Streaming algorithms for line simplification under the Fréchet distance

Streaming Algorithms for Line Simplification under the Fréchet Distance

M.A. Abam[*] M.de Berg[*] P. Hachenberger[*] A. Zarei[†]

Abstract

We study the following variant of the well-known line-simplification problem: we are getting a possibly infinite sequence of points $p_0, p_1, p_2, \ldots$ 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 algorithm, 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 points.

1 Introduction

Suppose 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 not more than a given (fixed) amount of available storage.

The problem described above is a streaming version of line simplification, one of the basic problems in GIS. Here 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 approximates P well. 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 as well as in computational geometry.

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 [5, 6, 7]. 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 $p_i, \ldots, p_j$ the Hausdorff distance or the Fréchet distance are usually used. Since we concentrate on the latter, the error of the simplification Q is now defined as the maximum Fréchet 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 line-simplification was first studied for the Fréchet distance by Godau [4]. Alt and Godau [2] proposed an algorithm to compute the Fréchet distance between two polygonal paths in quadratic time; combined with the approach of Imai and Iri [8] this can be used to compute an optimal solution to the min-δ or the min-k problem for the Fréchet distance. Since solving the line-simplification problem exactly is costly—the best known algorithm for the Fréchet distance takes quadratic time or more—Agarwal *et al.* [1] consider approximation algorithms. In particular, they consider the min-k problem for both the Hausdorff distance for x-monotone paths (in the plane) and the Fréchet distance for general paths (in

[*]Department of Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, the Netherlands. Email: {mabam,mdberg,phachenb}@win.tue.nl. MAA was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 612.065.307. MdB and PH were supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

[†]Computer Engineering Department, Sharif University of Technology and IPM School of Computer Science, Tehran, Iran. Email: zarei@mehr.sharif.edu.

d-dimensional space). They give 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 $\delta/2$. However, these algorithms cannot be used in a streaming setting, because the complexity of the produced simplification for an input path of n points can be $\Theta(n)$.

To state our problem more precisely, we first introduce some terminology and definitions. 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 a function *error* is given that assigns a non-negative 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 $k \leqslant \ell$ and $q_0 = p_0$ and $q_{k+1} = p_n$, 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 consider an error function based on the Fréchet distance, as defined next.

The Fréchet distance between two paths A and B, which we denote by $d_F(A, B)$, 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. The Fréchet distance between A and B is the minimum length of the leash needed for this walk, over all possible such walks. See [4] for a formal definition.

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*(ℓ) 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*, i.e., we allow $\mathcal{A}$ to use a $2k$-simplification, but we compare the error of this simplification to $Q_{Opt(k)}(n)$. Thus we define the competitive ratio of an algorithm $\mathcal{A}(2k)$ as

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

We say that an algorithm is *c-competitive* if its competitive ratio is at most c.

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 Section 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 Fréchet error function is monotone for arbitrary paths in the plane and how to implement the error oracles for this setting. Putting everything together leads to the following result. For paths in the plane and the Fréchet error function we can, for any fixed $\varepsilon > 0$, obtain a $(4\sqrt{2} + \varepsilon)$-competitive streaming algorithm that uses $O((k^2/\sqrt{\varepsilon}) \log^2(1/\varepsilon))$ additional storage and processes each input point in $O((k/\sqrt{\varepsilon}) \log^2(1/\varepsilon))$ amortized time.

2 A general simplification 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*$^*(p_i p_j)$. Later we will prove that the Fréchet error function is monotone, and we will show how to implement the oracle for this setting.

Suppose we have already handled the points $p_0, \ldots, p_n$. (We assume $n > \ell + 1$; otherwise 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 say that the error oracle is an *e-approximate error oracle* if for any link p_ip_j

$$error(p_ip_j) \leqslant error^*(p_ip_j) \leqslant e \cdot error(p_ip_j)$$

Theorem 1 *Suppose that the error function is c-monotone 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 $error(\sigma) \leqslant ce \cdot error(Q_{Opt(k)}(n))$ 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 $m \leqslant 2k+2$, 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's 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,

$$\begin{aligned} error(Q_{Opt(k)}(n)) &\geqslant \tfrac{1}{c} error(\sigma') \geqslant \tfrac{1}{c\cdot e} error^*(\sigma') \\ &\geqslant \tfrac{1}{c\cdot e} error^*(\sigma) \geqslant \tfrac{1}{c\cdot e} error(\sigma). \end{aligned}$$

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. □

3 An algorithm for 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 arbitrary paths in the plane. The first property we need is that the Fréchet error function is monotone. This has in fact already been proven by Agarwal *et al.* [1].

Lemma 2 [1] *The Fréchet error function is 2-monotone on arbitrary paths.*

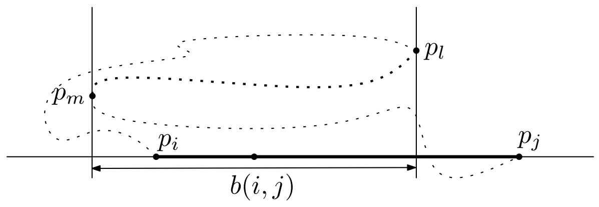

Figure 1: The largest back-path in direction p_ip_j.

Next we turn our attention to the implementation of the error oracle for the Fréchet error function. We use two parameters to approximate $error_F(p_ip_j)$: the width of the points of $P(i,j)$ in the direction p_ip_j and the length of the largest back-path in the direction of p_ip_j.

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. Let $w(i,j)$ be the width of the points in subpath $P(i,j)$ with respect to the direction $\overrightarrow{p_ip_j}$. Chan [3] has described a streaming algorithm for maintaining a core-set that can be used to approximate the width of a set in any direction. More precisely, given a data stream $p_0, p_1, \ldots$, he maintains an ε-core-set of size $O((1/\sqrt{\varepsilon})\log^2(1/\varepsilon))$ in $O(1/\sqrt{\varepsilon})$ amortized time per point; with this core-set one can get a $(1+\varepsilon)$-approximation of the width in any direction.

The largest back-path in direction p_ip_j is defined as follows. Assume without loss of generality that p_ip_j is horizontal with p_j to the 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 $\overrightarrow{p_ip_j}$, we go back when we move from p_l to p_m. The *length* of a back-path $P(l,m)$ on $P(i,j)$ is defined to be the length of the projection of p_lp_m onto a line parallel to p_ip_j, which is equal to $(p_l)_x - (p_m)_x$ since we assumed p_ip_j is horizontal. We define $b(i,j)$ to be the maximum length of any back-path on $P(i,j)$. See Figure 1 for an illustration.

Lemma 3 $\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2}) \leqslant error_F(p_ip_j) \leqslant 2\sqrt{2}\max(\frac{w(i,j)}{2}, \frac{b(i,j)}{2})$.

In the algorithm as presented in Section 2 we need to maintain (an approximation of) the error of each shortcut q_lq_{l+2} in the current simplification. According to the above lemma, in order to approximate $error_F(p_ip_j)$ it is enough if we can approximate $\max(w(i,j), b(i,j))$.

To approximate the width of the links q_lq_{l+2}, we must maintain a core-set for each link that might be needed at some later time in our simplification. These are the links q_iq_j, with $0 \leqslant i < j-1 < 2k+1$. So we need to maintain a core-set for each of these $O(k^2)$ links. Considering a new point $q_{2k+2} = p_{n+1}$, we must create $O(k)$ new core-sets, one for each of the links $q_i p_{n+1}$, with $0 \leqslant i \leqslant 2k$. We create such core-sets for the links $q_i p_{n+1}$, by copying the core-sets $q_i q_{2k+1}$ and 'inserting' point p_{n+1} to them using Chan's algorithm.

We also need to approximate the maximum length of a back-path on the path from q_l to q_{l+2}. For the moment let's 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) $xmax(i,j)$, which is defined as the maximum x-coordinate of any point on $P(i,j)$;
(iii) $xmin(i,j)$, which is defined as 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

$$\max\ (\ b(i,t),\ b(t,j),\ xmax(i,t) - xmin(t,j)\ ).$$

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 $xmax(i,j)$ and $xmin(i,j)$ 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 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 approximated well.

Lemma 4 *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 $\overrightarrow{d}$. Then we have: $b^* - \tan(\alpha) \cdot w \leqslant b \leqslant b^* + \tan(\alpha) \cdot (b^* + w)$.*

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 Chan's ε-core-set method, 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. We set

$$error^*_F(p_i p_j) \ := \ \sqrt{2} \cdot \max(w^*, b^* + \tan(\alpha) \cdot (b^* + w^*)).$$

Combing Lemma 3 with the observations above, we can prove the following lemma.

Lemma 5 *$error_F(p_i p_j) \leqslant error^*_F(p_i p_j) \leqslant 2\sqrt{2}(1 + \varepsilon)(1 + 4\tan(\alpha)) \cdot error_F(p_i p_j)$*

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

Theorem 6 *There is a streaming algorithm that maintains a $2k$-simplification for arbitrary paths under the Fréchet error function and that is $(4\sqrt{2} + \varepsilon)$-competitive with respect to $Opt(k)$. The algorithm uses $O(k^2 \frac{1}{\sqrt{\varepsilon}} \log^2(\frac{1}{\varepsilon}))$ additional storage and each point is processed in $O(k \frac{1}{\sqrt{\varepsilon}} \log^2(\frac{1}{\varepsilon}))$ amortized time.*

References

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

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

[3] T.M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.* 35:20–35 (2006).

[4] M. Godau. A natural metric for curves: Computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annu. Sympos. Theoret. Asp. Comput. Sci.(STACS)*, pages 127–136, 1991.

[5] M.T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discr. Comput. Geom.* 14:445–462 (1995).

[6] L.J. Guibas, J.E. Hershberger, J.S.B. Mitchell, and J.S. Snoeyink. Approximating polygons and subdivisions with minimum link paths. *Int. J. Comput. Geom. Appl.* 3:383–415 (1993).

[7] S.L. Hakimi and E.F. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graph. Models Image Process.* 53:132–136, 1991.

[8] H. Imai and M. Iri. Polygonal approximations of a curve-formulations and algorithms. In: G.T. Toussaint (ed.), *Computational Morphology*, North-Holland, pages 71–86, 1988.