varepsilon r$. However, as $\vec{a}, \vec{b}$ is contained in a ball of diameter $2r$, we have $W(Y, \vec{u}) = b - a \leq 2r$. Hence,

$$W(X, \vec{u}) \geq x - a = (x - b) + (b - a) > 3\varepsilon r + W(Y, \vec{u}) \geq \left(1 + \frac{3\varepsilon}{2}\right) W(Y, \vec{u}).$$

In particular, $W(Y, \vec{u}) < (1 - \varepsilon)W(X, \vec{u})$. □

Proof of Lemma 5.6. For a stream of points X, we will maintain an $\frac{\varepsilon}{5}$-kernel Y using Theorem 10. On a query for a minimum enclosing ball, we will compute an enclosing $B(\vec{c}, (1 + \frac{\varepsilon}{5})r)$ for Y (using the work of Kumar et al. [38]) such that there is no ball of radius r enclosing Y (or X). By Claim A.2, as Y is an $\frac{\varepsilon}{5}$ kernel of X, it holds that $X \subseteq B(\vec{c}, (1 + 3\frac{\varepsilon}{5})(1 + \frac{\varepsilon}{5})r) \subseteq B(\vec{c}, (1 + \varepsilon)r)$. We will return $B(\vec{c}, (1 + \varepsilon)r)$ as an answer. □

REFERENCES

[1] Mohammad Ali Abam, Mark de Berg, Peter Hachenberger, and Alireza Zarei. 2010. Streaming algorithms for line simplification. *Discret. Comput. Geom.* 43, 3 (2010), 497–515. https://doi.org/10.1007/s00454-008-9132-4

[2] Peyman Afshani and Anne Driemel. 2018. On the complexity of range searching among curves. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'18)*. 898–917. https://doi.org/10.1137/1.9781611975031.58

[3] Pankaj K. Agarwal, Rinat Ben Avraham, Haim Kaplan, and Micha Sharir. 2014. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.* 43, 2 (2014), 429–449. https://doi.org/10.1137/130920526

[4] Pankaj K. Agarwal and R. Sharathkumar. 2015. Streaming algorithms for extent problems in high dimensions. *Algorithmica* 72, 1 (2015), 83–98. https://doi.org/10.1007/s00453-013-9846-4

[5] Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. 2006. Fréchet distance for curves, revisited. In *Algorithms—ESA 2006*. Lecture Notes in Computer Science, Vol. 4168. Springer, 52–63. https://doi.org/10.1007/11841036_8

[6] Maria Sinziana Astefanoaei, Paul Cesaretti, Panagiota Katsikouli, Mayank Goswami, and Rik Sarkar. 2018. Multi-resolution sketches and locality sensitive hashing for fast trajectory processing. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL'18)*. ACM, New York, NY, 279–288. https://doi.org/10.1145/3274895.3274943

[11] Actually Zarrabi-Zadeh [45] bounds the space by $O(\varepsilon^{-\frac{d-1}{2}})$ while assuming that d is fixed and hence hiding exponential factors in d.

[7] Julian Baldus and Karl Bringmann. 2017. A fast implementation of near neighbors queries for Fréchet distance (GIS Cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'17)*. ACM, New York, NY, Article 99, 4 pages. https://doi.org/10.1145/3139958.3140062

[8] Sergey Bereg, Minghui Jiang, Wencheng Wang, Boting Yang, and Binhai Zhu. 2008. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proceedings of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*. 630–641. https://doi.org/10.1007/978-3-540-78773-0_54

[9] Karl Bringmann. 2014. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS'14)*. 661–670. https://doi.org/10.1109/FOCS.2014.76

[10] Karl Bringmann, Anne Driemel, André Nusser, and Ioannis Psarros. 2022. Tight bounds for approximate near neighbor searching for time series under the Fréchet distance. In *Proceedings of the Symposium on Discrete Algorithms (SODA'22)*.

[11] Karl Bringmann and Wolfgang Mulzer. 2016. Approximability of the discrete Fréchet distance. *J. Comput. Geom.* 7, 2 (2016), 46–76. https://doi.org/10.20382/jocg.v7i2a4

[12] Kevin Buchin, Yago Diez, Tom van Diggelen, and Wouter Meulemans. 2017. Efficient trajectory queries under the Fréchet distance (GIS Cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'17)*. ACM, New York, NY, Article 101, 4 pages. https://doi.org/10.1145/3139958.3140064

[13] Kevin Buchin, Tim Ophelders, and Bettina Speckmann. 2019. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 2887–2901. https://doi.org/10.1137/1.9781611975482.179

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

[15] Timothy M. Chan. 2006. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom.* 35, 1–2 (2006), 20–35. https://doi.org/10.1016/j.comgeo.2005.10.002

[16] Timothy M. Chan and Vinayak Pathak. 2014. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom.* 47, 2 (2014), 240–247. https://doi.org/10.1016/j.comgeo.2013.05.007

[17] Timothy M. Chan and Zahed Rahmati. 2018. An improved approximation algorithm for the discrete Fréchet distance. *Inf. Process. Lett.* 138 (2018), 72–74. https://doi.org/10.1016/j.ipl.2018.06.011

[18] Mark de Berg, Atlas F. Cook IV, and Joachim Gudmundsson. 2013. Fast Fréchet queries. *Comput. Geom.* 46, 6 (2013), 747–755. https://doi.org/10.1016/j.comgeo.2012.11.006

[19] Mark de Berg, Joachim Gudmundsson, and Ali D. Mehrabi. 2017. A dynamic data structure for approximate proximity queries in trajectory data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'17)*. Article 48, 4 pages. https://doi.org/10.1145/3139958.3140023

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

[21] Anne Driemel and Sariel Har-Peled. 2013. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Comput.* 42, 5 (2013), 1830–1866. https://doi.org/10.1137/120865112

[22] Anne Driemel, Sariel Har-Peled, and Carola Wenk. 2012. Approximating the Fréchet distance for realistic curves in near linear time. *Discret. Comput. Geom.* 48, 1 (2012), 94–127. https://doi.org/10.1007/s00454-012-9402-z

[23] Anne Driemel and Ioannis Psarros. 2021. ANN for time series under the Fréchet distance. In *Algorithms and Data Structures*. Lecture Notes in Computer Science, Vol. 12808. Springer, 315–328. https://doi.org/10.1007/978-3-030-83508-8_23

[24] Anne Driemel, Ioannis Psarros, and Melanie Schmidt. 2019. Sublinear data structures for short Fréchet queries. *CoRR* abs/1907.04420 (2019). http://arxiv.org/abs/1907.04420

[25] Anne Driemel and Francesco Silvestri. 2017. Locality-sensitive hashing of curves. In *Proceedings of the 33rd International Symposium on Computational Geometry*. Leibniz International Proceedings in Informatics, Vol. 77. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, Article 37, 16 pages. https://doi.org/10.4230/LIPIcs.SoCG.2017.37

[26] Fabian Dütsch and Jan Vahrenhold. 2017. A filter-and-refinement-algorithm for range queries based on the Fréchet distance (GIS Cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'17)*. ACM, New York, NY, Article 100, 4 pages. https://doi.org/10.1145/3139958.3140063

[27] Thomas Eiter and Heikki Mannila. 1994. *Computing Discrete Fréchet Distance*. Technical Report. Technische Universitat Wien.

[28] Ioannis Z. Emiris and Ioannis Psarros. 2018. Products of Euclidean metrics and applications to proximity questions among curves. In *Proceedings of the 34th International Symposium on Computational Geometry (SoCG'18)*. Article 37, 13 pages. https://doi.org/10.4230/LIPIcs.SoCG.2018.37

[29] Arnold Filtser, Omrit Filtser, and Matthew J. Katz. 2020. Approximate nearest neighbor for curves—Simple, efficient, and deterministic. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*. Article 48, 19 pages. https://doi.org/10.4230/LIPIcs.ICALP.2020.48
[30] Omrit Filtser. 2018. Universal approximate simplification under the discrete Fréchet distance. *Inf. Process. Lett.* 132 (2018), 22–27. https://doi.org/10.1016/j.ipl.2017.10.002
[31] Joachim Gudmundsson, Majid Mirzanezhad, Ali Mohades, and Carola Wenk. 2019. Fast Fréchet distance between curves with long edges. *Int. J. Comput. Geom. Appl.* 29, 2 (2019), 161–187. https://doi.org/10.1142/S0218195919500043
[32] Joachim Gudmundsson and Thomas Wolle. 2010. Towards automated football analysis: Algorithms and data structures. In *Proceedings of the 10th Australasian Conference on Mathematics and Computers in Sport*.
[33] Sariel Har-Peled. 2011. *Geometric Approximation Algorithms*. American Mathematical Society, Washington, DC.
[34] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.* 8, 1 (2012), 321–350. https://doi.org/10.4086/toc.2012.v008a014
[35] Piotr Indyk. 2002. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proceedings of the 8th Symposium on Computational Geometry*. ACM, New York, NY, 102–106. https://doi.org/10.1145/513400.513414
[36] Minghui Jiang, Ying Xu, and Binhai Zhu. 2008. Protein structure-structure alignment with discrete Fréchet distance. *J. Bioinform. Comput. Biol.* 6, 1 (2008), 51–64. https://doi.org/10.1142/S0219720008003278
[37] Omri Kahalon, Hung Le, Lazar Milenkovic, and Shay Solomon. 2021. Can't see the forest for the trees: Navigating metric spaces by bounded hop-diameter spanners. *CoRR* abs/2107.14221 (2021). https://arxiv.org/abs/2107.14221
[38] Piyush Kumar, Joseph S. B. Mitchell, and E. Alper Yildirim. 2003. Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions. In *Proceedings of the 5th Workshop on Algorithm Engineering and Experiments*. 45–55. https://doi.org/10.1145/996546.996548
[39] Ariane Mascret, Thomas Devogele, Iwan Le Berre, and Alain Hénaff. 2006. Coastline matching process based on the discrete Fréchet distance. In *Progress in Spatial Data Handling*. Springer, 383–400. https://doi.org/10.1007/3-540-35589-8_25
[40] Nimrod Megiddo. 1984. Linear programming in linear time when the dimension is fixed. *J. ACM* 31, 1 (1984), 114–127. https://doi.org/10.1145/2422.322418
[41] Rasmus Pagh and Flemming Friche Rodler. 2004. Cuckoo hashing. *J. Algorithms* 51, 2 (2004), 122–144. https://doi.org/10.1016/j.jalgor.2003.12.002
[42] Shay Solomon. 2013. Sparse Euclidean spanners with tiny diameter. *ACM Trans. Algorithms* 9, 3 (2013), Article 28, 33 pages. https://doi.org/10.1145/2483699.2483708
[43] Martin Werner and Dev Oliver. 2018. ACM SIGSPATIAL GIS Cup 2017: Range queries under Fréchet distance. *ACM SIGSPATIAL Special* 10, 1 (2018), 24–27. https://doi.org/10.1145/3231541.3231549
[44] Tim Wylie and Binhai Zhu. 2013. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Trans. Comput. Biology Bioinform.* 10, 6 (2013), 1372–1383. https://doi.org/10.1109/TCBB.2013.17
[45] Hamid Zarrabi-Zadeh. 2011. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica* 60, 1 (2011), 46–59. https://doi.org/10.1007/s00453-010-9392-2

Received 8 February 2022; revised 22 February 2023; accepted 21 June 2023