Discrete Comput Geom (2010) 43: 497–515
DOI 10.1007/s00454-008-9132-4

Streaming Algorithms for Line Simplification

M.A. Abam · M. de Berg · P. Hachenberger · A. Zarei

Received: 10 July 2008 / Revised: 16 December 2008 / Accepted: 16 December 2008 /
Published online: 13 January 2009

Abstract We study the following variant of the well-known line-simplification problem: we are getting a (possibly infinite) sequence of points $p_0, p_1, p_2, \ldots$ in the plane defining a polygonal path, and as we receive the points, we wish to maintain a simplification of the path seen so far. We study this problem in a streaming setting, where we only have a limited amount of storage, so that we cannot store all the points. We analyze the competitive ratio of our algorithms, allowing resource augmentation: we let our algorithm maintain a simplification with $2k$ (internal) points and compare the error of our simplification to the error of the optimal simplification with k

Communicated by Pankaj K. Agarwal.

M.A.A. was supported by the Netherlands Organization for Scientific Research (NWO) under project no. 612.065.307 and MADALGO, A Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation. M.dB. was supported by the Netherlands Organization for Scientific Research (NWO) under project no. 639.023.301. P.H. was supported by the Netherlands Organization for Scientific Research (NWO) under project no. 639.023.301. A.Z. was supported by Sharif University and a grant from IPM school of CS (no. CS1386-2-01).

M.A. Abam · P. Hachenberger
MADALGO, Department of Computing Science, Aarhus University, Aarhus, Denmark

M.A. Abam
e-mail: abam@madalgo.au.dk

P. Hachenberger
e-mail: phachenb@madalgo.au.dk

M. de Berg (✉)
Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
e-mail: mdberg@win.tue.nl

A. Zarei
Computer Engineering Department, Sharif University of Technology and IPM School of Computer Science, P.O. Box 11365-9517, Tehran, Iran
e-mail: zarei@mehr.sharif.edu

points. We obtain the algorithms with $O(1)$ competitive ratio for three cases: convex paths, where the error is measured using the Hausdorff distance (or Fréchet distance), xy-monotone paths, where the error is measured using the Hausdorff distance (or Fréchet distance), and general paths, where the error is measured using the Fréchet distance. In the first case the algorithm needs $O(k)$ additional storage, and in the latter two cases the algorithm needs $O(k^2)$ additional storage.

Keywords Line simplification · Streaming algorithms

1 Introduction

Motivation Suppose that we are tracking one, or maybe many, moving objects. Each object is equipped with a device that is continuously transmitting its position. Thus we are receiving a stream of data points that describes the path along which the object moves. The goal is to maintain this path for each object. We are interested in the scenario where we are tracking the objects over a very long period of time, as happens for instance when studying the migratory patterns of animals. In this situation it may be undesirable or even impossible to store the complete stream of data points. Instead we have to maintain an approximation of the input path. This leads us to the following problem: we are receiving a (possibly infinite) stream $p_0, p_1, p_2, \ldots$ of points in the plane, and we wish to maintain a simplification (of the part of the path seen so far) that is as close to the original path as possible, while using no more than a given (fixed) amount of available storage.

Related Work The problem described above is a streaming version of line simplification, one of the basic problems in GIS. In a line simplification problem one is given a polygonal path $P := p_0, p_1, \ldots, p_n$ in the plane, and the goal is to find a path $Q := q_0, q_1, \ldots, q_k$ with fewer vertices that well approximates the path P. In fact, this problem arises whenever we want to perform data reduction on a polygonal shape in the plane, and so it plays a role not only in GIS but also in areas like image processing and computer graphics. Line simplification has been studied extensively both in these application areas and in computational geometry. We study line simplification in a streaming setting, where we only have a limited amount of storage so that we cannot store all the points. A similar streaming model for geometric algorithms has been used by, e.g., Agarwal and Yu [2] and Zarrabi-Zadeh and Chan [19]. Also see Muthukrishnan's survey [18] on streaming algorithms.

The line-simplification problem has many variants. For example, we can require the sequence of vertices of Q to be a subsequence of P (with $q_0 = p_0$ and $q_k = p_n$)—this is sometimes called the *restricted version*—or we can allow arbitrary points as vertices. In this paper, as in most other papers, we consider the restricted version, and we limit our discussion to this version from now on; some results on the unrestricted version can be found in [10–12, 15]. In the restricted version, each link $q_l q_{l+1}$ of the simplification corresponds to a shortcut $p_i p_j$ (with $j > i$) of the original path, and the error of the link is defined as the distance between $p_i p_j$ and the subpath $p_i, \ldots, p_j$. To measure the distance between $p_i p_j$ and $p_i, \ldots, p_j$ one often uses the Hausdorff

distance, but the Fréchet distance can be used as well—see below for definitions. The error of the simplification Q is now defined as the maximum error of any of its links. Once the error measure has been defined, we can consider two types of optimization problems: the min-k and the min-δ problem. In the min-k problem, one is given the path P and a maximum error δ, and the goal is to find a simplification Q with as few vertices as possible whose error is at most δ. In the min-δ problem, one is given the path P and a maximum number of vertices k, and the goal is to find a simplification with the smallest possible error that uses at most k vertices.

The oldest and most popular algorithm for line simplification under the Hausdorff distance is the Douglas–Peucker algorithm [8]. A basic implementation of this algorithm runs in $O(n^2)$ time, but more careful implementations run in $O(n \log n)$ time [13] or even $O(n \log^* n)$ time [14]. However, the Douglas–Peucker algorithm is only heuristic and is not guaranteed to be optimal (in terms of the number of vertices used or the error of the resulting simplification). Imai and Iri [16] solved both versions of the problem by modeling it as a shortest-path problem on directed acyclic graphs. The running time of their method was proved to be quadratic or near quadratic by Chin and Chan [7] and Melkman and O'Rouke [17]. Finally, Agarwal and Varadarajan [1] improved the running time to $O(n^{4/3+\varepsilon})$ for any fixed $\varepsilon > 0$, for the L_1-metric and the so-called uniform metric—here $d(p, q) = |p_y - q_y|$ if $p_x = q_x$ and $d(p, q) = \infty$ otherwise—by implicitly representing the graph.

The line-simplification problem was first studied for the Fréchet distance by Godau [9]. Alt and Godau [4] proposed an algorithm to compute the Fréchet distance between two polygonal paths in quadratic time; combined with the approach of Imai and Iri [16], this can be used to compute an optimal solution to the min-δ or the min-k problem for the Fréchet distance.

Since exact solving the line-simplification problem is costly—the best known algorithm for the Hausdorff distance (under the L_2 metric) and for the Fréchet distance take quadratic time or more—Agarwal et al. [3] considered approximation algorithms. In particular, they considered the min-k problem for both the Hausdorff distance for x-monotone paths (in the plane) and the Fréchet distance for general paths (in d-dimensional space). They gave near-linear time algorithms that compute a simplification whose error is at most δ and whose number of vertices is at most the minimum number of vertices of a simplification of error at most $\delta/2$. Their algorithms are greedy and iterative. Because the algorithms are iterative, they can be used in an on-line setting, where the points are given one by one, and the simplification must be updated at each step. However, since they solve the min-k problem, they cannot be used in a streaming setting, because the complexity of the produced simplification for an input path of n points can be $\Theta(n)$. (Note that an iterative greedy approach can be used in the min-k problem—try to go as far as possible with each link, while staying within the error bound δ—but that for the min-δ problem this does not work.) Moreover, their algorithm for the Hausdorff distance does not work when the normal Euclidean distance is used in the definition of Hausdorff distance, but it does when the uniform distance is used. The other existing algorithms for line simplification cannot be used in a streaming setting either.

Definitions, Notation, and Problem Statement We first introduce some terminology to state more clearly the problem we wish to solve. Let $p_0, p_1, \ldots$ be the given stream

of input points. We use $P(n)$ to denote the path defined by the points $p_0, p_1, \ldots, p_n$, that is, the path connecting those points in order, and for any two points p, q on the path, we use $P(p, q)$ to denote the subpath from p to q. For two vertices p_i, p_j, we use $P(i, j)$ as a shorthand for $P(p_i, p_j)$. A segment $p_i p_j$ with $i < j$ is called a *link* or sometimes a *shortcut*. Thus $P(n)$ consists of the links $p_{i-1}p_i$ for $0 < i \leqslant n$. We assume that a function *error* is given that assigns a nonnegative error to each link $p_i p_j$. An *ℓ-simplification* of $P(n)$ is a polygonal path $Q := q_0, q_1, \ldots, q_k, q_{k+1}$, where $k \leqslant \ell$, $q_0 = p_0$, $q_{k+1} = p_n$, and $q_1, \ldots, q_k$ is a subsequence of $p_1, \ldots, p_{n-1}$. The error of a simplification Q for a given function *error*, denoted $error(Q)$, is defined as the maximum error of any of its links. We will consider two specific error functions for our simplifications, one based on the Hausdorff distance, and the other based on the Fréchet distance, as defined next. For two objects o_1 and o_2, we use $d(o_1, o_2)$ to denote the Euclidean distance between o_1 and o_2. (For two points p_i and p_j, we sometimes also use $|p_i p_j|$ to denote the Euclidean distance between p_i and p_j, which is equal to the length of the segment $p_i p_j$.)

- In the Hausdorff error function $error_{\mathrm{H}}$, the error of the link $p_i p_j$ is equal to $d_{\mathrm{H}}(p_i p_j, P(i, j))$, the Hausdorff distance of the subpath $P(i, j)$ to the segment $p_i p_j$. The Hausdorff distance is defined as $d_{\mathrm{H}}(p_i p_j, P(i, j)) := \max_{i \leqslant l \leqslant j} d(p_l, p_i p_j)$.
- The Fréchet distance between two paths A and B, which we denote by $d_{\mathrm{F}}(A, B)$, is defined as follows. Consider a man with a dog on a leash, with the man standing at the start point of A and the dog standing at the start point of B. Imagine that the man walks to the end of A and the dog walks to the end of B. During the walk, they can stop every now and then, but they are not allowed to go back along their paths. Now the Fréchet distance between A and B is the minimum length of the leash needed for this walk, over all possible such walks. More formally, $d_{\mathrm{F}}(A, B)$ is defined as follows. Let A and B be specified by functions $A : [0, 1] \to \mathbb{R}^2$ and $B : [0, 1] \to \mathbb{R}^2$. Any nondecreasing continuous function $\alpha : [0, 1] \to [0, 1]$ with $\alpha(0) = 0$ and $\alpha(1) = 1$ defines a reparameterization A_α of A by setting $A_\alpha(t) = A(\alpha(t))$. Similarly, any nondecreasing continuous function $\beta : [0, 1] \to [0, 1]$ with $\beta(0) = 0$ and $\beta(1) = 1$ defines a reparameterization B_β of B. The Fréchet distance $d_{\mathrm{F}}(A, B)$ between two paths A and B is now defined as

$$d_{\mathrm{F}}(A, B) := \inf_{\alpha,\beta} \max_{0 \leqslant t \leqslant 1} d\big(A_\alpha(t), B_\beta(t)\big),$$

  where the infimum is taken over all reparameterizations A_α of A and B_β of B. In the Fréchet error function $error_{\mathrm{F}}$, the error of the link $p_i p_j$ is the Fréchet distance of the subpath $P(i, j)$ to the segment $p_i p_j$, that is, $error_{\mathrm{F}}(p_i p_j) := d_{\mathrm{F}}(P(i, j), p_i p_j)$.

Now consider an algorithm $\mathcal{A} := \mathcal{A}(\ell)$ that maintains an ℓ-simplification for the input stream $p_0, p_1, \ldots$ for some given ℓ. Let $Q_{\mathcal{A}}(n)$ denote the simplification that $\mathcal{A}$ produces for the path $P(n)$. Let $Opt(\ell)$ denote an optimal off-line algorithm that produces an ℓ-simplification. Thus $error(Q_{Opt(\ell)}(n))$ is the minimum possible error of any ℓ-simplification of $P(n)$. We define the quality of $\mathcal{A}$ using the *competitive ratio*, as is standard for on-line algorithms. We also allow *resource augmentation*.

More precisely, we allow $\mathcal{A}$ to use a $2k$-simplification, but we compare the error of this simplification to $Q_{Opt(k)}(n)$. (This is similar to Agarwal et al. [3], who compare the quality of their solution to the min-k problem for a given maximum error δ to the optimal solution for maximum error $\delta/2$.) Thus we define the competitive ratio of an algorithm $\mathcal{A}(2k)$ as

$$\text{competitive ratio of } \mathcal{A}(2k) := \max_{n \geqslant 0} \frac{error(Q_{\mathcal{A}(2k)}(n))}{error(Q_{Opt(k)}(n))},$$

where $\frac{error(Q_{\mathcal{A}(2k)}(n))}{error(Q_{Opt(k)}(n))}$ is defined as 1 if $error(Q_{\mathcal{A}(2k)}(n)) = error(Q_{Opt(k)}(n)) = 0$. We say that an algorithm is *c-competitive* if its competitive ratio is at most c.

Our Results We present and analyze a simple general streaming algorithm for line simplification. Our analysis shows that the algorithm has good competitive ratio under two conditions: the error function that is used is *monotone*—see Sect. 2 for a definition—and there is an oracle that can approximate the error of any candidate link considered by the algorithm. We then continue to show that the Hausdorff error function is monotone on convex paths and on xy-monotone paths. (It is not monotone on general paths.) The Fréchet error function is monotone on general paths. Finally, we show how to implement the error oracles for these three settings. Putting everything together leads to the following results.

(i) For convex paths and the Hausdorff error function (or the Fréchet error function), we obtain a 3-competitive streaming algorithm using $O(k)$ additional storage that processes an input point in $O(\log k)$ time.
(ii) For xy-monotone paths and the Hausdorff error function (or the Fréchet error function). We can, for any fixed $\varepsilon > 0$, obtain a $(4+\varepsilon)$-competitive streaming algorithm that uses $O(k^2/\sqrt{\varepsilon})$ additional storage and processes each input point in $O(k \log(1/\varepsilon))$ amortized time.
(iii) For general paths and the Fréchet error function, we can, for any fixed $\varepsilon > 0$, obtain a $(4\sqrt{2}+\varepsilon)$-competitive streaming algorithm that uses $O(k^2/\sqrt{\varepsilon})$ additional storage and processes each input point in $O(k \log(1/\varepsilon))$ amortized time.

Finally, we give a negative result in Sect. 5. We show that, for the Hausdorff error function, it is not possible to have a streaming algorithm that maintains a path with less than $2k$ points whose competitive ratio (with respect to $Opt(k)$) is bounded, unless the algorithm uses $\Omega(n/k)$ additional storage.

2 A General Algorithm

In this section we describe a general strategy for maintaining an ℓ-simplification of an input stream $p_0, p_1, \ldots$ of points in the plane, and we show that it has a good competitive ratio under two conditions: the error function is *monotone* (as defined below), and we have an *error oracle* at our disposal that computes or approximates the error of a link. We denote the error computed by the oracle for a link $p_i p_j$ by $error^*(p_i p_j)$.

In later sections we will prove that the Hausdorff error metric is monotone on convex or xy-monotone paths and that the Fréchet error function is monotone on general paths, and we will show how to implement the oracles for these settings.

Our algorithm is quite simple. Suppose that we have already handled the points $p_0, \ldots, p_n$. (We assume that $n > \ell + 1$; until that moment, we can simply use all points and have zero error.) Let $Q := q_0, q_1, \ldots, q_\ell, q_{\ell+1}$ be the current simplification. Our algorithm will maintain a priority queue $\mathcal{Q}$ that stores the points q_i with $1 \leqslant i \leqslant \ell$, where the priority of a point is the error (as computed by the oracle) of the link $q_{i-1}q_{i+1}$. In other words, the priority of q_i is (an approximation of) the error that is incurred when q_i is removed from the simplification. Now the next point p_{n+1} is handled as follows:

1. Set $q_{\ell+2} := p_{n+1}$, thus obtaining an $(\ell+1)$-simplification of $P(n+1)$.
2. Compute $error^*(q_\ell q_{\ell+2})$ and insert $q_{\ell+1}$ into $\mathcal{Q}$ with this error as priority.
3. Extract the point q_s with minimum priority from $\mathcal{Q}$; remove q_s from the simplification.
4. Update the priorities of q_{s-1} and q_{s+1} in $\mathcal{Q}$.

Next we analyze the competitive ratio of our algorithm.

We say that a link $p_i p_j$ *encloses* a link $p_l p_m$ if $i \leqslant l \leqslant m \leqslant j$, and we say that *error* is a *c-monotone error function* for a path $P(n)$ if, for any two links $p_i p_j$ and $p_l p_m$ such that $p_i p_j$ encloses $p_l p_m$, we have

$$error(p_l p_m) \leqslant c \cdot error(p_i p_j).$$

In other words, an error function is c-monotone if the error of a link cannot be worse than c times the error of any link that encloses it.

Furthermore, we denote an error oracle as an *e-approximate error oracle* if

$$error(p_i p_j) \leqslant error^*(p_i p_j) \leqslant e \cdot error(p_i p_j)$$

for any link $p_i p_j$ for which the oracle is called by the algorithm above.

Theorem 2.1 *Suppose that we use a c-monotone error function and that we have an e-approximate error oracle at our disposal. Then the algorithm described above with $\ell = 2k$ is ce-competitive with respect to $Opt(k)$. The time the algorithm needs to update the simplification Q upon the arrival of a new point is $O(\log k)$ plus the time spent by the error oracle. Besides the storage needed for the simplification Q, the algorithm uses $O(k)$ storage plus the storage needed by the error oracle.*

Proof Consider an arbitrary $n \geqslant 0$, and let $Q(n)$ denote the $2k$-simplification produced by our algorithm. Since the error of $Q(n)$ is the maximum error of any of its links, we just need to show that $error(\sigma) \leqslant c \cdot e \cdot error(Q_{Opt(k)}(n))$ for any link σ in $Q(n)$. Let $m \leqslant n$ be such that σ appears in the simplification when we receive point p_m. If $m \leqslant 2k+1$, then $error(\sigma) = 0$, and we are done. Otherwise, let $Q(m-1) := q_0, \ldots, q_{2k+1}$ be the $2k$-simplification of $P(m-1)$. Upon the arrival of $p_m = q_{2k+2}$ we insert $q_{2k+1} = p_{m-1}$ into $\mathcal{Q}$. A simple counting argument shows that at least one of the shortcuts $q_{t-1}q_{t+1}$ for $1 \leqslant t \leqslant 2k+1$, let us call it σ', must be

enclosed by one of the at most $k+1$ links in $Q_{Opt(k)}(n)$. Since σ is the link with the smallest priority among all links in $\mathcal{Q}$ at that time, its approximated error is smaller than that of σ'. Therefore,

$$error\big(Q_{Opt(k)}(n)\big) \geqslant \frac{1}{c}\, error(\sigma') \geqslant \frac{1}{c \cdot e}\, error^*(\sigma') \geqslant \frac{1}{c \cdot e}\, error^*(\sigma) \geqslant \frac{1}{c \cdot e}\, error(\sigma).$$

We conclude that our algorithm is ce-competitive with respect to $Opt(k)$.

Besides the time and storage needed by the error oracle, the algorithm only needs $O(k)$ space to store the priority queue and $O(\log k)$ for each update of the priority queue. □

Remark The proof of Theorem 2.1 is similar to a proof by Buragohain et al. [6]. They show that any simplification with $2k$ links satisfying the so-called min-merge property is competitive against an optimal solution with k links. Although their proof is similar, their concept of min-merge property is not directly applicable in our setting, and so we cannot use their result directly.

3 The Hausdorff Error Function

The algorithm presented above has good competitive ratio if the error function being used is monotone and can be well approximated. In this section we show that these properties hold for the Hausdorff error function on convex and xy-monotone paths. (A path is convex if by connecting the last point to the first point on the path we obtain a convex polygon. A path is xy-monotone if any horizontal or vertical line intersects it in at most one point.) Note that for these two cases, the Hausdorff distance between a link $p_i p_j$ and the subpath $P(i, j)$ is identical to the Fréchet distance between them. Thus the results from this section hold for the Fréchet distance as well. They improve on the result that will be given in the next section for the Fréchet distance on general curves.

The following lemma gives results on the monotonicity of various types of paths under the Hausdorff error function.

Lemma 3.1 *The Hausdorff error function is* 1*-monotone on convex paths and* 2*-monotone on* xy*-monotone paths. Moreover, there is no constant* c *such that the Hausdorff error function is* c*-monotone on* y*-monotone paths.*

Proof It is easy to see that the Hausdorff error function is 1-monotone on convex paths. It is also not difficult, given any constant c, to give an example of an y-monotone path such that the Hausdorff error function is not c-monotone—a zigzag with four vertices such that the first and third are very close together and the second and fourth are very close together will do.

So now consider an xy-monotone path $p_0, \ldots, p_n$. Let $p_i p_j$ and $p_l p_m$ be two links such that $p_i p_j$ encloses $p_l p_m$, and let p_s be a point on the subpath $P(l, m)$

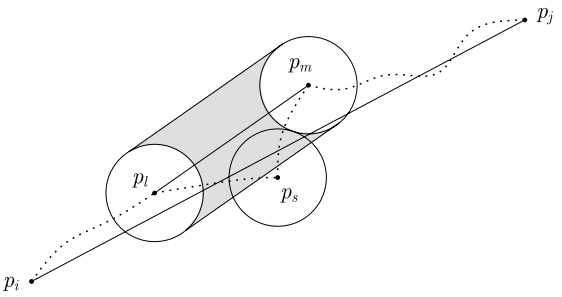

Fig. 1 The Hausdorff error function is 2-monotone on any xy-monotone path

such that $d(p_s, p_l p_m) = error_{\mathrm{H}}(p_l p_m)$. Consider the circles C_l, C_m, and C_s of radius $error_{\mathrm{H}}(p_i p_j)$ centered at points p_l, p_m, and p_s, see Fig. 1. Since the distance of the link $p_i p_j$ to the points p_l, p_s, and p_m is at most $error_{\mathrm{H}}(p_i p_j)$, it must intersect these circles. Let p'_s, p'_l, and p'_m be the orthogonal projections of p_s, p_l, and p_m onto the link $p_i p_j$. Clearly, p'_s, p'_l, and p'_m are inside C_s, C_l, and C_m, respectively. Since $P(i, j)$ is xy-monotone, p'_s lies between p'_l and p'_m, which implies

$$d(p'_s, p_l p_m) \leqslant \max\bigl(d(p'_l, p_l), d(p'_m, p_m)\bigr) \leqslant error_{\mathrm{H}}(p_i p_j).$$

Therefore,

$$\begin{aligned} error_{\mathrm{H}}(p_l p_m) &= error_{\mathrm{H}}(p_s, p_l p_m) \\ &\leqslant d(p_s, p'_s) + d(p'_s, p_l p_m) \\ &\leqslant 2\, error_{\mathrm{H}}(p_i p_j). \end{aligned}$$

Note that the link $p_i p_j$ can be tangent to C_s, C_l, and C_m, which shows that the monotonicity factor 2 is tight. □

The next step is to implement the error oracles for convex paths and for xy-monotone paths. We start with the case of convex paths.

3.1 The Error Oracle for Convex Paths

The idea of the error oracle is to maintain an approximation of the area enclosed by $p_i p_j$ and the path $P(i, j)$ for each link $p_i p_j$. Let $area(i, j)$ denote this area. If the two angles $\angle p_{i+1} p_i p_j$ and $\angle p_{j-1} p_j p_i$ are at most 90 degrees, we can deduce an approximation of $error_{\mathrm{H}}(p_i p_j)$ from $area(i, j)$ and $|p_i p_j|$. Indeed, if $d_{\mathrm{H}}(p_i p_j, P(i, j)) = d$, then the maximum area enclosed by $p_i p_j$ and $P(i, j)$ is achieved by a rectangle with base $p_i p_j$ and height d, and the minimum area is achieved by a triangle with base $p_i p_j$ and height d. Hence,

$$d_{\mathrm{H}}\bigl(p_i p_j, P(i, j)\bigr) \leqslant 2 \cdot area(i, j)/|p_i p_j| \leqslant 2 \cdot d_{\mathrm{H}}\bigl(p_i p_j, P(i, j)\bigr),$$

and so $2 \cdot area(i, j)/|p_i p_j|$ can be used as a 2-approximate error oracle. Unfortunately this approach does not work if $\angle p_{i+1} p_i p_j$ and/or $\angle p_{j-1} p_j p_i$ are bigger than 90 degrees. We therefore proceed as follows.

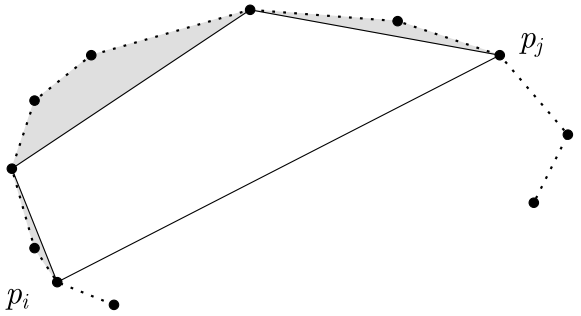

Fig. 2 The areas maintained by the error oracle for convex paths

For each shortcut $p_i p_j$ used in the current approximation, partition the path $P(i, j)$ into at most five pieces by splitting it at each vertex that is extreme in x- or y-direction. (If, say, there is more than one leftmost vertex on the path, we cut at the first such vertex.) The information we maintain for $p_i p_j$ is the set of cut points and the area enclosed by each such piece $P(l, m)$ and the corresponding shortcut $p_l p_m$, see Fig. 2. Notice that if $P(i, j)$ does not contain an extreme point, we simply maintain $area(i, j)$, as before.

Since $d_{\mathrm{H}}(p_i p_j, P(i, j))$ is the maximum of $d_{\mathrm{H}}(p_i p_j, P(l, m))$ over all pieces $P(l, m)$ into which $P(i, j)$ is cut, it is sufficient to approximate $d_{\mathrm{H}}(p_i p_j, P(l, m))$ for each piece. Note that $\angle p_{l+1} p_l p_m$ and $\angle p_{m-1} p_m p_l$ are at most 90 degrees. We approximate $d_{\mathrm{H}}(p_i p_j, P(l, m))$ by

$$\bigl(2 \cdot area(l, m)/|p_l p_m|\bigr) + d_{\mathrm{H}}\bigl(p_i p_j, p_l p_m\bigr).$$

We claim this gives us a 3-approximation. We have

$$\begin{aligned} d_{\mathrm{H}}\bigl(p_i p_j, P(l, m)\bigr) &\leqslant d_{\mathrm{H}}\bigl(p_l p_m, P(l, m)\bigr) + d_{\mathrm{H}}(p_i p_j, p_l p_m) \\ &\leqslant \frac{2 \cdot area(l, m)}{|p_l p_m|} + d_{\mathrm{H}}(p_i p_j, p_l p_m). \end{aligned}$$

On the other hand,

$$\begin{aligned} 3 \cdot d_{\mathrm{H}}\bigl(p_i p_j, P(l, m)\bigr) &\geqslant 3 \cdot \max\bigl(d_{\mathrm{H}}\bigl(p_l p_m, P(l, m)\bigr),\ d_{\mathrm{H}}(p_i p_j, p_l p_m)\bigr) \\ &\geqslant \frac{2 \cdot area(l, m)}{|p_l p_m|} + d_{\mathrm{H}}(p_i p_j, p_l p_m), \end{aligned}$$

so $(2 \cdot area(l, m)/|p_l p_m|) + d_{\mathrm{H}}(p_i p_j, p_l p_m)$ is indeed a 3-approximation of $d_{\mathrm{H}}(p_i p_j, P(l, m))$.

It remains to show that we can maintain this information as more points are received and the simplification changes. First consider Step 2 of the algorithm, where we need to compute $error^*(q_\ell q_{\ell+2})$. Since we have the information described above available for $q_\ell q_{\ell+1}$, and $q_{\ell+1}$ and $q_{\ell+2}$ are consecutive points of the original path P, we can compute the necessary information for $q_\ell q_{\ell+2}$ in $O(1)$ time. Updating the priorities of p_{s-1} and p_{s+1} after removing p_s in Step 4 can be performed in $O(1)$ time as follows. The priority assigned to p_{s-1}

after removing p_s is the approximation of $d_{\mathrm{H}}(q_{s-2}q_{s+1}, P(q_{s-2}, q_{s+1}))$, which is the maximum of $d_{\mathrm{H}}(q_{s-2}q_{s+1}, P(q_{s-2}, q_{s-1}))$, $d_{\mathrm{H}}(q_{s-2}q_{s+1}, P(q_{s-1}, q_s))$, and $d_{\mathrm{H}}(q_{s-2}q_{s+1}, P(q_s, q_{s+1}))$. Since the areas and the cutting points of subpaths $P(q_{s-2}, q_{s-1})$, $P(q_{s-1}, q_s)$, and $P(q_s, q_{s+1})$ are available, we can simply compute the priority of p_s as described above. The priority of p_{s+1} can be computed in a similar way.

Lemma 3.2 *There is a* 3*-approximate error oracle for the Hausdorff error function on convex paths that uses* $O(k)$ *storage and can be updated in* $O(1)$ *time.*

Theorem 3.3 *There is a streaming algorithm that maintains a* $2k$*-simplification for convex planar paths under the Hausdorff error function* (*or the Fréchet error function*) *and that is* 3*-competitive with respect to* $Opt(k)$*. The algorithm uses* $O(k)$ *additional storage, and each point is processed in* $O(\log k)$ *time.*

3.2 The Error Oracle for xy-Monotone Paths

We use the notion of width for approximating $error_{\mathrm{H}}$ of an xy-monotone path. The *width* of a set of points with respect to a given direction $\overrightarrow{d}$ is the minimum distance of two lines being parallel to $\overrightarrow{d}$ that enclose the point set.[1] Let $w(i, j)$ be the width of the points in subpath $P(i, j)$ with respect to the direction $\overrightarrow{p_i p_j}$. Since $P(i, j)$ is xy-monotone, it is contained inside the axis-parallel rectangle defined by p_i and p_j. Therefore, $w(i, j)/2 \leqslant error_{\mathrm{H}}(p_i p_j) \leqslant w(i, j)$, and $w(i, j)$ can be used as a 2-approximate error oracle for $error_{\mathrm{H}}(p_i p_j)$.

Agarwal and Yu [2] have described a streaming algorithm for maintaining a coreset that can be used to approximate the width of a set in any direction. More precisely, they maintain an ε-coreset of size $O(1/\sqrt{\varepsilon})$ in $O(\log(1/\varepsilon))$ amortized time per insertion. The width in a given direction can be efficiently computed from the coreset if we additionally maintain the convex hull of the coreset using the dynamic data structure by Brodal and Jacob [5]. This data structure uses linear space and can be updated in logarithmic time. Also it supports queries for the extreme point in a given direction in logarithmic time. Thus we can compute the extreme points that define the width in a given direction in $O(\log(1/\varepsilon))$ time. The coreset gives us a $(2+\varepsilon)$-approximate error oracle.

Lemma 3.4 *There is a* $(2+\varepsilon)$*-approximate error oracle for the Hausdorff error function on* xy*-monotone paths that uses* $O(k^2 + k/\sqrt{\varepsilon})$ *storage and has* $O(k \log(1/\varepsilon))$ *amortized update time.*

Proof Let $q_0, \ldots, q_{\ell+1}$ denote the current approximation with $\ell = 2k$. Although our algorithm only needs the approximate errors of the links $q_{i-1}q_{i+1}$ to decide which point q_s is erased next, we must maintain some information for each link that might

[1] Sometimes the width in a direction is defined as the smallest distance between two enclosing lines that are *orthogonal* to that direction, but for us it is more convenient to define it with respect to lines parallel to the given direction.

be needed at some later time in our simplification. More precisely, we maintain the following information:

- For each (potential) link $q_i q_j$ with $0 \leqslant i < j \leqslant \ell + 1$, we store a $(1+\varepsilon)$-approximation of the width of the subpath $P(q_i, q_j)$ in the direction of $q_i q_j$. This takes $O(k^2)$ storage in total.
- For each (potential) link $q_i q_{\ell+1}$ with $0 \leqslant i \leqslant \ell$, we maintain a coreset of all the points on the subpath $P(q_i, q_{\ell+1})$ and a dynamic convex-hull structure for this coreset. Since each coreset uses $O(1/\sqrt{\varepsilon})$ storage, the amount of storage for the coresets and convex hulls is $O(k/\sqrt{\varepsilon})$.

Now consider the arrival of a new point p_{n+1}. We need to add this point to the simplification and remove one of the existing points in the simplification. Since for each of the current links we know (an approximation) of its width, we also know its approximate error. Hence, we can decide which point to remove.

Next we must update the information we maintain. Note that p_{n+1} is our new last point of the simplification and that the only new (potential) links are links $q_i p_{n+1}$. We must obtain a coreset for each such link $q_i p_{n+1}$. Note that we no longer need the coreset for $q_i p_n$. Hence, we can simply take this coreset, insert p_{n+1} into it, and then update the dynamic convex-structure storing the coreset. After having obtained the $O(k)$ new coresets in this manner in $O(k \log(1/\varepsilon))$ amortized time, we can find the approximate widths of these links in the same amount of time. □

Theorem 3.5 *There is a streaming algorithm that maintains a $2k$-simplification for xy-monotone planar paths under the Hausdorff error function (or the Fréchet error function) and that is $(4+\varepsilon)$-competitive with respect to $Opt(k)$. The algorithm uses $O(k^2 + k/\sqrt{\varepsilon})$ additional storage, and each point is processed in $O(k \log(1/\varepsilon))$ amortized time.*

4 The Fréchet Error Function

We now turn our attention to the Fréchet error function. We will show that we can obtain an $O(1)$-competitive algorithm for general paths. The first property we need is that the Fréchet error function is monotone. This has in fact already been proven by Agarwal et al. [3].

Lemma 4.1 [3] *The Fréchet error function is 2-monotone on general paths.*

4.1 The Error Oracle

Next we consider the implementation of the error oracle for the Fréchet error function. We use two parameters to approximate $error_{\mathrm{F}}(p_i p_j)$. The first one is $w(i, j)$, the width of the points of $P(i, j)$ in the direction of $p_i p_j$, which we also used to approximate the Hausdorff error in the case of xy-monotone paths. The other parameter is the length of the largest back-path in the direction of $p_i p_j$, which is defined as follows. Assume without loss of generality that $p_i p_j$ is horizontal with p_j to the

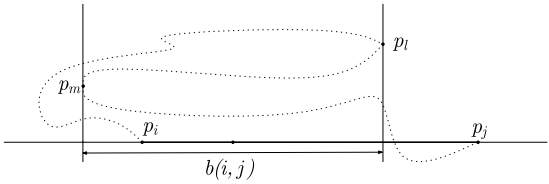

Fig. 3 Relation between Fréchet distance and back-paths

right of p_i. For two points p_l, p_m on the path $P(i,j)$ with $l < m$, we define $P(l,m)$ to be a *back-path on* $P(i,j)$ if $(p_m)_x < (p_l)_x$. In other words, $P(l,m)$ is a back-path if, relative to the direction $\overrightarrow{p_i p_j}$, we go back when we move from p_l to p_m, see Fig. 3. The *length* of a back-path $P(l,m)$ on $P(i,j)$ is defined to be the length of the projection of $p_l p_m$ onto a line parallel to $p_i p_j$, which is equal to $(p_l)_x - (p_m)_x$ since we assumed that $p_i p_j$ is horizontal. We define $b(i,j)$ to be the maximum length of any back-path on $P(i,j)$.

Lemma 4.2 $\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2}) \leqslant error_{\mathrm{F}}(p_i p_j) \leqslant 2\sqrt{2}\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2})$.

Proof As above, without loss of generality we assume that $p_i p_j$ is horizontal with p_j to the right of p_i.

We observe that $error_{\mathrm{F}}(p_i p_j) \geqslant error_{\mathrm{H}}(p_i p_j) \geqslant \frac{w(i,j)}{2}$. Next we will show that $\frac{b(i,j)}{2} \leqslant error_{\mathrm{F}}(p_i p_j)$. Consider a back-path $P(l,m)$ on $P(i,j)$ determining $b(i,j)$, as shown in Fig. 3. Let r be the point on the line through $p_i p_j$ midway between p_l and p_m, that is, the point on the line through $p_i p_j$ with x-coordinate $((p_l)_x + (p_m)_x)/2$. Note that r does not necessarily lie on $p_i p_j$. The Fréchet distance between $p_i p_j$ and $P(i,j)$ is determined by some optimal pair of parameterizations of $p_i p_j$ and $P(i,j)$ that identifies each point p of $P(i,j)$ with a point $\overline{p}$ on $p_i p_j$ in such a way that if p comes before q along $P(i,j)$, then $\overline{p}$ does not come later than $\overline{q}$ along $p_i p_j$. Now consider the images $\overline{p_l}$ and $\overline{p_m}$. If $\overline{p_l}$ lies to the left of r, then $|p_l \overline{p_l}| \geqslant \frac{b(i,j)}{2}$. If, on the other hand, $\overline{p_l}$ lies on or to the right of r, then $\overline{p_m}$ lies on or to the right of r as well, and we have $|p_m \overline{p_m}| \geqslant \frac{b(i,j)}{2}$. We conclude that $\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2}) \leqslant error_{\mathrm{F}}(p_i p_j)$, which proves the first part of the lemma.

For the second part of the lemma, we must show that $error_{\mathrm{F}}(p_i p_j) \leqslant \sqrt{2}\max(w(i,j), b(i,j))$. It is convenient to think about the Fréchet distance in terms of the man–dog metaphor. In these terms, we have to find a walking schedule where the man walks along $p_i p_j$ and the dog walks along $P(i,j)$ so that they never go back along their paths and their distance is never greater than $\sqrt{2}\max(w(i,j), b(i,j))$. We can find such a walk as follows. Denote the position of the man by p_{man} and the position of the dog by p_{dog}. Initially, $p_{\mathrm{man}} = p_{\mathrm{dog}} = p_i$. Let ℓ be the vertical line through p_i. Among all the intersection points of ℓ with $P(i,j)$, let p be one farthest along $P(i,j)$. (If ℓ does not intersect $P(i,j)$ except at p_i, then $p = p_i$.) We let the dog walk along $P(p_i, p)$, while the man waits at p_i. Let q be an arbitrary point on $P(p_i, p)$. Then there must be points p_l, p_m with $l < m$ such that $(p_l)_x \geqslant (p_i)_x$ and $(p_m)_x \leqslant (q)_x$. Hence, we have $|(q)_x - (p_i)_x| \leqslant (p_l)_x - (p_m)_x \leqslant b(i,j)$. Furthermore, $|(q)_y - (p_i)_y| \leqslant w(i,j)$. Hence, so far we have $|p_{\mathrm{man}} p_{\mathrm{dog}}| \leqslant \sqrt{2}\max(w(i,j), b(i,j))$.

We continue the walk as follows. Sweep ℓ to the right. Initially ℓ will intersect $P(p, p_j)$ in only one point. As long as this is the case, we set $p_{\text{man}} = \ell \cap p_i p_j$ and $p_{\text{dog}} = \ell \cap P(p, p_j)$. During this part, we clearly have $|p_{\text{man}} p_{\text{dog}}| \leqslant w(i, j)$. At some point, ℓ may intersect $P(p_{\text{dog}}, p_j)$ in one (or more) point(s) other than p_{dog}. When this happens, we take the intersection point p that is farthest along $P(p_{\text{dog}}, p_j)$, and let the dog proceed to p while the man waits at his current position. By the previous argument, $|p_{\text{man}} p_{\text{dog}}| \leqslant \sqrt{2}\max(w(i, j), b(i, j))$ during this phase. Then we continue to sweep ℓ to the right again, letting $p_{\text{man}} = \ell \cap p_i p_j$ and $p_{\text{dog}} = \ell \cap P(p, p_j)$. The process ends when the sweep line reaches p_j. We have thus found a walking schedule with $|p_{\text{man}} p_{\text{dog}}| \leqslant \sqrt{2}\max(w(i, j), b(i, j))$ at all times, finishing the proof of the lemma. □

According to the above lemma, in order to approximate $error_{\text{F}}(p_i p_j)$, it suffices to approximate $\max(w(i, j), b(i, j))$. In the previous section we already described how to approximate $w(i, j)$, when we were studying the Hausdorff error function for xy-monotone paths. Next we describe a method for approximating $b(i, j)$, and show how to combine these two methods to build the oracle for $error_{\text{F}}(p_i p_j)$. (Note that if there are no back-paths, then the Fréchet error is equal to the Hausdorff error, so the case of xy-monotone paths for Hausdorff error is a special case of our current setting.)

In the algorithm presented in Sect. 2 we need to maintain (an approximation of) the error of each shortcut $q_l q_{l+2}$ in the current simplification. For this, we need to know the maximum length of a back-path on the path from q_l to q_{l+2}. The operations we must do are to add a point $q_{\ell+2} = p_{n+1}$ at the end of the simplification and to remove a point q_s from the simplification. To this end we maintain the following information. For the moment let us assume that all we need is the maximum length of the back-path with respect to the positive x-direction. Then we maintain for each link $p_i p_j$ of the simplification the following values:

(i) $b(i, j)$, the maximum length of a back-path (w.r.t. the positive x-direction) on $P(i, j)$;
(ii) $x\max(i, j)$, the maximum x-coordinate of any point on $P(i, j)$;
(iii) $x\min(i, j)$, the minimum x-coordinate of any point on $P(i, j)$.

Now consider a shortcut $q_l q_{l+2}$. Let $q_l = p_i$, $q_{l+1} = p_t$, and $q_{l+2} = p_j$. Then $b(i, j)$, the maximum length of a back-path on $P(q_l, q_{l+2}) = P(i, j)$, is given by

$$\max\bigl(b(i, t), b(t, j), x\max(i, t) - x\min(t, j)\bigr).$$

Adding a point $q_{\ell+2}$ is easy, because we only have to compute the above three values for $q_{\ell+1}q_{\ell+2}$, which is trivial since $q_{\ell+1}$ and $q_{\ell+2}$ are consecutive points on the original path. Removing a point q_s can also be done in $O(1)$ time (let $q_{s-1} = p_i$ and $q_{s+1} = p_j$): above we have shown how to compute $b(i, j)$ from the available information for $q_{s-1}q_s$ and $q_s q_{s+1}$, and computing $x\max(i, j)$ and $x\min(i, j)$ is even easier.

Thus we can maintain the maximum length of a back-path. There is one catch, however: the procedure given above maintains the maximum length of a back-path *with respect to a fixed direction* (the positive x-direction). But in fact we need to

know for each $q_i q_{i+2}$ the maximum length of a back-path with respect to the direction $\overrightarrow{q_i q_{i+2}}$. These directions are different for each of the links, and, moreover, we do not know them in advance. To overcome this problem we define $2\pi/\alpha$ equally spaced canonical directions for a suitable $\alpha > 0$, and we maintain, for every link $p_i p_j$, the information described above for each direction. Now suppose that we need to know the maximum length of a back-path for $p_i p_j$ with respect to the direction $\overrightarrow{p_i p_j}$. Then we will use $b_{\overrightarrow{d}}(p_i p_j)$, the maximum length of a back-path with respect to $\overrightarrow{d}$ instead, where $\overrightarrow{d}$ is the canonical direction closest to $\overrightarrow{p_i p_j}$ in clockwise order. In general, using $\overrightarrow{d}$ may not give a good approximation of the maximum length of a back-path in direction $\overrightarrow{p_i p_j}$, even when α is small. However, the approximation is only bad when $w(i, j)$ is relatively large, which means that the Fréchet distance can still be well approximated. This is made precise in the following lemmas.

Lemma 4.3 *Let w be the width of $P(i, j)$ in direction $\overrightarrow{p_i p_j}$, let b be the maximum length of a back-path on $P(i, j)$ in direction $\overrightarrow{p_i p_j}$, and let b^* be the maximum length of a back-path on $P(i, j)$ in direction $\overrightarrow{d}$, where $\overrightarrow{d}$ is the canonical direction closest to $\overrightarrow{p_i p_j}$ in clockwise order. Then we have $b^* - \tan(\alpha) \cdot w \leqslant b \leqslant b^* + \tan(\alpha) \cdot (b^* + w)$.*

Proof We first show that $b \leqslant b^* + \tan(\alpha) \cdot (b^* + w)$. Let the sub-path $P(l, m)$ have the maximum back-path length in the direction $\overrightarrow{p_i p_j}$. Consider two half-lines originating from p_m and being parallel to $\overrightarrow{p_i p_j}$ and $\overrightarrow{d}$. Let β denote the angle between these two half-lines. Because $\overrightarrow{d}$ is the canonical direction closest to $\overrightarrow{p_i p_j}$ in clockwise order, clearly $\beta \leqslant \alpha$. Let p and q be the orthogonal projections of p_l onto the lines through p_m in directions $\overrightarrow{p_i p_j}$ and $\overrightarrow{d}$, respectively. We distinguish four cases, depending on the relation of direction $\overrightarrow{p_m p_l}$ to $\overrightarrow{p_i p_j}$ and $\overrightarrow{d}$. The direction $\overrightarrow{p_m p_l}$ can be counterclockwise to $\overrightarrow{p_i p_j}$, between $\overrightarrow{p_i p_j}$ and $\overrightarrow{d}$, or clockwise to $\overrightarrow{d}$. If $\overrightarrow{p_m p_l}$ is counterclockwise to $\overrightarrow{p_i p_j}$, we also distinguish whether the angle between $\overrightarrow{p_m p_l}$ and $\overrightarrow{p_i p_j}$ is less or more than $90 - \beta$ degrees. Note, that since $p_l p_m$ is a back-path, the angle between $\overrightarrow{p_m p_l}$ and $\overrightarrow{p_i p_j}$ cannot be larger than 90 degrees. All four cases are illustrated in Fig. 4. The corresponding proofs are as follows:

(a) $\overrightarrow{p_m p_l}$ is between 90 and $90 - \beta$ degrees counterclockwise to $\overrightarrow{p_i p_j}$:

$$\begin{aligned} b = |p_m q| &\leqslant |p_l q| \tan(\beta) \leqslant w \tan(\alpha) \\ &\leqslant b^* + \tan(\alpha) \cdot (b^* + w). \end{aligned}$$

(b) $\overrightarrow{p_m p_l}$ is less than $90 - \beta$ degrees counterclockwise to $\overrightarrow{p_i p_j}$:

$$\begin{aligned} b = |p_m q| &\leqslant |p_m p| + |pr| + |rq| \\ &\leqslant |p_m p| + |p_m p| \tan(\beta) + |p_l q| \tan(\beta) \\ &\leqslant b^* + \tan(\alpha) \cdot (b^* + w). \end{aligned}$$

(c) $\overrightarrow{p_m p_l}$ is between $\overrightarrow{p_i p_j}$ and $\overrightarrow{d}$:

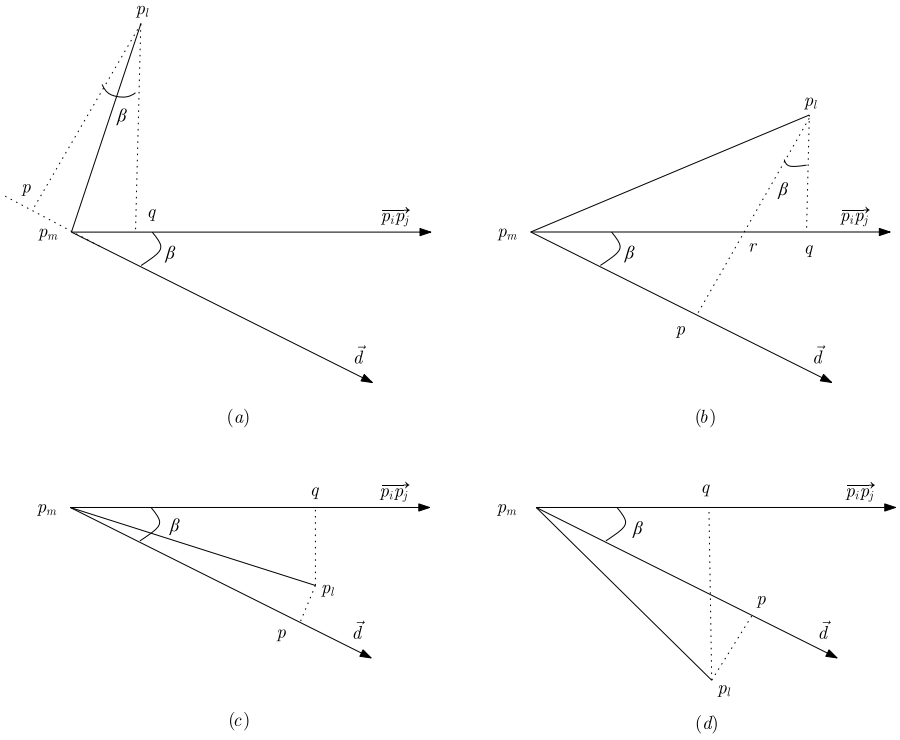

Fig. 4 Illustration for the proof of Lemma 4.3

$$\begin{aligned} b = |p_m q| &\leqslant |p_m p_l| \leqslant |p_m p| + |p p_l| \\ &\leqslant |p_m p| + |p_m p| \tan(\beta) \\ &\leqslant b^* + \tan(\alpha) \cdot (b^* + w). \end{aligned}$$

(d) $\overrightarrow{p_m p_l}$ is clockwise to $\vec{d}$ by at most $90 - \beta$ degrees:

$$b = |p_m q| \leqslant |p_m p| \leqslant b^* \leqslant b^* + \tan(\alpha) \cdot (b^* + w).$$

The same elementary arguments can be used to show that $b^* - \tan(\alpha) \cdot w \leqslant b$. □

The final oracle is now defined as follows. Let w^* be the approximation of the width of $P(i, j)$ in direction $\overrightarrow{p_i p_j}$ as given by Agarwal and Yu's ε-coreset method, and let b^* be the maximum length of a back-path on $P(i, j)$ in direction $\vec{d}$, where $\vec{d}$ is the canonical direction closest to $\overrightarrow{p_i p_j}$ in clockwise order. Then we set

$$error_{\mathrm{F}}{}^*(p_i p_j) := \sqrt{2} \cdot \max\bigl(w^*, b^* + \tan(\alpha) \cdot (b^* + w^*)\bigr).$$

Combining Lemma 4.2 with the observations above, we can prove the following lemma.

Lemma 4.4

$$error_{\mathrm{F}}(p_i p_j) \leqslant error_{\mathrm{F}}{}^*(p_i p_j) \leqslant 2\sqrt{2}(1+\varepsilon)\big(1+4\tan(\alpha)\big) \cdot error_{\mathrm{F}}(p_i p_j).$$

Proof Let w be the width of $P(i,j)$ in direction $\overrightarrow{p_i p_j}$, and let b be the maximum length of a back-path on $P(i,j)$ in direction $\overrightarrow{p_i p_j}$. Because w^* is the width of an ε-coreset, we have $w \leqslant w^* \leqslant (1+\varepsilon)w$. Using Lemma 4.2, we get

$$\begin{aligned} error_{\mathrm{F}}(p_i p_j) &\leqslant 2\sqrt{2} \cdot \max\left(\frac{w}{2}, \frac{b}{2}\right) \\ &\leqslant \sqrt{2} \cdot \max\big(w^*, b^* + \tan(\alpha) \cdot (b^* + w)\big) \\ &\leqslant \sqrt{2} \cdot \max\big(w^*, b^* + \tan(\alpha) \cdot (b^* + w^*)\big) \\ &= error_{\mathrm{F}}{}^*(p_i p_j). \end{aligned}$$

On the other hand,

$$\begin{aligned} error_{\mathrm{F}}{}^*(p_i p_j) &= \sqrt{2} \cdot \max\big(w^*, b^* + \tan(\alpha) \cdot (b^* + w^*)\big) \\ &\leqslant \sqrt{2} \cdot \max\big((1+\varepsilon)w, b + \tan(\alpha)w \\ &\quad + \tan(\alpha) \cdot \big(b + \tan(\alpha)w + (1+\varepsilon)w\big)\big) \\ &\leqslant \sqrt{2}(1+\varepsilon) \cdot \max\big(w, b + b\tan(\alpha) + 3w\tan(\alpha)\big) \\ &\leqslant \sqrt{2}(1+\varepsilon)\big(1+4\tan(\alpha)\big) \cdot \max(w, b) \\ &\leqslant 2\sqrt{2}(1+\varepsilon)\big(1+4\tan(\alpha)\big) \cdot \max\left(\frac{w}{2}, \frac{b}{2}\right) \\ &\leqslant 2\sqrt{2}(1+\varepsilon)\big(1+4\tan(\alpha)\big) \cdot error_{\mathrm{F}}(p_i p_j). \end{aligned}$$

Taking ε and α sufficiently small, we get our final result. □

Theorem 4.5 *There is a streaming algorithm that maintains a $2k$-simplification for general planar paths under the Fréchet error function and that is $(4\sqrt{2}+\varepsilon)$-competitive with respect to $Opt(k)$. The algorithm uses $O(k^2 + k/\sqrt{\varepsilon})$ additional storage, and each point is processed in $O(k \log(1/\varepsilon))$ amortized time.*

5 The Hausdorff Error Function for General Paths

In this section we show that for the Hausdorff error function, it is not possible to have a streaming algorithm that maintains a path with less than $2k$ points whose competitive ratio (with respect to $Opt(k)$) is bounded, unless the algorithm uses $\Omega(n/k)$ additional storage. In fact, this even holds when the input path is known to be y-monotone.

Theorem 5.1 *Let $\mathcal{A}$ be a streaming algorithm that maintains a $(2k-1)$-simplification for a general planar path $P(n)$ and that is able to store at most $m-1$ of*

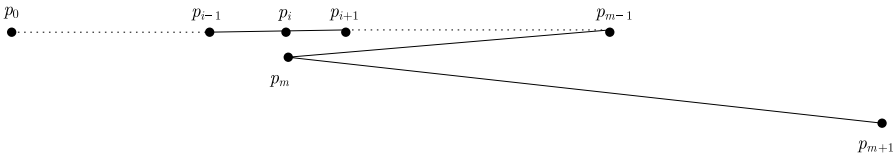

Fig. 5 Base path component for Theorem 5.1

the input points, where $2k+2 \leqslant m \leqslant n/k$. *For any* $c > 0$ *and* $n \geqslant km+1$, *there is a y-monotone path* $p_0, p_1, \ldots, p_n$ *such that* $error_{\mathrm{H}}(Q_{\mathcal{A}(2k-1)}(n)) > c \cdot error_{\mathrm{H}}(Q_{Opt(k)}(n))$.

Proof Figure 5 shows the basic component of the path which has the following properties:

(i) Points $p_0, \ldots, p_{m-1}$ are collinear.
(ii) $|p_i p_{i+1}| > c \cdot d_{\mathrm{H}}(p_{m-1}, p_i p_{m+1})$ for all $0 \leqslant i \leqslant m-2$.
(iii) $d_{\mathrm{H}}(p_{m-1}, p_{i-1} p_{m+1}) > c \cdot d_{\mathrm{H}}(p_{m-1}, p_i p_{m+1})$ for all $1 \leqslant i \leqslant m-1$.

To obtain a configuration with the above properties, we take a horizontal line ℓ and a point p_{m-1} on ℓ. We put p_{m+1} below ℓ and arbitrarily far from p_{m-1} to the right of p_{m-1} so that its distance to ℓ is greater than $(c+\epsilon)^{m-1}$, where $\epsilon > 0$ is an arbitrarily small number. We put p_i $(i = 0, \ldots, m-2)$ on ℓ to the left of p_{m-1} so that $p_i p_{m+1}$ is tangent to the circle whose radius is $(c+\epsilon)^{m-i-1}$ and whose center is p_{m-1}. The point p_i always exists, because $d_{\mathrm{H}}(p_{m+1}, \ell) > (c+\epsilon)^i$.

Let $\mathcal{A}$ be a simplification algorithm being able to store at most $m-1$ points. Upon the arrival of p_{m-1}, the algorithm $\mathcal{A}$ is required to delete one of the past points, because it cannot store m points. Let p_i with $1 \leqslant i \leqslant m-2$ be the deleted point, and let the next point, p_m, be slightly below p_i (i.e., $|p_i p_m| \cong 0$). Up to here, by choosing p_m in $Q_{\mathcal{A}(1)}(m)$, we have $error_{\mathrm{H}}(Q_{\mathcal{A}(1)}(m)) = error_{\mathrm{H}}(Q_{Opt(1)}(m)) = 0$. Now consider the next point p_{m+1}, which lies on its position according to our construction. Obviously, $Q_{Opt(1)}(m+1)$ is p_0, p_i, p_{m+1}, and its Hausdorff error is $d_{\mathrm{H}}(p_{m-1}, p_i p_{m+1})$. Since $\mathcal{A}$ has missed the point p_i, $Q_{\mathcal{A}(1)}(m+1)$ is p_0, p_j, p_{m+1} for some $j \neq i$. There are three possibilities for j:

1. $0 \leqslant j < i$: using property (iii), we have

$$\begin{aligned} error_{\mathrm{H}}\big(Q_{\mathcal{A}(1)}(m+1)\big) &= d_{\mathrm{H}}(p_{m-1}, p_j p_{m+1}) \\ &> c \cdot d_{\mathrm{H}}(p_{m-1}, p_i p_{m+1}) \\ &= c \cdot error_{\mathrm{H}}\big(Q_{Opt(1)}(m+1)\big). \end{aligned}$$

2. $i < j \leqslant m-1$: using property (ii), we have

$$\begin{aligned} error_{\mathrm{H}}\big(Q_{\mathcal{A}(1)}(m+1)\big) &\geqslant d_{\mathrm{H}}(p_m, p_j p_{m+1}) \cong |p_i p_j| \\ &> c \cdot d_{\mathrm{H}}(p_{m-1}, p_i p_{m+1}) \\ &= c \cdot error_{\mathrm{H}}\big(Q_{Opt(1)}(m+1)\big). \end{aligned}$$

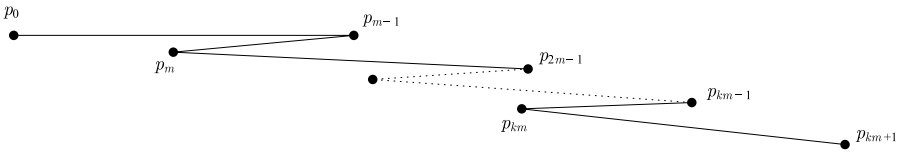

Fig. 6 A path that cannot be simplified within a bounded competitive ratio

3. $j = m$: using property (ii), we have

$$\begin{aligned} error_{\mathrm{H}}\big(Q_{\mathcal{A}(1)}(m+1)\big) &= d_{\mathrm{H}}(p_{m-1}, p_0 p_m) \cong |p_i p_{m-1}| \\ &> c \cdot d_{\mathrm{H}}(p_{m-1}, p_i p_{m+1}) \\ &= c \cdot error_{\mathrm{H}}\big(Q_{Opt(1)}(m+1)\big). \end{aligned}$$

Therefore, in order to be within a bounded competitive ratio, $\mathcal{A}$ must store at least the two points p_{m-1} and p_m, which leads to a 2-simplification.

We concatenate k of these components in such a way that for any two consecutive components, the first two points of the latter lie on the last two points of the former, as illustrated in Fig. 6. Other than the first and the last points, it is straightforward to show that $\mathcal{A}$ has to store two points of each component to be within a bounded competitive ratio. This implies $error_{\mathrm{H}}(Q_{\mathcal{A}(2k-1)}) > c \cdot error_{\mathrm{H}}(Q_{Opt(k)}(n))$. □

6 Concluding remarks

We presented the first line-simplification algorithms in the streaming model. We obtained algorithms with $O(1)$ competitive ratio for convex planar paths and xy-monotone planar paths under the Hausdorff error function (or the Fréchet error function) and for general planar paths under the Fréchet distance. Our results imply linear-time approximation when k is a constant (where the approximation factor is with respect to the optimal solution using half the number of links).

Our algorithms all use resource augmentation: they maintain a $2k$-simplification, but we compare the error of our simplification to the error of an optimal k-simplification. One obvious question is whether we can do with less, or maybe no, resource augmentation. We have shown that this is not the case for general planar paths under the Hausdorff error function, but note that we have not been able to give any $O(1)$-competitive algorithm for this case, not even with resource augmentation. Thus there is a significant gap between our positive and negative results.

Another aspect where improvement may be possible is the implementation of the error oracles, which need $O(k^2)$ storage for xy-monotone paths under the Hausdorff error function and for general paths under the Fréchet distance. For instance, if we can maintain coresets in a streaming setting such that one can also merge two coresets, then this will reduce the dependency on k in the storage from quadratic to linear. (Note that we need to be able to do an unbounded number of merges.)

Our general approach extends to higher dimensions (but the approximation factors and running times will change).

References

1. Agarwal, P.K., Varadarajan, K.R.: Efficient algorithms for approximating polygonal chains. Discrete Comput. Geom. **23**(2), 273–291 (2000)
2. Agarwal, P.K., Yu, H.: A space-optimal data-stream algorithm for coresets in the plane. In: Proceedings of ACM Symposium on Computational Geometry (SOCG), pp. 1–10 (2007)
3. Agarwal, P.K., Har-Peled, S., Mustafa, N.H., Wang, Y.: Near-linear time approximation algorithms for curve simplification. Algorithmica **42**, 203–219 (2005)
4. Alt, H., Godau, M.: Computing the Fréchet distance between two polygonal curves. Int. J. Comput. Geom. Appl. **5**, 75–91 (1995)
5. Brodal, G.S., Jacob, R.: Dynamic planar convex hull. In: Proceedings of Annual Symposium on Foundations of Computer Science (FOCS), pp. 617–626 (2002)
6. Buragohain, C., Gandhi, S., Hershberger, J., Sur, S.: Contour approximation in sensor networks. In: Proceedings of Distributed Computing in Sensor Systems (DCOSS), pp. 356–371 (2006)
7. Chan, W.S., Chin, F.: Approximation of polygonal curves with minimum number of line segments. In: Proceedings of Annual International Symposium on Algorithms and Computing (ISAAC). Lecture Notes in Computer Science, vol. 650, pp. 378–387. Springer, Berlin (1992)
8. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Can. Cartograph. **10**, 112–122 (1973)
9. Godau, M.: A natural metric for curves: Computing the distance for polygonal chains and approximation algorithms. In: Proceedings of Annual Symposium on Theoretical Aspects of Computer Science (STACS), pp. 127–136 (1991)
10. Goodrich, M.T.: Efficient piecewise-linear function approximation using the uniform metric. Discrete Comput. Geom. **14**, 445–462 (1995)
11. Guibas, L.J., Hershberger, J.E., Mitchell, J.S.B., Snoeyink, J.S.: Approximating polygons and subdivisions with minimum link paths. Int. J. Comput. Geom. Appl. **3**, 383–415 (1993)
12. Hakimi, S.L., Schmeichel, E.F.: Fitting polygonal functions to a set of points in the plane. CVGIP: Graph. Models Image Process. **53**, 132–136 (1991)
13. Hershberger, J., Snoeyink, J.: An $O(n \log n)$ implementation of the Douglas–Peucker algorithm for line simplification. In: Proceedings of ACM Symposium on Computational Geometry (SOCG), pp. 383–384 (1994)
14. Hershberger, J., Snoeyink, J.: Cartographic line simplification and polygon CSG formulae in $O(n \log^* n)$ time. In: Proceedings of International Workshop on Algorithms and Data Structures (WADS). Lecture Notes in Computer Science, vol. 1272, pp. 93–103. Springer, Berlin (1997)
15. Imai, H., Iri, M.: An optimal algorithm for approximating a piecewise linear function. J. Inf. Process. **9**(3), 159–162 (1986)
16. Imai, H., Iri, M.: Polygonal approximations of a curve-formulations and algorithms. In: Toussaint, G.T. (ed.) Computational Morphology, pp. 71–86. North-Holland, Amsterdam (1988)
17. Melkman, A., O'Rourke, J.: On polygonal chain approximation. In: Toussaint, G.T. (ed.) Computational Morphology, pp. 87–95. North-Holland, Amsterdam (1988)
18. Muthukrishnan, S.M.: Data Streams: Algorithms and Applications. Now Publishers (2005)
19. Zarrabi-Zadeh, H., Chan, T.: A simple streaming algorithm for minimum enclosing balls. In: Proceedings of Canadian Conference on Computational Geometry (CCCG), pp. 139–142 (2006)