International Journal of Machine Learning and Cybernetics (2020) 11:1483–1499
https://doi.org/10.1007/s13042-019-01052-y

ORIGINAL ARTICLE

An online PLA algorithm with maximum error bound for generating optimal mixed-segments

Huanyu Zhao[1,2,3] · Tongliang Li[1,2] · Genlang Chen[3] · Zhaowei Dong[4] · Mengya Bo[5] · Chaoyi Pang[3]

Received: 26 June 2019 / Accepted: 9 December 2019 / Published online: 20 December 2019

Abstract
Piecewise Linear Approximation (PLA) is an effective method used to represent and compress a time series. It divides a time series into a number of segments, each of which is approximated by a straight line. This division and approximation is done under a metric enforcing optimized storage and compressed data quality criteria. In this article, we propose a new optimal linear-time PLA algorithm (SemiMixedAlg) for generating a set of *mixed-connected* (continue and disconnected segments) with guaranteed maximum error and minimized storage. An efficient "k-length" strategy is designed to determine the location of *mixed* segments in order to minimize the storage of mixed-connected segments. Our experiments on 43 real-world data sets show that SemiMixedAlg achieves exactly the same results as that of PipeMixedAlg (Luo et al. in Piecewise linear approximation of streaming time series data with max-error guarantees. In: IEEE international conference on data engineering, pp 173—184); the only state of the art algorithm, but with much lower time and memory costs.

Keywords Time series · Piecewise linear approximation · Maximum error bound

1 Introduction

A time series is a sequence of data points where each data point is associated with a time stamp. This data is usually accumulated in the form of data streams over time, appearing in many real applications [1–3]. Because of the unbounded size of a data stream, storing the raw collection of such data not only takes up much disk space, but also makes the online analysis more difficult. Therefore, it is necessary to explore efficient compression methods to save data storage, alleviate data transmission. More importantly, it may accelerate data analyzing time, and improve the accuracy of analysis [4, 5]. There exists many strategies for this task, such as Fourier Transforms [6], Discrete Wavelet Transform [7], Symbolic Mapping [8], Piecewise Linear Approximation (PLA) [9–11] and Piecewise Aggregate Approximation [12], etc.

The basic idea of Piecewise Linear Approximation is to divide a time series into a number of fragments, each of which is approximated by a straight line under different criteria. Storing the starting and ending points of such lines can represent the corresponding fragments so that the original data stream can be effectively compressed. Due to the smoothness and intuition of this representation, PLA has been extensively studied for decades and successfully used in real applications. The PLA problem focused in this paper can be formulated as follows.

The Problem. Let $P = (p_1, \ldots, p_n)$ be a time series and δ be an error bound. We use $S_i = (p_{s_i}, p_{s_i+1}, \ldots, p_{e_i})$ to denote the *fragment* of P on time slot $[t_{s_i}, t_{e_i}]$(or $[s_i, e_i]$) $(s_i < e_i \leq n)$.

The problem L_∞-bound PLA is to divide P into k fragments $S_1, S_2, \ldots, S_k$; where each fragment $S_i = (p_{s_i}, p_{s_i+1}, \ldots, p_{e_i})$ $(i \in \{1, 2, \ldots, k\}$, $p_{s_1} = p_1$, $p_{e_i+1} = p_{s_{i+1}}$ and $p_{e_k} = p_n)$ can be approximated by a linear function $f_i(t)$ satisfying the pre-defined error bound of

✉ Huanyu Zhao
zhaohuanyu@163.com

✉ Genlang Chen
cgl@nit.zju.edu.cn

1 The Institute of Applied Mathematics, Hebei Academy of Sciences, Shijiazhuang, China

2 Hebei Authentication Technology Engineering Research Center, Shijiazhuang, China

3 The School of Computer and Data Engineering, Zhejiang University (NIT), Ningbo, China

4 Hebei University of Economics and Business, Shijiazhuang, China

5 The School of Information Science and Technology, Shijiazhuang Tiedao University, Shijiazhuang, China

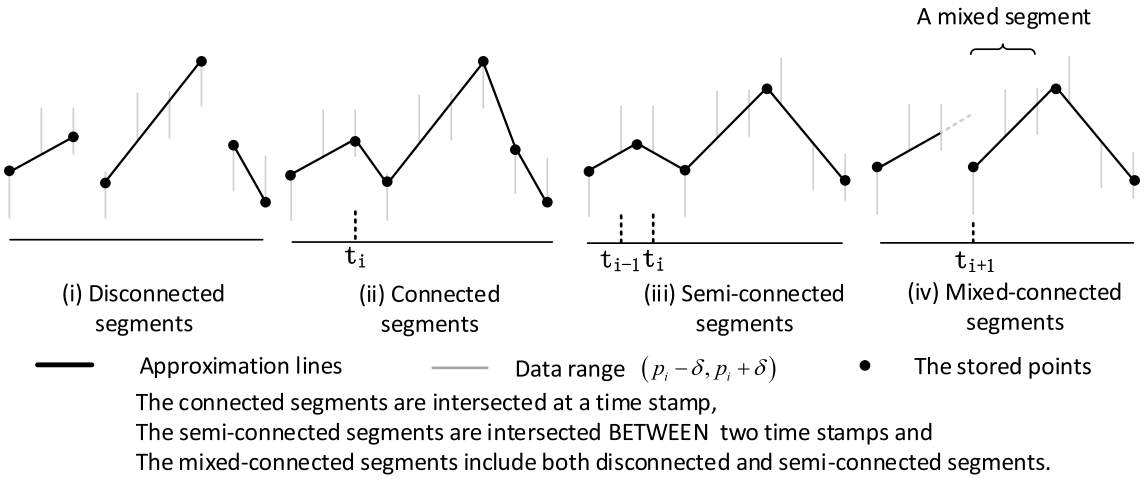


Fig. 1 The disconnected, connected, semi-connected and mixed-connected segments

$$|f_i(t) - p_t| \leq \delta \tag{1.1}$$

for $t \in \{s_i, s_i + 1, \ldots, e_i\}$. A linear function that satisfies Eq. 1.1 for the data points of S_i is called a (feasible) *representation* or *segment*[1] of S_i. We call it *connected* L_∞*-bound PLA* (or simply, *connected*) if there exist connected linear representations, i.e., $f_i(e_i) = f_{i+1}(s_{i+1})$ holds for each i $(1 \leq i \leq k-1)$ (Fig. 1ii). Otherwise, it is called *disconnected* (Fig. 1i). It is called *semi-connected* L_∞*-bound PLA* (or simply, *semi-connected*) if $f_i(x_i) = f_{i+1}(x_i)$ where $e_i \leq x_i \leq s_{i+1}$ holds for each i $(1 \leq i \leq k-1)$ (Fig. 1iii). Furthermore, it is called *mixed-connected* L_∞*-bound PLA* (or simply, *mixed-connected*) if semi-connected and disconnected representations coexist in $S_1, S_2, \ldots, S_k$ (Fig. 1iv), where a disconnected segment is referred to as a *mixed* segment. Uniformly, the segmentation result is called *optimal* if the storage[2] of constructed segments (or fragments) is minimum.

Why mixed-connected PLA?. The problems of optimal disconnected, semi-connected and mixed-connected L_∞-bound PLA have been solved by the linear-time algorithms of DisConnAlg, SemiConnAlg and PipeMixedAlg proposed in [13, 14] and [15] respectively. To our knowledge, there is no linear algorithm to solve the problem of optimal connected L_∞-bounded PLA. It is very difficult to construct optimal connected segments in linear-time cost. As a varied form, *semi-connected* generalizes *connected* segments by extending the intersection time stamp of adjacent line representations from integer to non-integer. The purpose of this generalization is to facilitate constructing the optimal continued (coexists semi-connected and connected) segments in lower time costs. The optimal mixed-connected problem first proposed in [15] is to further reduce the storage of output segments (semi-connected or disconnected). As showed in Fig. 1, the storage of disconnected, connected and semi-connected segments is $2 * 6 = 12, 2 * 6 = 12$ and $2 * 5 = 10$ respectively; while the mixed-connected one is $2 * 4 + 1 = 9$, where one extra storage unit is from the time-stamp t_{i+1} used to mark the disconnected location (i.e., t_{i+1} and two data values at t_{i+1}).

With the same error bound, we can achieve optimal disconnected, semi-connected and mixed-connected L_∞-bound PLA. Denoted its storages as sizeDisconn, sizeSemi and sizeMix respectively. We can derive the inequation 1.2 from the conclusions of [14, 15].

$$sizeMix \leq sizeSemi \leq sizeDisconn \tag{1.2}$$

We can use total square errors (L_2-error) to measure representation quality of PLA. To make it fair, we adjust the input error bounds to make optimal disconnected, semi-connected and mixed-connected segments have the same storage, and denote the L_2-errors of constructed segments as errorDis, errorSemi and errorMixed respectively. Having been verified in [15], the inequation 1.3 holds.

$$errorMixed \leq errorSemi \leq errorDis \tag{1.3}$$

Differences from PipeMixedAlg. Along with PipeMixedAlg in [15], another linear-time algorithm (Denoted as PipeConnAlg) can be derived for constructing optimal semi-connected segments. It is surprising and interesting that the output results of PipeConnAlg and SemiConnAlg are exactly the same [14]. But both time and memory costs of SemiConnAlg are lower than that of PipeConnAlg. The

[1] We will use S_i to denote both the i-th fragment and the i-th segment in the context when there is no confusion.

[2] Here, the unit of storage is a double or int type. For example, any point includes a time stamp and value, so the storage is 2.

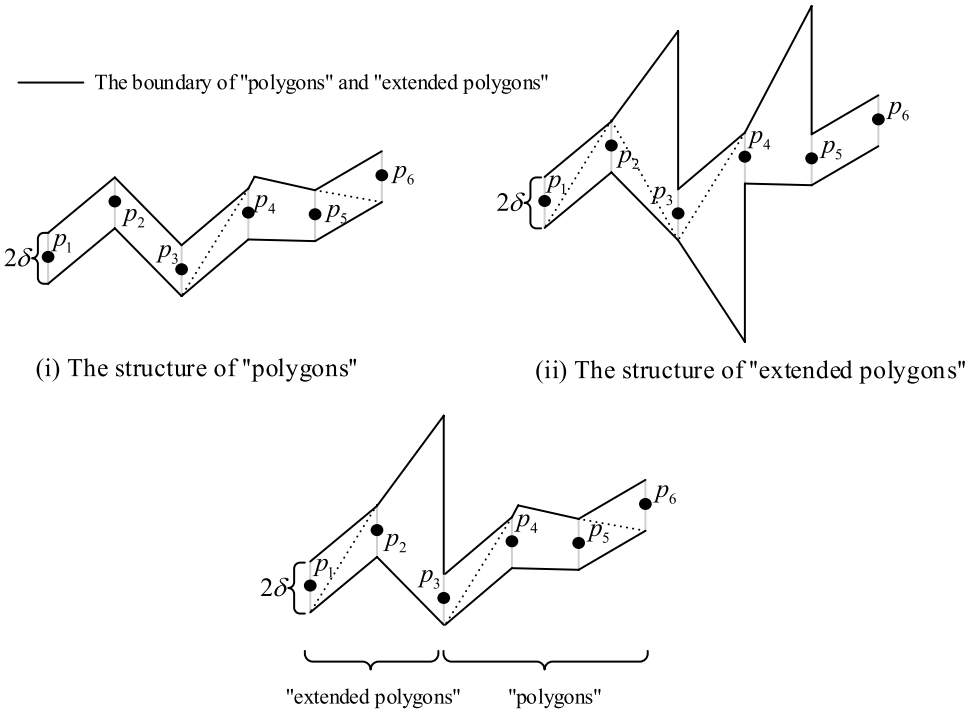


Fig. 2 The structures of "polygons", "extended polygons" and "mixed polygons"

performance differences are due to different mechanisms behind algorithms.

D1: PipeConnAlg extends from [16, 17] and works on a transformed space; While SemiConnAlg extends from [13, 18] and worked in time domain space.

D2: PipeConnAlg is based on the idea of searching the shortest paths in a pre-constructed pipeline, where the shortest path represents the set of segments with the least number segments or lowest storage; While SemiConnAlg is designed by searching the local optimal results to achieve global optimisation.

D3: PipeConnAlg is a *dynamic programming* (DP) algorithm that embraces a clever "early-output technique" to quickly extract part of the optimal solution from the DP to make the algorithm online. SemiConnAlg employs a series of theoretically proven deductions as shortcuts to a DP based computing.

One remaining question in [14] is how to solve the optimal mixed-connected L_∞-bound PLA by extending the idea of SemiConnAlg in time domain space. The key subproblem is how to efficiently disconnect the semi-connected segments resulted from SemiConnAlg, with acceptable time and memory costs. For solving this problem, an effective "k-length" strategy derived from Lemmas 4.1, 4.2 is designed to locate the starting time-stamp of mixed segments. Its methodology is different from the mechanism of extending PipeConnAlg to PipeMixedAlg:

M1: PipeConnAlg is based on "polygons" (Fig. 2i), special pipelines which include all feasible semi-connected representation lines. PipeMixedAlg generalizes "polygons" to "extended polygons" (Fig. 2i) to construct mixed-connected representations. Whereas, SemiMixedAlg and SemiConnAlg are based on "Convex hulls" (Fig. 4) defined in time domain space.

M2: PipeMixedAlg constructs segments by comparing the ending points of 5 "open visible regions", each of which represents a set of mixed-connected segments. If disconnecting a semi-connected segment results in the corresponding "open visible regions" to reach the point with the largest time stamp, then a mixed segment is constructed. Because of no theoretical property discovered to determine the location of a mixed segment, 5 "open visible regions" are used to check the starting point of a mixed segment. In fact, if the beginning is known in advance, the "extended polygons" can be further generalized to "mixed polygons" for reducing the search space of mixed-connected segments. As seen in Fig. 2iii, the fragments $(p_1, p_2, \ldots, p_6)$ are divided into mixed-connected segments of (p_1, p_2), (p_3, p_4) and (p_5, p_6) which are expressed as dotted lines. (p_3, p_4) is a mixed segment disconnected at the time stamp t_3. If we

know that t_3 is the starting position of a mixed segment beforehand, (p_3, p_4) and (p_5, p_6) can be determined without the need of constructing "extended polygons".

On the other hand, SemiMixedAlg locates possible mixed segments by checking the "status" of semi-connected segments (see Lemma 4.1), a property which can be directly accessed from the results of SemiConnAlg. Furthermore, a mixed segment is determined through trial and error using a set of bounded number of semi-connected segments (see Lemma 4.2).

M3: Due to always maintaining 5 "open visible regions", the time and memory costs of PipeMixedAlg are about 5 times of that PipeConnAlg. Since SemiMixedAlg only needs to store the "Convex hulls" of current constructing segment, the memory cost of SemiMixedAlg is close to that of SemiConnAlg in many situations. Again, because of the extra costs of reconstructing semi-connected segments in "k-length" strategy, the time cost of SemiMixedAlg is twice that of SemiConnAlg in the worst case.

Contributions. In this article, extending from the idea on disconnected and semi-connected segments of [13, 14], we design a new linear-time L_∞-PLA algorithm (termed as SemiMixedAlg) for the construction of optimal mix-connected segments. The contributions can be summarized as follows:

1. Design and implement SemiMixedAlg. Unlike PipeMixedAlg based on dynamic programming, SemiMixedAlg employs a series of theoretically proven deductions as shortcuts to a DP based computing. A number of proven properties that prohibit excessive costs and ensures the correctness of SemiMixedAlg have been obtained. In this regard, the deduction properties are about how to efficiently construct and locate a mixed segment. These properties have been presented within this article. According to these properties, it is concluded that SemiMixedAlg is a linear-time algorithm and generates optimal mixed-connected segments (Theorem 4.1).

2. Extensive experiments are conducted on 43 arbitrary data sets selected from a public domain [19] to verify the theoretical conclusions. Although SemiMixedAlg and PipeMixedAlg generate the same outputs on all data sets, SemiMixedAlg is about 5 times more efficient than that of PipeMixedAlg in both time and memory costs on all tested data sets. More importantly, as indicated in Fig. 12i, the run time of SemiMixedAlg scales significantly better than that of PipeMixedAlg on increased data sizes. This implies that SemiMixedAlg is much more preferable for processing large data sets.

The rest of the paper is organized as follows: Sect. 2 gives a comprehensive review of related work on PLA; Sect. 3 gives a brief introduction of DisConnAlg and SemiConnAlg; Sect. 4 explains the ideas, methodology, steps, pseudo code and properties of SemiMixedAlg; Sect. 5 shows the experimental results, comparing the performances of SemiMixedAlg and PipeMixedAlg; Sect. 6 concludes this article.

2 Related work

Most of the early research on PLA is focused on the holistic error [20–22]. This measurement norm was not suitable for online algorithms as there was no acknowledgment for continuous data inputs. More importantly, the constructed representation segments by the early approaches are not error-guaranteed on each individual data point [7, 23, 24]. We designed a special example to show the advantages of L_∞-bound PLA, which may reduce the complexity of some information models [25, 26].

Example 1 Let fragment P have 20 data points where

$$P = \{(1,72),(2,72),(3,70),(4,70),(5,70),(6,70),(7,69),(8,69),$$
$$(9,68),(10,68),(11,67),(12,67),(13,68),(14,68),(15,68),(16,68),$$
$$(17,51),(18,36),(19,24),(20,24)\}.$$

We adopt the algorithm SWAB [22] and SemiConnAlg [14] to compress this fragment. We set this error to be 0.45 and have the following output lines:

5 disconnected segments by SWAB:

$(1,72.2)(4,69.8)$;$(5,70.1667)(12,66.8333)$;$(13,68)(16,68)$;
$(17,51)(18,36)$;$(19,24)(20,24)$.

8 semi-connected segments by SemiConnAlg:

$(1,72.45),(2,71.55),(3,70.45),(6,69.55),(11.1923,67.3692),(16,67.85),$
$(17.9057,36.5972),(19,24.45),(20,24)$.

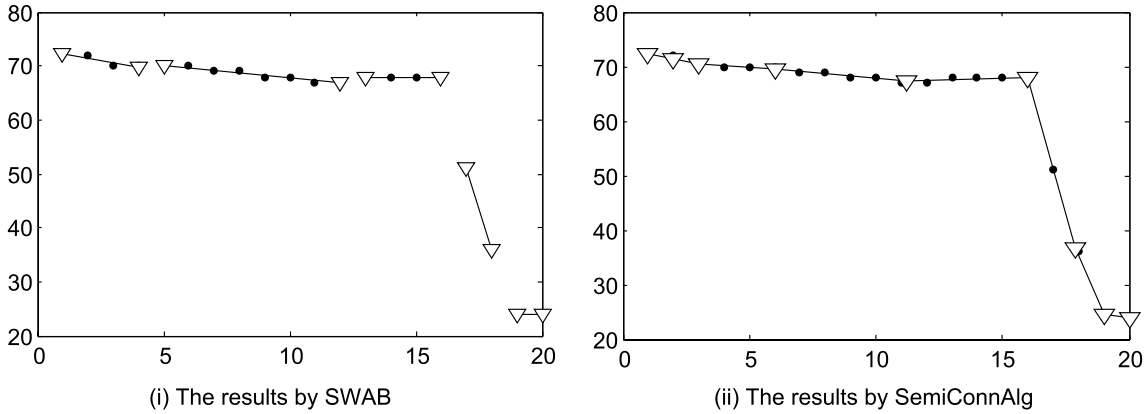


Fig. 3 The results of Example 1 by SWAB and SemiConnAlg

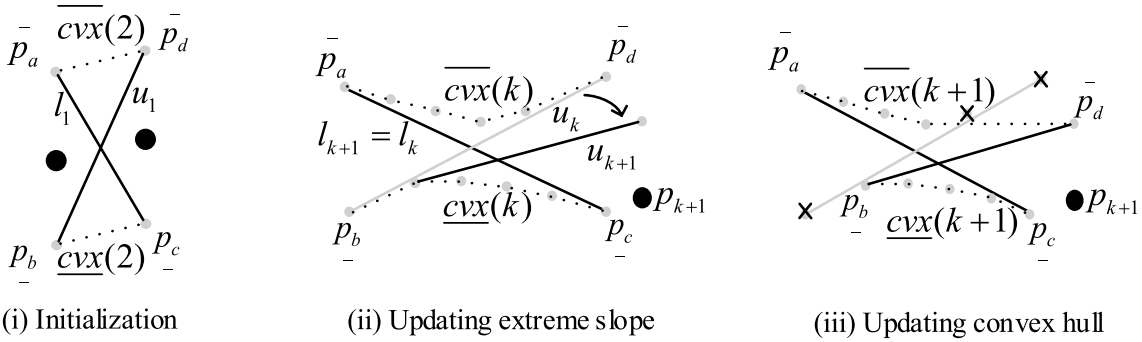
SWAB uses 10 points to represent the original dataset while SemiConnAlg uses 9 points (see Fig. 3), indicating that the compression performance of SemiConnAlg is better than that of SWAB.

Moreover, we can reconstruct the original data in lossless from the semi-connected representation by picking up the closest integers on the constructed semi-connected segments. For example, the reconstructed data at time stamp (1, 2, 3, 4) are (72.2, 71.4, 70.6, 69.8) and (72.45, 71.55, 70.45, 70.15) by the first line ((1,72.2), (4,69.8)) of SWAB and three lines ((1,72.45), (2,71.55)), ((2,71.55), (3,70.45)), ((3,70.45), (6,69.55)) of SemiConnAlg respectively. By using the nearest integer function, the approximation (72.45, 71.55, 70.45, 70.15) can simply convert to (72, 72, 70, 70). This can be hard to be realized from (72.2, 71.4, 70.6, 69.8) generated by SWAB as the approximation error on the segments is measured in L_2 norm.

For solving the L_∞-PLA problem, there are three kinds of algorithms minimizing the storages on different representative segments (refer Fig. 1):

- Disconnected Segments. Its goal is to find the minimum number of disconnected representative segments for a time series. In 1981, O'Rourke proposed the first linear-time algorithm (denoted as *DisOptPLA*) [27] where each expected segment is generated by incrementally maintaining a convex polygon in a transformed space. Different from O'Rourke's ideas, Elmeleegy et al. [28] gave another optimal strategy in time domain space. Xie et al. [13] improved the method of [28] and provided a linear-time algorithm (termed as DisConnAlg) with good practical performances.
- Connected Segments. The goal is to construct the minimum number of connected representative segments. Motivated from the work on disconnected segments [27] and [17], Hakimi et al. [16] gave an off-line method by searching the shortest path in a constructed "pipeline". In 2015, Luo et al. [15] designed an efficient linear time algorithm (termed as PipeConnAlg) based on dynamic programming to make the idea of [16] to be online by filtering pipelines. Recently, Zhao et al. [14] gave a "shrinking-extension" based linear time algorithm (termed as SemiConnAlg) through deeply analyzing the relationship of adjacent disconnected segments generated by DisConnAlg of [13]. This algorithm is very efficient in practice and outputs the optimal semi-connected segments which are exactly the same to that of [15].
- Mixed-connected Segments. The goal is to further reduce the storage on representative segments by employing both disconnected and connected segments. Luo et al. [15] proposed the "mixed-connection" concept, and developed an algorithm (termed as PipeMixedAlg) that achieved smaller output storage than that of simply using disconnected or semi-connected segmentations. PipeMixedAlg smartly converts PipeConnAlg to construct mixed-connected segments under the motivation expressed in Fig. 8 of Sect. 4.2.

In addition to the above optimal methods, there are many heuristic L_∞-PLA algorithms [9, 13, 28, 29] for achieving less storage with connected segments. For example, Liu et al. [9] propose an FSW (Feasible Space Window) algorithm that uses the Feasible Space Window method to construct connected segments from a fixed initial point. In [28], the connected version algorithm is very similar with FSW. At the end of [13], they also indicate that the idea of DisConnAlg can be used to construct connected line segments through constructing the next segment from the feasible space of the last data point of the previous segment.



(i) Initialization (ii) Updating extreme slope (iii) Updating convex hull

Fig. 4 The process of updating extreme lines and convex hulls

3 Preliminary

In this section, we describe the methodologies of DisConnAlg and SemiConnAlg in order to illustrate SemiMixedAlg clearly.

3.1 DisConnAlg

Proposed in [13], DisConnAlg is an optimal linear time algorithm for constructing disconnected segments for time series. To minimize the number of generated segments, it ensures that each segment approximates the maximum number of data points. This is done by incrementally adjusting the range of all feasible lines in the process of adding consecutive points. Whenever no feasible lines can approximate the new point within the error bound, the construction process is stopped and a new line segment begins.

In the process of generating a segment, two extreme lines which represent the bounds of all feasible lines are iteratively updated as new data is introduced. To reduce the computation cost on updating extreme lines, two "convex hulls" are incrementally maintained to avoid the extensive search cost.

More specifically, suppose that $S = (p_1, p_2, \ldots, p_n)$ is a fragment constructed from DisConnAlg and let $u[1, k+1]$ and $l[1, k+1]$ be the slopes (gradients) of the upper and lower extreme lines when p_k is approximated for $k \in \{1, \ldots, n-1\}$. Then, from [13],

$$\begin{cases} l[1,k+1] &= \max\limits_{1\le i<j\le k} \frac{(y_j-\delta)-(y_i+\delta)}{(t_j-t_i)} \\ &= \frac{(y_c-\delta)-(y_a+\delta)}{(t_c-t_a)}, \\ u[1,k+1] &= \min\limits_{1\le i<j\le k} \frac{(y_j+\delta)-(y_i-\delta)}{(t_j-t_i)} \\ &= \frac{(y_d+\delta)-(y_b-\delta)}{(t_d-t_b)}, \end{cases} \tag{3.1}$$

where the extreme points p_a, p_b, p_c and p_d are defined by

$$\begin{cases} \{a,c\} &= arg \max\limits_{1\le i<j\le k} \frac{(y_j-\delta)-(y_i+\delta)}{(t_j-t_i)}, (a<c), \\ \{b,d\} &= arg \min\limits_{1\le i<j\le k} \frac{(y_j+\delta)-(y_i-\delta)}{(t_j-t_i)}, (b<d). \end{cases} \tag{3.2}$$

From Eq. 3.1, we can derive the following formula.

$$\begin{cases} l[1,k+1] &= \max\limits_{1\le i<j\le k+1} \{\frac{(y_j-\delta)-(y_i+\delta)}{(t_j-t_i)}, l[1,k]\}, \\ u[1,k+1] &= \min\limits_{1\le i<j\le k+1} \{\frac{(y_j+\delta)-(y_i-\delta)}{(t_j-t_i)}, u[1,k]\}. \end{cases} \tag{3.3}$$

According to the fact (Lemma 1 of [13]) that all data range $[y_i - \delta, y_i + \delta](1 \le i \le k)$ are intersected by $line(\overline{p_a}, \underline{p_c})$ and $line(\underline{p_b}, \overline{p_d})$, Eq. 3.3 can be simplified to:

$$\begin{cases} l[1,k+1] &= \max\limits_{a\le i\le d} \{\frac{(y_{k+1}-\delta)-(y_i+\delta)}{(t_{k+1}-t_i)}, l[1,k]\}, \\ u[1,k+1] &= \min\limits_{b\le i\le c} \{\frac{(y_{k+1}+\delta)-(y_i-\delta)}{(t_{k+1}-t_i)}, u[1,k]\}, \end{cases} \tag{3.4}$$

Again, from Lemma 2 of [13], i.e., the computation of extreme lines can be constrained to the points on the convex hulls, Eq. 3.4 can be rewritten into:

$$\begin{cases} l[1,k+1] &= \max\limits_{\overline{p_i}\in\overline{cvx}(k)} \{\frac{(y_{k+1}-\delta)-(y_i+\delta)}{(t_{k+1}-t_i)}, l[1,k]\}, \\ u[1,k+1] &= \min\limits_{\underline{p_i}\in\underline{cvx}(k)} \{\frac{(y_{k+1}+\delta)-(y_i-\delta)}{(t_{k+1}-t_i)}, u[1,k]\}, \end{cases} \tag{3.5}$$

where $\overline{cvx}(k)$ and $\underline{cvx}(k)$ are the set of convex points of $\{\overline{p_a}, \ldots, \overline{p_d}\}$ and $\{\underline{p_b}, \ldots, \underline{p_c}\}$ respectively. The points p_a, p_b, p_c and p_d are determined by Eq. 3.2.

To update the extreme lines for a new data point, these convex hulls need to be maintained and updated. Updating a convex hull may require deleting some points to incorporate the newly introduced points. Since the number of points in the convex hulls can be significantly smaller than the total number of data points in the relevant intervals, the convex

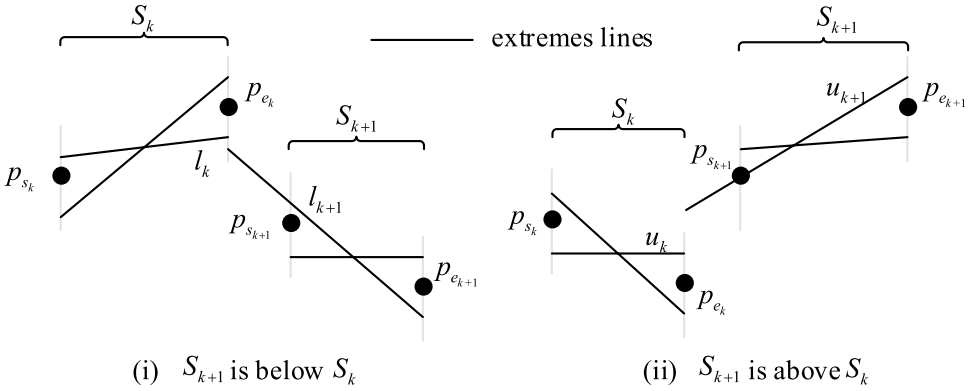


Fig. 5 Two relationships between S_k and S_{k+1}

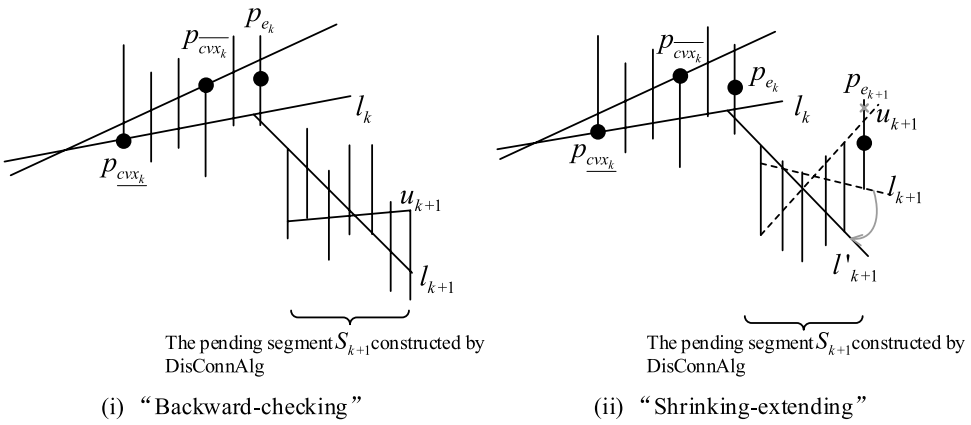


Fig. 6 "Backward-checking" and "Shrinking-extending" strategies

hulls can be efficiently updated using the triangle check technique of [30] through recursively examining the three most recent consecutive points and then moving backwards. If the middle point is above or on the line formed by the other two points, then the middle point is removed. This process is continued for the remaining three most recent consecutive points until the middle point is no longer being removed.

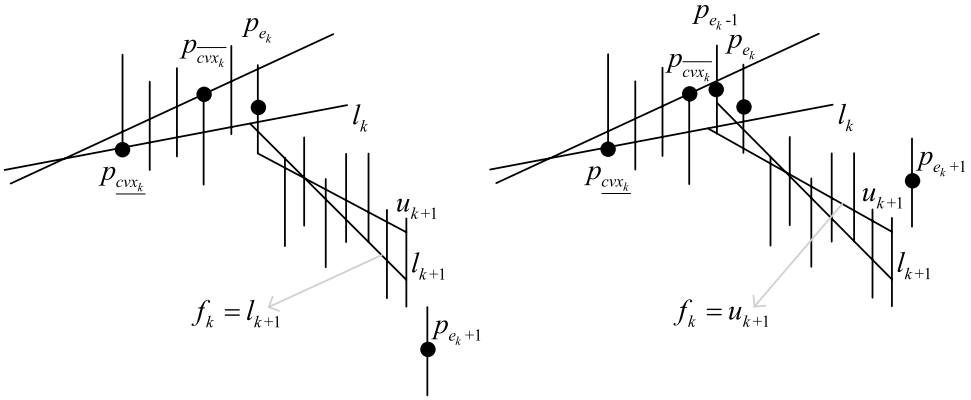
Figure 4 illustrates the process of updating the extreme lines and convex hulls. Within this figure, both the extreme lines and convex hulls are initialized from points p_1 and p_2. When the next point p_{k+1} is introduced, we use the Theorem 2 of [13] and Eq. 3.5 to determine the new extreme lines u_{k+1} and l_{k+1}. As seen in Fig. 4iii, the new convex hulls can be updated by removing the earliest point in $\overline{cvx}(k)$ to form $\overline{cvx}(k+1)$, and deleting the two latest points in $\underline{cvx}(k)$ to form $\underline{cvx}(k+1)$.

3.2 SemiConnAlg

For two adjacent disconnected segments S_k and S_{k+1} constructed by DisConnAlg, there are two kinds of relationships between them. As Fig. 5 shows, S_{k+1} is below or above S_k. This can be derived by checking whether $p_{s_{k+1}}$ is below or above l_k or u_k respectively. In addition, as indicated in Fig. 5i, l_{k+1} can approximate the point p_{e_k} by extending S_{k+1} backwards, which inspires the idea of constructing semi-connected segments from the disconnected ones. In the rest of this subsection, we introduce how to make S_{k+1} semi-connect to S_k when S_{k+1} is below S_k. The other case is similar.

Extended from DisConnAlg, SemiConnAlg generalizes the concept of "connected" and achieves the optimal set of semi-connected segments by shrinking and extending the pending disconnected segments constructed from DisConnAlg. The basic steps of SemiConnAlg are: (1) construct the first disconnected segments S_1 by DisConnAlg; (2) construct the $(k+1)$-th $(k \geq 1)$ disconnected segment from p_{e_k+1} by DisConnAlg. Using the "backward-checking" strategy, check whether S_{k+1} can semi-connected to S_k. I.e, whether any two representation lines of S_k can be intersected with S_{k+1} while all points in S_k and S_{k+1} still satisfy the maximum error bound; (3) conduct the "shrinking-extending" strategy if the "backward-checking" strategy fails. This process would discard the points of S_{k+1} one by one from the direction in which data is being introduced until "backward-checking" strategy succeeds. This step guarantees that "backward-checking" must succeed when the shrunk S_{k+1} only contains one remaining point.

For example, Fig. 6i shows that the pending segment S_{k+1} can directly semi-connect to the previous segment S_k by the "backward-checking" strategy, where $p_{\underline{cvx}_k}$ and $p_{\overline{cvx}_k}$ repre-



(i) Status=(down,down) (ii) Status=(down,up)

Fig. 7 The selection method of f_{k+1} in S_{k+1}

sent points $\underline{p_c}$ and $\overline{p_d}$ determined by formula 3.2. Figure 6ii shows the process of "shrinking-extending", i.e, S_{k+1} can extend to approximate p_{e_k} after deleting $p_{e_{k+1}}$. At this time, the updated extreme line l'_k can semi-connect with l_k.

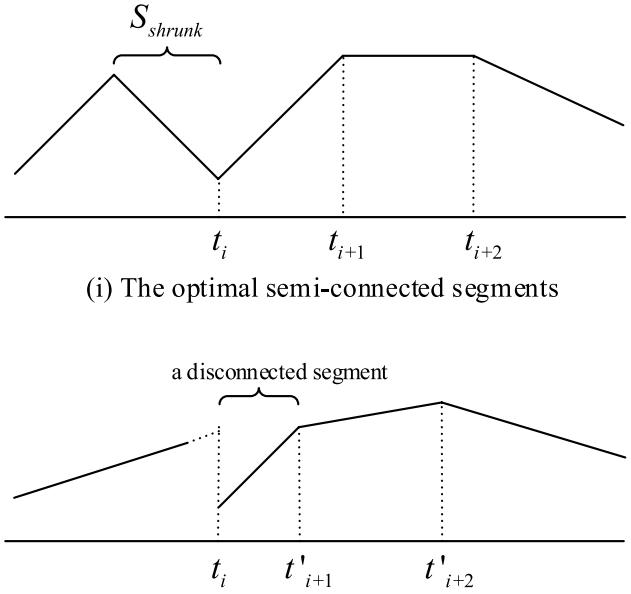
With this, the next problem to face is how to determine the representation lines f_{k+1} of S_{k+1}. In fact, the determination of f_{k+1} depends on the relationships of S_{k+1} with S_k and S_{k+2}. We use a position variable (named as $status = (a, b)$) to represent them, where a and b are in $\{up, down\}$ and represents the relative positions between S_{k+1} and S_k, between S_{k+2} and S_{k+1} respectively. For example, if S_{k+1} is below S_k, then $a = down$. Once $status = (a, b)$ is instantiated, the selected f_{k+1} from S_{k+1} and S_k are decided (and vice versa). Specifically, Fig. 7 shows the selection method. If $status = (down, down)$ or $status = (down, up)$, f_{k+1} is the lower or upper extremes lines respectively. As Fig. 7ii shows, if $f_{k+1} = u_{k+1}$, S_{k+1} needs to be extended to the point p_{e_k-1} to ensure that "backward-checking" succeeds. But this extension is not laissez-faire. From this figure, it can be verified that if it reaches the point $p_{\underline{cvx_k}}$, the updated u_{k+1} semi-connects to l_k. This characteristic implies that the earlier constructed semi-connected segments do not need to be modified and thus ensures the linear-time complexity of SemiConnAlg.

For a clearer discussion of SemiMixedAlg, we list some of the key properties of SemiConnAlg:

1 Each currently generated semi-connected segment will reach the farthest possible data point by preserving semi-connectedness. That is, starting from p_1, there is no $(k+1)$ number of semi-connected segments with the ending time stamp of e'_{k+1} ensuring that $(e'_{k+1} > e_{k+1})$ holds, where e_{k+1} is the ending time stamp after constructing the semi-connected segment S_{k+1} by SemiConnAlg.
2 If the semi-connected segment is generated by using the "shrinking-extending" strategy, its status with two adjacent segments is opposite, i.e, $status = (down, up)$ or $status = (up, down)$.
3 Let $t(Dis)$ and $t(Semi)$ be the time cost of DisConnAlg and SemiConnAlg conducted on a time series with the same error bound, $t(Dis) \leq t(Semi) \leq 4 * t(Dis)$ holds.

4 SemiMixedAlg

In this section, we first show how to use an optimal set of semi-connected segments to further reduce the output storage. We then describe the "k-length" strategy and explain how to use this strategy to construct mixed-connected segments from the results by SemiConnAlg. Lastly, we give the



(i) The optimal semi-connected segments

(ii) The derived mixed segments

Fig. 8 Mixed-connection

pseudo code of SemiMixedAlg and some properties for the computational efficiency.

4.1 Mixed-connection

The purpose of introducing mixed-connected segments is to further reduce the output storage of optimal semi-connected. As showed in Fig. 8(i), the storage of optimal semi-connected segments (denoted as S) is $2 * 6 = 12$ units. When we disconnect S at t_i, a set of mixed-connected segments may be derived. The storage of this new set is smaller than that of S as 3 units are required at t_i; resulting in a total storage of $3 + 2 * 4 = 11$ units.

In the process of SemiConnAlg, each currently produced segment reaches the farthest point (in the direction which data is being introduced) forwardly, but cannot reach the farthest one backwardly. If we conduct SemiConnAlg backwardly from the ending point, then the semi-connected points at t_{i+2} and t_{i+1} would be moved backwardly becoming t'_{i+2} and t'_{i+1} respectively. These movements may cause segment S_{shrunk} to piled up and be replaced with a disconnected segment at t_i; resulting in a reduction in storage by one unit.

In the following, Lemma 4.1 indicates that the starting time stamp of a mixed segment of S_{mixed} is the same with the starting time stamp of pending disconnected segment that needs to be shrunk so as to semi-connect to S_{i-1}.

Lemma 4.1 *For a data fragment P, let the optimal semi-connected segments by SemiConnAlg be $S_{semi} = (S_1, S_2, \dots, S_k)$. Suppose there exists a set of mixed-connected segments S_{mixed} derived from S_{semi}, including only one disconnected segment S'_i. Suppose that S_i is constructed from pending disconnected segment Sp_i. Then*: (1). *In order to semi-connect to S_{i-1}, Sp_i needs to be readjusted by using the "shrinking-extending" strategy*; (2). *S'_i can start at the same starting time stamp of Sp_i.*

Proof If S_i can semi-connect to S_{i-1} by the "backward-checking" strategy without shrinking, then the two segments, S_i and Sp_i, can have the same largest end time stamp. In this situation, the replacement of S_i with disconnected Sp_i would cause one more unit in storage, which is contradictory to the requirement of optimal storage. Thus (1) is proven.

Next, since S'_i is disconnected, the end time stamp of S'_i is no larger than that of Sp_i. Thus (2) is proven. □

Lemma 4.1 guarantees that a mixed segment in a set of mixed-connected segments is a pending disconnected segment by DisConnAlg, which would fail to directly semi-connect with the previous one. The following "k-length" strategy is used to construct disconnected segments in a mixed representation.

Strategy: "k-length". Let S_i be a segment that fails to semi-connect to S_{i-1} through backward-checking. Let Sp_i be the pending disconnected segment used to generate S_i by the shrinking and extension strategy. Suppose that S_{i+k} is the k-th semi-connected segment constructed from S_i and S'_{i+k-1} is the $(k-1)$-th semi-connected segment from Sp_i. Both S_{i+k} and S'_{i+k-1} are conducted by SemiConnAlg. For the end time stamps e'_{i+k-1} of S'_{i+k-1} and e_{i+k} of S_{i+k}, we say S_i is "k-length", denoted as $tryLength(S_i) = k$, if

$$e_{i+k} \le e'_{i+k-1}. \tag{4.1}$$

Clearly, Sp_i is a mixed segment in mixed-connected representations if $tryLength(S_i) = k$. With Lemma 4.1, S_i can be constructed from Sp_i. Intuitively, if $tryLength(S_i) = k$, there would exist a k such that the end time stamp of S'_{i+k-1} is no smaller than that of S_{i+k}. This indicates that S_{i+k} can be extended backwardly to approximate more points, making the semi-connected point of S_{i+k} shift to an earlier data point. If this process for the semi-connected segments is performed recursively, this can eventually lead to S_i being removed and replaced by Sp_i.

For computational efficiency, we would generally choose the smallest k ($k > 1$) in Eq. 4.1.

Let $P = (p_1, p_2, \dots, p_n)$ be a time series. Continued from the definition on "k-length", the value of $tryLength(S_i)$ is specified in Lemma 4.2 for checking the "shrunk" segments only.

Lemma 4.2 *Suppose that $tryLength(S_i) = k$ where the end time stamp e_{i+k} of S_{i+k} is less than t_n. Then S_{i+k} is generated through "shrinking-extending" strategy to semi-connect to its previous segment.*

Proof If S_{i+k} is generated through the "shrinking-extending" strategy to semi-connect to S_{i+k-1}, then the lemma is proven. Otherwise, without loss of generality, we assume that S_{i+h} is the closest to S_{i+k} generated through "shrinking-extending" strategy. That is, each S_j $(i + h < j \le k)$ can semi-connect to its previous segment without requiring "shrinking" operations (i.e., S_j semi-connects to its previous segment by the "backward-checking" strategy without shrinking).

Again, since S_{i+k} is the longest representing segment starting from time stamp s_{i+k}, we have $s_{i+k} \le s'_{i+k-1}$ holds for the start time stamp of S'_{i+k-1}. Repeat the above process until we have $s_{i+h+1} \le s'_{i+h}$ holds for the start time stamps of S_{i+h+1} and S'_{i+h}, which is $e_{i+h} \le e'_{i+h-1}$ equivalently. From the definition of the "k-length" strategy, we therefore have $tryLength(S_i) = h$ and S_{i+h} is generated through "shrinking-extending" strategy to semi-connect to S_{i+h-1}. □

Algorithm 1 Function SemiMixedAlg(P, δ)

Input:
time sequence $P = (p_1, p_2, \cdots, p_n, \ldots)$, error bound δ
Output:
The mixed-connected points $Inter = (inter_0, inter_1, inter_2, \ldots)$, where $inter_i = (t_i, y_i)$ if $inter_i$ is a connected point, $inter_i = (t_i, y_{i_1}, y_{i_2})$ if $inter_i$ is a mixed point.
Description:
1: Use DisConnAlg to generate the first segment (S, u, l, t_s, t_e) in which S is the segment, u and l are lower and upper extreme lines of S, t_s and t_e are the start and end time stamps of S
2: Set the number of segments $k = 1$
3: Set *extrm*: if pt_{e+1} is under l, then set *extrm* to be l. Otherwise, set *extrm* to be u
4: Set *status*: $a = \Phi$, $b = down$ if pt_{e+1} is located at below line l or $b = up$ if pt_{e+1} is located at above line u.
5: Store the intersection of *extrm* at time tag t_1 into $inter_0$.
6: **while** (not finished segmenting time series) **do**
7: Use DisConnAlg to generate the next disconnected segment (S', u', l', t'_s, t'_e) in which S' is the segment, u' and l' are lower and upper extreme lines of S', $t'_s = t_e + 1$ and t'_e are the start and end time stamps of S'
8: **if** (Use the "backward-checking" strategy to check whether S is semi-connected by S') **then**
9: **if** $(b = down)$ **then**
10: Store the intersection point $inter_k$ of l' and *extrm*
11: update $extrm = l'$
12: **end if**
13: **if** $(b = up)$ **then**
14: Store the intersection point $inter_k$ of u' and *extrm*
15: update $extrm = u'$
16: **end if**
17: **else**
18: Shrink and backwards extend S' to achieve the semi-connected segment $(S'', u'', l'', t''_s, t''_e)$
19: Do SemiMixedAlg from S'' until another shrunk and extended segment with the end time stamp t^*_e is generated, push the semi-connected points $(inter^*_0, inter^*_1, ..., inter^*_{num-1})$ into *Inter*.
20: Construct the $(num-1)$-th semi-connected segment by SemiMixedAlg from S', record the end time stamp of $(num-1)$-th segment is t^{**}_e, store the semi-connected points into a temp list $Inter^{**} = (inter^{**}_0, inter^{**}_1, ..., inter^{**}_{num-2})$, where $inter^{**}_0 = (t^{**}_0, y^{**}_0)$
21: **if** $(t^{**}_e >= t^*_e)$ **then**
22: Update $inter^{**} = (t^{**}_0, y^{**}_0, y^{**}_1)$, where y^{**}_1 is the value of intersection of *extrm* at t^{**}_0
23: Back pop *num* number of elements from *Inter*
24: Push $Inter^{**}$ into *Inter*
25: **end if**
26: **end if**
27: Continue
28: **end while**

4.2 Algorithm

The pseudo code of SemiMixedAlg is depicted in Algorithm 1. Its steps are explained as follows:

Lines 1–5: Construct the first disconnected segment (S, u, l, t_s, t_e) by DisConnAlg in which S is the segment, l and u are lower and upper extreme lines of S, $t_s = t_1$ and t_e are the start and end time stamps of S. Set the number of semi-connected segment $k = 1$.

To initialize $status = (a, b)$ and the selected extreme line (*extrm*), we check the relative position of u (or l) to the next data point p_{t_e+1}. The algorithm sets $status = (\phi, down)$ and $extrm = l$ if p_{t_e+1} is located below line l and, otherwise, sets $status = (\phi, up)$ and $extrm = u$ if p_{t_e+1} is located above line u. Let $inter_0$ be the point of *extrm* at time stamp t_1.

Lines 6–7: Construct segment (S', u', l', t'_s, t'_e) by DisConnAlg starting from $t'_s = t_e + 1$ as the pending disconnected segment.

Lines 8–16: If S' can semi-connect to S by the "backward checking" strategy (without shrinking), store the intersection $inter_k$ of l' and *extrem* (or u' and *extrem*) with $b = Down$ (or $b = Up$), and update $extrm = l'$ (or $extrm = u'$) respectively.

Lines 18–19: Otherwise, the "shrinking-extending" strategy is used to have $(S', u'', l'', t''_s, t''_e)$ derived from Sp in order to semi-connect with S. From $t''_e + 1$, do step 2–3 until the "shrinking-extending" is used again. Let the end time-stamp of the shrunk and extended segment be t^*_e. In this process, push the semi-connected points $Inter^* = (inter^*_0, inter^*_1, \cdots, inter^*_{num-1})$ into *Inter*.

Then, starting from $t'_e + 1$; do $num - 1$ semi-connected segments by following steps 2–3. Meanwhile, record the semi-connected points in another temp list $Inter^{**} = (inter^{**}_0, inter^{**}_1, \ldots, inter^{**}_{num-2})$. Let the end time-stamp of $num - 1$ semi-connected segments be t^{**}_e.

Lines 21–27: If $t^{**}_e >= t^*_e$, a mixed segment is generated. Update $inter^{**}_0 = (t^{**}_0, y^{**}_0, y^{**}_1)$, where y^{**}_1 is the intersection value of *extrm* at t^{**}_0. Back pop *num* number of elements from *Inter* and push $Inter^{**}$ into *Inter*.

Lines 28–29: Repeat the above from Line 6 for the last segment until all data points have been processed. Output the stored data *Inter*.

The following theorem is the major result of this paper.

Theorem 4.1 *For any given time series $P = (p_1, p_2, \ldots, p_n)$ and an error bound δ, SemiMixedAlg is a linear-time algorithm that generates mixed-connected segments with minimum storage.*

Proof For any given time series $P = (p_1, p_2, \ldots, p_n)$ and an error bound δ, let the generated k number of mixed-connected segments be $S_{mixed} = (S_1, \ldots, S_k)$, where $M_j, (1 \le j \le m)$ denotes the the m number of disconnected segments. The storage of S_{mixed} is $2 * (k + 1) + m$, which indicates that the only way to reduce storage further is to reduce k into $k - 1$ by adding one more disconnected segment. This implies that the representation needs to replace two semi-connected segments with one disconnected segment, contradictory to the fact that two semi-connected segments cannot be covered by any single disconnected segment and the "k-length" strategy is used for traversal segments.

Comparing both the time costs of SemiConnAlg and SemiMixedAlg, it can be seen that SemiConnAlg consists of a lower time cost. This extra cost of SemiMixedAlg is the computation cost on the "k-length" strategy. In the worst case, the number of original data points from the "k-length" checking is the same as the number of points in P excluding S_1. So the time cost $T(SemiMixed)$ of SemiMixedAlg is $T(SemiConn) <= T(SemiMixed) <= 2 * T(SemiConn)$,

Table 1 Storage costs of SemiMixedAlg and PipeMixedAlg

Data set	Length	Storage SemiOpt=PipeOpt	Compression rate
Coffee	8036	674	0.042
ECG200	9700	1758	0.091
Beef	14,130	152	0.005
OliveOil	17,131	627	0.018
synthetic	18,301	11,867	**0.324**
GunPoint	22,650	1965	0.043
Lighting7	23,360	1317	0.028
ItalyPowerDemand	25,726	12,830	**0.249**
Trace	27,600	1061	0.019
FaceFour	30,888	4357	0.071
Lighting2	38,918	772	0.01
SonyAIBORobotSurface	42,671	16,751	**0.196**
SonyAIBORobotSurfaceII	62,899	27,007	**0.215**
Adiac	69,207	1841	0.013
MedicalImages	76,000	8064	0.053
SwedishLeaf	80,626	7142	0.044
FISH	81201	1596	0.01
TwoLeadECG	94,537	15,029	0.079
OSULeaf	103,576	5960	0.029
DiatomSizeReduction	105,877	3332	0.016
MoteStrain	106,420	12,231	0.057
CBF	116100	66,240	**0.285**
CricketX	117,391	5083	0.022
CricketY	117,391	4686	0.02
CricketZ	117,391	4977	0.021
ECGFiveDays	117,958	11,704	0.05
50words	123,305	1550	0.006
WordsSynonyms	172,898	5749	0.017
FaceAll	223,081	41,848	0.094
FacesUCR	27,0601	51,584	0.095
Haptics	336,645	3967	0.006
Symbols	397,006	10,918	0.014
TwoPatterns	516,000	231,610	**0.224**
ChlorineConcentration	641,281	44,445	0.035
wafer	943092	80,998	0.043
InlineSkate	1,035,650	6507	0.003
uWaveGestureLibraryX	1,131,913	39,428	0.017
uWaveGestureLibraryY	1,131,913	38,122	0.017
uWaveGestureLibraryZ	1,131,913	41,004	0.018
yoga	1,281,000	49,998	0.02
CinCECG	2,263,201	15,810	0.003
MALLAT	2,403,626	71,285	0.015
StarLightCurves	8,441,901	89,713	0.005

where $T(SemiConn)$ is the cost of SemiConnAlg on P. Therefore, SemiMixedAlg is a linear-time algorithm.

□

5 Experiment

To verify the characteristics of our algorithm, we compare our algorithm with SemiConnAlg, PipeConnAlg and PipeMixedAlg on a wide variety of real data sets. We thank the authors of [15] for giving us their original source code for

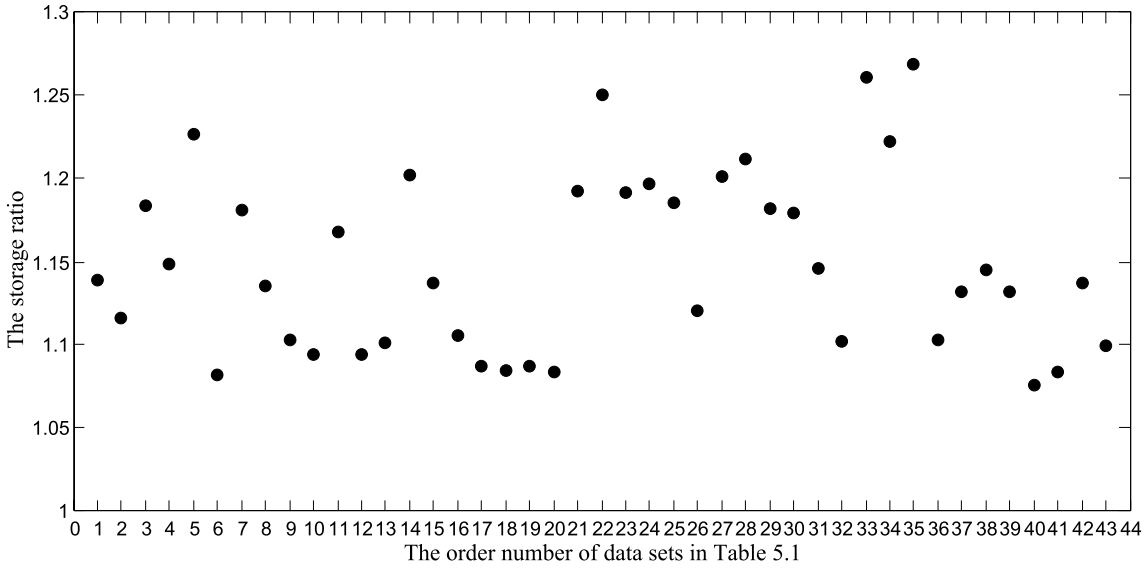


Fig. 9 The storage ratio of SemiConnAlg and SemiMixedAlg

PipeConnAlg and PipeMixedAlg. Our experiments are conducted with the same computing environment, i.e., all algorithms are written in C++ and conducted on a PC with Intel Core 2.4 GHz CPU and 8G memory. We tested the storage, time cost and maximum memory cost on the 43 data sets from the UCR Time Series Archive [19]. These data sets are listed in Table 1 in the order of their sizes. We have tested all the data sets on the error bounds consisting of value ranges of 2.5%, 5% and 10%. Due to the similarities, we only show the results on the 5% error bound.

In the scalability tests, we explore the time cost and memory cost with varying data lengths on the error bound 5% of the value range for all 43 data sets. We first divide each data set into 20 fragments and compute the accumulated time and maximum memory costs on each fragment. We then compute the time and memory cost on every ratio with varying the error bound from 1% to 10% of data range.

Storage on outputs. The storage costs of SemiMixedAlg are listed in Table 1. Since the distributions of those sets are varying, the achieved compression rate on the 43 data sets are different. Except the items expressed in boldface, the outputs of the rest 37 data sets are less than 10% of the original storages. A compared result of SemiConnAlg and SemiMixedAlg is shown in Fig. 9. Storage ratios of different scaled data sets were plotted below. This plot verified that the storage of SemiMixedAlg is less than that of SemiConnAlg. Specifically, the average storage is less than that of SemiConnAlg by 13%.

The output storage of SemiMixedAlg and PipeMixedAlg are exactly the same. The reasons are:

1. Let the representation lines constructed by SemiConnAlg and PipeConnAlg be $f_i(e_i) = f_{i+1}(s_{i+1})$ $(1 \le i \le k-1)$ and $f'_i(e'_i) = f'_{i+1}(s'_{i+1})$ $(1 \le i \le k'-1)$ respectively. Verified from [14], $k = k'$, $s_i = s'_i$, $e_i = e'_i$ and $f_i = f'_i$ hold, where f_i and f'_i are derived from the extremes lines of i-th semi-connected segment.
2. Although the mechanisms of searching the mixed segments in SemiMixedAlg and PipeMixedAlg are different, the starting time stamp and the number of mixed segments are exactly the same. PipeMixedAlg determines the locations of mixed segments by using 5 "open visible regions". It then checks which one of these regions can reach the farthest point (i.e., the point with the maximum time stamp). While SemiMixedAlg uses a property driven "k-length" strategy to yield the same results.
3. Suppose we divide a time series P into a number of fragments at the starting time stamp of mixed segments. The mixed-connected segments can be achieved by conducting SemiConnAlg or PipeConnAlg on each fragment.

Table 2 Time costs of SemiMixedAlg, PipeMixedAlg, SemiConnAlg and PipeConnAlg

Order	T(SemiInM) =T(SemiC)	T(MixInM)	T(SemiM)	T(PipeC)	T(PipeM)
1	3.75	2.23	5.98	20	41
2	3.73	2.45	6.18	20	30
3	7.35	2.65	10	19	60
4	8.64	4.2	12.84	30	95
5	9.18	8.06	17.24	40	77
6	9.62	6.38	16	44	130
7	8.23	3.85	12.08	36	85
8	15.42	11.92	27.34	49	123
9	8.85	5.33	14.18	34	106
10	14.3	9.38	23.67	50	124
11	9.13	5.56	14.69	46	134
12	20.1	14.71	34.81	65	219
13	27.69	19.89	47.58	114	284
14	26.88	9.89	36.77	87	260
15	31.55	19.43	50.97	111	322
16	47.5	22.29	69.79	123	387
17	54.15	19.54	73.69	114	420
18	46.34	29.44	75.78	173	444
19	47.43	26.42	73.85	142	491
20	39.37	24.2	63.57	197	506
21	61.72	35.7	97.41	175	499
22	76.24	54.41	130.65	211	570
23	50.72	20.07	70.79	167	480
24	52.17	21.52	73.7	175	462
25	51.86	19.68	71.54	181	506
26	54.32	29.3	83.62	218	526
27	42.41	12.85	55.26	168	467
28	90.64	32.46	123.1	310	800
29	118.84	60.26	179.1	325	1113
30	145.31	74.5	219.81	409	1161
31	126.83	73.19	200.01	424	1078
32	156.03	100.53	256.56	593	1587
33	269.87	250.14	520.01	825	2187
34	235.74	105.8	341.55	789	2388
35	477.11	426.03	903.14	1372	3812
36	357.43	173.48	530.92	1228	3543
37	516.45	252.38	768.83	1441	4151
38	498.16	255.28	753.44	1572	4128
39	503.24	271	774.24	1712	4715
40	489.21	316.15	805.36	1599	4323
41	742.34	330.61	1072.96	3224	8341
42	1193.21	705.79	1899	3687	11,828
43	3200.45	2090.79	5291.24	12,255	35,567

From 1 and 2, the outputs from SemiMixedAlg and PipeMixedAlg are exactly the same.

Due to the exact same outputs of SemiMixedAlg and PipeMixedAlg, the representation lines constructed by SemiMixedAlg yield the smallest mean square error than that of optimal disconnected and semi-connected results, which is verified in [15].

Time cost. For 43 data sets, we report the time costs of PipeConnAlg, PipeMixedAlg, semi-connected processing and mixed-connected processing in SemiMixedAlg, denoted by $T(PipeC)$, $T(PipeM)$, $T(SemiInM)$ and $T(MixInM)$ respectively. Let the time cost of SemiConnAlg and SemiMixedAlg be $T(SemiC)$ and $T(SemiM)$. Then $T(SemiC) = T(SemiInM)$ and $T(SemiM) = T(SemiInM) + T(MixInM)$ holds. All the results are listed in Table 2. For each data set, we use the entire run time of SemiConnAlg, SemiMixedAlg, PipeConnAlg and PipeMixedAlg divided by the length of data set to obtain the average time cost on updating one point. The average time cost on 43 data sets are showed in Fig. 10. Table 2 and Fig. 10 indicate:

1. $T(SemiC) \leq T(SemiM) < 2 * T(SemiC)$ holds for all data sets, which verifies the time costs between SemiConnAlg and SemiMixedAlg in the proof of Theorem 4.1. From Table 2, it can see that the maximum and minimum ratio of SemiMixedAlg and SemiConnAlg are 1.93 and 1.36 on 33-rd and 17-th data sets respectively. The reason for the resulting different ratios on various data sets is the diverse trying number by the "k-length" strategy. If the trying number is greater than 2, extra time cost is required.
2. From Table 2, we can see that $T(PipeC)$ is about 4 times of $T(SemiC)$, which is also verified in the report of [14]. In addition, $T(PipeM)$ is about 3 times of $T(PipeC)$.
3. The time cost of PipeMixedAlg is about 6 times that of SemiMixedAlg. On 43 tested data sets, the average ratio of $T(PipeM)$ and $T(SemiM)$ is 6.22.

Memory cost. Let $M(SemiC)$, $M(SemiM)$, $M(PipeC)$ and $M(PipeM)$ be the maximum memory costs of SemiConnAlg, SemiMixedAlg, PipeConnAlg and PipeMixedAlg respectively. $M(SemiM)$ includes two parts: the maximum memory cost of SemiConnAlg used in SemiMixedAlg, denoted by $M(SemiInM)$ and the maximum memory usage of the list to store the temporary intersection (see Line 20 in Algorithm 1), denoted by $M(Trying)$. The maximum memory cost of each data set is illustrated in Table 3 and Fig. 11. They demonstrate that:

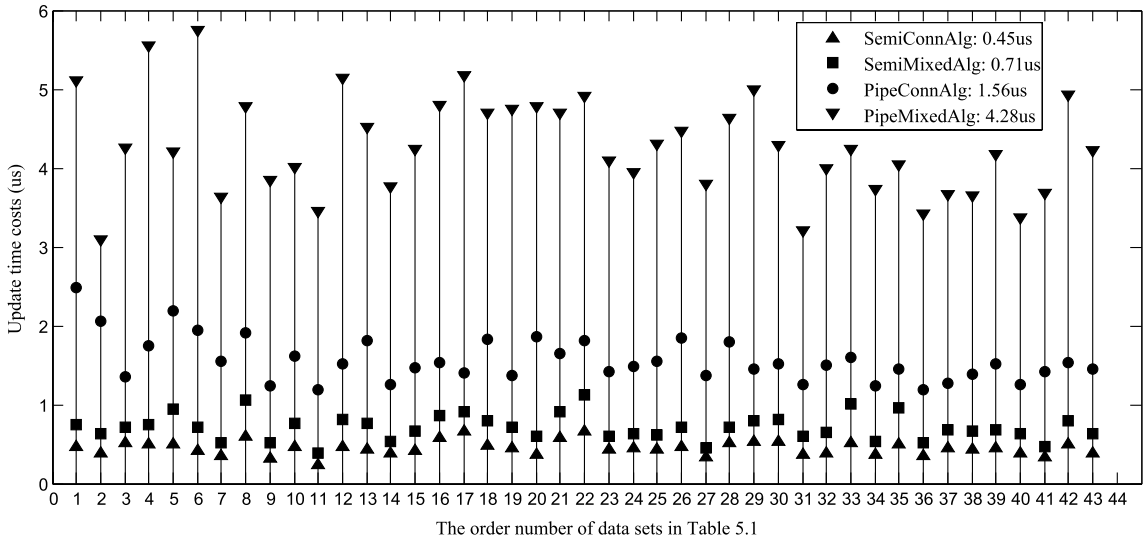


Fig. 10 Average time costs of SemiMixedAlg, PipeMixedAlg, SemiConnAlg and PipeConnAlg

1. $M(SemiInM)$ is equal to $M(SemiC)$ on most tested data sets, excluding the 35th, 41st and 42nd data sets. It indicates that the largest memory cost of SemiConnAlg in "k-length" strategy has a tiny impact on $M(SemiInM)$. This is because the maximum storage of "convex hulls" in "k-length" process is very close to the one on semi-connected segments.
2. $M(trying)$ relates to the size of "k-length", which can be very small on the tested data sets. $M(SemiM)$ is almost the same as $M(SemiC)$.
3. The change from $M(SemiC)$ to $M(SemiM)$ on memory cost is lower than $T(PipeC)$ to $T(PipeM)$. Similar to the situation on time cost, the difference can be ascribed to the maintenance of 5 open visible regions in PipeMixedAlg. The average of $M(PipeM)$ on all tested data sets is about 4 times of that $M(SemiM)$.

Scalability. Since PipeMixedAlg is the only state of the art algorithm to construct optimal mixed-connected segments, we compare it with SemiMixedAlg to verify the better performance of SemiMixedAlg. Since the results on all data sets are very similar, we only report the results on the largest data set "StarLightCurves".

As showed in Fig. 12i, ii, the time costs of these two algorithms are increased linearly with the added data size. But the increasing tendency of SemiMixedAlg is much slower than that of PipeMixedAlg. In addition, the average memory cost on updating each data point is close to constant for both algorithms, and the memory size of PipeMixedAlg is about 4 times of that SemiMixedAlg. As depicted in Fig. 12iii, iv, it can be seen that SemiMixedAlg does not only consist of lower time and memory costs, but is also more stable than that of PipeMixedAlg on the increased error bound.

6 Conlusion

In this paper, a new linear time PLA algorithm SemiMixedAlg based on SemiConnAlg is presented for constructing mixed segments with guaranteed max-error bound. We prove that SemiMixedAlg achieves the minimum storage on mixed-connection. The extensive experiments on 43 data sets indicate that SemiMixedAlg has lower memory requirements and is much more efficient and stable than that of PipeMixedAlg on large data sets.

Table 3 The maximum memory costs of SemiMixedAlg, PipeMixedAlg, SemiConnAlg and PipeConnAlg

Order	M(SemiC)	M(SemiInM)	M(Trying)	M(SemiM)	T(PipeC)	T(PipeM)
1	0.27	0.27	0.06	0.33	0.34	1.43
2	0.14	0.14	0.16	0.3	0.23	0.97
3	0.28	0.28	0.03	0.31	0.45	1.45
4	0.41	0.41	0.05	0.46	0.55	1.85
5	0.12	0.12	0.13	0.25	0.2	0.95
6	0.23	0.23	0.07	0.3	0.34	1.41
7	0.21	0.21	0.09	0.3	0.28	1.06
8	0.12	0.12	0.12	0.24	0.19	0.88
9	0.19	0.19	0.21	0.4	0.27	1.15
10	0.16	0.16	0.12	0.28	0.23	1.19
11	0.25	0.25	0.11	0.36	0.31	1.42
12	0.12	0.12	0.2	0.32	0.2	0.98
13	0.12	0.12	0.19	0.31	0.2	0.95
14	0.46	0.46	0.04	0.5	0.66	2.06
15	0.59	0.59	0.1	0.69	0.74	2.33
16	0.25	0.25	0.09	0.34	0.38	1.48
17	0.73	0.73	0.04	0.77	0.81	3.23
18	0.15	0.15	0.11	0.26	0.23	0.92
19	0.33	0.33	0.13	0.46	0.39	1.72
20	0.5	0.5	0.11	0.61	0.65	2.04
21	0.25	0.25	0.09	0.34	0.34	1.46
22	0.13	0.13	0.09	0.22	0.2	1.02
23	0.22	0.22	0.07	0.29	0.28	1.26
24	0.21	0.22	0.09	0.31	0.3	1.38
25	0.21	0.21	0.07	0.28	0.28	1.14
26	0.21	0.21	0.14	0.35	0.27	1.1
27	0.71	0.71	0.04	0.75	0.73	3.09
28	1.18	1.18	0.05	1.23	1.4	4.3
29	0.2	0.2	0.11	0.31	0.26	1.28
30	0.19	0.19	0.12	0.31	0.27	1.23
31	0.54	0.54	0.07	0.61	0.6	2.16
32	0.3	0.3	0.12	0.42	0.44	1.6
33	0.13	0.13	0.09	0.22	0.21	0.97
34	0.19	0.19	0.07	0.26	0.27	1.31
35	0.33	0.35	0.07	0.42	0.42	1.45
36	0.27	0.27	0.07	0.34	0.39	1.62
37	0.52	0.52	0.1	0.62	0.55	2.04
38	0.38	0.38	0.1	0.48	0.5	1.8
39	0.43	0.43	0.1	0.53	0.52	1.96
40	1.02	1.02	0.18	1.2	1.09	3.96
41	0.42	0.5	0.12	0.62	0.61	2.3
42	0.93	1.05	0.06	1.11	1.23	5.37
43	1.25	1.25	0.21	1.46	1.44	5.5

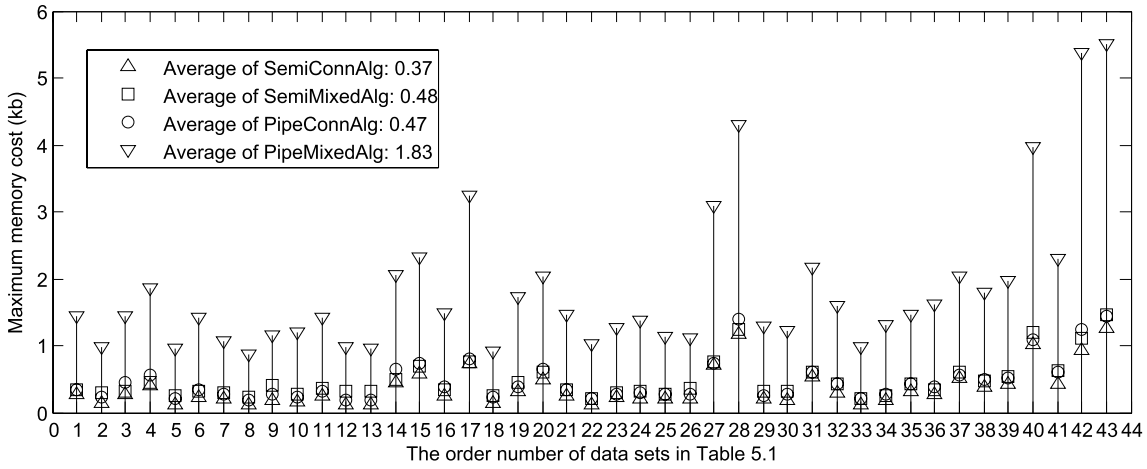


Fig. 11 The maximum memory costs of SemiMixedAlg, PipeMixedAlg, SemiConnAlg and PipeConnAlg

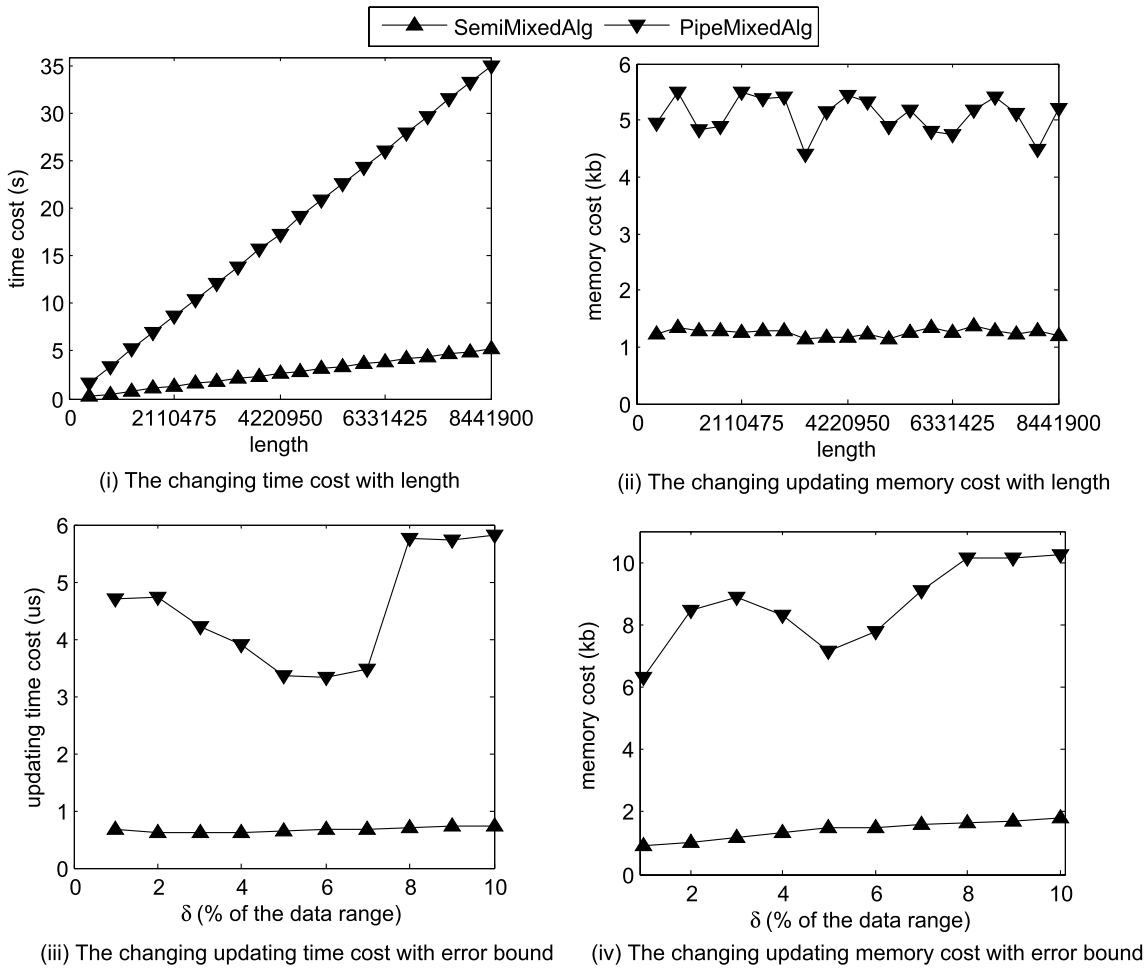


Fig. 12 The time and memory costs of "StarLightCurves" on varying data sizes and error bound

Acknowledgements We thank the authors of [15] for giving us their source codes for the comparison tests in this article. This work was partially supported by the Natural Science Foundation of China (No. 61572022), and Hebei "One Hundred Plan" Project (No. E2012100006).

References

1. Letchford A, Gao J, Zheng L (2013) Filtering financial time series by least squares. Int J Mach Learn Cybern 4(2):149–154
2. Pulkkinen A, Cox BT, Arridge SR, Goh H, Kaipio JP, Tarvainen T (2016) Direct estimation of optical parameters from photoacoustic time series in quantitative photoacoustic tomography. IEEE Trans Med Imaging 35(11):2497–2508
3. Wang R, Chow C-Y, Lyu Y, Lee VCS, Kwong S, Li Y, Zeng J (2018) TaxiRec: recommending road clusters to taxi drivers using ranking-based extreme learning machines. IEEE Trans Knowl Data Eng 30(3):585–598
4. Wang X, Zhang T, Wang R (2019) Non-iterative deep learning: incorporating restricted Boltzmann machine into multilayer random weight neural networks. IEEE Trans Syst Man Cybern Syst 49(7):1299–1380
5. Wang R, Wang X, Kwong S, Chen X (2017) Incorporating diversity and informativeness in multiple-instance active learning. IEEE Trans Fuzzy Syst 25(6):1460–1475
6. Rafiei D, Mendelzon A (1998) Efficient retrieval of similar time sequences using DFT. eprint arXiv:cs/9809033, pp 249–257
7. Pang C, Zhang Q, Zhou X, Hansen DP, Wang S, Maeder AJ (2013) Computing unrestricted synopses under maximum error bound. Algorithmica 65(1):1–42
8. Perng C, Wang H, Zhang SR, Parker DS (2000) Landmarks: a new model for similarity-based pattern querying in time series databases. In: Proceedings of the 16th international conference on data engineering. IEEE Computer Society, pp 33–42
9. Liu X, Lin Z, Wang H (2008) Novel online methods for time series segmentation. IEEE Trans Knowl Data Eng 20(12):1616–1626
10. Palpanas T, Vlachos M, Keogh E, Gunopulos D, Truppel W (2004) Online amnesic approximation of streaming time series. In: 20th international conference on data engineering, 2004. Proceedings, pp 339–349
11. Palpanas T, Vlachos M, Keogh E, Gunopulos D (2008) Streaming time series summarization using user-defined amnesic functions. IEEE Trans Knowl Data Eng 20:992–1006
12. Keogh EJ, Pazzani MJ (1999) Scaling up dynamic time warping to massive dataset. In: principles of data mining and knowledge discovery, Third European conference, PKDD '99, Prague, Czech Republic, September 15–18, 1999, Proceedings, pp 1–11
13. Xie Q, Pang C, Zhou X, Zhang X, Deng K (2014) Maximum error-bounded piecewise linear representation for online stream approximation. VLDB J 23(6):915–937
14. Zhao H, Pang C, Kotagiri R, Pang CK, Li T (2017) An optimal piecewise linear approximation algorithm on semi-connected segmentation under maximum error bound. Technical report, Zhejiang University
15. Luo G, Yi K, Cheng S-W Li Z, Fan W, Cheng H, Yadong M (2015) Piecewise linear approximation of streaming time series data with max-error guarantees. In: IEEE international conference on data engineering, pp 173–184
16. Hakimi SL, Schmeichel EF (1991) Fitting polygonal functions to a set of points in the plane. Academic Press, Inc, Cambridge, pp 132–136
17. Imai H, Iri M (1986) An optimal algorithm for approximating a piecewise linear function. Inf Process Lett 9(3):159–162
18. Zhao H, Dong Z, Li T, Wang X, Pang C (2016) Segmenting time series with connected lines under maximum error bound. Inf Sci 345(C):1–8
19. Keogh E, Xi X, Wei L, Ratanamahatana C (2011) The ucr time series classification/clustering homepage
20. Appel U, Brandt AV (1983) Adaptive sequential segmentation of piecewise stationary time series. Inf Sci 29(1):27–56
21. Bellman R (1961) On the approximation of curves by line segments using dynamic programming. Commun. ACM 4(6):284
22. Keogh E, Chu S, Hart D, Pazzani M (2001) An online algorithm for segmenting time series. In: Proceedings of the 2001 IEEE international conference on data mining, ICDM '01. IEEE Computer Society, pp 289–296
23. Garofalakis M, Gibbons PB (2004) Probabilistic wavelet synopses. ACM Trans Database Syst 29(1):43–90
24. Garofalakis M, Kumar A (2005) Wavelet synopses for general error metrics. ACM Trans Database Syst 30(4):888–928
25. Wang X, Xing H-J, Li Y (2015) A study on relationship between generalization abilities and fuzziness of base classifiers in ensemble learning. IEEE Trans Fuzzy Syst 23(5):1638–1654
26. Wang X, Wang R, Chen X (2018) Discovering the relationship between generalization and uncertainty by incorporating complexity of classification. IEEE Trans Cybern 48(2):703–715
27. O'Rourke J (1981) An on-line algorithm for fitting straight lines between data ranges. Commun ACM 24(9):574–578
28. Elmeleegy H, Elmagarmid AK, Cecchet E, Aref WG, Zwaenepoel W (2009) Online piece-wise linear approximation of numerical streams with precision guarantees. Proc Vldb Endow 2(1):145–156
29. Zhao H, Li G, Zhang H, Xue Y (2016) An improved algorithm for segmenting online time series with error bound guarantee. Int J Mach Learn Cybern 7(3):365–374
30. Berg MD, Cheong O, van Kreveld M, Overmars M (2008) Computational geometry algorithms and applications. Springer, Berlin