Simplification of Trajectory Streams*

Siu-Wing Cheng[†] Haoqiang Huang[‡] Le Jiang[§]

August 28, 2025

Abstract

While there are software systems that simplify trajectory streams on the fly, few curve simplification algorithms with quality guarantees fit the streaming requirements. We present streaming algorithms for two such problems under the Fréchet distance d_F in $\mathbb{R}^d$ for some constant $d \geq 2$.

Consider a polygonal curve τ in $\mathbb{R}^d$ in a stream. We present a streaming algorithm that, for any $\varepsilon \in (0,1)$ and $\delta > 0$, produces a curve σ such that $d_F(\sigma, \tau[v_1, v_i]) \leq (1+\varepsilon)\delta$ and $|\sigma| \leq 2\,\mathrm{opt} - 2$, where $\tau[v_1, v_i]$ is the prefix in the stream so far, and $\mathrm{opt} = \min\{|\sigma'| : d_F(\sigma', \tau[v_1, v_i]) \leq \delta\}$. Let $\alpha = 2(d-1)\lfloor d/2 \rfloor^2 + d$. The working storage is $O(\varepsilon^{-\alpha})$. Each vertex is processed in $O(\varepsilon^{-\alpha}\log\frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(\varepsilon^{-\alpha})$ time for $d \geq 4$. Thus, the whole τ can be simplified in $O(\varepsilon^{-\alpha}|\tau|\log\frac{1}{\varepsilon})$ time. Ignoring polynomial factors in $1/\varepsilon$, this running time is a factor $|\tau|$ faster than the best static algorithm that offers the same guarantees.

We present another streaming algorithm that, for any integer $k \geq 2$ and any $\varepsilon \in (0, \frac{1}{17})$, maintains a curve σ such that $|\sigma| \leq 2k-2$ and $d_F(\sigma, \tau[v_1, v_i]) \leq (1+\varepsilon)\cdot\min\{d_F(\sigma', \tau[v_1, v_i]) : |\sigma'| \leq k\}$, where $\tau[v_1, v_i]$ is the prefix in the stream so far. The working storage is $O((k\varepsilon^{-1} + \varepsilon^{-(\alpha+1)})\log\frac{1}{\varepsilon})$. Each vertex is processed in $O(k\varepsilon^{-(\alpha+1)}\log^2\frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(k\varepsilon^{-(\alpha+1)}\log\frac{1}{\varepsilon})$ time for $d \geq 4$.

1 Introduction

The pervasive use of GPS sensors has enabled tracking of moving objects. For example, a car fleet can use real-time information about its vehicles to deploy them in response to dynamic demands. The sensor periodically samples the location of the moving object and sends it as a vertex to the remote cloud server for storage and processing. The remote server interprets the sequence of vertices received as a polygonal curve (trajectory).

For a massive stream, it has been reported [6, 8, 19, 20] that sending all vertices uses too much network bandwidth and storage at the server, and it may aggravate issues like out-of-order and duplicate data points. Software systems have been built for the sensor side to simplify trajectory streams on the fly (e.g. [17, 18, 19, 28, 29]). A local buffer is used. Every incoming vertex in the stream triggers a new round of computation that uses only the incoming vertex and the data in the local buffer. At the end of a round, these systems may determine the next vertex and send it to the cloud server, but they may also send nothing.

*This research is supported by the Research Grants Council, Hong Kong, China (project no. 16208923).

†Department of Computer Science and Engineering, HKUST, Clear Water Bay, Hong Kong. Email: scheng@cse.ust.hk. ORCID: https://orcid.org/0000-0002-3557-9935

‡Department of Computer Science and Engineering, HKUST, Clear Water Bay, Hong Kong. Email: haoqiang.huang@connect.ust.hk. ORCID: https://orcid.org/0000-0003-1497-6226

§USTC, Hefei, China. Email: lejiang@mail.ustc.edu.cn. ORCID: https://orcid.org/0009-0009-8379-0266

It is important to keep the local buffer small [17, 18, 19, 28, 29]. It means a small working storage and less computation in processing the next vertex in the stream, which enables real-time response and requires less computing power. In theoretical computer science, streaming algorithms have three goals [23]. Let n be the number of items in the stream so far. First, the working storage should be $o(n)$ if not $\text{polylog}(n)$. Second, an item in the stream should be processed in $\text{polylog}(n)$ time or at worst $o(n)$ time because items are arriving continuously. Third, although the best solution is unattainable as the entire input is not processed as a whole, the solution should be provably satisfactory. Let τ be the curve in the stream. The abovementioned software systems only deal with some local error measures between the simplified curve σ and τ. We want to provide guarantees on the size of σ and the global similarity between σ and τ.

A popular similarity measure is the *Fréchet distance* [15]. Let $\rho_\tau : [0,1] \to \mathbb{R}^d$ be a *parameterization* of τ such that, as t increases from 0 to 1, $\rho_\tau(t)$ moves from the beginning of τ to its end without backtracking. It is possible that $\rho_\tau(t) = \rho_\tau(t')$ for two distinct $t, t' \in [0,1]$. We can similarly define a parameterization ρ_σ for σ. These parameterizations induce a *matching* $\mathcal{M}$ between $\rho_\sigma(t)$ and $\rho_\tau(t)$ for all $t \in [0,1]$. A point may have multiple matching partners. The Fréchet distance is $d_F(\sigma,\tau) = \inf_{\mathcal{M}} \max_{t\in[0,1]} d(\rho_\sigma(t), \rho_\tau(t))$. We call a matching that realizes the Fréchet distance a *Fréchet matching*.

We study two problems. The *δ-simplification problem* is to compute, for a given $\delta > 0$, a curve σ of the minimum size (number of vertices) such that $d_F(\sigma,\tau) \le \delta$. The *$k$-simplification problem* is to compute, for a given integer $k \ge 2$, a curve σ of size k that minimizes $d_F(\sigma,\tau)$.

Given a polygonal curve $\tau = (v_1, v_2, \ldots)$, $|\tau|$ denotes its number of vertices, and for any $i < j$, $\tau[v_i, v_j]$ denotes the subcurve $(v_i, v_{i+1}, \ldots, v_j)$. For a subset $P \subset \mathbb{R}^d$, $d(x,P) = \inf_{p\in P} d(x,p)$. Given two points $x, y \in \mathbb{R}^d$, xy denotes the oriented line segment from x to y.

1.1 Related works

Static $\boldsymbol{\delta}$-simplification. Let $n = |\tau|$. In $\mathbb{R}$, if the vertices of σ are restricted to lie on τ (anywhere), then $|\sigma|$ can be minimized in $O(n)$ time [25]; if the vertices of σ must be a subset of those of τ, it takes $O(n)$ time to compute σ such that $|\sigma|$ is at most two more than the minimum [11]. In $\mathbb{R}$, there is a data structure [26] that, for any given δ, reports a curve σ with the minimum $|\sigma|$ such that the vertices of σ lie on τ (anywhere) and $d_F(\sigma,\tau) \le \delta$. The query time is $O(|\sigma|)$, and the preprocessing time is $O(n)$.

In $\mathbb{R}^2$, a curve σ with minimum $|\sigma|$ such that $d_F(\sigma,\tau) \le \delta$ can be computed in $O(n^2 \log^2 n)$ time [16]. In $\mathbb{R}^d$ for $d \ge 3$, if the vertices of σ must be a subset of those of τ, then $|\sigma|$ can be minimized in $O(n^3)$ time [5, 25, 27]. If there is no restriction on the vertices of σ, it has been recently shown that $|\sigma|$ can be minimized in $O\big((|\sigma|n)^{O(d|\sigma|)}\big)$ time [10].

Faster algorithms have been developed by allowing inexact output. Let $\kappa(\tau, r)$ be the minimum simplified curve size for an error r. In $\mathbb{R}^2$, it was shown [3] that for any $\delta > 0$, a curve σ can computed in $O(n \log n)$ time such that $d_F(\sigma,\tau) \le \delta$ and $|\sigma| \le \kappa(\tau, \delta/2)$. It is possible that $\kappa(\tau,\delta/2) \gg \kappa(\tau,\delta)$. A better control of the curve size has been obtained later. In $\mathbb{R}^d$, it was shown [25] that for any $\varepsilon \in (0,1)$ and $\delta > 0$, a curve σ can be constructed in $O\big(\varepsilon^{2-2d} n^2 \log n \log\log n\big)$ time such that $d_F(\sigma,\tau) \le (1+\varepsilon)\delta$ and $|\sigma| \le 2\kappa(\tau,\delta) - 2$.[1] Later, it was shown [9] that for any $\alpha, \varepsilon \in (0,1)$ and $\delta > 0$, a curve σ can be computed in $\tilde{O}\big(n^{O(1/\alpha)} \cdot (d/(\alpha\varepsilon))^{O(d/\alpha)}\big)$ time such that $d_F(\sigma,\tau) \le (1+\varepsilon)\delta$ and $|\sigma| \le (1+\alpha)\cdot\kappa(\tau,\delta)$.

The algorithms above for $d \ge 2$ do not fit the streaming setting [3, 5, 9, 10, 16, 25, 27]. In [3], the simplified curve σ is a subsequence of the vertices in τ, and the vertices of σ are picked from τ in a traversal of τ. Given the last pick v_i, for any $j > i$, whether v_j should be included in σ is determined by examining the subcurve $\tau[v_i, v_j]$. This requires an $O(n)$ working

[1] In [25], the first and last vertices of σ are restricted to be the same as those of τ.

storage which is too large. If the algorithms in [5, 9, 10, 16, 25, 27] are used in the streaming setting, the processing time per vertex would be $O(n)$ or more which is too high.

Static k-simplification. In $\mathbb{R}$, for any given k, the data structure for δ-simplification in [26] can be queried in $O(k)$ time to report a curve σ such that $|\sigma| = k$, the vertices of σ lie on τ (anywhere), and $d_F(\sigma, \tau)$ is minimized. In $\mathbb{R}^d$, if the vertices of σ must be a subset of those of τ, it was shown [15] that a solution σ can be computed in $O(n^4 \log n)$ time such that $d_F(\sigma, \tau) \leq 7 \cdot \min\bigl\{d_F(\sigma', \tau) : |\sigma'| \leq k$, no restriction on the vertices of $\sigma'\bigr\}$. The factor 7 can be reduced to 4 by a better analysis [3]. Nevertheless, if this algorithm is used in the streaming setting, the vertex processing time would be $O(n^3 \log n)$ or more, which is way too high.

Streaming line simplification. A *line simplification* σ of a stream τ is a subsequence of the vertices such that the first and last vertices of σ are the first and last vertices in the stream so far. In $\mathbb{R}^2$, for any integer $k \geq 2$ and any $\varepsilon \in (0, 1)$, it was shown [1, 2] how to maintain σ such that $|\sigma| \leq 2k$ and $d_F(\sigma, \tau) \leq (4\sqrt{2} + \varepsilon) \cdot \mathrm{opt}_k$, where $\mathrm{opt}_k = \min\{d_F(\sigma', \tau) :$ line simplification σ' of τ, $|\sigma'| \leq k\}$. The working storage is $O(k^2 + k\varepsilon^{-1/2})$. Each vertex is processed in $O(k \log \frac{1}{\varepsilon})$ amortized time. There are also streaming software systems for this problem that minimize some local error measures (e.g. [14, 21, 22, 24]); however, they do not provide any guarantee on the global similarity between τ and the output curve.

Discrete Fréchet distance. The *discrete* Fréchet distance $d_{\mathrm{dF}}(\sigma, \tau)$ is obtained with the restriction that the parameterizations ρ_σ and ρ_τ must match the vertices of σ to those of τ and vice versa. Note that $d_{\mathrm{dF}}(\sigma, \tau) \geq d_F(\sigma, \tau)$ and $d_{\mathrm{dF}}(\sigma, \tau) \gg d_F(\sigma, \tau)$ in the worst case.

There are several results in $\mathbb{R}^3$ [4]. A curve σ with the minimum $|\sigma|$ such that $d_{\mathrm{dF}}(\sigma, \tau) \leq \delta$ can be computed in $O(n \log n)$ time; if the vertices of σ must be a subset of those of τ, the computation time increases to $O(n^2)$. On the other hand, if k is given instead of δ, a curve σ such that $|\sigma| = k$ and $d_{\mathrm{dF}}(\sigma, \tau)$ is minimized can be computed in $O(kn \log n \log(n/k))$ time; if the vertices of σ must a subset of those of τ, the computation time increases to $O(n^3)$.

In $\mathbb{R}^d$, a streaming k-simplification algorithm was proposed in [12] with guarantees on $d_{\mathrm{dF}}(\sigma, \tau)$, where τ is the curve seen in the stream so far. It was shown that for any integer $k \geq 2$ and any $\varepsilon \in (0, 1)$, a curve σ can be computed such that $|\sigma| \leq k$ and $d_{\mathrm{dF}}(\sigma, \tau) \leq 8 \cdot \min\{d_{\mathrm{dF}}(\sigma', \tau) : |\sigma'| \leq k\}$. The working storage is $O(kd)$.

Improved results have been obtained subsequently [13]. For the streaming δ-simplification problem, there is a streaming algorithm in $\mathbb{R}^d$ that, for any $\gamma > 1$, computes a curve σ such that $d_{\mathrm{dF}}(\sigma, \tau) \leq \delta$ and $|\sigma| \leq \kappa(\tau, \delta/\gamma)$. This algorithm relies on a streaming algorithm for maintaining a γ-approximate minimum closing ball (MEB) of points in $\mathbb{R}^d$; the processing time per vertex is asymptotically the same as the γ-approximate MEB streaming algorithm. There are two streaming k-simplification algorithms in [13]. The first one uses $O(\frac{1}{\varepsilon} kd \log \frac{1}{\varepsilon}) + O(\varepsilon)^{-(d+1)/2} \log^2 \frac{1}{\varepsilon}$ working storage and maintains a curve σ such that $|\sigma| \leq k$ and $d_{\mathrm{dF}}(\sigma, \tau) \leq (1 + \varepsilon) \cdot \min\{d_{\mathrm{dF}}(\sigma', \tau) : |\sigma'| \leq k\}$. The second algorithm uses $O(\frac{1}{\varepsilon} kd \log \frac{1}{\varepsilon})$ working storage and maintains a curve σ such that $|\sigma| \leq k$ and $d_{\mathrm{dF}}(\sigma, \tau) \leq (1.22 + \varepsilon) \cdot \min\{d_{\mathrm{dF}}(\sigma', \tau) : |\sigma'| \leq k\}$.

1.2 Our results

We present streaming algorithms for the δ-simplification and k-simplification problems in $\mathbb{R}^d$ for $d \geq 2$ under the Fréchet distance. Let $\tau = (v_1, v_2, \ldots)$ be the input stream.

Streaming δ-simplification. Our algorithm outputs vertices occasionally and maintains some vertices in the working storage. The simplified curve σ for the prefix $\tau[v_1, v_i]$ in the stream so far consists of vertices that have been output and vertices in the working storage. Vertices that have been output cannot be modified, so they form a prefix of all simplified curves to be produced in the future for the rest of the stream.

For any $\varepsilon \in (0,1)$ and any $\delta > 0$, our algorithm produces a curve σ for the prefix $\tau[v_1, v_i]$ in the stream so far such that $d_F(\sigma, \tau[v_1, v_i]) \leq (1+\varepsilon)\delta$ and $|\sigma| \leq 2\kappa(\tau[v_1, v_i], \delta) - 2$. Let $\alpha = 2(d-1)\lfloor d/2 \rfloor^2 + d$. The working storage is $O(\varepsilon^{-\alpha})$. Each vertex in the stream is processed in $O(\varepsilon^{-\alpha} \log \frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(\varepsilon^{-\alpha})$ time for $d \geq 4$. If our algorithm is used in the static case, the running time on a curve of size n is $O(\varepsilon^{-\alpha} n \log \frac{1}{\varepsilon})$. Ignoring polynomial factors in $1/\varepsilon$, our time bound is a factor n smaller than the $O(\varepsilon^{2-2d} n^2 \log n \log\log n)$ running time of the best static algorithm that achieves the same error and size bounds [25].

Intuitively, our streaming algorithm finds a line segment that stabs as many balls as possible that are centered at consecutive vertices of τ with radii $(1+O(\varepsilon))\delta$. If such a line segment ceases to exist upon the arrival of a new vertex, we start a new line segment. Concatenating these segments forms the simplified curve. For efficiency, we approximate each ball by covering it with grid cells of $O(\varepsilon\delta)$ width. In [25], vertex balls and their covers by grid cells of $O(\varepsilon\delta)$ width are also used. Connections between all pairs of balls are computed to create a graph such that the simplified curve corresponds to a shortest path in the graph. In contrast, we maintain some geometric structures so that we can read off the next line segment from them. We prove an $O(\varepsilon^{-\alpha})$ bound on the total size of these structures, which yields the desired working storage and processing time per vertex.

Streaming *k*-simplification. A memory budget may cap the size of the simplified curve, motivating the streaming k-simplification problem. For example, in some wildlife tracking applications, the location-acquisition device is not readily accessible after deployment, and data is only offloaded from it after an extended period [20].

Our algorithm maintains the simplified curve σ in the working storage for the prefix $\tau[v_1, v_i]$ in the stream so far. For any $k \geq 2$ and any $\varepsilon \in (0, \frac{1}{17})$, our algorithm guarantees that $|\sigma| \leq 2k-2$ and $d_F(\sigma, \tau[v_1, v_i]) \leq (1+\varepsilon) \cdot \min\{d_F(\sigma', \tau[v_1, v_i]) : |\sigma'| \leq k\}$. Recall that $\alpha = 2(d-1)\lfloor d/2 \rfloor^2 + d$. The working storage is $O((k\varepsilon^{-1} + \varepsilon^{-(\alpha+1)}) \log \frac{1}{\varepsilon})$. Each vertex in the stream is processed in $O(k\varepsilon^{-(\alpha+1)} \log^2 \frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(k\varepsilon^{-(\alpha+1)} \log \frac{1}{\varepsilon})$ time for $d \geq 4$. We first simplify as in the case of δ-simplification. When the simplified curve reaches a size of $2k-1$, the key idea is to run our δ-simplification algorithm with a suitable error tolerance to bring its size down to $2k-2$.

There is only one prior streaming k-simplification algorithm [1, 2]. It works in $\mathbb{R}^2$ and requires the vertices of the simplified curve σ to be a subset of those of τ. It uses $O(k^2 + k\varepsilon^{-1/2})$ working storage, processes each vertex in $O(k \log \frac{1}{\varepsilon})$ amortized time, and guarantees that $|\sigma| \leq 2k$ and $d_F(\sigma, \tau) \leq (4\sqrt{2} + \varepsilon) \cdot$ optimum under the requirement that the vertices of σ must be a subset of those of τ. For $d = 2$, our algorithm uses $O((k\varepsilon^{-1} + \varepsilon^{-5}) \log \frac{1}{\varepsilon})$ working storage, processes each vertex in $O(k\varepsilon^{-5} \log^2 \frac{1}{\varepsilon})$ worst-case time, and guarantees that $|\sigma| \leq 2k-2$ and $d_F(\sigma, \tau) \leq (1+\varepsilon) \cdot$ optimum. Ignoring the restriction on the vertices of σ, we offer a better approximation ratio for $d_F(\sigma, \tau)$. Although our vertex processing time bound is larger, it is worst-case instead of amortized. The comparison between working storage depends on the relative magnitudes of k and $1/\varepsilon$.

2 Streaming δ-simplification

Given a subset $X \subseteq \mathbb{R}^d$, we use $\mathrm{conv}(X)$ to denote the convex hull of X. For any point $x \in \mathbb{R}^d$, let B_x denote the d-ball centered at x with radius δ. Consider the infinite d-dimenisonal grid with x as a grid vertex and side length $\varepsilon\delta/(2\sqrt{d})$. Let G_x be the subset of grid cells that intersect the ball centered at x with radius $(1+\varepsilon/2)\delta$. We say that an oriented line segment *s stabs the objects* $O_1, \ldots, O_m$ *in order* if there exist points $x_i \in s \cap O_i$ for $i \in [m]$ such that $x_1, \ldots, x_m$ appear in this order along s [16].

2.1 Algorithm

The high-level idea is to find the longest sequence $v_1, v_2, \ldots, v_i$ of vertices such that there is a segment s_1 that stabs B_{v_a} for $a \in [i]$ in order. Then, restart to find the longest sequence $v_{i+1}, \ldots, v_j$ such that there is a segment s_2 that stabs B_{v_a} for $a \in [i+1, j]$ in order. Repeating this way gives a sequence of segments $s_1, s_2, \ldots$. Concatenating these segments gives the simplified curve. The problem is the large working storage: a long vertex sequence may be needed to determine a line segment. We use several ideas to overcome this problem.

First, we approximate B_{v_a} by $\mathrm{conv}(G_{v_a})$ and find a line segment that stabs the longest sequence $\mathrm{conv}(G_{v_1}), \ldots, \mathrm{conv}(G_{v_i})$ in order. We will show that it suffices for this line segment to start from the set P of grid points in G_{v_1}. So $|P| = O(\varepsilon^{-d})$.

Second, as the vertices arrive in the stream, for each point $p \in P$, we construct a structure $S_a[p]$ inductively as follows:

- Set $S_1[p] = \{p\}$ for all $p \in P$. We write $S_1[p] = p$ for convenience.
- For all $a \geq 1$, set $S_{a+1}[p] = \mathrm{conv}(G_{v_{a+1}}) \cap F(S_a[p], p)$ for all $p \in P$, where $F(S_a[p], p) = \{y \in \mathbb{R}^d : py \cap S_a[p] \neq \emptyset\}$.

If $p \in S_a[p]$, then $F(S_a[p], p) = \mathbb{R}^d$ and hence $S_{a+1}[p] = \mathrm{conv}(G_{v_{a+1}})$. If $S_a[p] = \emptyset$, then $F(S_a[p], p) = \emptyset$ and hence $S_{a+1}[p] = \emptyset$. Otherwise, by viewing p as a light source, $F(S_a[p], p)$ is the unbounded convex polytope consisting of $S_a[p]$ and the non-illuminated subset of $\mathbb{R}^d$.

We will show that for $a \geq 2$, $S_a[p]$ is equal to $\{x \in \mathrm{conv}(G_{v_a}) : px \text{ stabs } \mathrm{conv}(G_{v_1}), \ldots, \mathrm{conv}(G_{v_a}) \text{ in order}\}$. We will also show that for each $p \in P$, $S_a[p]$ and $F(S_a[p], p)$ have $\mathrm{poly}(1/\varepsilon)$ complexities, and they can be constructed in $\mathrm{poly}(1/\varepsilon)$ time. The array S_a is deleted after computing S_{a+1}, which keeps the working storage small.

We pause when we encounter a vertex v_{i+1} such that $S_{i+1}[p] = \emptyset$ for all $p \in P$. We choose any vertex q of any non-empty $S_i[p]$ (which must exist) and output the segment pq. Since pq stabs $\mathrm{conv}(G_{v_1}), \ldots, \mathrm{conv}(G_{v_i})$ in order, we can show that $d_F(pq, \tau[v_1, v_i]) \leq (1+\varepsilon)\delta$. All is well except that S_i has been deleted after computing S_{i+1}. A fix is that after processing a vertex v_a, we store a segment pq in a buffer $\tilde{\sigma}$ for an arbitrary vertex q of an arbitrary non-empty $S_a[p]$. We output the content of $\tilde{\sigma}$ after discovering that $S_{i+1}[p] = \emptyset$ for all $p \in P$.

Afterward, we reset P as the set of grid points in $G_{v_{i+1}}$ and restart the above processing from v_{i+1}. That is, reset $S_{i+1}[p] = p$ for all $p \in P$. One of the points in P will be the start of the next segment that will be output in the future. The start of this next segment may be far from $\mathrm{conv}(G_{v_i})$, which contains the end of $\tilde{\sigma}$ that was just output. For $a = i+1, i+2, \ldots$, we set $S_{a+1}[p] = \mathrm{conv}(G_{v_{a+1}}) \cap F(S_a[p], p)$ for all $p \in P$ until $S_a[p]$ becomes empty for all $p \in P$ again. In summary, the simplified curve σ is the concatenation of the output segments $s_1, s_2, \ldots$, i.e., the end of s_j is connected to the start of s_{j+1}. Figure 1 illustrates a few steps of this algorithm. The procedure Simplify(ε, δ) in Algorithm 1 gives the details.

2.2 Analysis

Given a polytope $Q \subset \mathbb{R}^d$, let $|Q|$ denote its complexity, i.e., the total number of its faces of all dimensions.

Lemma 1. *Let P be the set of grid points in G_{v_i}. Assume that $S_i[p] = p$ for all $p \in P$. Take any $p \in P$ and any index $j \geq i+1$ such that $S_a[p] \neq \emptyset$ for all $a \in [i, j-1]$.*

(i) $S_j[p] = \{x \in \mathrm{conv}(G_{v_j}) : px \text{ stabs } \mathrm{conv}(G_{v_i}), \ldots, \mathrm{conv}(G_{v_j}) \text{ in order}\}$.

(ii) *$S_j[p]$ is a convex polytope that can be computed in $O\big(|S_{j-1}[p]| + N \log N + N^{\lfloor d/2 \rfloor}\big)$ time, where N is any upper bound on the number of support hyperplanes of $F(S_{j-1}[p], p)$ such that $N = \Omega(\varepsilon^{(1-d)\lfloor d/2 \rfloor})$.*

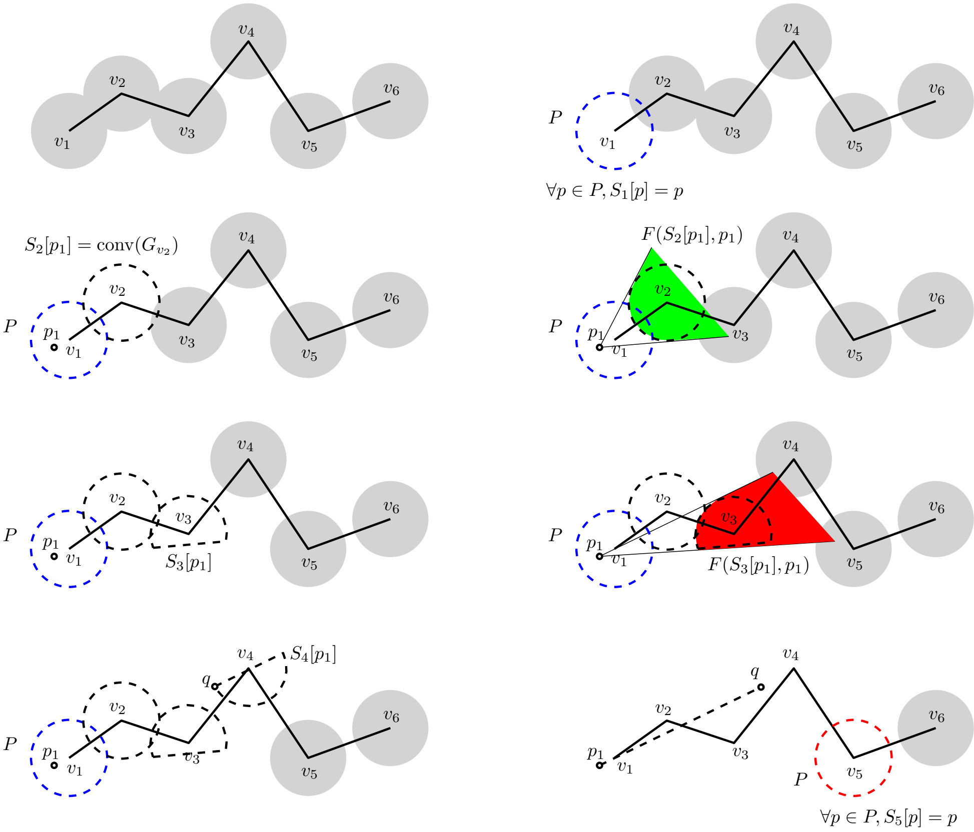

Figure 1: The shaded balls denote the $\mathrm{conv}(G_{v_a})$'s. The vertices arrive in order (from left to right and top to bottom). The first P consists of the grid points inside the blue dashed ball. Let p_1 be a grid point in P. Note that $F(S_1[p_1], p_1) = \mathbb{R}^d$. So $S_2[p_1] = \mathrm{conv}(G_{v_2})$ and $F(S_2[p_1], p_1)$ is the green unbounded convex region that is bounded by the tangents from p_1 to $S_2[p_1]$. $S_3[p_1] = \mathrm{conv}(G_{v_3}) \cap F(S_2[p_1], p_1)$ is denoted by the dashed clipped ball around v_3. Similarly, $F(S_3[p_1], p_1)$ is the red unbounded convex region that is bounded by the tangents from p_1 to $S_3[p_1]$. $S_4[p_1]$ is denoted by the dashed clipped ball around v_4. In this example, $S_5[p] = \emptyset$ for all $p \in P$, so we output $\tilde{\sigma}$ which may be equal to (p_1, q) for a vertex q of $S_4[p_1]$. We reset P as the set of grid points in G_{v_5} (inside the red dashed ball) and reset $S_5[p] = p$ for all $p \in P$.

Proof. When $j = i + 1$, (i) is true as $S_{i+1}[p] = \mathrm{conv}(G_{v_{i+1}})$. Assume inductively that (i) holds for $j - 1$ for some $j \geq i + 2$. For any $x \in F(S_{j-1}[p], p)$, px intersects $S_{j-1}[p]$ by definition, which implies that px stabs $\mathrm{conv}(G_{v_i}), \ldots, \mathrm{conv}(G_{v_{j-1}})$ in order by induction assumption. Therefore, for every $x \in \mathrm{conv}(G_{v_j}) \cap F(S_{j-1}[p], p)$, px stabs $\mathrm{conv}(G_{v_i}), \ldots, \mathrm{conv}(G_{v_j})$ in order. Conversely, for every $x \in \mathrm{conv}(G_{v_j})$, if px stabs $\mathrm{conv}(G_{v_i}), \ldots, \mathrm{conv}(G_{v_j})$ in order, then px contains a point in $S_{j-1}[p]$ by induction assumption, which implies that $x \in F(S_{j-1}[p], p)$.

Consider (ii). If $p \in S_{j-1}[p]$, then $F(S_{j-1}[p], p) = \mathbb{R}^d$ and $S_j[p] = \mathrm{conv}(G_{v_j})$. This case can be handled in $O(|S_{j-1}[p]|)$ time. If $p \notin S_{j-1}[p]$, then $F(S_{j-1}[p], p)$ has two types of support hyperplanes. The first type consists of support hyperplanes of $S_{j-1}[p]$ that separate p from $S_{j-1}[p]$. The second type consists of hyperplanes that pass through p and are tangent to $S_{j-1}[p]$ at some $(d-2)$-dimensional faces of $S_{j-1}[p]$. Since the side length of G_{v_j} is $\varepsilon\delta/(2\sqrt{d})$, the number of cells in G_{v_j} that intersect the boundary of B_{v_j} is $O(\varepsilon^{1-d})$. Hence, $\mathrm{conv}(G_{v_j})$ has $O(\varepsilon^{(1-d)\lfloor d/2 \rfloor})$ complexity which is $O(N)$, and $S_j[p]$ can be computed in $O(N \log N + N^{\lfloor d/2 \rfloor})$ time [7]. □

Algorithm 1 Simplify

Input: A stream $\tau = (v_1, v_2, \ldots)$, ε, and δ.
Output: The output curve σ consists of the sequence of vertices output so far and the vertices in the buffer $\tilde{\sigma}$. After processing the last vertex v_i at the end of the stream, Simplify returns (P, S_i), which will be useful for the k-simplification problem.

function Simplify(ε, δ)
  read v_1 from the data stream; $i \leftarrow 1$; *init* $\leftarrow$ true; $\tilde{\sigma} \leftarrow \emptyset$
  while true do
    if *init* = true **then**
      init $\leftarrow$ false
      output the vertices in $\tilde{\sigma}$ ▷ output the line segment in $\tilde{\sigma}$
      $\tilde{\sigma} \leftarrow (v_i)$ ▷ view v_i as a degenerate line segment
      $P \leftarrow$ the grid points of G_{v_i}
      $S_i[p] \leftarrow p$ for all $p \in P$
    else
      choose a vertex q of some non-empty $S_i[p]$
      $\tilde{\sigma} \leftarrow (p, q)$ ▷ update the last edge of the simplified curve
    end if
    if end of data stream **then**
      output $\tilde{\sigma}$
      return (P, S_i) ▷ will be useful for k-simplification
    else
      read v_{i+1} from the data stream
    end if
    $S_{i+1}[p] \leftarrow \mathrm{conv}(G_{v_{i+1}}) \cap F(S_i[p], p)$ for all $p \in P$
    delete S_i
    init $\leftarrow$ true if $S_{i+1}[p] = \emptyset$ for all $p \in P$
    $i \leftarrow i + 1$
  end while
end function

Next, we prove the solution quality guarantees.

Lemma 2. *Let $\tau[v_1, v_i]$ be the prefix in the stream so far. The simplified curve σ satisfies the properties that $d_F(\sigma, \tau[v_1, v_i]) \leq (1+\varepsilon)\delta$ and $|\sigma| \leq 2\kappa(\tau[v_1, v_i], \delta) - 2$.*

Proof. Let σ be the concatenation of line segments $s_1, s_2, \ldots$. That is, the end of s_j is connected to the start of s_{j+1} for $j \geq 1$ to produce σ.

The start of s_1 is a grid point p_1 of G_{v_1}, the end of s_1 is a vertex p_b of $S_b[p_1] \subseteq \mathrm{conv}(G_{v_b})$ for some $b > 1$, and the start of s_2 is a grid point p_{b+1} in $G_{v_{b+1}}$. By Lemma 1(i), s_1 stabs $\mathrm{conv}(G_{v_1}), \ldots, \mathrm{conv}(G_{v_b})$ in order. Let $p_2, \ldots, p_{b-1}$ be some points on s_1 in this order such that $p_a \in s_1 \cap \mathrm{conv}(G_{v_a})$ for $a \in [2, b-1]$. Then, $d(v_a, p_a) \leq (1+\varepsilon)\delta$ for $a \in [b]$. So we can match $v_a v_{a+1}$ to $p_a p_{a+1}$ for $a \in [b-1]$ by linear interpolation within a distance of $(1+\varepsilon)\delta$. Similarly, we can match $v_b v_{b+1}$ to the segment $p_b p_{b+1}$ between s_1 and s_2 by linear interpolation within a distance of $(1+\varepsilon)\delta$. Continuing this way shows that $d_F(\sigma, \tau[v_1, v_i]) \leq (1+\varepsilon)\delta$.

Let $k = \kappa(\tau[v_1, v_i], \delta)$. Let $\gamma = (u_1, u_2, \ldots, u_k)$ be a polygonal curve of size k such that $d_F(\gamma, \tau[v_1, v_i]) \leq \delta$. For $a \in [i]$, let x_a be a point on γ mapped to v_a by a Fréchet matching between γ and $\tau[v_1, v_i]$. We can assume that $x_1 = u_1$ and $x_i = u_k$. Let $b = \max\{a \in [i] : x_a \in u_1 u_2\}$. So $x_1 u_2$ stabs balls of radii δ centered at $v_1, v_2, \ldots, v_b$ in order. Refer to Figure 2. There

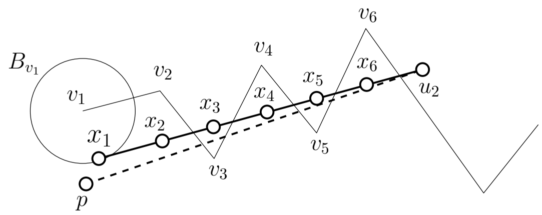

Figure 2: The points x_i for $i \in [6]$ are matched to v_i for $i \in [6]$ by the Fréchet matching.

is a grid point p of G_{v_1} within a distance $\varepsilon\delta/2$ from x_1. By a linear interpolation between x_1u_2 and pu_2, for $i \in [b]$, x_i is mapped to a point on pu_2 at distance no more than $(1+\varepsilon/2)\delta$ from v_i. That is, pu_2 stabs the balls centered at $v_1, v_2, \ldots, v_b$ with radii $(1+\varepsilon/2)\delta$ in order. It follows that pu_2 stabs $\mathrm{conv}(G_{v_1}), \ldots, \mathrm{conv}(G_{v_b})$ in order, which implies that $S_a[p] \neq \emptyset$ for $a \in [b]$ by Lemma 1(i). Therefore, Simplify outputs the first segment when it encounters v_{c+1} for some $c \geq b$. We charge the first output segment to u_1u_2. The vertex v_{c+1} is matched to the point x_{c+1} that lies on u_ju_{j+1} for some $j \geq 2$. We repeat the above argument to $x_{c+1}u_{j+1}$. This way, every output segment is charged to a unique edge of γ, so there are at most $k-1$ output segments. Concatenating them uses at most $k-2$ more edges. Hence, $|\sigma| \leq 2k-2$. ☐

We are now ready to prove the performance guarantees offered by Simplify.

Theorem 3. *Let τ be a polygonal curve in $\mathbb{R}^d$ that arrives in a data stream. Let $\alpha = 2(d-1)\lfloor d/2\rfloor^2 + d$. For every $\delta > 0$ and every $\varepsilon \in (0,1)$, the output curve σ of* Simplify(ε,δ) *satisfies $d_F(\sigma,\tau) \leq (1+\varepsilon)\delta$ and $|\sigma| \leq 2\kappa(\tau,\delta) - 2$. The working storage is $O(\varepsilon^{-\alpha})$. Each vertex is processed in $O(\varepsilon^{-\alpha}\log\frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(\varepsilon^{-\alpha})$ time for $d \geq 4$.*

Proof. The guarantees on σ follow from Lemma 2. We will bound the number of support hyperplanes and complexities of $S_j[p]$ and $F(S_{j-1}[p],p)$. The working storage and processing time then follow from Lemma 1(ii) and the fact that $|P| = O(\varepsilon^{-d})$.

A *bounding halfspace* of a convex polytope O is a halfspace that contains O and is bounded by the support hyperplane of a facet of O. We count the bounding halfspaces of $S_j[p]$. Some are bounding halfspaces of $F(S_{j-1}[p],p)$ whose boundaries pass through p and are tangent to $S_{j-1}[p]$. We call them the *pivot* bounding halfspaces. The remaining ones are bounding halfspaces of $\mathrm{conv}(G_{v_a})$ for some $a \leq j$. We call these *non-pivot* bounding halfspaces.

Since $\mathrm{conv}(G_{v_a})$'s are translates of each other, their facets share a set V of unit outward normals. We have $|V| \leq |\mathrm{conv}(G_{v_a})| = O(\varepsilon^{(1-d)\lfloor d/2\rfloor})$. No two facets of $S_j[p]$ have the same vector in V because the two corresponding bounding halfspaces would be identical or nested—neither is possible. This feature helps us to prevent a cascading growth in the complexities of $S_j[p]$ in the inductive construction. It also implies that there are $O(\varepsilon^{(1-d)\lfloor d/2\rfloor})$ non-pivot bounding halfspaces of $S_j[p]$.

Take a pivot bounding halfspace h of $S_j[p]$. Suppose that h was introduced for the first time as a bounding halfspace of $S_i[p]$ for some $i \leq j$. The boundary of h, denoted by ∂h, passes through p and is tangent to $S_{i-1}[p]$ at a $(d-2)$-dimensional face f of $S_{i-1}[p]$. Let g_1 and g_2 be the bounding halfspaces of $S_{i-1}[p]$ whose boundaries contain the two facets of $S_{i-1}[p]$ incident to f. We make three observations. First, $F(S_{i-1}[p],p)$ lies inside the cone C_i of rays that shoot from p towards $S_{i-1}[p]$. Second, h is a bounding halfspace of C_i. Third, $S_a[p] \subseteq C_i$ for all $a \in [i,j]$.

We claim that neither ∂g_1 nor ∂g_2 passes through p. Neither g_1 nor g_2 is equal to h because h was not introduced before $S_i[p]$. If both ∂g_1 and ∂g_2 pass through p, then f spans a $(d-2)$-dimensional affine subspace ℓ that passes through p. But then ∂h meets C_i at ℓ only, which is

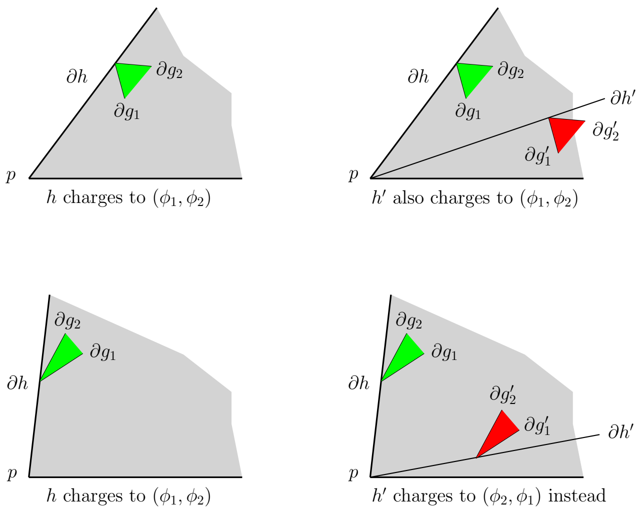

Figure 3: The shaded cones denote C_i. Let g'_1 and g'_2 be the bounding halfspaces of $S_b[p]$ such that $\partial h'$ is spanned by p and $\partial g'_1 \cap \partial g'_2$. Without loss of generality, assume that g_i and g'_i have the same unit outward normal ϕ_i for $i \in \{1, 2\}$. A local neighborhood of $g_1 \cap g_2$ is shown in green, and a local neighborhood of $g'_1 \cap g'_2$ is shown in red. In the first row, both h and h' are charged to (ϕ_1, ϕ_2), and $\partial h'$ separates $g'_1 \cap g'_2 \cap C_i$ from ∂h. Note that $S_b[p] \subseteq g'_1 \cap g'_2 \cap C_i$. In the second row, the alternative configuration is shown in which $\partial h'$ does not separate $g'_1 \cap g'_2 \cap C_i$ from ∂h. In this case, a ray that shoots from p to any interior point of $g'_1 \cap g'_2$ arbitrarily close to $\partial g'_1 \cap \partial g'_2$ must intersect $\partial g'_2$ before $\partial g'_1$. Therefore, h' is charged to (ϕ_2, ϕ_1) instead of (ϕ_1, ϕ_2).

a contradiction because ∂h should support a facet of $S_j[p]$. If either ∂g_1 or ∂g_2 passes through p, say ∂g_2, then the affine subspace spanned by f does not pass through p. But then p and f span a unique hyperplane, giving the contradiction that $g_2 = h$.

By our claim, g_1 and g_2 are non-pivot bounding halfspaces of $S_{i-1}[p]$. We charge h to (ϕ_1, ϕ_2) or (ϕ_2, ϕ_1), where ϕ_1 and ϕ_2 are the outward unit normals of g_1 and g_2, respectively. The pair (ϕ_1, ϕ_2) is charged if a ray that shoots from p to an interior point of $g_1 \cap g_2$ arbitrarily near $\partial g_1 \cap \partial g_2$ hits ∂g_1 before ∂g_2. Otherwise, the pair (ϕ_2, ϕ_1) is charged. Suppose that h is charged to (ϕ_1, ϕ_2). We argue that (ϕ_1, ϕ_2) is not charged again at some $(d-2)$-dimensional face of $S_a[p]$ for all $a \in [i, j-1]$ due to another pivot bounding halfspace of $S_j[p]$.

Assume to the contrary that (ϕ_1, ϕ_2) is charged again at some $(d-2)$-dimensional face of $S_b[p]$ for some $b \in [i, j-1]$ due to a pivot bounding halfspace h' of $S_j[p]$. Either the dimension of $\partial h' \cap C_i$ is zero or one, or the dimension of $\partial h' \cap C_i$ is two. In the first case, $\partial h'$ cannot support any facet of $S_j[p]$, a contradiction. In the second case, since h' is charged to the same ordered pair (ϕ_1, ϕ_2), $\partial h'$ must separate $S_b[p]$ from ∂h. Figure 3 illustrates the configuration. Therefore, the cone of rays C_b that shoot from p towards $S_b[p]$ meets ∂h only at p. However, $S_j[p] \subseteq C_b$, which means that ∂h cannot support any facet of $S_j[p]$, a contradiction.

In all, $S_j[p]$ has $O(\varepsilon^{(1-d)\lfloor d/2 \rfloor})$ non-pivot bounding halfspaces and at most $|V|(|V|-1) = O(\varepsilon^{2(1-d)\lfloor d/2 \rfloor})$ pivot bounding halfspaces. The same bounds hold for $S_{j-1}[p]$. If a support hyperplane L of $F(S_{j-1}[p], p)$ is not a support hyperplane of $S_{j-1}[p]$, then L passes through p and is tangent to $S_{j-1}[p]$ at some $(d-2)$-dimensional face f. Since L is different from the support hyperplanes of the two facets of $S_{j-1}[p]$ incident to f, these two hyperplanes do not pass through p. So the two facets incident to f induce some non-pivot bounding halfspaces of $S_{j-1}[p]$, implying that $F(S_{j-1}[p], p)$ has $O(\varepsilon^{2(1-d)\lfloor d/2 \rfloor})$ bounding halfspaces. It follows that $|S_j[p]|$ and $|F(S_{j-1}[p], p)|$ are $O(\varepsilon^{2(1-d)\lfloor d/2 \rfloor^2})$. $\square$

3 Streaming algorithm for the k-simplification problem

3.1 Algorithm

Let σ_i^* be the optimal solution for the k-simplification problem for $\tau[v_1, v_i]$. If we knew $\delta_i^* = d_F(\sigma_i^*, \tau[v_1, v_i])$, we could call SIMPLIFY$(\varepsilon, \delta_i^*)$ to obtain an approximate solution for the k-simplification problem. Therefore, we try to estimate δ_i^* on the fly.

We maintain an output simplified curve σ_i in the working storage after processing every vertex v_i in the stream. The first simplified curve σ_{2k-1} is obtained by calling SIMPLIFY, via an invocation of a new procedure COMPRESS, on $\tau[v_1, v_{2k-1}]$ with δ equal to some value δ_{2k-1}. Given a curve of size $2k-1$, COMPRESS simplifies it to a curve of size at most $2k-2$.

For $i \geq 2k$, when a vertex v_i arrives in the stream, either we update the last edge of σ_{i-1} to produce σ_i as in SIMPLIFY, or we start a new segment (initially just the vertex v_i) as in SIMPLIFY. Suppose that we start a new segment. If $|\sigma_{i-1}| < 2k-2$, we can append v_i to σ_{i-1} to form σ_i. Otherwise, σ_i is obtained by calling SIMPLIFY, via an invocation of COMPRESS, on the concatenation $\sigma_{i-1} \circ (v_i)$ with δ equal to some value δ_i to be specified later. We will prove in Lemmas 4–8 that $\delta_i \leq (1 + O(\varepsilon))\delta_i^*$. Afterward, we repeat the above to process the next vertex in the stream. The above processing is elaborated in REDUCE in Algorithm 2.

We define a curve τ_i for every $i \geq 2k-1$ in the comments in lines 5, 10, and 21 of REDUCE. These curves facilitate the analysis, but they are not maintained by the algorithm as variables. For each i, τ_i is the curve from which the simplification σ_i is computed. Therefore, $\tau_{2k-1} = \tau[v_1, v_{2k-1}]$ (line 5). Consider any $i \geq 2k$. If we call COMPRESS to simplify $\sigma_{i-1} \circ (v_i)$ to a curve σ_i of size at most $2k-2$, we set $\tau_i = \sigma_{i-1} \circ (v_i)$ (line 21). Otherwise, σ_i is obtained by simplifying $\tau_{i-1} \circ (v_i)$ whether we start a new segment at v_i or not, so $\tau_i = \tau_{i-1} \circ (v_i)$ (line 10).

The input parameter r of REDUCE needs an explanation. We will show how to compute a lower bound $\delta_{\min} > 0$ for $d_F(\sigma_{2k-1}^*, \tau[v_1, v_{2k-1}])$. For $i \geq 2k-1$, let $r_i \geq 1$ and $s_i \geq 0$ be integers such that

$$\frac{1}{\varepsilon^{s_i}}\delta_{\min} \leq \frac{(1+\varepsilon)^{r_i-1}}{\varepsilon^{s_i}}\delta_{\min} \leq \delta_i^* \leq \frac{(1+\varepsilon)^{r_i}}{\varepsilon^{s_i}}\delta_{\min} \leq \frac{1}{\varepsilon^{s_i+1}}\delta_{\min}.$$

We will show that if we can start the processing of the data stream with an initial error tolerance of $\delta_{2k-2} = \varepsilon(1+\varepsilon)^{r_i+1}(1-4\varepsilon)^{-1}\delta_{\min}$, then the output curve σ_i will be within a Fréchet distance of $\varepsilon^{-s_i}(1+\varepsilon)^{r_i+1}(1-4\varepsilon)^{-1}\delta_{\min}$ from $\tau[v_1, v_i]$. By the definitions of r_i and s_i, the error is at most $(1 + O(\varepsilon))\delta_i^*$. The correct values of r_i and s_i are unknown to us. Nevertheless, no matter what s_i is, we have $1 \leq r_i \leq \lfloor \log_{1+\varepsilon} \frac{1}{\varepsilon} \rfloor$. We launch $\lfloor \log_{1+\varepsilon} \frac{1}{\varepsilon} \rfloor = O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon})$ independent runs of REDUCE for each r between 1 and $\lfloor \log_{1+\varepsilon} \frac{1}{\varepsilon} \rfloor$. After processing the arriving vertex v_i, the run with the minimum δ_i gives the right answer. This idea of using multiple independent runs to capture the right δ was proposed in [13] for streaming k-simplification under the discrete Fréchet distance. We still have to obtain s_i. This is achieved by establishing a lower bound for δ_i^* (Lemma 5) and computing a particular upper bound for δ_i^* in line 2 of COMPRESS.

We assume that no three consecutive vertices in the stream are collinear. We enforce it using $O(1)$ more working storage as follows. Remember the last two vertices (x, y) in the stream, but do not feed y to REDUCE yet. Wait until the next vertex z arrives. If x, y and z are not colinear, feed y to REDUCE and remember (y, z) as the last two vertices. If x, y and z are collinear, then drop y, make (x, z) the last two vertices, and wait for the next vertex.

Algorithm 3 shows the procedure COMPRESS. If COMPRESS is called when processing v_i, its task is to simplify $\tau_i = \sigma_{i-1} \circ (v_i)$ to a curve σ_i of size at most $2k-2$ with error estimate $\delta_i = \varepsilon^{-t}\delta_{i-1}$, where t is some judiciously chosen positive integer.

Algorithm 2 REDUCE

Input: A data stream $\tau = (v_1, v_2, \ldots)$ and three parameters r, ε, and k.
Output: Let $\tau[v_1, v_i]$ be the prefix in the stream so far. A curve σ_i is maintained such that $|\sigma_i| \leq 2k-2$ and $d_F(\sigma_i, \tau[v_1, v_i]) \leq (1+O(\varepsilon))\delta_i$.

procedure REDUCE(r, ε, k)
  read the first $2k-1$ vertices $v_1, \ldots, v_{2k-1}$ of τ
  $\delta_{\min} \leftarrow \frac{1}{2}\min\{d(v_i, v_{i-1}v_{i+1}) : i \in [2, 2k-2]\}$
  $\delta_{2k-2} \leftarrow \varepsilon(1+\varepsilon)^{r+1}(1-4\varepsilon)^{-1}\delta_{\min}$
  $\sigma_{2k-2} \leftarrow \tau[v_1, v_{2k-2}]$ ▷ $\tau_{2k-1} \leftarrow \tau[v_1, v_{2k-1}]$
  $(\sigma_{2k-1}, \delta_{2k-1}, P, S_{2k-1}) \leftarrow$ COMPRESS$(\tau[v_1, v_{2k-1}], \varepsilon, k, \delta_{2k-2})$
  $i \leftarrow 2k$
  while true do
    read v_i from the data stream
    $\delta_i \leftarrow \delta_{i-1}$ ▷ $\tau_i \leftarrow \tau_{i-1} \circ (v_i)$
    $S_i[p] \leftarrow \mathrm{conv}(G_{v_i}) \cap F(S_{i-1}[p], p)$ for all $p \in P$, where G_{v_i} is defined with $\delta = \delta_i$
    if $S_i[p] \neq$ null for some $p \in P$ **then**
      ▷ update the last possibly degenerate segment as in SIMPLIFY
      $q \leftarrow$ any vertex of $S_i[p]$
      $\sigma_i \leftarrow \sigma_{i-1}$ with the last segment replaced by pq
    else if $|\sigma_{i-1}| < 2k-2$ **then** ▷ start a new segment as in SIMPLIFY
      $P \leftarrow$ the grid points of G_{v_i} ▷ G_{v_i} is defined with $\delta = \delta_i$
      $S_i[p] \leftarrow p$ for all $p \in P$
      $\sigma_i \leftarrow \sigma_{i-1} \circ (v_i)$ ▷ treat v_i as a degenerate segment
    else ▷ simplify $\sigma_{i-1} \circ (v_i)$ to produce σ_i
      $(\sigma_i, \delta_i, P, S_i) \leftarrow$ COMPRESS$(\sigma_{i-1} \circ (v_i), \varepsilon, k, \delta_{i-1})$ ▷ $\tau_i = \sigma_{i-1} \circ (v_i)$
    end if
    delete σ_{i-1}, δ_{i-1}, and S_{i-1}
    $i \leftarrow i+1$
  end while
end procedure

Algorithm 3 COMPRESS

Input: A polygonal curve $(x_1, \ldots, x_{2k-1})$ and three parameters ε, k, and δ.
Output: $(\zeta, \varepsilon^{-t}\delta, P, S)$, where ζ is the output curve of calling SIMPLIFY$(\varepsilon, \varepsilon^{-t}\delta)$ on $(x_1, \ldots, x_{2k-1})$ for some $t \in \mathbb{N}$, and (P, S) are the output returned by SIMPLIFY after processing x_{2k-1}.

function COMPRESS$((x_1, \ldots, x_{2k-1}), \varepsilon, k, \delta)$
  $t \leftarrow \min\left\{i \in \mathbb{N} : i \geq 1, \varepsilon^{-i}\delta \geq \frac{1}{2}\min\{d(x_j, x_{j-1}x_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\}\right\}$
  $\zeta \leftarrow$ output curve produced by calling SIMPLIFY$(\varepsilon, \varepsilon^{-t}\delta)$ on $(x_1, \ldots, x_{2k-1})$
  $(P, S) \leftarrow$ output returned by the above call of SIMPLIFY after processing x_{2k-1}
  return $(\zeta, \varepsilon^{-t}\delta, P, S)$
end function

3.2 Analysis

Lemma 4 below gives a lower bound for the Fréchet distance between a curve of size m and another curve of size n such that $m \geq 2n - 1$. An application of this result shows that $\delta_{\min}$ computed in line 3 of REDUCE is a lower bound for $d_F(\sigma^*_{2k-1}, \tau[v_1, v_{2k-1}])$ as $|\sigma^*_{2k-1}| = k$.

Lemma 4. *Let $\xi = (p_1, \ldots, p_m)$ and $\zeta = (q_1, \ldots, q_n)$ be any two curves such that $m \geq 2n - 1$. Then, $d_F(\xi, \zeta) \geq \frac{1}{2} \min\{d(p_i, p_{i-1}p_{i+1}) : i \in [2, m-1]\}$.*

Proof. Let $\mathcal{M}$ be a Fréchet matching between ξ and ζ. Since $m \geq 2n - 1$, there must exist an index $c \in [2, m-1]$ such that $\mathcal{M}$ matches $\xi[p_{c-1}, p_{c+1}]$ to an edge of ζ. Let xy be the line segment to which $\xi[p_{c-1}, p_{c+1}]$ is matched by $\mathcal{M}$. Let $r = d_{\mathcal{M}}(\xi[p_{c-1}, p_{c+1}], xy)$. We claim that $r \geq \frac{1}{2} d(p_c, p_{c-1}p_{c+1})$. Since $d(p_{c-1}, x) \leq r$ and $d(p_{c+1}, y) \leq r$, by linear interpolation, every point on xy is at a distance no more than r from $p_{c-1}p_{c+1}$. Let z be the point in xy that is matched with p_c by $\mathcal{M}$. We have $d(p_c, p_{c-1}p_{c+1}) \leq d(p_c, z) + d(z, p_{c-1}p_{c+1})$. We have shown that $d(z, p_{c-1}p_{c+1}) \leq r$, so $d(p_c, p_{c-1}p_{c+1}) \leq 2r$, completing the proof of our claim. Hence, $d_F(\xi, \zeta) \geq r \geq \frac{1}{2} d(p_c, p_{c-1}p_{c+1}) \geq \frac{1}{2} \min\{d(p_i, p_{i-1}p_{i+1}) : i \in [2, m-1]\}$. □

Lemma 5 below shows that a curve $(p_1, \ldots, p_{2k-1})$ of size $2k-1$ can be simplified to a curve of size at most $2k-2$ using an error tolerance of $\frac{1}{2} \min\{d(p_j, p_{j-1}p_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\}$. By Lemma 5, COMPRESS always returns a curve of size at most $2k - 2$.

Lemma 5. *Let $\xi = (p_1, \ldots, p_{2k-1})$. Assume that no three consecutive vertices are collinear. Let $\hat{\delta}$ be the minimum value such that calling* SIMPLIFY$(\varepsilon, \hat{\delta})$ *on ξ produces a curve of size at most $2k - 2$. Then,*

$$\hat{\delta} \leq \frac{1}{2} \min\{d(p_j, p_{j-1}p_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\} \leq (1 + \varepsilon)\hat{\delta}.$$

Proof. Consider the run of SIMPLIFY$(\varepsilon, \hat{\delta})$ on ξ. Let $p_{i_1}, p_{i_2} \ldots$ be the vertices in order along ξ at which SIMPLIFY starts a new segment. Note that $i_1 = 1$ and $i_{j+1} \geq i_j + 2$ for all j.

SIMPLIFY replaces each subcurve $\xi[p_{i_j}, p_{i_{j+1}-1}]$ by a segment. The simplified curve is a concatenation of these segments. Since $|\xi| = 2k - 1$, there must be an index j such that $i_{j+1} > i_j + 2$ so that SIMPLIFY$(\varepsilon, \hat{\delta})$ simplifies ξ to a curve of size at most $2k - 2$. Let $b = \min\{j : i_{j+1} > i_j + 2\}$. Because $i_1 = 1$ and $i_{j+1} = i_j + 2$ for all $j \in [b-1]$, we know that i_j is odd for every $j \in [b]$ and $i_b + 1$ is even. It follows from the choice of b that SIMPLIFY does not start a new segment at p_{i_b+2}. By the working of SIMPLIFY$(\varepsilon, \hat{\delta})$, there is a grid point p in $G_{p_{i_b}}$ (defined with $\delta = \hat{\delta}$) such that $S_{i_b+2}[p] \neq \emptyset$. By the reasoning in the proof of Lemma 2, for any point $x \in S_{i_b+2}[p]$, $d_F(px, \xi[p_{i_b}, p_{i_b+2}]) \leq (1 + \varepsilon)\hat{\delta}$. On the other hand, by Lemma 4, $d_F(px, \xi[p_{i_b}, p_{i_b+2}]) \geq \frac{1}{2} d(p_{i_b+1}, p_{i_b}p_{i_b+2})$. Hence, $(1 + \varepsilon)\hat{\delta} \geq \frac{1}{2} d(p_{i_b+1}, p_{i_b}p_{i_b+2}) \geq \frac{1}{2} \min\{d(p_j, p_{j-1}p_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\}$.

It remains to argue that $\hat{\delta} \leq \frac{1}{2} \min\{d(p_j, p_{j-1}p_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\}$. Let $\delta = \frac{1}{2} \min\{d(p_j, p_{j-1}p_{j+1}) : j \in [2, 2k-2], j \text{ is even}\}$. It suffices to show that calling SIMPLIFY(ε, δ) on ξ returns a curve of size at most $2k - 2$. Let $p_{a_1}, p_{a_2}, \ldots$ be the vertices in order along ξ at which SIMPLIFY(ε, δ) starts a new segment. Let c be the even number in $[2, 2k-2]$ such that $d(p_c, p_{c-1}p_{c+1}) = \min\{d(p_j, p_{j-1}p_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\}$.

Suppose that there is an index j that satisfies $a_j \leq c - 1$ and $a_{j+1} > a_j + 2$. Since $\xi[p_{a_j}, p_{a_{j+1}-1}]$ is simplified to a line segment, the prefix $\xi[p_1, p_{a_{j+1}-1}]$ is simplified to a curve of size at most $a_{j+1} - 2$, i.e., at least one vertex less. We are done because the size of the whole simplified curve must be at most $2k - 2$.

The remaining possibility is that $a_{j+1} = a_j + 2$ for all $a_j \in [c-1]$. Since $a_1 = 1$ and c is even, we have $a_1 = 1, a_2 = 3, a_3 = 5, \ldots, a_j = c - 1$. It follows that SIMPLIFY starts a new segment at p_{c-1}. Let $r = d(p_c, p_{c-1}p_{c+1})$.

We claim that there is a line segment xy such that $d_F(xy, \xi[p_{c-1}, p_{c+1}]) \leq r/2$. Let z be the nearest point in $p_{c-1}p_{c+1}$ to p_c. So zp_c has length r. Let z' be the midpoint of zp_c. Translate $p_{c-1}p_{c+1}$ by the vector $z' - z$ to obtain a segment xy. Observe that $z' \in xy$ and $d(x, p_{c-1}) = d(y, p_{c+1}) = d_F(z', p_c) = r/2$. By linear interpolations between xz' and $p_{c-1}p_c$ and between $z'y$ and p_cp_{c+1}, we have $d_F(xy, \xi[p_{c-1}, p_{c+1}]) \leq r/2$, establishing our claim.

By the choice of c, $\frac{1}{2}r = \frac{1}{2}d(p_c, p_{c-1}c_{c+1}) = \delta$. Then, our claim implies that xy stabs balls of radii δ centered at p_{c-1}, p_c, p_{c+1} in order. There is a grid point p' in $G_{p_{c-1}}$ (defined using δ) within a distance $\varepsilon\delta/2$ from x. By a linear interpolation between $p'y$ and xy, we know that $p'y$ stabs $\mathrm{conv}(G_{p_{c-1}}), \mathrm{conv}(G_{p_c}), \mathrm{conv}(G_{p_{c+1}})$ in order. Therefore, $S_{c+1}[p'] \neq \emptyset$ by Lemma 1(i), implying that Simplify does not start a new segment at p_{c+1}. Hence, Simplify replaces a subcurve $\xi[p_{c-1}, p_e]$ for some $e \geq c+1$ by a segment and produces a curve of size at most $2k-2$. This proves that $\delta \geq \hat{\delta}$. □

Recall the curves τ_i for $i \geq 2k-1$ defined in the comments in lines 5, 10, and 21. The next result follows immediately from the working of Reduce.

Lemma 6. *For $i \geq 2k-1$, σ_i computed by* Reduce *can also be produced by calling* Simplify(ε, δ_i) *on τ_i.*

We can also show that τ_i is a faithful approximation of $\tau[v_1, v_i]$.

Lemma 7. *Assume that $\varepsilon \in (0, 1/3]$. For $i \geq 2k-1$, $d_F(\tau_i, \tau[v_1, v_i]) \leq 2\varepsilon\delta_i$.*

Proof. We prove the lemma by induction on i. As $\tau_{2k-1} = \tau[v_1, v_{2k-1}]$, $d_F(\tau_{2k-1}, \tau[v_1, v_{2k-1}])$ is zero. Assume that the lemma is true for some $i-1 \geq 2k-1$. There are two cases.

If $\delta_i = \delta_{i-1}$, then $\tau_i = \tau_{i-1} \circ (v_i)$. Since $d_F(\tau_{i-1}, \tau[v_1, v_{i-1}]) \leq 2\varepsilon\delta_{i-1}$, the last vertex of τ_{i-1} is within a distance $2\varepsilon\delta_{i-1}$ from v_{i-1}. A linear interpolation between $v_{i-1}v_i$ and the last edge of τ_i shows that $d_F(\tau_i, \tau[v_1, v_i]) \leq d_F(\tau_{i-1}, \tau[v_1, v_{i-1}]) \leq 2\varepsilon\delta_{i-1} = 2\varepsilon\delta_i$.

If $\delta_i > \delta_{i-1}$, then $\tau_i = \sigma_{i-1} \circ (v_i)$. Compress ensures that $\delta_i \geq \varepsilon^{-1}\delta_{i-1}$. By Lemma 6 and Theorem 3, $d_F(\tau_{i-1}, \sigma_{i-1}) \leq (1+\varepsilon)\delta_{i-1}$. Therefore, $d_F(\sigma_{i-1}, \tau[v_1, v_{i-1}]) \leq d_F(\tau_{i-1}, \sigma_{i-1}) + d_F(\tau_{i-1}, \tau[v_1, v_{i-1}]) \leq (1+3\varepsilon)\delta_{i-1} \leq \varepsilon(1+3\varepsilon)\delta_i \leq 2\varepsilon\delta_i$. A linear interpolation between $v_{i-1}v_i$ and the last edge of τ_i gives $d_F(\tau_i, \tau[v_1, v_i]) \leq d_F(\sigma_{i-1}, \tau[v_1, v_{i-1}]) \leq 2\varepsilon\delta_i$. □

Recall that σ_i^* is the curve of size k at minimum Fréchet distance from $\tau[v_1, v_i]$ and that $\delta_i^* = d_F(\sigma_i^*, \tau[v_1, v_i])$. Also, recall that for $i \geq 2k-1$, $r_i \geq 1$ and $s_i \geq 0$ are integers that satisfy the following inequalities:

$$\frac{1}{\varepsilon^{s_i}}\delta_{\min} \leq \frac{(1+\varepsilon)^{r_i-1}}{\varepsilon^{s_i}}\delta_{\min} \leq \delta_i^* \leq \frac{(1+\varepsilon)^{r_i}}{\varepsilon^{s_i}}\delta_{\min} \leq \frac{1}{\varepsilon^{s_i+1}}\delta_{\min}. \tag{1}$$

By Lemma 4, $\delta_{\min} \leq \delta_{2k-1}^*$. Also, $\delta_a^* \leq \delta_b^*$ for all $a < b$ because a Fréchet matching between σ_b^* and $\tau[v_1, v_b]$ also matches $\tau[v_1, v_a]$ to a curve of size at most k. Therefore, $\delta_{\min} \leq \delta_i^*$ for $i \geq 2k-1$, which implies that r_i and s_i are well defined for $i \geq 2k-1$. Note that r_i is an integer between 1 and $\lfloor \log_{1+\varepsilon}(1/\varepsilon) \rfloor$.

Lemma 8. *For all $i \geq 2k-1$ and $\varepsilon \in \left(0, \frac{1}{17}\right)$,* Reduce$(r_i, \varepsilon, k)$ *computes $\delta_i \leq (1+8\varepsilon)\delta_i^*$.*

Proof. We prove that Reduce(r_i, ε, k) computes $\delta_i \leq \varepsilon^{-s_i}(1+\varepsilon)^{r_i+1}(1-4\varepsilon)^{-1}\delta_{\min}$. By (1), this is at most $(1+\varepsilon)^2(1-4\varepsilon)^{-1}\delta_i^* \leq (1+8\varepsilon)\delta_i^*$ for $\varepsilon \leq 1/17$.

Consider the case of $i = 2k-1$. We examine the call Reduce$(r_{2k-1}, \varepsilon, k)$. By Theorem 3, calling Simplify$(\varepsilon, \delta_{2k-1}^*)$ on $\tau_{2k-1} = \tau[v_1, v_{2k-1}]$ produces a curve of size at most $2k-2$. Then, Lemma 5 and (1) imply that $\frac{1}{2}\min\{d(v_j, v_{j-1}v_{j+1}) : j \in [2, 2k-2],\ j \text{ is even}\} \leq (1+\varepsilon)\delta_{2k-1}^* \leq \varepsilon^{-s_{2k-1}}(1+\varepsilon)^{r_{2k-1}+1}\delta_{\min}$. Since the input parameter r of Reduce is equal to r_{2k-1}, line 4 of

Reduce sets $\delta_{2k-2} = \varepsilon(1+\varepsilon)^{r_{2k-1}+1}(1-4\varepsilon)^{-1}\delta_{\min}$. As a result, line 2 of Compress covers $\varepsilon^{-i}\delta_{2k-2}$ for $i \geq 1$, which are $\varepsilon^{-j}(1+\varepsilon)^{r_{2k-1}+1}(1-4\varepsilon)^{-1}\delta_{\min}$ for $j \geq 0$. Therefore, line 2 of Compress must set δ_{2k-1} to a value no more than $\varepsilon^{-s_{2k-1}}(1+\varepsilon)^{r_{2k-1}+1}(1-4\varepsilon)^{-1}\delta_{\min}$ because this quantity is greater than $\frac{1}{2}\min\bigl\{d(v_j, v_{j-1}v_{j+1}) : j \in [2, 2k-2],\, j \text{ is even}\bigr\}$. The base case is thus taken care of.

Consider an index $i > 2k-1$. We examine the call Reduce(r_i, ε, k). Define:

$$\Delta = \varepsilon^{-s_i}(1+\varepsilon)^{r_i+1}\delta_{\min}. \tag{2}$$

Our goal is to prove that $\delta_i \leq (1-4\varepsilon)^{-1}\Delta$. We rewrite the middle inequalities in (1) as

$$(1+\varepsilon)^{-2}\Delta \leq \delta_i^* \leq (1+\varepsilon)^{-1}\Delta. \tag{3}$$

Define:

$$a = \max\bigl\{j \in [2k-2, i] : \delta_j \leq \varepsilon(1-4\varepsilon)^{-1}\Delta\bigr\}. \tag{4}$$

The index a is well defined because one can verify that $\delta_{2k-2} = \varepsilon(1+\varepsilon)^{r_i+1}(1-4\varepsilon)^{-1}\delta_{\min} \leq \varepsilon(1-4\varepsilon)^{-1}\Delta$.

By (4), (2), line 4 of Reduce, and line 2 of Compress, $\delta_a = \varepsilon^h(1-4\varepsilon)^{-1}\Delta$ for some integer $h \geq 1$. If $a = i$, then $\delta_i = \delta_a \leq \varepsilon(1-4\varepsilon)^{-1}\Delta$, so we are done. Suppose that $a < i$. Note that $\delta_{a+1} \neq \delta_a$ by (4), which implies that $\delta_{a+1} > \delta_a$. So $\tau_{a+1} = \sigma_a \circ (v_{a+1})$, and Compress$(\tau_{a+1}, \varepsilon, k, \delta_a)$ is called to produce σ_{a+1} and δ_{a+1}. We have

$$\begin{aligned} d_F(\sigma_{a+1}^*, \tau_{a+1}) &\leq d_F(\sigma_{a+1}^*, \tau[v_1, v_{a+1}]) + d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]) \\ &= \delta_{a+1}^* + d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]) \\ &\leq \delta_i^* + d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]). \end{aligned} \tag{5}$$

If $a+1 = 2k-1$, then $\tau_{a+1} = \tau_{2k-1} = \tau[v_1, v_{2k-1}]$, so $d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]) = 0$. Suppose that $a+1 > 2k-1$. By Lemma 6 and Theorem 3, $d_F(\sigma_a, \tau_a) \leq (1+\varepsilon)\delta_a$. By Lemma 7, $d_F(\tau_a, \tau[v_1, v_a]) \leq 2\varepsilon\delta_a$. So $d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]) \leq d_F(\sigma_a, \tau[v_1, v_a]) \leq d_F(\sigma_a, \tau_a) + d_F(\tau_a, \tau[v_1, v_a]) \leq (1+3\varepsilon)\delta_a$. Since $\delta_a \leq \varepsilon(1-4\varepsilon)^{-1}\Delta$, we conclude that

$$d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]) \leq \varepsilon(1+3\varepsilon)(1-4\varepsilon)^{-1}\Delta. \tag{6}$$

Plugging (6) into (5) and using (3), we obtain the following inequality for $\varepsilon < 1/17$:

$$d_F(\sigma_{a+1}^*, \tau_{a+1}) \leq \left(\frac{1}{1+\varepsilon} + \frac{\varepsilon + 3\varepsilon^2}{1-4\varepsilon}\right)\Delta \leq \frac{\Delta}{(1+\varepsilon)(1-4\varepsilon)}. \tag{7}$$

Consider the call of Simplify in Compress$(\tau_{a+1}, \varepsilon, k, \delta_a)$. By (7) and Theorem 3, calling Simplify on τ_{a+1} with an error tolerance of $\frac{1}{(1+\varepsilon)(1-4\varepsilon)}\Delta$ will produce a curve of size at most $2k-2$. Then, by Lemma 5, $\frac{1}{2}\min\bigl\{d(v_j, v_{j-1}v_{j+1}) : j \in [2, 2k-2],\, j \text{ is even}\bigr\} \leq (1+\varepsilon) \cdot \frac{1}{(1+\varepsilon)(1-4\varepsilon)}\Delta = (1-4\varepsilon)^{-1}\Delta$. Line 2 of Compress ensures that $\delta_{a+1} = \varepsilon^{-t}\delta_a = \varepsilon^{h-t}(1-4\varepsilon)^{-1}\Delta$ for the smallest integer $t \geq 1$ such that $\delta_{a+1} \geq \frac{1}{2}\min\bigl\{d(v_j, v_{j-1}v_{j+1}) : j \in [2, 2k-2],\, j \text{ is even}\bigr\}$. Hence, t cannot be greater than h. But t cannot be less than h because $\delta_{a+1} > \varepsilon(1-4\varepsilon)^{-1}\Delta$ by (4). So $\delta_{a+1} = (1-4\varepsilon)^{-1}\Delta$. For every $b \in [a+2, i]$,

$$\begin{aligned} d_F(\sigma_b^*, \tau_{a+1} \circ \tau[v_{a+2}, v_b]) &\leq d_F(\sigma_b^*, \tau[v_1, v_b]) + d_F(\tau_{a+1} \circ \tau[v_{a+2}, v_b], \tau[v_1, v_b]) \\ &\leq \delta_b^* + d_F(\tau_{a+1}, \tau[v_1, v_{a+1}]) \\ &\leq \delta_i^* + \varepsilon(1+3\varepsilon)(1-4\varepsilon)^{-1}\Delta \qquad (\because (6)) \\ &< (1-4\varepsilon)^{-1}\Delta \qquad (\because (3) \text{ and } \varepsilon \leq 1/17) \\ &= \delta_{a+1}. \end{aligned}$$

Then, when processing v_{a+2}, Lemma 6 and Theorem 3 imply that REDUCE executes lines 9–19 with $\delta_{a+2} = \delta_{a+1} = (1-4\varepsilon)^{-1}\Delta$ to produce σ_{a+2} of size at most $2k-2$. Applying this reasoning inductively gives $\delta_i = (1-4\varepsilon)^{-1}\Delta$. $\square$

Our main result on streaming k-simplification are as follows.

Theorem 9. *Let τ be a polygonal curve in $\mathbb{R}^d$ that arrives in a data stream. Let $\alpha = 2(d-1)\lfloor d/2\rfloor^2 + d$. There is a streaming algorithm that, for any integer $k \geq 2$ and any $\varepsilon \in (0, \frac{1}{17})$, maintains a curve σ such that $|\sigma| \leq 2k-2$ and $d_F(\tau,\sigma) \leq (1+\varepsilon)\cdot\min\{d_F(\tau,\sigma') : |\sigma'| \leq k\}$. It uses $O\big((k\varepsilon^{-1} + \varepsilon^{-(\alpha+1)})\log\frac{1}{\varepsilon}\big)$ working storage and processes each vertex of τ in $O(k\varepsilon^{-(\alpha+1)}\log^2\frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(k\varepsilon^{-(\alpha+1)}\log\frac{1}{\varepsilon})$ time for $d \geq 4$.*

Proof. Let $\delta^* = \min\{d_F(\tau,\sigma') : |\sigma'| \leq k\}$. By Lemma 8, if we launch $\lfloor \log_{1+\varepsilon/10}(10/\varepsilon)\rfloor = O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ runs of REDUCE using $\varepsilon/10$ instead of ε, there is a run that outputs a curve σ such that $d_F(\tau,\sigma) \leq (1+\varepsilon/10)(1+8\varepsilon/10)\delta^* \leq (1+\varepsilon)\delta^*$.

Each run of REDUCE uses $O(k)$ space to maintain the output curve and uses another $O(\varepsilon^{-\alpha})$ working storage by Theorem 3. This gives a total of $O\big((k\varepsilon^{-1} + \varepsilon^{-(\alpha+1)})\log\frac{1}{\varepsilon}\big)$ working storage over all runs. The vertex processing time is the highest when COMPRESS is called to simplify a curve of size $2k-1$. By Theorem 3, this call takes $O(k\varepsilon^{-\alpha}\log\frac{1}{\varepsilon})$ time for $d \in \{2,3\}$ and $O(k\varepsilon^{-\alpha})$ time for $d \geq 4$. Summing over all $O(\frac{1}{\varepsilon}\log\frac{1}{\varepsilon})$ runs gives the total processing time. $\square$

4 Conclusion

We present streaming algorithms for the δ-simplification and k-simplification problems. Both use little working storage and process the next vertex in the stream efficiently. Moreover, they offer provable guarantees on the size of the output curve and the Fréchet distance between the input and output curves. There are some natural research questions. In this work, the vertices of the output curve may not be a subset of those of the input curve. What guarantees can be obtained if this requirement is imposed? Can other curve proximity problems be solved efficiently in the streaming setting?

References

[1] M.A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. In *Proceedings of the Annual Symposium on Computational Geometry*, pages 175–183, 2007.

[2] M.A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. *Discrete and Computational Geometry*, 43:497–515, 2010.

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

[4] S. Bereg, M. Jiang, W. Wang, B. Yang, and B. Zhu. Simplifying 3D polygonal chains under the discrete Fréchet distance. In *Proceedings of the Latin American Symposium on Theoretical Informatics*, pages 630–641, 2008.

[5] K. Bringmann and B.R. Chaudhury. Polyline simplification has cubic complexity. In *Proceedings of the International Symposium on Computational Geometry*, pages 18:1–18:16, 2019.

[6] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, 2006.

[7] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[8] M. Chen, M. Xu, and P. Fränti. A fast $o(n)$ multiresolution polygonal approximation algorithm for gps trajectory simplification. *IEEE Transactions on Image Processing*, 21(5):2770–2785, 2012.

[9] S.-W. Cheng and H. Huang. Curve simplification and clustering under Fréchet distance. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 1414–1432, 2023.

[10] S.-W. Cheng and H. Huang. Solving Fréchet distance problems by algebraic geometric methods. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 4502–4513, 2024.

[11] A. Driemel, A. Krivošija, and C. Sohler. Clustering time series under the Fréchet distance. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 766–785, 2016.

[12] A. Driemel, I. Psarros, and M. Schmidt. Sublinear data structures for short Fréchet queries, 2019. arXiv:1907.04420.

[13] A. Filtser and O. Filtser. Static and streaming data structures for Fréchet distance queries. *ACM Transactions on Algorithms*, 19(4):39:1–39:36, 2023.

[14] Y. Gao, Z. Fang, J. Xu, S. Gong, C. Shen, and L. Chen. An efficient and distributed framework for real-time trajectory stream clustering. *IEEE Transactions on Knowledge and Data Engineering*, 36(5):1857–1873, 2024.

[15] M. Godau. A natural metric for curves - computing the distance for polygonal chains and approximation algorithms. In *Proceedings of the Annual Symposium on Theoretical Aspects of Computer Science*, pages 127–136, 1991.

[16] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, and J.S. Snoeyink. Approximating polygons and subdivisions with minimum-link paths. *International Journal of Computational Geometry & Applications*, 3(4):383–415, 1993.

[17] X. Lin, J. Jiang, S. Ma, Y. Zuo, and C. Hu. One-pass trajectory simplification using the synchronous Euclidean distance. *The VLDB Journal*, 28:897–921, 2018.

[18] X. Lin, S. Ma, J. Jiang, Y. Hou, and T. Wo. Error bounded line simplification algorithms for trajectory compression: an experimental evaluation. *ACM Transactions on Database Systems*, 46(3):11:1–11:44, 2021.

[19] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. In *Proceedings of the VLDB Endowment*, volume 10, pages 841–852, 2017.

[20] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: error-bounded trajectory compression on the go. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 987–998, 2015.

[21] J. Muckell, J.-H. Hwang, V. Patil, C.T. Lawson, F. Ping, and S. Ravi. SQUISH: an online approach for GPS trajectory compression. In *Proceedings of the International Conference on Computing for Geospatial Research & Applications*, pages 13:1–13:8, 2011.

[22] J. Muckell, P.W. Olsen, J.-H. Hwang, C.T. Lawson, and S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.

[23] S. Muthukrishnan. *Data Streams: Algorithms and Applications.* now Publishers Inc., 2005.

[24] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *Proceedings of the IEEE International Conference on Scientific and Statistical Database Management*, pages 275–284, 2006.

[25] M. van de Kerkhof, I. Kostitsyna, M. Löffler, M. Mirzanezhad, and C. Wenk. Global curve simplification. In *Proceedings of the European Symposium on Algorithms*, pages 67:1–67:14, 2019.

[26] T. van der Horst and T. Ophelders. Computing minimum complexity 1D curve simplifications under the Fréchet distance. In *The 39th European Workshop on Computational Geometry*, 2023.

[27] M. van Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. In *Proceedings of the International Symposium on Computational Geometry*, pages 56:1–56:14, 2018.

[28] Z. Wang, C. Long, G. Cong, and Q. Zhang. Error-bounded online trajectory simplification with multi-agent reinforcement learning. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1758–1768, 2021.

[29] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H.T. Shen. Trajectory simplification: an experimental study and quality analysis. In *Proceedings of the VLDB Endowment*, volume 11, pages 934–946, 2018.