Piecewise Linear Approximation of Streaming Time Series Data with Max-error Guarantees

Ge Luo Ke Yi Siu-Wing Cheng Zhenguo Li Wei Fan Cheng He Yadong Mu

HKUST Huawei Noah's Ark Lab AT&T Labs

***Abstract*—Given a time series $S = ((x_1, y_1), (x_2, y_2), \dots)$ and a prescribed error bound ε, the *piecewise linear approximation (PLA)* problem with max-error guarantees is to construct a piecewise linear function f such that $|f(x_i) - y_i| \le \varepsilon$ for all i. In addition, we would like to have an *online* algorithm that takes the time series as the records arrive in a streaming fashion, and outputs the pieces of f on-the-fly. This problem has applications wherever time series data is being continuously collected, but the data collection device has limited local buffer space and communication bandwidth, so that the data has to be compressed and sent back during the collection process.**

Prior work addressed two versions of the problem, where either f consists of disjoint segments, or f is required to be a continuous piecewise linear function. In both cases, existing algorithms can produce a function f that has the minimum number of pieces while meeting the prescribed error bound ε. However, we observe that neither minimizes the true representation size of f, i.e., the number of parameters required to represent f. In this paper, we design an online algorithm that generates the optimal PLA in terms of representation size while meeting the prescribed max-error guarantee. Our experiments on many real-world data sets show that our algorithm can reduce the representation size of f by around 15% on average compared with the current best methods, while still requiring $O(1)$ processing time per data record and small space.

I. Introduction

Piecewise linear approximation (PLA) for time series data is a classic problem in data compression and signal tracking that dates back to the 1960's [1]. In recent years, the problem has rejuvenated itself, due to the proliferation of ubiquitous data collection devices that continuously capture almost every measurable data source: temperature, humidity, pollution levels, and even people's locations. Since the data collection devices have limited local buffer space and data communication is costly, it is important to be able to compress and send back the data on-the-fly, which calls for *online* algorithms that take the data record one by one, and construct the compressed representation of the time series as the data is streaming in.

Two basic criteria for measuring the quality of the compression are error and size. The most commonly used error form is the ℓ_p-error for $p = 1, 2$, or ∞. Let the time series be $S = ((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n))$, where $x_1 < x_2 < \cdots < x_n$. The ℓ_p-error of a function f for approximating S is $(\sum_{i=1}^{n} |f(x_i) - y_i|^p)^{1/p}$. In particular, when $p = \infty$, the ℓ_∞-error (a.k.a. the max-error) is $\max_i |f(x_i) - y_i|$. This paper, like most prior work that studied this problem in the online setting, uses the ℓ_∞-error, i.e., for a prescribed ε, we would like to construct a piecewise linear function f such that $|f(x_i) - y_i| \le \varepsilon$ for all i. This is because the ℓ_1/ℓ_2-error is ill-suited for online algorithms as it is a sum of errors over the entire time series. When the algorithm has no knowledge about the future, in particular the length of the time series n, it is impossible to properly allocate the allowed error budget over time. Another advantage of the ℓ_∞-error is that it gives us a guarantee on *any* data record in the time series, while the ℓ_1/ℓ_2-error only ensures that the "average" error is good without a bound on any particular record. Admittedly, the ℓ_∞-error is sensitive to outliers, but one could remove them before feeding the stream to the PLA algorithm, and there is abundant work on outlier removal from streaming data (e.g. [2], [3]).

In terms of size, the natural measure is the number of pieces that f has. Specifically, two versions of the problem have been studied: either f is required to be a continuous piecewise linear function, or f consists of disjoint segments. Both cases have been solved optimally (i.e., minimizing the number of pieces of f) by online algorithms that have $O(1)$ amortized processing time per data record using small working space[1]. The disjoint case was first solved optimally by O'Rourke [4] in 1981. The same algorithm was rediscovered recently by Elmeleegy et al. [5], who were probably unaware of O'Rourke's work. This algorithm is called the *slide filter*[2] in [5]. The continuous case was solved optimally by Hakimi and Schmeichel [6]. Elmeleegy et al. [5] also presented an algorithm for the continuous case (called the *swing filter*), but it is not optimal.

We revisit the PLA problem by asking if the number of pieces used by f is really the right measure to optimize. Clearly, the disjoint case uses fewer segments than the continuous case due to fewer restrictions, but it also needs more parameters to represent a segment. Figure 1 shows an example where two consecutive segments are either joint or disjoint at a knot, from which we see that the former only requires 2 parameters to represent while the latter requires 3. Previous work require all the knots to be of the same type, thus does not really optimize the number of parameters needed to represent f. To optimize the representation size of f thus calls for an

[1]The worst-case space complexity of these algorithms is actually $O(n)$, but it occurs only on rather contrived inputs; on most real data sets, the space usage is essentially constant.

[2]The slide filter algorithm, as presented in [5], actually has a worst-case running time $O(n)$ per data record, but it can be easily improved to $O(1)$ by using an observation made in [4] that the convex hull structure used in this algorithm can in fact be updated in $O(1)$ time.

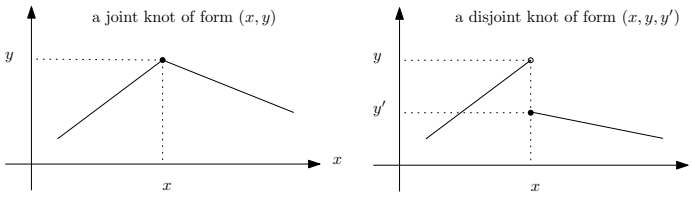


Fig. 1. A joint knot needs 2 parameters to represent, while a disjoint knot needs 3.

adaptive solution that uses a mixture of joint and disjoint knots. We call such a piecewise linear function a *mixed-type* PLA.

Figure 2 shows a concrete example where we try to construct a PLA for a time series consisting of 8 data records. The best (in terms of number of pieces) disjoint PLA has 3 segments: $\overline{A^uD'}, \overline{D^uG'}, \overline{G^lH^u}$, with a representation size of $3 \times 2 = 6$ (ignoring the starting point of the first segment and the ending point of the last segment, which are needed in any method); the best continuous PLA has 4 segments $\overline{A^uB'D^uE'H}$, with a representation size of $2 \times 3 = 6$. However, the best mixed-type PLA consists of 3 segments $\overline{A^uD'}, \overline{D^uE'H}$, with one disjoint knot and one joint knot, resulting in a representation size of $3 + 2 = 5$, which is better than both.

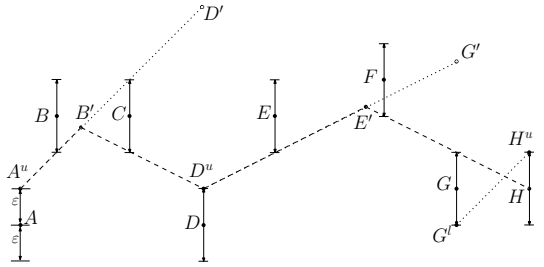


Fig. 2. Different ways to approximate a time series $S = (A, B, C, D, E, F, G, H)$ with ℓ_∞-error ε. The vertical segment centered at each data point has length 2ε. The best disjoint PLA is $\overline{A^uD'}, \overline{D^uG'}, \overline{G^lH^u}$; the best continuous PLA is $\overline{A^uB'D^uE'H}$; the best mixed-type PLA is $\overline{A^uD'}, \overline{D^uE'H}$.

A. Problem definition

Before formally defining the problem we study in this paper, we first describe how to represent a mixed-type PLA. A mixed-type PLA is a piecewise linear function f in which any two consecutive segments are connected via either a joint or disjoint knot. In theory, this requires an extra Boolean array to record the knot types, but in practice, we can exploit the fact that the x-coordinate, which is the timestamp in the time series, is always positive. So we can just use a single array R to record all the parameters needed to represent f. More precisely, for every joint knot (x, y) (see Figure 1), its coordinates are simply stored as they are, while for a disjoint knot (x, y, y'), we store it as $(-x, y, y')$. This way, when we obtain f from R, upon reading a negative entry, we know that this represents a disjoint knot, so we should read the next 3 entries and revert the x-coordinate; otherwise, we know it is a joint knot and just read the next 2 entries.

Thus, we define the representation size of f as the size of the array R, i.e.,

$$size(f) = 2 \times (\#\text{ joint knots}) + 3 \times (\#\text{ disjoint knots}).$$

Note that we have ignored the starting point of the first segment and the ending point of the last segment, to make the definition cleaner.

Now we can formally define the mixed-type PLA problem:

Problem 1 (Mixed-type PLA): Given a time series stream $S = ((x_1, y_1), \ldots, (x_n, y_n))$ where $x_1 < x_2 < \cdots < x_n$, and an error ε, the goal is to construct a mixed-type PLA f such that $size(f)$ is minimized, while $|f(x_i) - y_i| \le \varepsilon$ for all i.

In addition, we would like to solve the problem online, i.e., the records of S are given one at a time. The algorithm has no knowledge about the future, including n, and it should output the pieces of f on-the-fly.

B. Our contributions

In this paper, we design an efficient online algorithm to compute the optimal mixed-type PLA for streaming time series data. The algorithm has $O(1)$ amortized processing time per data record and uses small working space. We have conducted extensive experiments on many real-world data sets, and the results demonstrate 15% reduction on average in terms of the representation size of the constructed PLA, compared with the optimal continuous PLA or disjoint PLA. Note that a reduction in the PLA size brings savings in all downstream processing of the time series data, such as clustering, indexing, similarity search, storage, etc.

A major technical challenge is that, while a simple greedy approach can find the optimal solution for both the continuous and the disjoint PLA problem [4], [6], no greedy algorithms would work for the mixed-type PLA problem. We thus develop a dynamic programming algorithm, based on some nontrivial geometric properties of PLAs. Secondly, as dynamic programming typically works only for offline problems, yielding the optimal solution only after all input data has been processed, we propose a novel early-output technique that can extract part of the optimal solution from the dynamic program as soon as possible, thus making the algorithm online. Our experiments demonstrate that this early-output technique is highly effective on real-world data sets, usually causing a delay of only 3 or 4 segments.

Our algorithm almost always uses small space; in our experiments with many real-world time series data sets, the space usage is always just about 1KB. However, theoretically speaking, there are still some highly contrived inputs on which our algorithm uses $O(n)$ space in the worst case. In Section VI, we show that this is inherently unavoidable by presenting a corresponding lower bound. The lower bound not only holds for the mixed-type PLA problem, it also holds for the continuous and the disjoint PLA problem, explaining why previous algorithms on these problems also inevitably require $O(n)$ space in the worst case.

To summarize, our contributions are as follows.

1) We observe that neither the continuous nor the disjoint PLA problem studied previously truly optimizes the representation size of the PLA, and propose a new formulation of the problem allowing the PLA to adaptively use a mixture of joint and disjoint knots, so as to minimize the representation size.
2) We design an optimal online algorithm with $O(1)$ processing time per data record, while using small working space. The algorithm is quite different from existing algorithms for the continuous or disjoint PLA problem, which all use a simple greedy approach.
3) We have performed extensive experiments on many real-world data sets. The results demonstrate a reduction of 15% in terms of representation size by using the optimal mixed-type PLA, compared with current best methods.
4) We prove an $\Omega(n)$ worst-case space lower bound on any algorithm for solving any version of the PLA problem. This means no algorithm can always guarantee small space usage if the problem is to be solved optimally.

II. Related Work

Constructing good PLAs to approximate time series data is a fundamental problem in data compression, statistics, and databases. There has been extensive work on this problem, on both the case where the PLA is a continuous piecewise linear function and the case where it consists of disjoint segments. But our work is the first to consider a mixed-type PLA, which truly optimizes the representation size.

For both the continuous and the disjoint case, past work simply used the number of segments as the size of the PLA. There are two ways to formulate this problem. The instance studied in this paper assumes a given error bound ε, and tries to minimize the size of the PLA. Flipping the question around, one may also ask that for a given size k, how to construct the optimal PLA with at most k segments that minimizes the error. The former is often called the *min-k* problem while the latter the *min-ε* problem. Note that, however, the min-ε version is not compatible with the online requirement, since the optimal k-segment PLA (for a fixed k) inevitably changes as more data arrives, so it is not possible to output the optimal solution in an online fashion.

Different error measures have been considered, most commonly the $\ell_1/\ell_2/\ell_\infty$-error. Under ℓ_∞-error, the min-k problem is exactly the problem studied in this paper, and past work has already been reviewed in Section I. Lazaridis and Mehrotra [7] studied a restricted version of this problem where the PLA is replaced by a piecewise constant function. For the min-ε problem, the continuous case is known as the *polygonal fitting problem* in computational geometry. It was first solved by Hakimi and Schmeichel [6] in $O(n^2 \log n)$ time, which was improved to $O(n^2)$ by Wang et al. [8] and further to $O(n \log n)$ by Goodrich [9]. The disjoint case was solved by Chen and Wang [10].

Under ℓ_1/ℓ_2-error, the min-ε problem using k disjoint segments is also known as the *histogram construction* problem in the database literature. A histogram can be considered as a piecewise constant function, and it turns out that all algorithms for the piecewise constant case also apply to the disjoint piecewise linear case. Jagadish et al. [11] were the first to consider this problem and presented an $O(n^2 k)$ dynamic programming algorithm. Guha et al. [12] gave near-linear time algorithms, although only returning approximately optimal solutions. Their algorithms also work in the streaming model, but are not online algorithms as the solution is only constructed at the end of the stream. The min-k problem can be solved by using a binary search on k while invoking a min-ε algorithm. The continuous PLA problem (either the min-ε or the min-k version) under ℓ_1/ℓ_2-error turns out to be much more difficult. Currently it only has an approximation algorithm by Aronov et al. [13], and it is not even known if the problem is NP-hard.

As the ℓ_1/ℓ_2-error is ill-suited for the online setting, compromises have been sought. A widely used online PLA algorithm that heuristically optimizes the ℓ_1/ℓ_2-error is the SWAB algorithm of Keogh et al. [14]. Instead of imposing an ℓ_1/ℓ_2-error constraint on the entire time series, it puts an error bound on each segment of the PLA, and combines a greedy algorithm with a bottom-up heuristic to optimize the overall error. This algorithm produces a disjoint PLA. One can also use ℓ_∞-error in SWAB, but the result is not optimal. Palpanas et al. [15] proposed another online PLA algorithm using a variant of the ℓ_2-error that puts less emphasis on the past, hence the name "amnesic".

Finally, there are many other compressed representations for time series data, such as *Discrete Fourier Transform* [16], *Piecewise Aggregate Approximation* [17], *Discrete Wavelet Transform* [18], *Adaptive Piecewise Constant Approximation* [19], [7], and *Chebyshev Polynomials* [20]. Each one has its advantages and disadvantages. Nevertheless, PLA remains one of the most commonly used representations for time series data, as argued in [14], [15], [21].

III. Concepts and Properties

In this section, we introduce a series of concepts with regard to the PLA problem and their properties, which will be essential to the correctness and optimality of our algorithm.

We model the time series as a sequence of points in the plane. Let $S = (p_1, p_2, \ldots, p_n)$ be the time series consisting of n points, where $p_i = (x_i, y_i)$ and $x_1 < x_2 < \cdots < x_n$. We use the following notation throughout the paper: For two points p, q, $\overline{pq}$ denotes the line segment with p, q as the endpoints, and $\overleftrightarrow{pq}$ denotes the (infinite) line passing through p and q. We use $\overleftrightarrow{pq}(\cdot)$ to denote the linear function corresponding to the line $\overleftrightarrow{pq}$, namely, $\overleftrightarrow{pq}(x)$ is the y-coordinate of the point on the line $\overleftrightarrow{pq}$ at x.

A. Mixed-type PLA fitting through an extended polygon

We first convert Problem 1 into the problem of fitting a mixed-type PLA through an extended polygon.

For a prescribed error ε, for each point p_i, we create two new points $p_i^u = (x_i, y_i + \varepsilon)$ and $p_i^l = (x_i, y_i - \varepsilon)$. Obviously, any valid PLA that approximates S with ℓ_∞-error at most ε must intersect the segment $\overline{p_i^u p_i^l}$ for every i. This restricts

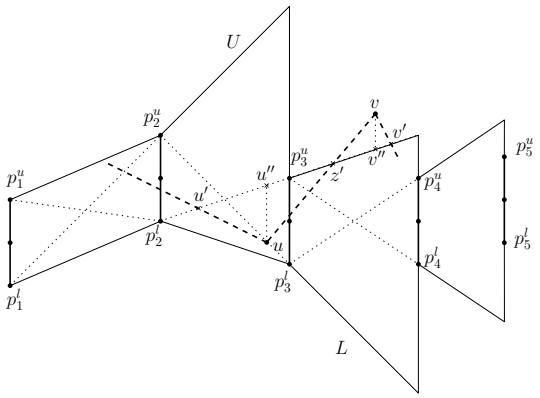


Fig. 3. The extended polygon P

the position of the PLA only at every x_i; below we define a polygon P that restricts the PLA at all other x-coordinates.

The *extended polygon* of S, denoted by P, is the following region:

$$P = \{(x, y) \mid x_i \le x \le x_{i+1}, l_i(x) \le y \le u_i(x), i = 1, \ldots, n-1\},$$

where

$$l_i(x) = \begin{cases} \overleftrightarrow{p_1^l p_2^l}(x) & \text{if } i = 1 \\ \min\{\overleftrightarrow{p_{i-1}^u p_i^l}(x), \overleftrightarrow{p_i^l p_{i+1}^l}(x)\} & \text{if } 2 \le i \le n-1 \end{cases}$$

and

$$u_i(x) = \begin{cases} \overleftrightarrow{p_1^u p_2^u}(x) & \text{if } i = 1 \\ \max\{\overleftrightarrow{p_{i-1}^l p_i^u}(x), \overleftrightarrow{p_i^u p_{i+1}^u}(x) & \text{if } 2 \le i \le n-1 \end{cases}.$$

Please see Figure 3 for an example. Intuitively, the polygon is upper bounded by the polygonal line $\overline{p_1^u p_2^u \ldots p_n^u}$, but we expand it by extending the segment $\overline{p_{i-1}^l p_i^u}$ to the right until x_{i+1}, for $i = 2, \ldots, n-1$. We do a similar extension for the lower boundary. Note that the extension does not necessarily expand the polygon; for example in Figure 3, extending $\overline{p_1^u p_2^l}$ does not expand the polygon.

We can see that P is bounded from the left by $\overline{p_1^u p_1^l}$, which we call the *initial window*, bounded from the right by $\overline{p_n^u p_n^l}$, which we call the *final window*, bounded from above by an *upper chain* from p_1^u to p_n^u, denoted by U, and bounded from below by a *lower chain* from p_1^l to p_n^l, denoted by L.

Note that we can easily construct P in an online fashion, since each portion of P only depends on 3 consecutive points in the time series. It will also soon be clear that we do not need to store the entire P in memory; the portions of P corresponding to old data can be discarded as soon as the algorithm is done with them.

The benefit of introducing the extended polygon is that it restricts the position of the optimal PLA. More precisely, we define the *mixed-type PLA fitting* problem as follows, and then reduce the original PLA problem to it.

Problem 2 (Mixed-type PLA fitting): Given an extended polygon P, the goal is to construct a mixed-type piecewise linear function f that starts from somewhere on the initial window of P, goes entirely inside P, and finally reaches somewhere on the final window of P, and find such an f with the smallest $size(f)$.

The following lemma reduces the original mixed-type PLA problem (Problem 1) to Problem 2.

Lemma 1: An optimal solution for the mixed-type PLA fitting problem on P is also an optimal solution for the mixed-type PLA problem on S.

Proof: It is obvious that any valid solution to the fitting problem on P is also a valid solution for the PLA problem on S. So to prove the lemma, it suffices to show that, for any optimal solution f for the PLA problem on S, if it does not fit in P, we can do some transformation to fit into P without increasing its size.

Let f be an optimal solution for the PLA problem on S. First, observe that f cannot have more than one knot between x_i and x_{i+1} for any i. Otherwise we could replace two knots (whatever their types are) by a new disjoint knot, which would reduce the size of the PLA by at least 1. Second, a disjoint knot between x_i and x_{i+1} for any i can always be aligned to x_{i+1} by extending(resp. shrinking) the preceding(resp. succeeding) segment of this knot.

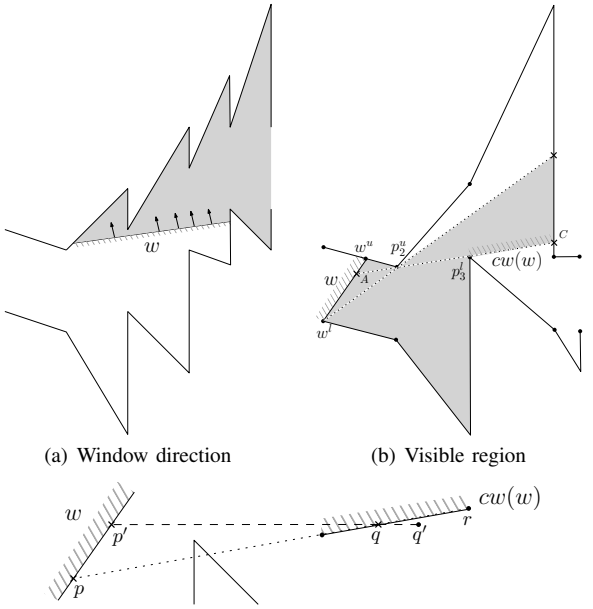
Without loss of generality, we only consider the case where f goes above U somewhere between x_i and x_{i+1}. Let $\overline{uv}$ be the segment that crosses U, where u is inside P and v is outside (see Figure 3, where $i = 3$). Since f must go below p_{i+1}^u, v must be on the left side of x_{i+1}. Similarly, since f must go above p_{i-1}^l, u must be on the right of x_{i-1}. Since v is above U, it must be above $\overleftrightarrow{p_{i-1}^l p_i^u}$, whose slope is therefore smaller than that of $\overline{uv}$.

Now let us consider the knots at u and v, which we denote by $knot_u$ and $knot_v$, respectively. If both knots are disjoint, then we can simply replace $\overline{uv}$ by $\overline{u''v''}$, where u'' (resp. v'') is the intersection between $\overleftrightarrow{p_{i-1}^l p_i^u}$ and the vertical line placed at u (resp. v). If $knot_u$ is joint and $knot_v$ is disjoint, we consider the segment preceding $\overline{uv}$. Suppose it intersects $\overline{p_{i-1}^l p_i^u}$ at u'. Then we can replace $\overline{uv}$ by $\overline{u'v''}$. If $knot_u$ is disjoint and $knot_v$ is joint, we similarly consider the succeeding segment of $\overline{uv}$, which intersects $\overline{p_{i-1}^l z}$ at v', and replace $\overline{uv}$ by $\overline{u''v'}$. Finally, if both knots are joint, then we can replace it by $\overline{u'v'}$. It should be clear that these operations do not increase the size of f. Doing so for every segment that is outside P will eventually make all parts of f inside P. ■

B. Visible regions and closing windows

From now on, we can focus on the mixed-type PLA fitting problem on the extended polygon P.

We first introduce some terms. Given an extended polygon P, a *window* w of P is a segment $\overline{w^u w^l}$ with one endpoint w^u on U and the other endpoint w^l on L, while the whole segment is inside P. For example, the initial window $\overline{p_1^u p_1^l}$ and the final windows $\overline{p_n^u p_n^l}$ are both special windows. A window w (except the initial and final window) cuts P into two sub-polygons. The one that connects with the initial window is said to be *on the left side* of w, while the other is *on the right side*. Note that this left-right relationship is relative to the polygon



(a) Window direction (b) Visible region

(c) An observation on the closing window

Fig. 4. Windows, visible regions, and closing windows

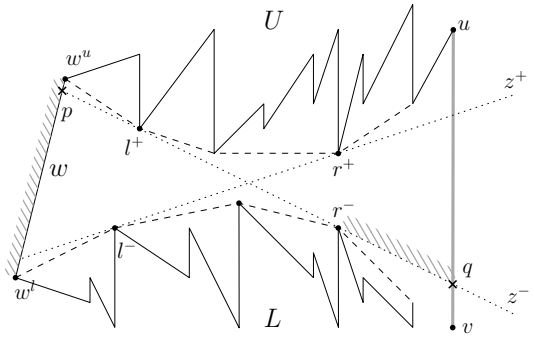


Fig. 5. Algorithm for computing the visible region and closing window.

P. For the example in Figure 4(a), the shaded region is on the right side of w, although part of it is immediate to the left of w.

A point q is said to be *visible* from w if q is on the right side of w and can be reached by a ray shooting from some point p on w, i.e., the segment $\overline{pq}$ completely lies inside P. The *visible region* of w, denoted as $vr(w)$, is thus the set of all points visible from w. Intuitively, if we imagine w as a light source (emitting light only to the right), then $vr(w)$ is simply the area that can be lit up by w (see Figure 4(b)).

Suppose $vr(w)$ does not reach the final window. Then $vr(w)$ is a polygon inside P, which is bounded from the right by another window. We call this window the *closing window* of w, denoted by $cw(w)$ (see Figure 4(b)). We have the following observation with respect to the closing window: If we extend $cw(w)$ to the left, the line will hit w without leaving P. To see why this is true, see Figure 4(c). We extend $cw(w)$ to the left until it hits w, and denote the resulting segment $\overline{pr}$ where p is on w and r is the right endpoint of $cw(w)$. Suppose $\overline{pr}$ is intersected by the boundary of P. Consider an arbitrary point q on $cw(w)$, which must be visible from some point p' on w other than p. Since $\overline{pq}$ and $cw(w)$ are not coincident, that means that p' must be able to see some q' on the right side of $cw(w)$. This contradicts with the definition of the visible region. We will call $\overline{pr}$ the *generating segment* of the visible region and denote it by $gen(vr(w))$. Note that this segment is the one that goes the furthest (relative to P) starting from somewhere on w.

Computing the visible region and the closing window. For a given window w, $vr(w)$ and $cw(w)$ can be efficiently computed in an online fashion, with amortized $O(1)$ time per data point, by the algorithm in [22]. For self-containment, we briefly describe the algorithm below, adapted to our setting.

Starting from w, the algorithm maintains the convex hull of U (bounding from below), as well as the convex hull of L (bounding from above). See the dashed lines in Figure 5. In addition, it maintains two supporting lines z^+, z^-, which separate the two convex hulls, and among all such separating lines, z^+ (resp. z^-) has the maximum (resp. minimum) slope. These two supporting lines are tangent to the two convex hulls at four vertices, l^+, r^+, l^-, r^-.

Suppose a new vertical segment $\overline{uv}$ on U arrives; the case with L is symmetric. We first update the convex hull of U by inserting v into it. Then there are 3 cases: 1) if v is above z^+, there is nothing more to do; 2) if v is below z^+ but above z^-, we rotate z^+ by setting r^+ to be v, and moving l^- to the right if necessary; 3) if v is below z^-, we compute the crossing point q between z^- and $\overline{uv}$, close the visible region and set the closing window $cw(w) = \overline{r^- q}$.

From the algorithm above, it is clear that the right endpoint of the closing window is always on a vertical segment of U or L. And it was shown in [22] that the amortized processing time per data point of this algorithm is $O(1)$. Its space complexity depends on the size of the two convex hulls, which could be $O(n)$ in the worst case. But in reality it is essentially a constant.

Two key properties. We now prove two key properties with regard to closing windows, which will be essential for the correctness and optimality of our algorithm.

For two windows w_1, w_2 of P, if w_2 is completely on the right side of w_1, we say w_2 is on the right side of w_1, denoted by $w_1 \preceq w_2$. The relationship $\preceq$ defines a partial order on the windows of P; if two windows cross each other, then they are not comparable under $\preceq$.

The first property is quite intuitive.

Lemma 2 (Order-preserving property): For any two windows w_1, w_2 of P, if $w_1 \preceq w_2$, then $cw(w_1) \preceq cw(w_2)$.

Proof: First, it is trivial to see that $cw(w_2) \preceq cw(w_1)$ is impossible. This is because any ray shooting from w_1 must cross w_2 to get to the right side of w_2, hence any point (on the right side of w_2) that can be seen from w_1 can also be seen from w_2.

Now suppose that $cw(w_1)$ crosses $cw(w_2)$ at some point q as shown in Figure 6. Consider the generating segment of

$vr(w_1)$, i.e., the one obtained by extending $cw(w_1)$ to the left, which hits w_1 at some point p. Since $\overline{pq}$ is completely inside P, it must cross w_2 at some point p'. Next, let q' be some point on $\overleftrightarrow{pq}$ but on the right side of $cw(w_2)$. It is visible from w_2 from p', but this contradicts with the fact that q' is on the right side of $cw(w_2)$ so should not be visible from w_2. ■

The second property is less obvious.

Lemma 3 (Non-crossing property): For any two windows w_1, w_2 of P, $cw(w_1)$ and $cw(w_2)$ do not cross.

Proof: Lemma 2 already has established that if w_1 and w_2 do not cross, then $cw(w_1)$ and $cw(w_2)$ do not cross. So we just consider the case where w_1 and w_2 cross each other. Let their crossing point be p. Suppose for contradiction that $cw(w_1)$ crosses $cw(w_2)$ at point q. Consider the generating segment of $vr(w_1)$ by extending $cw(w_1)$ to the left, which hits w_1 at p_1. Further consider the segment $\overline{p_1q}$. If $\overline{p_1q}$ crosses w_2, then by the same argument as in the proof of Lemma 2, we can find some q' on that right side of $cw(w_2)$ that is visible from w_2, which causes a contradiction. So $\overline{p_1q}$ cannot cross w_2. Similarly, the generating segment of $vr(w_2)$, $\overline{p_2q}$ for some point p_2 on w_2, cannot cross w_1. Therefore, except for the directions of the two closing windows, Figure 7 shows the only possible situation.

Clearly p_2 is visible from w_1 since $\overline{pp_2}$ is completely in P and p_2 is on the right side of w_1. And we know $\overline{p_2q}$ is completely inside P. Now, if the direction of $cw(w_1)$ is not the same as shown in Figure 7, then $\overline{p_2q}$ will be completely on the right side of $cw(w_1)$, which implies that p_2 will not be visible from w_1. Therefore, we conclude that the direction of $cw(w_1)$ must be as shown in the figure. Using a symmetric argument, the direction of $cw(w_2)$ must also be as shown.

Now focus on $\overline{p_1q}$ where p_1 and q are both on the generating segment of $vr(w_1)$. Since the direction of $cw(w_1)$ is as shown, that means if we rotate $\overline{p_1q}$ clockwise slightly according to a center placed on somewhere along this segment, it will remain in $vr(w_1)$, but we cannot rotate it counterclockwise according to a center placed on anywhere along this segment without leaving $vr(w_1)$. This implies that there exist a point l^- on the lower chain L and a point r^+ on the upper chain U that both touch $\overline{p_1q}$ as shown in the figure. However, since $\overline{p_2q}$ is inside P, point l^- must be on or below $\overline{p_2q}$. This leaves the only possibility which is that both p_1 and p_2 must coincide with p. This in turn makes $cw(w_1)$ and $cw(w_2)$ coincide, but with opposite directions. This is not possible, since every window of P has a unique direction. ■

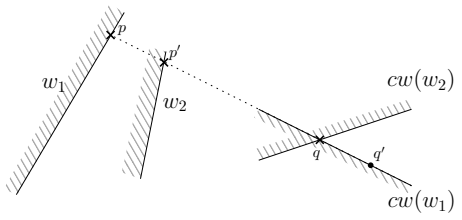


Fig. 6. The order-preserving property. The shade of each window indicates the left side.

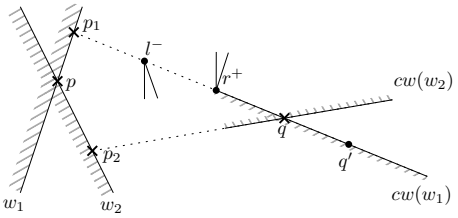


Fig. 7. The non-crossing property. The shade of each window indicates the left side.

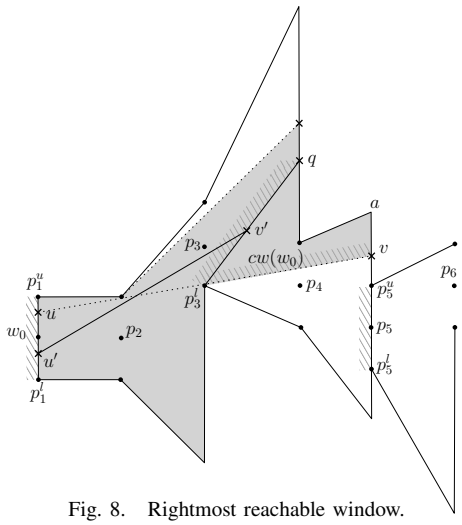


Fig. 8. Rightmost reachable window.

C. Rightmost reachable window

Recall that the mixed-type PLA fitting problem (Problem 2) asks us to find the f with the smallest $size(f)$ that starts from the initial window of P and goes to its final window. In our algorithm, we will flip the question around, and ask that for a given size k, what is the rightmost point reachable by an f with $size(f) = k$. Then, the smallest k such that f reaches some point on the final window of P yields the optimal solution to the original problem.

However, it is difficult to develop a dynamic programming algorithm based on the "rightmost reachable point" concept. Another technical problem is that the rightmost reachable point is not unique. For example, in Figure 8, for $k = 0$, all points on the segment $\overline{av}$ are the rightmost points reachable from the initial window w_0. Thus, we consider the *rightmost reachable window* by an f with $size(f) = k$ starting from w_0, and denote this window by $C[k]$. Recall that a window of P is a segment with one endpoint on the upper chain U and the other endpoint on the lower chain L, and the "rightmost" one is defined relative to P under the partial order $\preceq$. However, as $\preceq$ is a partial order and not all windows are comparable, it is not clear if this "rightmost reachable window" is a valid definition at all. Below, we show that $C[k]$ is indeed well-defined and unique for any k (except for the case $k = 1$ since no PLA has size 1).

We start by analyzing some simple cases.

- For $k = 0$, which means that no knots can be used, the problem is exactly to compute $vr(w_0)$. By the definition of the visible region, if w_0 can "see" the final window

directly, then we have already found the optimal solution with $size(f) = 0$. Otherwise, $vr(w_0)$ is bounded by the closing window $cw(w_0)$, and there is a segment (the generating segment $gen(vr(w_0))$) that starts from somewhere on w_0 and goes through $cw(w_0)$, so we have $C[0] = cw(w_0)$. In the example of Figure 8, $gen(vr(w_0)) = \overline{uv}$.

- Next, consider the case $k = 2$, which means that the PLA must consist of 2 segments connected by a joint knot. We will argue that $C[2] = cw(cw(w_0))$, i.e., in order to go the furthest to the right, the second segment of the PLA should just start from somewhere on $cw(w_0)$. To see why this is true, consider any PLA with 2 segments connected by a joint knot v' (see Figure 8). The first segment of the PLA is $\overline{u'v'}$. Since $C[0] = cw(w_0)$ is the rightmost reachable window with one segment, v' must be on the left side of $C[0]$ or on $C[0]$. Thus, any ray shooting from v' must penetrate $C[0]$, so any point reachable by a segment starting from v' must also be reachable by a segment starting from somewhere on $C[0]$. Therefore, to find the rightmost reachable window for any PLA with $k = 2$, it is sufficient to restrict the starting point of the second segment to $C[0]$, i.e., $C[2] = cw(C[0]) = cw(cw(w_0))$.
- When $k = 3$, the PLA must consist of 2 segments connected by a disjoint knot. For any window w, we let its *next window*, $nw(w)$, to be the window $\overline{p_i^u p_i^l}$ on its immediate right that is defined by a data point. For example in Figure 8, $cw(w_0)$'s next window is $nw(cw(w_0)) = \overline{p_5^u p_5^l}$. We claim that $C[3] = cw(nw(C[0])) = cw(nw(cw(w_0)))$. Again, by the definition of the rightmost reachable window $C[0]$, its next window $\overline{p_5^u p_5^l}$ cannot be reached by the first segment of the PLA. However, the first segment's right endpoint can have the same x-coordinate as the next window $\overline{p_5^u p_5^l}$. Thus, the second segment is restricted to have a starting point on $\overline{p_5^u p_5^l}$, so we must have $C[3] = cw(nw(C[0]))$.
- For $k \geq 4$, the PLA consists of two or more knots. Consider the last knot. If it is a joint knot, using similar arguments as the $k = 2$ case above, we have $C[k] = cw(C[k-2])$. If it is a disjoint knot and $k \geq 5$, using similar arguments as the $k = 3$ case above, we have $C[k] = cw(nw(C[k-3]))$.

To summarize, we have the following recursive definition of $C[k]$:

$$C[k] = \begin{cases} cw(w_0) & \text{if } k = 0 \\ cw(nw(C[k-3])) & \text{if } k = 3 \text{ or } (k \geq 5 \text{ and} \\ & cw(C[k-2]) \preceq cw(nw(C[k-3]))) \\ cw(C[k-2]) & \text{else.} \end{cases}$$

Note that by the non-crossing lemma (Lemma 3), $cw(C[k-2])$ and $cw(nw(C[k-3]))$ must be comparable as both are closing windows.

This recursion clearly yields a well-defined and unique $C[k]$ for any $k \neq 1$. Below we establish its correctness and optimality.

Lemma 4: Let $C[k]$ be defined as above. For any $k \neq 1$, we have that (1) there exists a PLA f of $size(f) = k$ that fits inside P and its last segment contains $C[k]$; and (2) for any PLA f of $size(f) = k$ that fits inside P, it cannot reach the right side of $C[k]$.

Proof: The proof for the base cases $k = 0, 2, 3$ is straightforward following the discussion above. Now we assume by induction that the lemma is true for $0, 2, 3, \ldots, k-1$, and will prove that it is true for k, where $k \geq 4$.

The (1) part is straightforward, since $C[k]$ is set to either $cw(nw(C[k-3]))$ or $cw(C[k-2])$. By the induction hypothesis, there exists a PLA f_1 of $size(f_1) = k-3$ and its last segment contains $C[k-3]$. If $C[k] = cw(nw(C[k-3]))$. we add a segment to f_1 (preceded by a disjoint knot) that starts at $nw(C[k-3])$ and goes through $cw(nw(C[k-3]))$. This gives us the desired f of $size(f) = k$. Similarly, by the induction hypothesis, there exists a PLA f_2 of $size(f_2) = k-2$ and its last segment contains $C[k-2]$. If $C[k] = cw(C[k-2])$, we add a segment to f_2 (preceded by a joint knot) that starts at $C[k-2]$ and goes through $cw(C[k-2])$.

Next, we consider the (2) part. By Lemma 3, $cw(nw(C[k-3]))$ and $cw(C[k-2])$ must be comparable, and we take the one on the right and set it as $C[k]$. Suppose that there exists an f of $size(f) = k$ that reaches the right side of $C[k]$. Consider its last knot v', and let f_1 be the PLA after removing v' and the last segment of f. If v' is a joint knot, then f_1 must have size $k-2$. By the induction hypothesis, f_1 cannot reach the right side of $C[k-2]$, so v' must be on the left side of $C[k-2]$. By Lemma 2, it is impossible for a segment to start from v' and reach the right side of $cw(C[k-2]) \preceq C[k]$. If v' is a disjoint knot, then f_1 must have size $k-3$. By the induction hypothesis, f_1 cannot reach the right side of $C[k-3]$, hence also $nw(C[k-3])$. Thus, the last segment of f must start from $nw(C[k-3])$ or earlier. Again by Lemma 2, it is impossible for a segment to start from $nw(C[k-3])$ and reach the right side of $cw(nw(C[k-3])) \preceq C[k]$.

Summarizing these cases, we conclude that $C[k]$ is the rightmost reachable window for any PLA of size k. ■

IV. The Algorithm

Based on the recurrence of $C[k]$, it is easy to have a dynamic programming algorithm. Some care, however, needs to be taken to make the algorithm online while using small working space.

A. Dynamic programming

We first describe the data structures needed for the dynamic programming algorithm. Obviously we need the array $C[\cdot]$, which is computed using the recurrence. For each entry $C[k]$, we also need a pointer $pred[k]$ that points to either $k-3$ or $k-2$ depending on whether $C[k]$ is set to $cw(nw(C[k-3]))$ or $cw(C[k-2])$. This can actually be implemented as a bit array as each $pred[k]$ has only two possibilities. As the base cases, we set $pred[2] = pred[3] = 0$ since $C[1]$ and $C[-1]$

Algorithm 1 UPDATE(p)

1: **for** each $vr \in lvr$ **do**
2: **if** vr is open **then**
3: $vr.update(p_t)$;
4: **end if**
5: **end for**
6: **while** $vr(nw(C[k-3]))$, $vr(C[k-2])$ are both closed **do**
7: **if** $cw(nw(C[k-3])) \succeq cw(C[k-2])$ **then**
8: $C[k] \leftarrow cw(nw(C[k-3]))$, $pred[k] \leftarrow k-3$, $ref[k-3] \leftarrow ref[k-3]+1$;
9: **else**
10: $C[k] \leftarrow cw(C[k-2])$, $pred[k] \leftarrow k-2$, $ref[k-2] \leftarrow ref[k-2]+1$;
11: **end if**
12: discard $vr(nw(C[k-3]))$, $vr(C[k-2])$ from lvr;
13: create $vr(C[k])$, $vr(nw(C[k]))$ and insert them to lvr;
14: $k \leftarrow k+1$;
15: PRUNE($C[\cdot], pred[\cdot]$);
16: **end while**
17: EARLY-OUTPUT($C[\cdot], pred[\cdot]$);

are both ill-defined; and we further set $pred[0] = null$ for convenience.

As the algorithm takes incoming data points, it executes a number of instances of the algorithm for computing visible regions as described in Section III-B. Abusing notation, we also use $vr(w)$ to denote the data structures needed for computing the visible region of w. When $vr(w)$ is closed by a closing window, we have obtained $cw(w)$ and can discard $vr(w)$. Before reaching the closing window $cw(w)$, we say $vr(w)$ is open, and we maintain a list lvr which keeps at most 5 open visible regions, namely, $vr(nw(C[k-3]))$, $vr(C[k-2])$, $vr(nw(C[k-2]))$, $vr(C[k-1])$, $vr(nw(C[k-1]))$ during the period when computing $C[k]$. Initially, we set $lvr = \{vr(w_0)\}$ and $k = 0$.

When a data point p arrives, we first update all the open visible regions in the list lvr. If p closes one or more regions, then we obtained their closing windows, and can move ahead with the dynamic programming using the recurrence. The algorithm is outlined in Algorithm 1. Note that we have not considered the base cases $k = 0, 2, 3, 4$ in the pseudocode for better clarity, which are easy to handle. There are two functions PRUNE(), EARLY-OUTPUT() and another array $ref[\cdot]$ used in the algorithm, which will be explained later.

When the stream finishes, the current k corresponds to the optimal size of the PLA. To construct the actual PLA, we look at $vr(nw(C[k-3]))$ or $vr(C[k-2])$. Note that at least one of them must be open, since otherwise the algorithm would have computed $C[k]$ and incremented k. If $vr(nw(C[k-3]))$ is open, the last piece of the PLA can be any line segment starting from $nw(C[k-3])$ and reaching the final window, e.g., one of the supporting lines z^+ or z^- maintained by the visible-region algorithm. We also know that the knot preceding the last segment is a disjoint knot. Then following the pointers starting from $pred[k-3]$, we can construct the PLA iteratively. The case with $vr(C[k-2])$ being open can be handled similarly.

Algorithm 2 PRUNE($C[\cdot], pred[\cdot]$)

1: $i \leftarrow k-4$;
2: **while** $i > 0$ and $ref[i] = 0$ **do**
3: $i' \leftarrow pred[i]$;
4: delete $C[i], pred[i], ref[i]$;
5: $ref[i'] \leftarrow ref[i'] - 1$;
6: $i \leftarrow i'$;
7: **end while**

Note that at any time, there are at most 5 open visible regions in lvr. Since updating each visible region takes amortized constant time, the total time spent by the algorithm in lines 1–5 is $O(1)$ amortized. The **while** loop in lines 6–16 also takes $O(1)$ time amortized, since each iteration increases k by 1, and k is at most n. So the total running time of the update algorithm is $O(1)$ amortized, excluding PRUNE() and EARLY-OUTPUT().

However, the space usage of the algorithm is large, required by the two arrays $C[\cdot]$ and $pred[\cdot]$. More seriously, the algorithm is not online, as it has to wait until the end of the stream, and then traces back following the $pred$ pointers to reconstruct the optimal solution. In the rest of this section, we show how to tackle these two issues.

B. Pruning the array

We first consider how to reduce the number of entries in the array $C[\cdot]$, hence $pred[\cdot]$. The idea is to delete those entries that are impossible to be part of the optimal solution. For this purpose, we introduce another array $ref[\cdot]$, with $ref[k]$ being the number of times $C[k]$ has been pointed to. Note that this array can be easily maintained whenever we add a new $C[k]$ and set its $pred[k]$ pointer.

An important observation is that since $pred[k]$ can only be $k-2$ or $k-3$, $ref[i]$ will never be incremented for any $i < k-3$. If $ref[i] = 0$ for such an i, that means $C[i]$ will never be used and thus can be discarded. After deleting $C[k]$ (and also $pred[k]$), we will also decrement $ref[pred[k]]$, which in turn may trigger iterative deletions. The pruning algorithm is outlined in Algorithm 2. Note that to facilitate the pruning, it is actually easier to implement $C[\cdot]$, $pred[\cdot]$, and $ref[\cdot]$ as a list of objects with 3 fields.

Clearly, PRUNE() takes amortized constant time since each entry of $C[\cdot]$ is deleted at most once.

C. Early output

Next, we consider the problem of outputting the optimal solution while the input data arrives over time. In general, dynamic programming algorithms have to wait for the entire space of subproblems to be computed to start reconstructing the optimal solution. However, we observe an interesting special property of the subproblem state diagram, which allows us to output the optimal solution as early as possible.

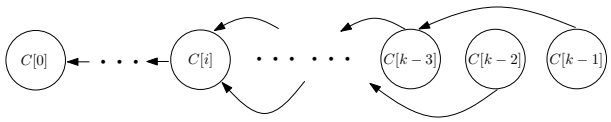


Fig. 9. The structure of $C[\cdot]$.

First, since a backward pointer $pred$ jumps at most 3 steps, any solution has to pass through at least one of $C[k-1]$, $C[k-2]$, or $C[k-3]$. Meanwhile, each entry of $C[\cdot]$ has exactly one backward pointer (except the first entry). Suppose we trace the solution backward with $C[k-1]$, $C[k-2]$, and $C[k-3]$ as the starting point, respectively, and the 3 traces first meet at some $C[i]$. After $C[i]$, the 3 traces will always stay together. This means that the common part of the 3 traces must be a prefix of each trace, and it must be a prefix of the optimal solution, thus can be outputted.

On the other hand, consider any $C[j]$ for $j < i$ in the pruned array. We will show that $ref[j] = 1$. Obviously, $ref[j] > 0$ since we have pruned all entries with $ref[j] = 0$. Meanwhile, observe that no backward pointers jump over $C[i]$, since $C[i]$ is common to all 3 traces by definition (note that there is no other trace). Thus, all references to $C[j]$ must come from some entry before $C[i]$ (including $C[i]$), which means that the total number of references to $C[0], \ldots, C[i-1]$ must equal to the total number of entries that have not been pruned in $C[0], \ldots, C[i]$, minus 1 (the first entry does not have a backward pointer), which equals to the number of entries in $C[0], \ldots, C[i-1]$. Thus, every $C[j], j < i$ must have reference count exactly equal to 1, since if some entry has 2 or more references, then there must be another with 0 reference, which violates the precondition that all such entries have already been pruned.

The above observations imply that the structure of the $C[\cdot]$ array (implemented as a list) has a very special form, as illustrated in Figure 9. Therefore, we can output the "tail" part as soon as possible. After outputting the corresponding solution, the "tail" can also be deleted (thus, the real picture as maintained by the algorithm does not have the tail).

Since we implement $C[\cdot]$ as a list, the early-output algorithm is actually quite simple, as illustrated in Algorithm 3. The algorithm also takes amortized constant time since each entry of $C[\cdot]$ is deleted at most once.

Algorithm 3 EARLY-OUTPUT($C[\cdot]$)

```
1: head ← the first entry in C[·];
2: next ← the second entry in C[·];
3: while ref[head] = 1 and ref[next] = 1 do
4:    if next < k − 3 then
5:       output the corresponding piece of C[head];
6:       discard C[head];
7:       head ← next;
8:       next ← the next entry after C[next];
9:    end if
10: end while
```

Note that the non-tail part of $C[\cdot]$ can still get very long, and linear in n in the worst case. However, such cases only happen on contrived inputs. In practice, the non-tail part is very short (usually 3 or 4 entries).

Theorem 1: Our algorithm solves the mixed-type PLA problem optimally, spending amortized $O(1)$ time per data point in the stream. It outputs the partial optimal solution as early as possible. Its worst-case space complexity is $O(n)$.

V. EXPERIMENTS

A. Experimental setup

In this section, we report our experimental findings on a variety of real-world data sets. Like prior work [5], [7] on the PLA problem, we used the sea surface temperature data[3] which consists 210851 temperature readings with 10-minute intervals. In addition, we also picked 8 representative data sets from the UCR time series data archive[4]. These data sets exhibit different characteristics in terms of smoothness, stationarity, etc.

We compared four algorithms: (1) *Swing filter*, (2) *Slide filter*, (3) *Cont-PLA*, and (4) *Mixed-PLA*, where (1) is a heuristic algorithm introduced in [5] that generates a suboptimal disjoint PLA; (2) is the optimal disjoint PLA algorithm of O'Rourke [4] which was rediscovered in [5]; (3) is the optimal continuous PLA algorithm of Hakimi and Schmeichel [6]; and (4) is our algorithm. All experiments were conducted on a machine with 16G RAM and a 3.00GHz processor.

B. PLA size

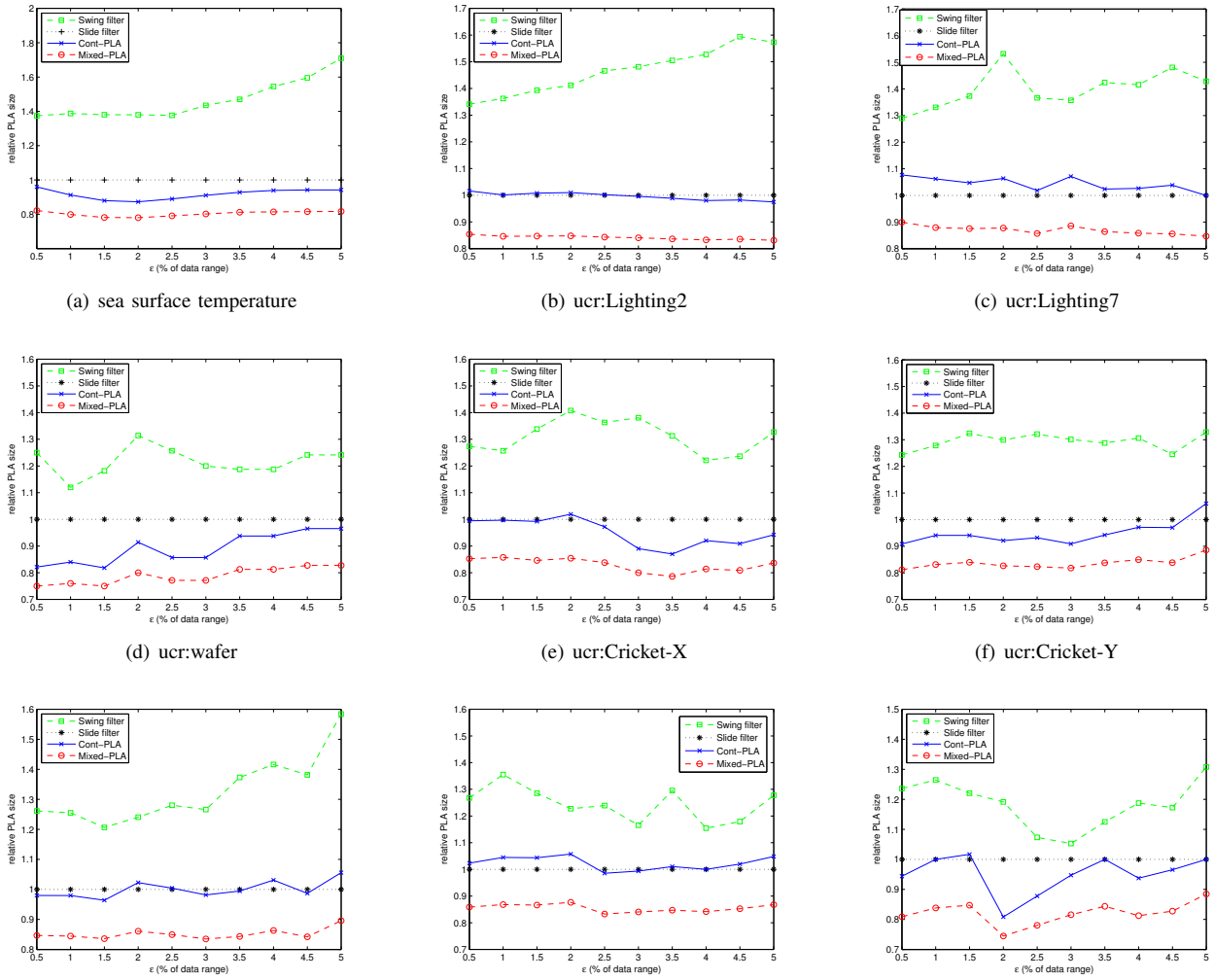
We first measure the size of the generated PLAs of these algorithms for a given ℓ_∞-error ε. We first found the maximum and minimum value in each data set and computed their difference. Then we set ε to be $0.5\%, 1\%, \ldots, 5\%$ of this difference and ran each of the 4 algorithms. The resulting PLA sizes for the 9 data sets are shown in Figure 10. For ease of comparison, we have normalized their sizes with that of Slide filter being 1.

From the plots, we see the following trends. First, Swing filter generates the largest PLA and is worse than the other algorithms by a large margin. Between Slide filter and Cont-PLA, the two algorithms that generate the optimal disjoint and continuous PLAs respectively, there is no clear winner, with perhaps Cont-PLA being better on more cases, and the relative advantage seems to depend on both the data set and the choice of ε. Finally, Mixed-PLA consistently beats Slide filter and Cont-PLA by roughly 15%.

In terms of ℓ_∞-error, all algorithms exactly attain the given ε as they all try to minimize the PLA size within the allowed ℓ_∞ constraint. We also looked at the ℓ_2-error of the PLAs returned by the 4 algorithms. Note that to be fair, we should compare PLAs of the same size, so we plot the results in the plane with ℓ_2-error and the PLA size as the two dimensions, and the results on two of the data sets are shown in Figure

[3] http://www.pmel.noaa.gov/tao

[4] http://www.cs.ucr.edu/~eamonn/time_series_data/



(a) sea surface temperature (b) ucr:Lighting2 (c) ucr:Lighting7

(d) ucr:wafer (e) ucr:Cricket-X (f) ucr:Cricket-Y

(g) ucr:Cricket-Z (h) ucr:FaceFour (i) ucr:Mote

Fig. 10. Normalized PLA size

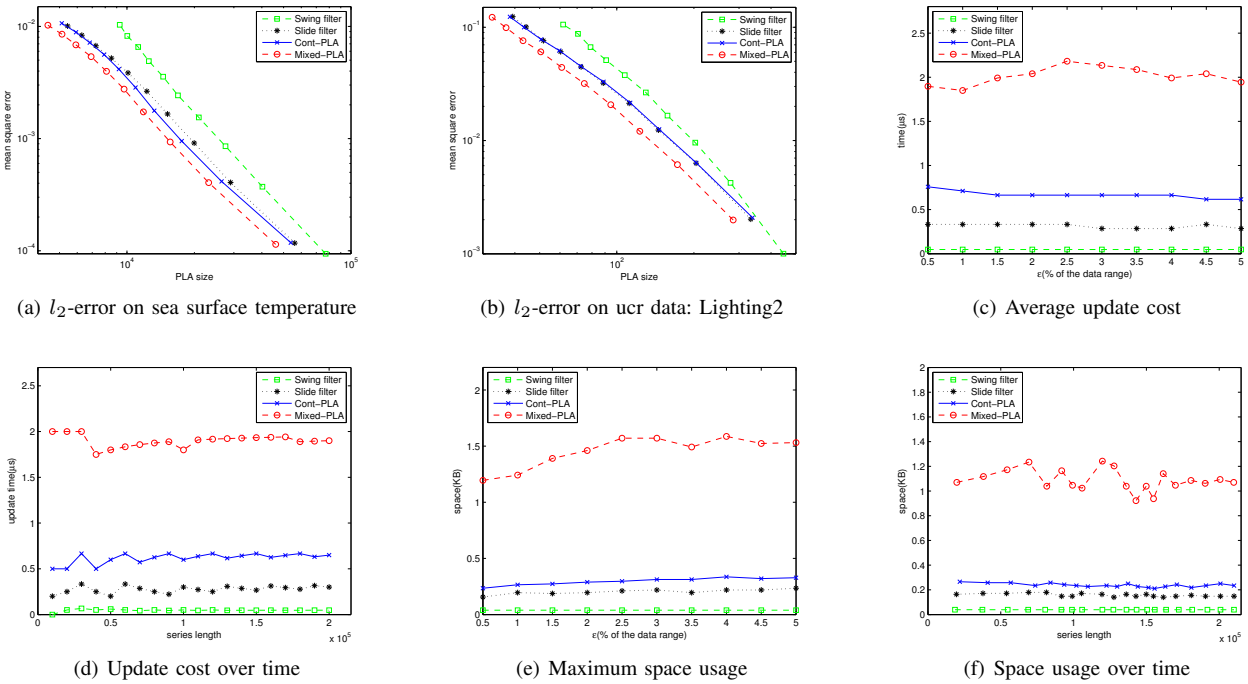
11(a) and 11(b) (results on other data sets are similar). Note that the points are not aligned since we cannot control the PLA size (we controlled the ℓ_∞-error ε). To our pleasant surprise, Mixed-PLA seems to yield the smallest ℓ_2 error, although none of these algorithms really takes ℓ_2-error into consideration.

C. Space and time

Next, we investigate the space and time costs of the 4 algorithms. The space and time costs of these algorithms are not sensitive to data characteristics, so we only report the results on the sea surface temperature data set, which is the largest among the 9. We first vary ε, and plot the average update cost per data record in Figure 11(c). Recall that theoretically, all algorithms have $O(1)$ amortized update time. In practice, the worst algorithm in terms of quality (i.e., PLA size) usually is also the fastest. The Mixed-PLA algorithm is slowest, due to the need of running up to 5 instances of the visible region algorithm. Anyway, all algorithms are quite efficient, with the average update time per data record below 2 microsecond. We also have checked the update cost over time (Figure 11(d)), and are assured that it is not affected by n, the length of the time series.

Although we do not have a good worst-case bound on the space usage, in reality the space usage is low for all algorithms. In Figure 11(e) we plot the maximum space usage for different values of ε. Again, algorithms with lower quality tend to use smaller space. Our algorithm uses the most space, but still it is just around 1KB. In Figure 11(f), we plot the space usage over time. Again, the space usage does not increase as n.

Although our algorithm uses more space and time than the previous algorithms, these are just one-time costs, while the benefits of a smaller PLA (with the same ℓ_∞-error and smaller ℓ_2-error) are long-term, reducing the cost of any downstream processing including time series analysis, clustering, indexing, similarity search, storage, etc. In addition, the space and time costs are still very low, with 1KB of working space and 2 microsecond of processing time, these costs are almost negligible compared to their long-term benefits.



(a) l_2-error on sea surface temperature

(b) l_2-error on ucr data: Lighting2

(c) Average update cost

(d) Update cost over time

(e) Maximum space usage

(f) Space usage over time

Fig. 11. ℓ_2-error, space and time cost

D. Output delay

Finally, we examine the effectiveness of the early-output technique, and see how much delay the algorithm has when returning the PLA. We have tested all 9 data sets, with $\varepsilon = 5\%$, and recorded maximum delay when the segments of PLA are outputted over the entire time series. Table I shows that the early-output technique is highly effective, with the maximum delay being just 3 or 4 segments.

Data set	a	b	c	d	e	f	g	h	i
Delay	4	3	2	3	3	3	3	3	2

TABLE I
MAXIMUM OUTPUT DELAY

VI. Lower Bound

Our algorithm has very small space overhead as shown in the experiments, but its worst-case space complexity is still $O(n)$. In this section, we show that this is inherently unavoidable, by proving corresponding a lower bound. In addition, we show that this is not specific to the mixed-type PLA problem. It applies to the continuous and disjoint PLA problem as well. In fact, we consider the following simple problem, which is a special case of the mixed-type, continuous, or the disjoint PLA problem, so lower bounds proved for this problem also hold for the more general problems.

Problem 3 (One-line approximation): Given a time series stream $S = ((x_1, y_1), \ldots, (x_n, y_n))$ where $x_1 < x_2 < \cdots < x_n$, and an error ε, the goal is to find a linear function f such that $|f(x_i) - y_i| \leq \varepsilon$ for all i. Return "no solution" if such an f does not exist.

To get a space lower bound for the one-line approximation problem, we use a reduction from the INDEX problem in communication complexity. In the INDEX problem, Alice has an array $A[\cdot]$ of n bits, and Bob has an index j. Bob needs to know the value of $A[j]$, and Alice can only send one message to Bob. It is well known that the INDEX problem has communication complexity $\Omega(n)$ [23].

Theorem 2: The worst-case space complexity of the one-line approximation problem is $\Omega(n)$.

Proof: (sketch) Let $\mathcal{A}$ be any algorithm for the one-line approximation problem. The reduction from the INDEX problem to the one-line approximation problem consists of the following steps.

1) First, Alice constructs an instance of the one-line approximation problem $S = (q_1, \ldots, q_n)$ from her input array $A[\cdot]$. More precisely, for $i = 1, \ldots, n$, she creates a point $q_i = (i, y_i)$, where
$$y_i = \begin{cases} i(i+1)/2 + \delta & \text{if } A[i] = 1 \\ i(i+1)/2 & \text{if } A[i] = 0, \end{cases}$$
where $\delta > 0$ is a sufficiently small constant.
2) Alice runs algorithm $\mathcal{A}$ on S with $\varepsilon = n^2$. When $\mathcal{S}$ has processed all the n points, Alice sends the memory content of $\mathcal{A}$ to Bob.
3) Let $p_* = (t_*, m_*)$ where $t_* = j + 1/2$ and $m_* = \varepsilon - j^2/2 + \delta/2$. Bob creates two data points $q_{n+1} = (n+1, n^2 + m_* + (n+1)t_*)$ and $q_{n+2} = (n+2, -n^2 + m_* + (n+2)t_*)$.
4) After receiving the memory content of $\mathcal{A}$ from Alice, Bob continues the execution of $\mathcal{A}$ to take two more

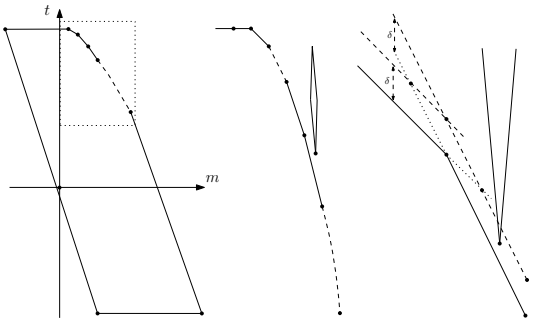


Fig. 12. The reduction.

points q_{n+1} and q_{n+2}. Then Bob asks $\mathcal{A}$ whether a solution to the one-line approximation problem exists. If the answer is yes, Bob declares $A[j] = 1$, otherwise $A[j] = 0$.

It is obvious that in this reduction, the communication cost is a lower bound on the memory usage of $\mathcal{A}$, so the space complexity of $\mathcal{A}$ is $\Omega(n)$.

Below, we show the correctness of the reduction. Due to space constraint, we only sketch the high-level ideas and leave out the tedious calculations.

We write any linear function in the form $f = -tx + m$, and consider the parameter space (t, m). The requirement of f approximate a point (x_i, y_i) within ε error translates to the constraint $y_i - \varepsilon \leq -tx_i + m \leq y_i + \varepsilon$, which corresponds to a slab (i.e., the region bounded by two parallel lines) in the parameter space. Thus, the one-line approximation problem on the $n+2$ data points has a solution iff the $n+2$ corresponding slabs have a nonempty intersection in the parameter space.

The intersection of the slabs corresponding to $q_1, \ldots, q_n$ will look like the left figure in Figure 12. We call this region R. The two slabs corresponding to q_{n+1} and q_{n_2} have an intersection that is a skinny parallelogram whose bottom vertex is exactly p_*, as shown in the middle figure of Figure 12. The position of the parallelogram is determined by the value of j held by Bob. We can set the value of δ such that, if $A[j] = 0$, the parallelogram will not intersect R, but if $A[j] = 1$, it will intersect R (see the dashed lines in the right figure of Figure 12). Thus, deciding whether all the $n + 2$ slabs have a nonempty intersection will also tell us whether $A[j] = 1$ or 0. ■

As the one-line approximation problem is a special case of the mixed-type, continuous, and disjoint PLA problem, the same lower bounds also hold for these problems.

Corollary 1: The worst-case space complexity of the mixed-type, continuous, and disjoint PLA problem is $\Omega(n)$.

VII. Conclusion

In this paper, we have revisited the classical PLA problem over streaming time series data, and presented an online algorithm that truly optimizes the representation size of the PLA. The algorithm has $O(1)$ update cost per data record, and uses small working space. The worst-case space complexity is $O(n)$, but the space usage on all real-world data sets is very low. We have also complemented this by a space lower bound of $\Omega(n)$ showing that no algorithm can always guarantee small working space.

References

[1] S. H. Cameron, "Piece-wise linear approximations," DTIC Document, Tech. Rep., 1966.

[2] C. Aggarwal, "On abnormality detection in spuriously populated data streams," in *Proc. SIAM International Conference on Data Mining*, 2005.

[3] I. Assent, P. Kranen, C. Baldauf, and T. Seidl, "Anyout: Anytime outlier detection on streaming data," in *Proc. International Conference on Database Systems for Advanced Applications*, 2012.

[4] J. O'Rourke, "An on-line algorithm for fitting straight lines between data ranges," *Communications of the ACM*, vol. 24, no. 9, 1981.

[5] H. Elmeleegy, A. K. Elmagarmid, E. Cecchet, W. G. Aref, and W. Zwaenepoel, "Online piece-wise linear approximation of numerical streams with precision guarantees," in *Proc. International Conference on Very Large Data Bases*, 2009.

[6] S. L. Hakimi and E. F. Schmeichel, "Fitting polygonal functions to a set of points in the plane," *CVGIP: Graph. Models Image Process.*, vol. 53, no. 2, pp. 132–136, 1991. [Online]. Available: http://dx.doi.org/10.1016/1049-9652(91)90056-P

[7] I. Lazaridis and S. Mehrotra, "Capturing sensor-generated time series with quality guarantees," in *Proc. IEEE International Conference on Data Engineering*, 2003.

[8] D. Wang, N. Huang, H. Chao, and R. Lee, "Plane sweep algorithms for the polynomial approximation problems with applications," in *Proc. International Symposium on Algorithms and Computation*, 1993.

[9] M. Goodrich, "Efficient piecewise-linear function approximation using the uniform metric," in *Proc. Annual Symposium on Computational Geometry*, 1994.

[10] D. Z. Chen and H. Wang, "Approximating points by a piecewise linear function," *Algorithmica*, vol. 66, pp. 682–713, 2013.

[11] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel, "Optimal histograms with quality guarantees," in *Proc. International Conference on Very Large Data Bases*, 1998.

[12] S. Guha, N. Koudas, and K. Shim, "Approximation and streaming algorithms for histogram construction problems," *ACM Transactions on Database Systems*, vol. 31, no. 1, pp. 396–438, 2006.

[13] B. Aronov, T. Asano, N. Katoh, K. Mehlhorn, and T. Tokuyama, "Polyline fitting of planar points under min-sum criteria," *International Journal of Computational Geometry & Applications*, vol. 16, no. 2-3, pp. 97–116, 2006.

[14] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proc. IEEE International Conference on Data Mining*, 2001.

[15] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos, and W. Truppel, "Online amnesic approximation of streaming time series," in *Proc. IEEE International Conference on Data Engineering*, 2004.

[16] D. Rafiei, "On similarity-based queries for time series data," in *Proc. IEEE International Conference on Data Engineering*, 1999.

[17] B. Yi and C. Faloutsos, "Fast time sequence indexing for arbitrary lp-norms," in *Proc. International Conference on Very Large Data Bases*, 2000.

[18] I. Popivanov and R. J. Miller, "Similarity search over time series data using wavelets," in *Proc. IEEE International Conference on Data Engineering*, 2002.

[19] K. Chakrabarti, E. J. Keogh, S. Mehrotra, and M. J. Pazzani, "Locally adaptive dimensionality reduction for indexing large time series databases," *ACM Transactions on Database Systems*, vol. 27, no. 2, pp. 188–228, 2002.

[20] Y. Cai and R. Ng, "Indexing spatio-temporal trajectories with chebyshev polynomials," in *Proc. ACM SIGMOD International Conference on Management of Data*, 2004.

[21] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. X. Yu, "Indexable PLA for efficient similarity search," in *Proc. International Conference on Very Large Data Bases*, 2007.

[22] H. Imai and M. Iri, "An optimal algorithm for approximating a piecewise linear function," *Information Processing Letters*, vol. 9, no. 3, pp. 159–162, 1986. [Online]. Available: http://ci.nii.ac.jp/naid/110002673423/en/

[23] E. Kushilevitz and N. Nisan, *Communication Complexity*. Cambridge University Press, 1997.