An Optimal Online Semi-Connected PLA Algorithm With Maximum Error Bound

Huanyu Zhao, Chaoyi Pang, Ramamohanarao Kotagiri, Christopher K. Pang, Ke Deng, Jian Yang, and Tongliang Li

Abstract—Piecewise Linear Approximation (PLA) is one of the most widely used approaches for representing a time series with a set of approximated line segments. With this compressed form of representation, many large complicated time series can be efficiently stored, transmitted and analyzed. In this article, with the introduced concept of “semi-connection” that allowing two representation lines to be connected at a point between two consecutive time stamps, we propose a new optimal linear-time PLA algorithm SemiOptConnAlg for generating the least number of semi-connected line segments with guaranteed maximum error bound. With extended experimental tests, we demonstrate that the proposed algorithm is very efficient in execution time and achieves better performances than the state-of-art solutions.

Index Terms—Piecewise linear approximation (PLA), time series, semi-connected, bounded error in approximation, online algorithm, data stream

1 Introduction

A time series is a sequence of data points, each represented by a real number, arranged according to its time sequence. This form of data can be extracted from many fields, including medical and commerce. A good compression strategy for such data can greatly reduce data storage size, expedite data transmission and even improve query efficiency [16], [19].

There are many methods to compress a time series. These include Discrete Wavelet Transform [16], Symbolic Mapping [18], Histograms [19] and Piecewise Linear Approximation (PLA) [10], [12], [13], etc. PLA is a fundamental research problem and has been extensively studied for decades under different criteria. The basic idea of PLA is to represent a time series with a number of straight lines so that the original time series can be precisely approximated and more efficiently processed. In this article, we shall study this problem bounded with maximum error in approximation. This can be expressed as follows.

- H. Zhao, C. Pang, and T. Li are with the School of Computer & Data Engineering, Zhejiang University (NIT); Zhejiang University Ningbo Institute; The Institute of Applied Mathematics, Hebei Academy of Sciences; Hebei Authentication Technology Engineering Research Center, Shijiazhuang, China. E-mail: {huanyuzhao, chaoyi.pang}@qq.com, litongliang@tom.com.
- R. Kotagiri is with the University of Melbourne, Parkville, VIC 3010, Australia. E-mail: kotagiri@unimelb.edu.au.
- C.K. Pang is with the University of Queensland, Brisbane 4072, Australia. E-mail: christopherpang13@gmail.com.
- K. Deng is with the RMIT University, Melbourne, VIC 3000, Australia. E-mail: ke.deng@rmit.edu.au.
- J. Yang is with the Macquarie University, Sydney 2109, Australia. E-mail: jian.yang@mq.edu.au.

Manuscript received 9 July 2019; revised 14 Jan. 2020; accepted 12 Mar. 2020. Date of publication 17 Mar. 2020; date of current version 7 Dec. 2021. (Corresponding Authors: Chaoyi Pang and Tongliang Li.) Recommended for acceptance by J. M. Phillips. Digital Object Identifier no. 10.1109/TKDE.2020.2981319

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 of time slot $[s_i,e_i]$ ($s_i<e_i\le n$). 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 (i.e., *segment*) $f_i(t)$ satisfying the pre-defined error bound of

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

where the domain of t is

$$\begin{cases} t\in[s_1,e_i+1) & i=1,\\ t\in(s_i-1,e_i+1) & 1<i<k,\\ t\in(s_i-1,e_i] & i=k.\end{cases}$$

A linear function that satisfies Equation (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 ($1\le i\le k-1$) (Fig. 1 ii). Otherwise, it is called *disconnected* (Fig. 1i). The segmentation result is called *optimal* if the number of constructed segments (or fragments) k is minimum. Furthermore, the segmentation result is called *semi-connected L_∞-bound PLA* (or simply, *semi-connected*) if $f_i(x_i)=f_{i+1}(x_i)$ where $e_i\le x_i\le s_{i+1}$ holds[2] for each i ($1\le i\le k-1$) (Fig. 1 iii). Conceptually, semi-connection is an extension of connection mentioned above.

1. We will use S_i to denote both the ith fragment and the ith segment in the context when there is no confusion.
2. In this article, we do not consider (integer or decimal) data internal storage expression in bits.

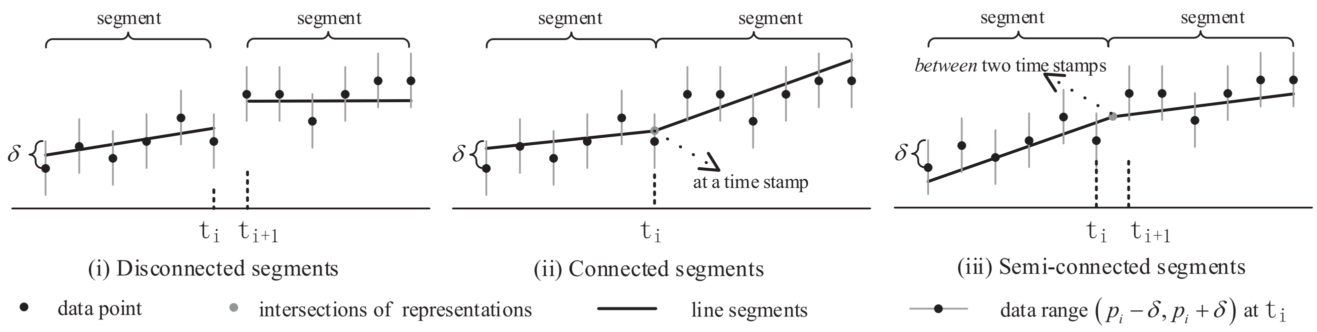

segment
segment
δ
t_i t_{i+1}
(i) Disconnected segments
(ii) Connected segments
at a time stamp
(iii) Semi-connected segments
between two time stamps
● data point
● intersections of representations
line segments
data range $(p_i - \delta, p_i + \delta)$ at t_i

Fig. 1. The disconnected/connected/semi-connected segments.

In this article, we shall propose an optimal linear-time L_∞-bound PLA algorithm for constructing semi-connected segments.

Related Work. On the research of L_∞-bound PLA, Rourke, *et al.* [15] proposed the first optimal disconnected linear-time PLA algorithm in 1981. Recently, Xie, *et al.* [21] gave another completely different optimal method for the same problem.[3] The general idea of their approaches [15], [21] is to adjust each segment to approximate the greatest number of data points. To do this, their methods need to determine the range of all feasible segments and incrementally update it for a consecutive data point. Whenever a new point cannot be approximated by a segment within the error bound, a new segment begins. In the process of determining the range of feasible segments, the algorithm of Rourke [15] maintains a convex polygon in the slope-offset space while DisConnAlg [21] maintains two minimized (partial) convex hulls in the time-value space (time domain) progressively. Generally, these partial convex hulls are much smaller in size than that of a convex polygon. As a result, DisConnAlg not only achieves a speed of up to 4 times faster, but a lower memory cost of 27 percent on the tested data sets. DisConnAlg generates the minimum number of line segments through the "forward-checking" strategy that maximally extends each "local" segment in the forward direction. Furthermore, at the end of the article [21], Xie, *et al.* also indicate that the idea of DisConnAlg can be used to construct connected line segments through the construction of next segment using the feasible region of the previous segment.[4] We denote this algorithm that generates connected segments in [21] as XieConnAlg in this article.

For the construction of connected segments, there were many heuristic algorithms in literature. Liu, *et al.* [13] proposed an Feasible Space Window (FSW) algorithm to construct connected segments from a fixed initial point. Qi, *et al.* [20], [22] extended FSW from straight-line segments to curve (polynomial) segments. Combining the ideas of DisConnAlg and XieConnAlg, Zhao, *et al.* [23] presented a linear time algorithm (denoted as ConnSegAlg) which outperforms FSW and XieConnAlg by generating fewer connected segments. However, it is difficult to directly convert XieConnAlg and ConnSegAlg of [21], [23] to build an optimal connected representation, especially in linear time complexity. For example, ConnSegAlg may lead to readjusting all previously generated segments to guarantee that the current segment reaches the maximum possible time stamp. This could require both higher costs in time and storage. Through scrutinizing the whole process, we have found that the problem can be simplified if we allow two consecutive segments to be connected *between* two time tags. That is, two segments semi-connect at a point of interval $[t_i, t_{i+1}]$ (Fig. 1 iii).

In fact, this research problem of constructing optimal semi-connected segments[5] was first solved by Imai and Hakimi, *et al.* [7], [8]. Hakimi, *et al.* [7] proposed an online method by searching the shortest path in a constructed "pipeline": Any line in the "pipeline" can guarantee that the reconstructed points is within a pre-defined error bound. The idea of "pipeline" is originally from Suri [25] for finding a minimum-link monotone polygonal chain within a simple polygon. In 2015, with the idea of [7], [15], Luo, *et al.* [14] designed an online PLA algorithm, which is termed as PipeMixedAlg in this article, to construct mixed (i.e., semi-connected and disconnected) segments to minimize the storage size for the outputs. It 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. As a byproduct, PipeMixedAlg can be modified to obtain a linear-time PLA algorithm, which is referred to as PipeOptConnAlg[6] in this article, for generating semi-connected segments under the prescribed maximum error bound.

There were also many PLA research upon variant error criterions [26], [27], [28], [29], [30] in computational geometry. These research results are either infeasible or less obvious to be used for generating semi-connected segments with linear time complexity. For example, Egyed, *et al.* [26] proposed a linear time algorithm for constructing one straight line stabbing a series of convex objects with maximum error bound. Agarwal, *et al.* [28] proposed a nonlinear time algorithm for solving connected PLA problem in x-monotone polygonal chains. In streaming setting, Abam, *et al.* [29] discussed the "restricted" line simplification problem with Fréchet Distance. In the end of [27], Guibas, *et al.* proposed a theoretical greedy algorithm to compute an "unrestricted" minimum link chain of fatten objects with disjointing. Different from the "pipeline" methodology, this

3. optimalPLR algorithm of [21] is denoted as DisConnAlg in this article.
4. Refer to Section 6.3.1 of [21] for details.
5. Those papers of [7], [8] were actually about "semi-connection" even though they used the term of "connection".
6. In fact, PipeOptConnAlg represents the kind of algorithms [7], [8], [14] based on "pipeline" idea.

theoretical result is derived from some structural properties of convex objects, which is quite similar to the mechanism of our proposed algorithm. At the end of their article, the authors raised an open problem on how to design an efficient implementation as their future work. From their results, it is still not clear how to obtain a linear-time semi-connected PLA algorithm.

There were many other research focused on constructing a pre-defined (or the smallest) number of straight line segments with minimal (or with pre-defined) holistic errors (i.e., L_2-norm) between the data points in the original time series and the corresponding data points in the representation lines, respectively. These include Top-down, Bottom-up [9] and evolutionary algorithms [4]. Bellman, *et al.* [2] proposed an optimal algorithm for online processing which used the dynamic programming framework to minimize the holistic approximation error recursively as new data points are introduced. Due to the time complexity of $O(kn^2)$ where k is the number of segments and n is the whole length of the time series, this algorithm is expensive for processing large data. As such, some heuristic strategies were suggested to improve the efficiency of data processing. This was evident in the Sliding Window (SW) algorithm [1] and improved SW (SWAB) [9].

To generate L_2-norm representations as in SW or SWAB, the number of processed data points needs to be fixed. Let k be the fixed-window size. Since a data stream is naturally unbounded in size, those constructed representations with SW or SWAB were not error-guaranteed on each individual data point and their approximation qualities were unjustified for the given error bound on a k-sized sliding window which was moving from the very left of the time series to the right. Consequently, this means that these algorithms would be insufficient in obtaining error-guaranteed analytical results for the given error bound [5], [6]. Furthermore, although any two adjacent disconnected segments can be connected using an added naive segment to link them, the total number of segments (storage) is far from the smallest (optimal). For example, as stated in Section 4, the average storage cost for the outputs of PipeOptConnAlg is about 67 percent of that DisConnAlg in all tested situations.

Our contributions. The contributions of this article can be summarized as follows:

1) A new algorithm SemiOptConnAlg that constructs the optimal number of semi-connected segments within a given error bound in L_∞ was proposed. This algorithm processes data streams *progressively* with linear time complexity and extremely low memory requirement. Different from the "pipeline" based algorithm like PipeOptConnAlg, SemiOptConnAlg is designed in the time-value space and optimized according to a series of reduction properties.
2) We tested on 43 data sets arbitrarily selected from a public domain [11] to verify our theoretical conclusions by comparing SemiOptConnAlg with DisConnAlg and PipeOptConnAlg in terms of time costs, memory costs, the number of constructed segments (storage), the sensitivity of δ values on error bound, and the scalability on data size. The results show that: SemiOptConnAlg has achieved lower time and memory costs and outputs the minimum number of semi-connected segments. The storage and representation lines outputted by SemiOptConnAlg are exactly the same as PipeOptConnAlg. Its scalability evaluations on time costs, maximum space usage and varied maximum errors outperform that of PipeOptConnAlg.

The rest of the article is organized as follows: Section 2 is the preliminaries; Section 3 gives the detailed method and the proofs on properties; Section 4 is the experimental results, including the performance comparisons among DisConnAlg, PipeOptConnAlg and SemiOptConnAlg; Section 5 concludes this article. Table 1 summarizes all the symbols frequently used throughout this article.

TABLE 1
Notations

Symbol	Meaning
δ	a given error bound on each data point
P	a time series $P = (p_1, p_2, \ldots, p_n)$
$p_i = (t_i, y_i)$	ith data point in time series
$\underline{p_i} = (t_i, y_i - \delta)$	data point with deleted tolerant error
$\overline{p_i} = (t_i, y_i + \delta)$	data point with added tolerant error
$line(p_i, p_j)$	line that passes point p_i and p_j
S_i	ith segment $S_i = (p_{s_i}, \ldots, p_{e_i})$
p_{s_i}, p_{e_i}	start and end points of S_i
$f_i(t)$	line segment of S_i
u_i, $y = u_i(t)$	max (upper) extreme line in S_i
l_i, $y = l_i(t)$	min (lower) extreme line in S_i
$\overline{cvx}(i)/\underline{cvx}(i)$	upper/lower convex hull after updating the ith point in a fragment
$\overline{cvx_i}$ (or $\underline{cvx_i}$)	upper (or lower) convex hull of S_i
$\underline{p_{cvx_i}}$ (or $\overline{p_{\overline{cvx_i}}}$)	the data points with the biggest time stamp of $\underline{cvx_i}$ (or $\overline{cvx_i}$)
$inter_l$ (or $inter_u$)	intersection of l_{i+1} (or u_{i+1}) with l_i or u_i

2 Preliminary

In this section, we provide a more focused discussion on how the methods of DisConnAlg [21], ConnSegAlg [23] and PipeOptConnAlg [14] relate to the discussed problem.

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 highest 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[7] which represent the bounds of all feasible lines are iteratively updated as new data is introduced. To efficiently update the 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 slope (gradient)

7. The extreme lines are the special lines with minimum and maximum slopes that represent the processed points.

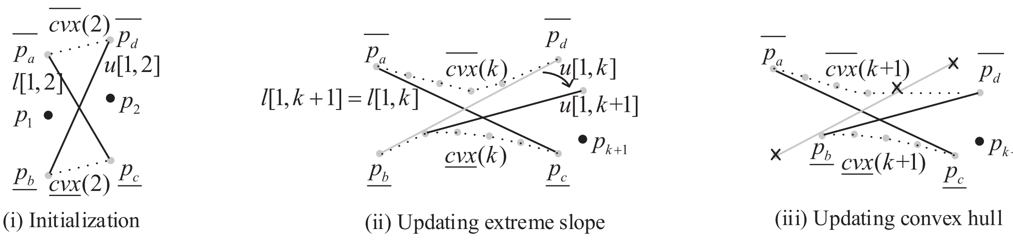
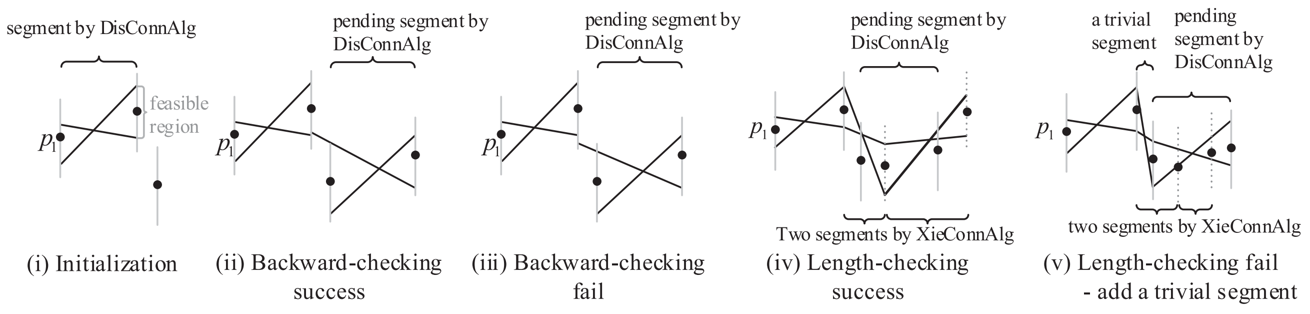


Fig. 2. The process of updating extreme lines and convex hulls.



segment by DisConnAlg
feasible region
pending segment by DisConnAlg
a trivial segment
Two segments by XieConnAlg
two segments by XieConnAlg
(i) Initialization
(ii) Backward-checking success
(iii) Backward-checking fail
(iv) Length-checking success
(v) Length-checking fail - add a trivial segment

``backward-checking" : Check whether two extreme lines of the pending segment are intersected in the feasible region of previous constructed segment.
``length-checking" : Check whether the last time stamp in two segments by XieConnAlg is further than that in the pending segment.

Fig. 3. The idea of "backward-checking" and "length-checking".

of the upper and lower extreme lines when p_k is approximated for $k \in \{1, \ldots, n-1\}$. Then, from [21],

$$\begin{cases} l[1,k+1] &= \max_{\overline{p_i} \in \overline{cvx}(k)} \{\frac{(y_{k+1}-\delta)-(y_i+\delta)}{(t_{k+1}-t_i)}, l[1,k]\}, \\ u[1,k+1] &= \min_{\underline{p_i} \in \underline{cvx}(k)} \{\frac{(y_{k+1}+\delta)-(y_i-\delta)}{(t_{k+1}-t_i)}, u[1,k]\}, \end{cases} \tag{2.1}$$

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 extreme points p_a, p_b, p_c and $\overline{p_d}$ are defined by

$$\begin{cases} \{a, c\} &= \arg\min_{1 \le i < j \le k} \frac{(y_j-\delta)-(y_i+\delta)}{(t_j-t_i)}, (a < c), \\ \{b, d\} &= \arg\min_{1 \le i < j \le k} \frac{(y_j+\delta)-(y_i-\delta)}{(t_j-t_i)}, (b < d). \end{cases} \tag{2.2}$$

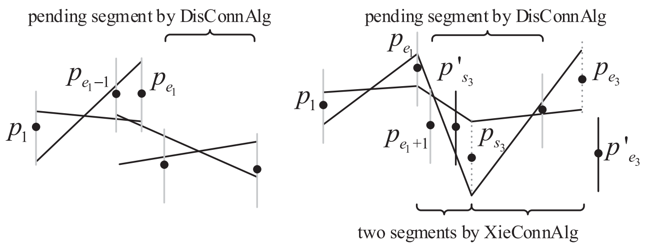
To update the extreme lines for a new data point, these convex hulls need to be maintained. Since the number of points in the convex hulls can be significantly smaller than the total number of data points in the relevant intervals, the convex hulls can be efficiently updated using the triangle check technique of [3] through recursively examining the three most recent consecutive points and then propagating backwards. When updating the upper/lower convex hull, if the middle point is above/below 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.

Fig. 2 illustrates the process of updating the extreme lines and convex hulls. Initially, both the extreme lines and convex hulls are from points p_1 and p_2 (Fig. 2i). When the next point p_{k+1} is introduced, we use Theorem 2 from [21] and Equation (2.1) to determine the new extreme lines $u[1, k+1]$ and $l[1, k+1]$. As seen in Figs. 2 ii and 2 iii, the new convex hulls $\overline{cvx}(k+1)$ and $\underline{cvx}(k+1)$ can be constructed by removing the earliest point in $\underline{cvx}(k)$ and deleting the two latest points in $\overline{cvx}(k)$, respectively.

DisConnAlg has been used for constructing connected segments in XieConnAlg. To make segment S_{i+1} connect to segment S_i, XieConnAlg employs DisConnAlg to build S_{i+1} beginning from the last data point p_{e_i} of S_i within the restricted region of two extreme lines of S_i at t_{e_i} (i.e., feasible region) as seen in Fig. 3i. To further minimize the number of connected segments, ConnSegAlg [23] uses two strategies, "backward-checking" and "length-checking", to build connected segments. As denoted in Fig. 3, "backward-checking" is used to check and decide if the current (pending) segment[8] constructed from DisConnAlg can be connected to the previous segment. Since DisConnAlg is an optimal algorithm, if the check succeeds, the current segment reaches the farthest data point and can approximate the data with the largest possible time stamp. Otherwise, it indicates that the last data point of this segment cannot be approximated with a single segment that connects to its previous segment. Whereas "length-checking" is to ensure if two connected segments constructed by XieConnAlg [21] can stretch further (i.e., larger time stamp) than the single segment generated by DisConnAlg and output the two connected segments if it is true. Otherwise, a trivial segment is added to connect the two disconnected parts.

As mentioned above, the "backward-checking" strategy only checks a single earlier point in the backwards direction. This is insufficient for obtaining an optimal solution. As indicated in Fig. 4i, the pending segment can intersect at an earlier time stamp. In this figure, "backward-checking" fails at point

8. A *pending* segment is the segment we are currently operating. It is called a *pended* segment once the updates are done.



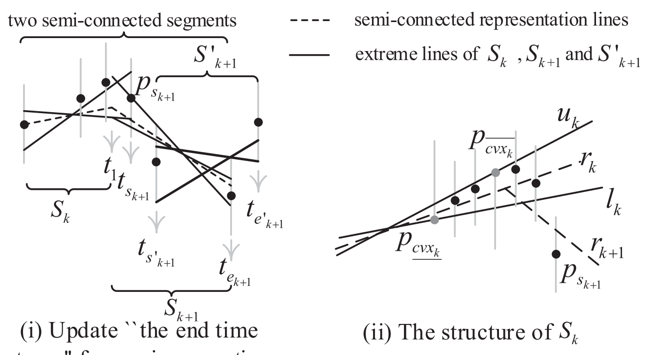
(i) Backward-checking can be improved (ii) Length-checking can be improved

Fig. 4. Insufficiency of previous strategies.

p_{e_1}, and the two disconnected segments are connected at point p_{e_1-1}. Similarly, the "length-checking" strategy is not designed to make the second connected segment reach the farthest possible point. As depicted in Fig. 4 ii, suppose that points p_{e_1+1} and p_{s_3} are checked by the "length-checking" strategy. There may exist a point p'_{s_3} between p_{e_1+1} and p_{s_3} that makes the further point p'_{e_3} approximated.

Motivated from the above issues on "backward-checking" and "length-checking", we introduce the concept of "semi-connectivity" for representing a time series with the smallest number of semi-connected segments. The semi-connected line segments are the connected lines that are not restricted to be connected at exact time indexes. Our general idea is to construct a pended segment that is semi-connected to the closest earlier segment and, at the same time, approximates the "furthest" possible successive data point (i.e., the data point with largest time tag). In next section, we shall show that this "locally maximal" property can be obtained by shrinking and extending a pending segment in the forwards and backwards directions.

Almost all existed algorithms on constructing optimal semi-connected segments, such as PipeOptConnAlg, are "pipeline" based. Its main idea is: (1) Construct a polygon between up-boundary and low-boundary; (2) Search the furthest visible window within this polygon; (3) Output the line intersecting the adjacent visible windows as a segment. For two point p and q, let the linear function passing through p and q be $line_{p,q}(t) = line(p,q)$. As depicted in Fig. 6, we initialize window w_1 to be the line with $\underline{p_1}$ and $\overline{p_1}$ as endpoints. To construct a polygon, let up-boundary $bu_1(t) = line_{\overline{p_1},\overline{p_2}}(t)$ and low-boundary $bl_1(t) = line_{\underline{p_1},\underline{p_2}}(t)$. For $i > 1$, set



(i) Update ``the end time stamp" for semi-connection (ii) The structure of S_k

Fig. 5. Semi-connectivity.

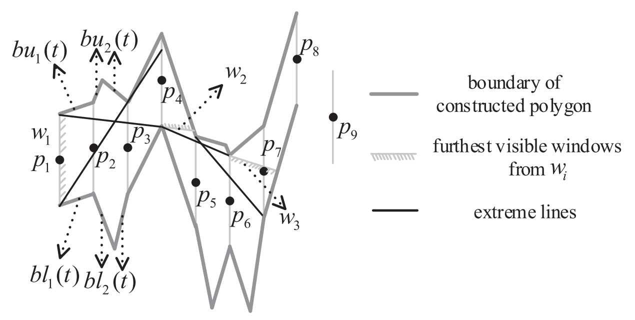


Fig. 6. The main idea of PipeOptConnAlg.

$$\begin{cases} bu_i(t) = \max(line_{\overline{p_i}\,\overline{p_{i+1}}}(t), \min(line_{\underline{p_{i-1}}\,\overline{p_i}}(t), line_{\underline{p_{i+2}}\,\overline{p_{i+1}}}(t))),\\ bl_i(t) = \min(line_{\underline{p_i}\,\underline{p_{i+1}}}(t), \max(line_{\overline{p_{i-1}}\,\underline{p_i}}(t), line_{\overline{p_{i+2}}\,\underline{p_{i+1}}}(t))). \end{cases}$$

To get window w_2 from w_1 from the polygon, it needs to construct two convex hulls from up-boundary and low-boundary of the polygon, respectively. The two convex hulls are then maintained for point p_5. Window w_2 is determined by the extreme lines of two convex hulls [14]. At this time, the region on the right of w_2 is not shined by w_1. Iteratively construct w_i from w_{i-1} until the time series is ended.

3 Optimal Semi-Connected Algorithm

Suppose we have the first two segments S_1 and S_2 constructed by DisConnAlg [21]. These two segments are "locally maximal" which reaches the furthest possible data points. If S_2 can not semi-connect to S_1, S_2 needs to be shrunk by deleting the latest data point and extended backwards. This process is repeated until S_2 semi-connects to S_1. Lemma 3.1 confirms that the modified S_2 is locally maximal as it approximates the "furthest" possible successive data point. By repeating the above process and updating S_2, we can construct two semi-connected segments.

Let $S'_{k+1} = (p_{s'_{k+1}}, \ldots, p_{e'_{k+1}})$ be the $(k+1)$th pending segment constructed by DisConnAlg. With the above steps, S'_{k+1} is "transformed" to $S_{k+1} = (p_{s_{k+1}}, \ldots, p_{e_{k+1}})$ which semi-connects to segment S_k by repeatedly removing its end point and adding an earlier point in S_k. Generally, $t_{s_{k+1}} \leq t_{s'_{k+1}}$ and $t_{e_{k+1}} \leq t_{e'_{k+1}}$ hold. In Lemma 3.1, we will show that it needs at least $k+1$ semi-connected segments to approximate time series $P = (p_1, p_2, \ldots, p_{e_{k+1}})$. As such, we say that this constructed S_{k+1} is "locally maximal" and approximates the "furthest" possible data point. For example, in Fig. 5i, the start and end time stamp of the $(k+1)$th pending segment S'_{k+1} are $t_{s'_{k+1}}$ and $t_{e'_{k+1}}$, respectively. After S_{k+1} is built by deleting $p_{e'_{k+1}}$ and extending $p_{s_{k+1}}$, the end time stamp of S_k becomes t_1, sacrificing a data point $p_{s_{k+1}}$ for semi-connecting to S_{k+1}. At this stage, S_{k+1} reaches the "furthest" possible data point.

3.1 Construct Semi-Connection Using Extreme Lines

Suppose that we have constructed semi-connected segments S_1, $S_2, \ldots,$ and S_k. Let S_{k+1} be the pending segment generated by DisConnAlg starting at data point p_{e_k+1}. Let the extreme lines of S_k and S_{k+1} be l_k, u_k and l_{k+1}, u_{k+1}, respectively. In the following, we will show how to update S_{k+1} to semi-connect S_k via extreme lines.

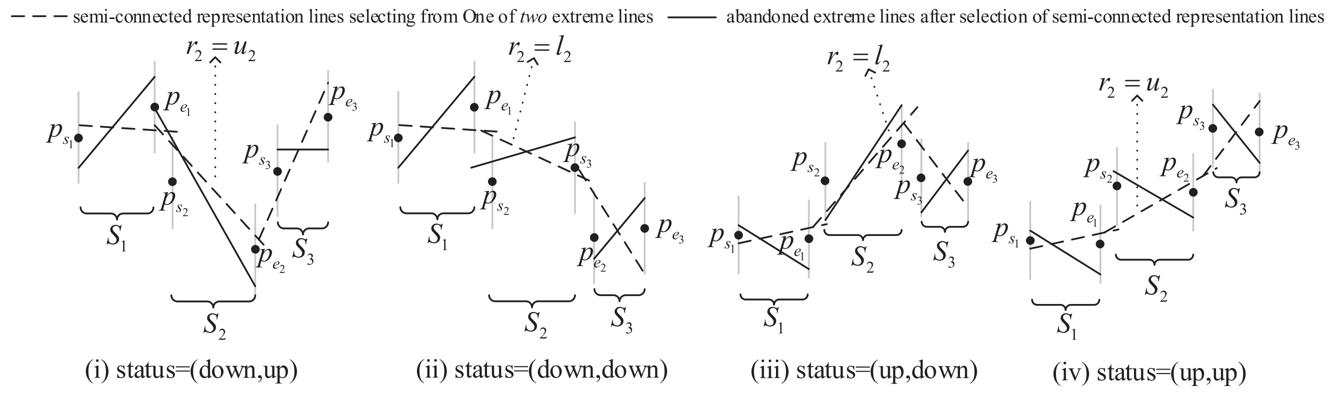


Fig. 7. The patterns of status.

Fig. 5 ii depicts the structure of S_k from three aspects: (1) The extreme lines l_k and u_k represent the lower and upper representation lines of S_k to semi-connect S_{k-1}; (2) $p_{\underline{cvx_k}}$ and $p_{\overline{cvx_k}}$ are on l_k and u_k , respectively; (3) S_k cannot approximate the data point $p_{s_{k+1}}$ as it is located either below l_k or above u_k.

The following reduction property is to locate the feasible representations of S_k and S_{k+1} for semi-connection. It indicates that a feasible representation line of S_{k+1} is semi-connected to a feasible representation line of S_k if and only if l_{k+1} is semi-connected to l_k (or u_{k+1} is semi-connected to u_k) according to the locations of $p_{s_{k+1}}$.

Property 3.1. *Let r_k and r_{k+1} be any feasible representation lines of S_k and S_{k+1}, respectively.*

1) *When $p_{s_{k+1}}$ is located below l_k, S_{k+1} is semi-connected to S_k if and only if r_{k+1} is semi-connected to l_k, and the intersection point of r_{k+1} and l_k is not on the left side of data point $p_{\underline{cvx_k}}$.*
2) *When $p_{s_{k+1}}$ is located above u_k, S_{k+1} is semi-connected to S_k if and only if r_{k+1} is semi-connected to u_k, and the intersection point of r_{k+1} and u_k is on the left side of data point $p_{\overline{cvx_k}}$.*

The proof of Property 3.1 is quite straightforward. First, the extreme lines of l_k and u_k are the special cases of representation line r_k. Second, if the intersection point of r_k and r_{k+1} is before (i.e., on the left side of) $p_{\underline{cvx_k}}$ (or $p_{\overline{cvx_k}}$), then the approximation error of r_{k+1} at point $p_{\underline{cvx_k}}$ (or $p_{\overline{cvx_k}}$) is above the predefined tolerance and contradicts to the hypothesis that r_{k+1} is a representation line.

In the process of deciding semi-connection, Property 3.1 implies that *only one extreme line and its data points on the right side of $p_{\underline{cvx_k}}$ (or $p_{\overline{cvx_k}}$) in S_k* need to be examined. However, the selection of which extreme lines are used as representations merely depends on the location of the start point of the next segment. Therefore, we use $status = (a, b)$, where $a, b \in \{up, down\}$, for the selection of representative extreme lines in SemiOptConnAlg. We define operation "!" as: if a is $down$ (or up), then $!a$ is up (or $down$). Once $status = (a, b)$ is instantiated, the selected semi-connected extreme lines from S_{k+1} and S_k are decided (and vice versa).

Example 1. Suppose three segments S_1, S_2 and S_3 are semi-connected. Let u_i, l_i, p_{s_i} and p_{e_i} be the upper and lower extreme line, the start point and the end point of segment S_i, respectively. One way of choosing semi-connected extreme lines is shown in Fig. 7. If $b =$"down" in $status = (a, b)$ of S_2, then semi-connected representation line r_2 is set to be the lower extremes line l_2 of S_2, or else set r_2 to be u_2.

One remaining question, which will be discussed next, is how to set the values of $status$ prior to the process of constructing semi-connected segments. This is very important for the efficiency of SemiOptConnAlg, as without it, the number of updated extreme lines can be much more than needed.

Without loss of generality, let $inter_l$ and $inter_u$ be the intersection points of l_{k+1} and u_{k+1} with l_k, respectively. The steps of constructing a semi-connected segment from S_{k+1} can be summarized into the following three cases:

Case 1. Both extreme lines of S_{k+1} are semi-connected to S_k. Under this situation: (i) For any feasible representation lines r_k of S_k and r_{k+1} of S_{k+1}, r_{k+1} is semi-connected to r_k; (ii) $status = (a, b)$ is determined by the positions of data points $p_{s_{k+1}}$ and $p_{s_{k+2}}$ (i.e., the start points of next segments). For example, if $p_{s_{k+1}}$ is above (up) line u_k and $p_{s_{k+2}}$ is below ($down$) line l_{k+1}, then $status = (up, down)$ with $a =$"up" and $b =$"down".

In Figs. 8i and 8 ii, both l_{k+1} and u_{k+1} can be used to approximate the points in S_k that are located after $inter_l$ and $inter_u$, respectively. In Fig. 8i, since $p_{s_{k+1}}$ is below (down) line l_k and $p_{s_{k+2}}$ is above (up) line u_{k+1}, $status = (down, up)$ holds. Therefore, the selected representations are l_k and u_{k+1} in this situation. Similarly, if $status = (down, down)$, the selected representations are l_k and l_{k+1} as shown in Fig. 8 ii.

Case 2. Only one of the extreme lines of S_{k+1} is semi-connected to S_k. In this situation, we would extend the segment S_{k+1} built from DisConnAlg backwards. This can be done by repeatedly adding the last data point of S_k into S_{k+1}. The extreme lines and the convex hulls of S_{k+1} can be updated by the Equations (2.1) and (2.2) backwardly. This process updates the extreme line of S_{k+1} that was not semi-connected to S_k into a semi-connected line.

In Fig. 8 iv, l_{k+1} can approximate the points in S_k that are located after $inter_l$. In this situation, u_{k+1} is neither intersected with l_k nor approximated to the last point in S_k that is located after $inter_u$. The updated u_{k+1} (denoted as u'_{k+1}) intersect with l_k at point $inter'_u$ and u'_{k+1} can approximate the last point in S_k located after $inter'_u$. If $status = (down, up)$, we select the

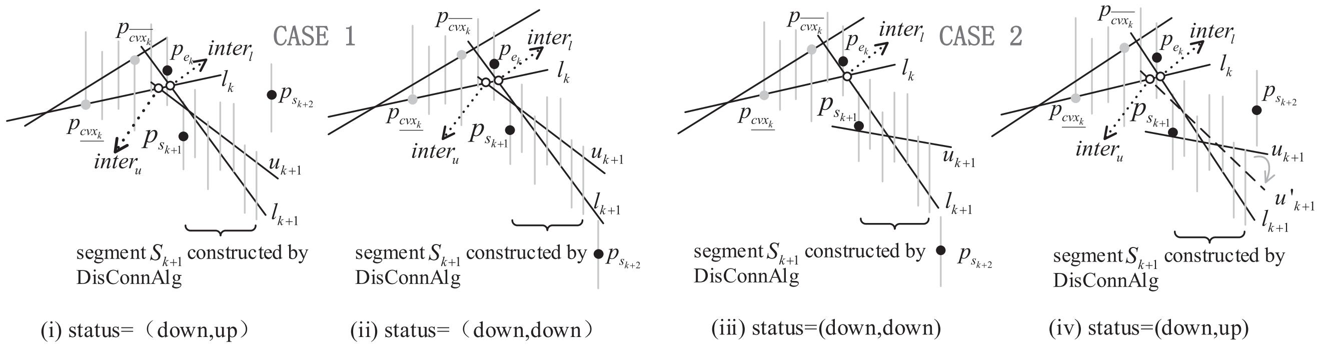


Fig. 8. S_k and S_{k+1} in Case 1 and Case 2.

updated extreme line u'_{k+1} as the representation line, making it "closer" to the next segment.

It should be noted that u_{k+1} would be neither selected as a representation nor used to construct any representations if $status = (down, down)$. Under this situation, u_{k+1} does not need to be updated as it will not affect the selection of semi-connected representations. Thus, as shown in Fig. 8 iii, we select the extreme line l_{k+1} as the representation line.

Like the situation of Case 1, $status = (a, b)$ is determined by the positions of data points $p_{s_{k+1}}$ and $p_{s_{k+2}}$.

Case 3. Both extreme lines of S_{k+1} are not semi-connected to S_k. To make S_{k+1} semi-connected to S_k, S_{k+1} need to be shrunk from the side of data arrival and extended backwardly to see if the updated S_{k+1} is semi-connected to S_k. If not, the procedure is repeated until the updated S_{k+1} satisfies the condition described in Case 2. From Property 3.2, $status = (a, b)$ is determined by the value of its first parameter. That is, if $p_{s_{k+1}}$ is above (up) line u_k, then $p_{s_{k+2}}$ is below (down) line l_{k+1}.

In Fig. 9, both l_{k+1} and u_{k+1} cannot semi-connect to l_k when S_{k+1} covers the data point $p_{s_{k+2}-1}$. After shrinking (removing) the data point $p_{s_{k+2}-1}$ from S_{k+1}, the updated extreme line l'_{k+1} is intersected with l_k and it can be used to approximate the data point p_{e_k}. At this time, u_{k+1} is not changed and Case 3 is converted into Case 2. Note that $status = (down, down)$ is updated into $status = (down, up)$ as confirmed by Property 3.2.

Property 3.2. *Suppose that S_{k+1} is not semi-connected to S_k. Then $status = (up, down)$ or $status = (down, up)$.*

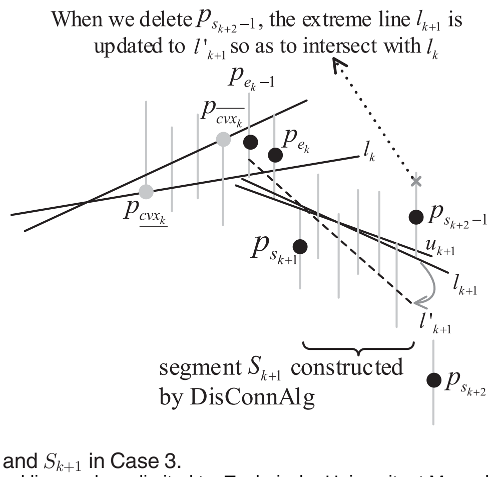

When we delete $p_{s_{k+2}-1}$, the extreme line l_{k+1} is updated to l'_{k+1} so as to intersect with l_k
segment S_{k+1} constructed by DisConnAlg

Fig. 9. S_k and S_{k+1} in Case 3.

Proof. We only illustrate on the condition of $status = (down, up)$. The proof for $status = (up, down)$ is very similar to that of $status = (down, up)$.

Since $p_{s_{k+1}}$ is below (down) line l_k and l_{k+1} is not semi-connected to S_k, the process of removing the last data point $p_{e_{k+1}}$ of S_{k+1} and extending S_{k+1} backwards can decrease the slops of l_{k+1} and u_{k+1}. Again, since the lower extreme line of trivial segment $S_t = (p_{e_k}, p_{e_k+1})$ is always semi-connected to S_k, there exist a minimum number $h > 0$ such that the extreme lines of $S' = (p_{s_{k+1}}, p_{s_{k+1}+1}, \ldots, p_{e_k-h})$ are semi-connected to S_k. Let u' be the upper extreme line of S'. Since data point p_{e_k-h+1} is in S_{k+1} but not in S', p_{e_k-h+1} is located above (up) line u', leading to $status = (down, up)$. □

As mentioned in Case 1 and Case 2, $status = (a, b)$ is determined by the positions of data points $p_{s_{k+1}}$ and $p_{s_{k+2}}$, which, in turn, is determined by S_k and S_{k+1}; the disjoint segments generated from DisConnAlg of [21]. From the above statement, $status = (a, b)$ can be instantiated after S_k and S_{k+1} are built. However, it must be before the process of the semi-connection check or the update of S_{k+1}. While in Case 3, $status = (a, b)$ can be instantiated after S_{k+1} being built and the semi-connection check with S_{k+1}.

In summary, we have the following conditions for Case 2 and Case 3. This suggests that the decision of Case 2 or Case 3 can be done by checking one extreme line. Subsequently, $status = (a, b)$ can be instantiated by checking one extreme line.

1. Extension strategy: In Case 2, S_{k+1} needs to be extended backwards.
   This strategy exists in Case 2 if (i) $p_{s_{k+1}}$ is below l_k and l_{k+1} is semi-connected to l_k, or (ii) $p_{s_{k+1}}$ is above u_k and u_{k+1} is semi-connected to u_k.
2. Shrinking strategy: In Case 3, S_{k+1} needs to be shrunk from the last data point and then extended backwards. This strategy exists in Case 3 if (i) $p_{s_{k+1}}$ is below l_k and l_{k+1} is not semi-connected to l_k, or (ii) $p_{s_{k+1}}$ is above u_k and u_{k+1} is not semi-connected to u_k.

3.2 Algorithm

In the following, we explain the pseudo code of SemiOptConnAlg (Algorithm 1). The outline of Algorithm 1 is summarized into the following steps.

Step 1. (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-segment $k = 1$. Based on the position of data point p_{t_e+1} that is relevant to u or l, initialize $status = (a, b)$ and $extrm$ (the selected extreme line). Set $a = \phi$, $b = down$, and $extrm = l$ if p_{t_e+1} located below line l. Set $b = up$, $extrm = u$ if p_{t_e+1} located above line u. Set $inter_0$ to be the point of $extrm$ at time t_1 and store it.

Step 2. (Lines 6 and 7). Construct the disconnected segment (S', u', l', t'_s, t'_e) by DisConnAlg starting from $t'_s = t_e + 1$ as the current pending segment.

Step 3. (Lines 8-12). This part is relevant to Case 3. In Case 3, none of the extreme lines of l' and u' are semi-connected to the previous segment. Our strategy is to transform Case 3 to Case 2 or Case 1.

The shrinking and extending process as described in Case 3 will be conducted. Thus, p'_e will be updated to be the shrunk time tag and p'_s will be updated to be the extended time tag. Based on Property 3.2, update $status$ as $a = b, b = !a$. Go to Step 4.

If it is not Case 3, go to Step 4.

Step 4. (Lines 13-17). This part is relevant to Case 2. In Case 2, only one of the extreme lines of l' and u' are semi-connected to the previous segment.

The extending process is conducted as described in Case 2. p'_s is updated to be the extended time tag and p'_e is not changed. Based on the position of data point $p_{t'_e+1}$ that is relevant to u' or l', update $status$, i.e, set $a = b$ and $b = down$ if $p_{t'_e+1}$ located below line l' or $b = up$ if $p_{t'_e+1}$ located above line u'. Go to Step 6.

If it is not Case 2, go to Step 5.

Step 5. (Lines 18-20). This part is relevant to Case 1 where both extreme lines of l' and u' are semi-connected to segment S. According to Property 3.1, it implies that both extreme lines of l' and u' are semi-connected to extreme line l or extreme line u, respectively. Based on the position of data point $p_{t'_e+1}$ that is relevant to u' or l', update $status$, i.e, set $a = b$ and $b = down$ if $p_{t'_e+1}$ located below line l' or $b = up$ if $p_{t'_e+1}$ located above line u'.

Step 6. (Lines 21-31). If $b = down$ in $status$, store the intersection point $inter_k$ of l' and $extrm$, and update $extrm = l'$. If $b = up$ in $status$, store the intersection point $inter_k$ of u' and $extrm$, and update $extrm = u'$. Update $k = k + 1$ and $t_e = t'_e$.

Step 7. Repeat the above from Step 2 for the last segment until all data points have been processed. Output the stored data.

3.3 Properties

In the following, we first prove that SemiOptConnAlg is optimal which generates the least number of semi-connected segments. We then measure the storage and the actual running time of SemiOptConnAlg and DisConnAlg. Lastly, we briefly indicate that SemiOptConnAlg and PipeOptConnAlg can generate the exact same representing segments.

To prove SemiOptConnAlg is optimal, we need the following lemma. It says that each currently generated segment in SemiOptConnAlg reaches the farthest possible data point by preserving semi-connectedness.

Algorithm 1. Function SemiOptConnAlg (P, δ)

```
Input:
    time sequence $P = (p_1, p_2, \ldots, p_n, \ldots)$, error bound $\delta$
Output:
    The semi-connected points $(inter_0, inter_1, inter_2, \ldots)$
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 $p_{t_e+1}$ is under $l$, then set $extrm$ to be $l$. Other-
    wise, set $extrm$ to be $u$
 4: Set $status$: $a = \Phi$, $b = down$ if $p_{t_e+1}$ located below line $l$ or
    $b = up$ if $p_{t_e+1}$ located above line $u$.
 5: Store the intersection $inter_0$ of $extrm$ at time tag $t_1$.
 6: while (not finished segmenting time series) do
 7:    Use DisConnAlg to generate the next disconnected seg-
       ment $(S', u', l', t'_s, t'_e)$ 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 (Case 3) then
 9:       do "Shrinking strategy"
10:       Update $t'_e$ and $t'_s$
11:       Update $status$ by $a = b, b = !a$
12:    end if
13:    if (Case 2) then
14:       do "Extending strategy"
15:       Update $t'_e$ and $t'_s$
16:       Update $status$ according to the position of data point
          $p_{t'_e+1}$ that relevant to $u'$ or $l'$.
17:    end if
18:    if (Case 1) then
19:       Update $status$ according to the position of data point
          $p_{t'_e+1}$ that relevant to $u'$ or $l'$
20:    end if
21:    if ($b == Down$) then
22:       Store the intersection point $inter_k$ of $l'$ and $extrm$
23:       update $extrm = l'$
24:    end if
25:    if ($b == Up$) then
26:       Store the intersection point $inter_k$ of $u'$ and $extrm$
27:       update $extrm = u'$
28:    end if
29:    $k = k + 1$
30:    $t_e = t'_e$
31: end while
```

Lemma 3.1. *Given a time series $P = (p_1, p_2, \ldots, p_n)$ and an error bound δ. Let $S_1, S_2, \ldots, S_k$ be the semi-connected segments constructed up to point p_{e_k} from SemiOptConnAlg where $e_k < n$. That is, S_k is semi-connected with S_{k-1} but it cannot approximate data point p_{e_k+1}.*

Proof. Clearly, the claims are true for $k = 1$ as S_1 is maximum and constructed from DisConnAlg. Suppose that the claims are true for $k < h$ where $h > 1$. We prove the claims are true for $k = h$ in the following.

If the lemma does not hold true for $k = h$, we assume that there exists an optimal semi-connected solution $S' = (S'_1, S'_2, \ldots, S'_m)$ where $e'_m = e_h + 1$ and $m \leq h$ hold for the last data point $p_{e'_m}$ of S'_m.

TABLE 2
Number of segments

Data set (Order number)	length	Number of segment		Processing time (ms)					Data set (Order number)	length	Number of segment		Processing time (ms)				
		Semi=Pipe	Dis / 2k-1	$T(D)$	$T(P)$	$T(S)$	$T(DInS)$	$T(PinS)$			Semi=Pipe	Dis / 2k-1	$T(D)$	$T(P)$	$T(S)$	$T(DInS)$	$T(PinS)$
50words(1)	123305	2490	1625/3249	23.7±1.2	185.5±24.6	59.9±4.2	43.4±2.8	16.5±1.4	CricketY(23)	117391	5095	3549/7097	21.3±2.8	164.9±10.3	49±3.3	34.8±2.4	14.2±0.9
Adiac(2)	69207	2264	1532/3063	13.5±0.7	108.9±4.3	35.9±9.8	24.3±6.7	11.6±3.1	CricketZ(24)	117391	5245	3669/7337	19.7±1.6	163.7±7.5	47.5±0.2	34.3±0	13.2±0.2
Beef(3)	14130	90	61/121	3±0.4	19.4±1.6	6.5±0	6.5±0	0±0	Diatom(25)	105877	2309	2045/4089	25.9±3.8	152.2±15	32.7±0.2	28.6±0.1	4.1±0.1
Coffee(4)	8036	560	476/951	1.4±0.5	12.8±1.4	3.1±0.6	2±0.1	1.1±0.5	ECGFiveDays(26)	117958	8572	7419/14837	24.8±3.3	158.4±8.5	44.2±1.5	36.3±1.2	7.9±0.3
OliveOil(5)	17131	450	391/781	3.5±0.7	20.4±1.9	6.6±3.3	5.7±2.8	0.9±0.5	Haptics(27)	336645	4624	3028/6055	64.1±4.2	433.1±22.1	183±10.2	102±5.8	81±4.4
CBF(6)	116100	65625	40183/80365	13±1.4	150.4±11.5	51.1±24.5	24.9±12.2	26.2±12.3	InlineSkate(28)	1035650	5013	4240/8479	153.8±4.4	1307.9±66.5	294.4±23.7	216.1±16	78.3±7.7
ECG200(7)	9700	1959	1376/2751	1.8±0.4	13.3±1.3	5.5±1.3	3.4±1	2.1±0.3	MedicalImages(29)	76000	5937	4633/9265	14.3±0.5	113.8±12.2	28±5.8	20.7±4.4	7.3±1.4
FaceAll(8)	223081	38693	29994/59987	34.4±1.4	328.7±19.4	104.3±13.2	54.4±14.2	50±1.1	MoteStrain(30)	106420	11085	7486/14971	20.2±0.9	138.6±11.9	46.1±3	34.6±2.2	11.5±0.8
FaceFour(9)	30888	4972	3632/7263	4.8±0.4	40.2±2.7	10.9±0.4	6.5±0.3	4.4±0.1	SonyAIBOI(31)	42671	12765	10583/21165	7.1±2.3	61.4±4.7	15.5±1.2	7.3±0.5	8.2±0.7
FISH(10)	81201	993	854/1707	19.1±3.1	120.5±6.3	30.8±0.6	23.3±0.6	7.5±0	SonyAIBOII(32)	62899	21042	17088/34175	7.6±0.5	93.5±11.3	23.7±1.5	10.9±0.7	12.8±0.8
GunPoint(11)	22650	1160	983/1965	5.2±1.1	29.4±3.1	7.3±0.3	6.8±0.3	0.5±0	Symbols(33)	397006	8751	7174/14347	77.2±4	553.4±19.8	148.4±9	111.1±6.8	37.3±2.2
Lighting2(12)	38918	1274	866/1731	5.5±0.5	48.8±4.9	11.8±0.4	8.9±0.2	2.9±0.2	TwoLeadECG(34)	94537	11488	9510/19019	18±2.4	135.8±11.3	34.5±3.2	23.8±0.8	10.7±2.4
Lighting7(13)	23360	1801	1175/2349	3.2±0.4	30.2±1.2	8.9±1.1	6.8±0.8	2±0.3	WordsSynonyms(35)	172898	5380	4124/8247	34.8±2.9	239.8±11.2	74.6±2.9	56.1±2.2	18.5±0.7
OSULeaf(14)	103576	4466	3795/7589	21.8±3.1	151±12	41.3±4.7	28.9±3.3	12.4±1.4	CinCECG(36)	2263201	14731	12001/24001	365.9±6.5	2770.8±59.7	711.9±34.9	572.1±28.2	139.8±6.7
SwedishLeaf(15)	80626	5707	4415/8829	16.3±1	126.8±7.9	31.7±1.9	22.8±1.4	9±0.5	FacesUCR(37)	270601	47834	37101/74201	42.8±2.4	385±14	129.3±9.7	67.8±6.2	61.5±3.5
synthetic(16)	18301	10643	6707/13413	1.8±0.6	25.9±3	6.7±1.7	3.3±0.9	3.4±0.8	ItalyPower(38)	25726	9713	7376/14751	3±0	34.3±2	8.4±0.2	4.7±0.1	3.7±0.1
Trace(17)	27600	759	627/1253	4.5±1.9	38.9±4.2	7.8±0.1	6.4±0.1	1.4±0	MALLAT(39)	2403626	61032	51763/103525	585.7±93.6	2708±28.7	819.8±42.7	621.9±31.9	197.9±10.8
TwoPatterns(18)	516000	214311	129045/258089	69.2±1.9	691.9±46.2	207.9±5.2	137.5±2.6	70.4±2.6	StarLight(40)	8441901	72023	58599/117197	1974.8±155.9	10926.3±298.2	2809.5±17.7	2282.2±15.9	527.3±1.8
yoga(19)	1281000	41592	36078/72155	273±21.9	1598.7±177.5	450.3±33.5	333.7±23.8	116.6±9.7	uWaveGestX(41)	1131913	32117	25709/51417	253.8±15.9	1305.2±19.3	424.9±9.7	315.6±6.2	109.3±3.5
wafer(20)	943092	62836	38309/76617	220.4±12	1440.5±33.1	443.9±26.1	403.4±25.3	40.5±0.8	uWaveGestY(42)	1131913	32374	25396/50791	251.2±13.6	1288.4±13.8	429.3±1.5	318.2±0.8	111.1±0.7
Chlorine(21)	641281	106495	62393/124785	107.2±4.4	747.1±34.6	330±8.7	197.3±5.3	132.7±3.4	uWaveGestZ(43)	1131913	34087	27169/54337	256.2±21.4	1432.2±30.2	422.7±5.4	313.3±2.3	109.4±3.1
CricketX(22)	117391	5247	3672/7343	20.2±2.1	162±15.1	46.5±1.6	33.9±0.8	12.6±0.8									

From the hypothesis stated above, let $\tilde{S}_{h-1}$ be the $(h-1)$th segment constructed from SemiOptConnAlg before the construction of S_h (i.e., $\tilde{S}_{h-1}$ is the segment of S_{h-1} without shrinking). For the last data point $p_{\tilde{e}_{h-1}}$ of $\tilde{S}_{h-1}$, according to the assumption and the hypothesis, $e_{h-1} \leq \tilde{e}_{h-1}$, $e'_{m-1} \leq \tilde{e}_{h-1}$ and $h-1 \leq m-1$ hold. Therefore, $h = m$ holds.

On the other hand, since the set of data points of S_h is a subset of S'_h, S'_h can semi-connect to $\tilde{S}_{h-1}$. The segment constructed by DisConnAlg started from point $p_{\tilde{e}_{h-1}+1}$ can reach point p_{e_h+1} and can semi-connect to $\tilde{S}_{h-1}$, which is contradictory to the hypothesis that point p_{e_k+1} is not in S_k. Thus, the claims are proven. □

Theorem 3.1. *SemiOptConnAlg is a linear-time algorithm that generates minimum number of semi-connected segments for any given time series $P = (p_1, p_2, \ldots, p_n)$ and error bound δ.*

Proof. Since the proof on the least number of segments is directly from Lemma 3.1, we only give the proof on time complexity in the following.

As shown in Algorithm 1, SemiOptConnAlg employs DisConnAlg to construct the pending disconnected segment and checks whether it is semi-connected to the previous segment. This check is implemented by the three designed cases discussed in Section 3.1. In the while loop, the costs of constructing/updating a new segment are of three parts: the cost of constructing a new disconnected segment, the cost of shrinking data points and the cost of extending data points. Suppose the pending disconnected segment S_{k+1}. Let the number of points in S_{k+1} be m_{k+1}. According to the illustrations on page 10 of paper [21], the cost of constructing S_{k+1} is dominated by the costs of updating convex hulls and is bound by $(4+c_1)m_{k+1}$ [21], where c_1 is a constant number that summarizes other costs in the process. In the process of shrinking data points (let the number of shrunk points be $m' \leq m_{k+1}$), we do not need to update the convex hulls in order to compute the extreme lines as the extreme lines have been stored in advance. So the cost can be denoted as $c_2 m'$.

Similarly, the cost of extending S_{k+1} to semi-connect to S_k is bound by $(4+c_1)(m_{k+1} - m' + m_k)$ where m_k is the number of points in S_k. Thus, the total cost of constructing a new semi-connected segment is bound by $(4+c_1)m_{k+1} + c_2 m' + (4+c_1)(m_{k+1} - m' + m_k) \leq (6+2c_1)m_{k+1} + (4+c_1)m_k$. We therefore conclude that the time complexity of SemiOptConnAlg algorithm is $O(n)$. □

Let the number of segments constructed from SemiOptConnAlg, ConnSegAlg and DisConnAlg be m, h and k, respectively. According to Theorem 2.1 [23], $k \leq h \leq 2k-1$ holds. Then $k \leq m \leq 2k-1$ holds as $m \leq h$ holds. Based on this, we have the bounds on storage costs for SemiOpt ConnAlg.

Property 3.3. *Let r and r' be the number of outputted data points of SemiOptConnAlg and DisConnAlg, respectively. Then $(r'+2)/2 \leq r \leq r'$.*

Since each disconnected segment needs to store two points while the whole semi-connected segments only need to store the turning points plus the start and end points, we have $r = m+1$ and $r' = 2k$. Thus, $(r'+2)/2 \leq r \leq r'$ holds.

Property 3.4 indicates that the running time of SemiOptConnAlg is within the four times that of DisConnAlg.

Property 3.4. *For a given time series P and an error bound δ, let $t(S)$ and $t(D)$ be the processing time of SemiOptConnAlg and DisConnAlg, respectively. Then $t(D) \leq t(S) \leq 4 \times t(D)$ holds.*

Proof. Since SemiOptConnAlg uses DisConnAlg as basic operations, we can rewrite $t(S)$ into two parts. That is,

$$t(S) = t(DInS) + t(PInS),$$

where $t(DInS)$ and $t(PInS)$ are the time cost on calling DisConnAlg and the time cost on the post processing phase, respectively.

Obviously, the best case of SemiOptConnAlg is that each pending disconnected segment can semi-connect to the previous one without shrinking any point. At this time, $t(D) = t(DInS)$ and $t(PInS) \geq 0$ hold due to the extra computations on the"backward-checking" process. Thus, $t(D) \leq t(S)$ holds.

In the worst case of $t(DInS)$, each pending disconnected segment constructed by DisConnAlg cannot semi-connect to its previous segment until only the start point of this segment is left after repeatedly shrinking. DisConnAlg is required to build a segment start from the

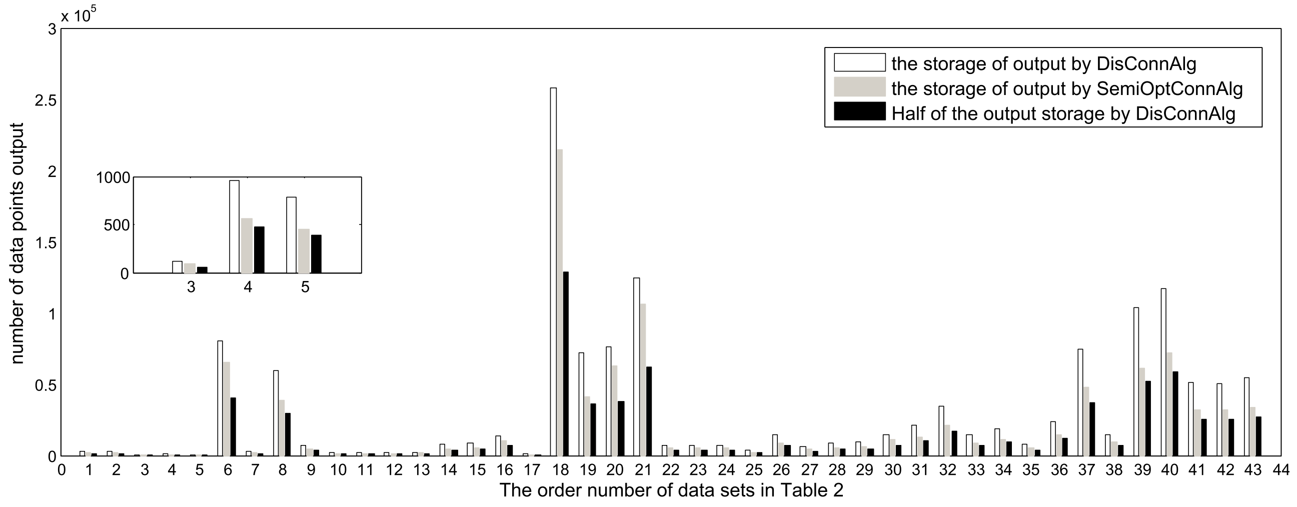


Fig. 10. Storage costs of SemiOptConnAlg and DisConnAlg on 43 data sets.

next data point. This may cause DisConnAlg to build segments starting from two adjacent data points. Therefore, $t(DInS) \leq 2 \times t(D)$.

In the worst case of $t(PInS)$, the shrunk segment only deletes the latest point and is extended to semi-connect with the previous segment at the second data point. $t(PInS)$ is determined by the number of points in the shrunk and extended segment, which is bound by the length of two disconnected segments from DisConnAlg. So $t(PInS)$ is bound by $2 \times t(D)$. Thus, we have $t(S) \leq 4 \times t(D)$. □

Let k be the number of segments generated from DisConnAlg on time series P. From the proof of Property 3.4, we know that, in the process of constructing semi-connected segments, the number of calls on DisConnAlg from SemiOptConnAlg is bound by $2k$.

Next, we will briefly indicate that SemiOptConnAlg and PipeOptConnAlg can generate the exact same representing segments. Since both SemiOptConnAlg and PipeOptConnAlg are optimal and generate the same number of semi-connected representative segments for any given time series P and error bound δ, these two algorithms will construct the same extreme lines.[9] Taken the extreme lines as the outputted semi-connected segments, we therefore have the following property.

Property 3.5. *For any time series $P = (p_1, p_2, \ldots, p_n)$ and an error bound δ, SemiOptConnAlg and PipeOptConnAlg generate the same set of representative segments.*

4 EXPERIMENTS

In this section, we evaluate the performance of SemiOptConnAlg by experimentally comparing it against DisConnAlg and PipeOptConnAlg on a wide range of data sets. The comparisons are made in terms of the number of segments constructed, storage costs, time costs, memory costs, scalability and representation quality.

9. Otherwise, these two algorithms would output different number of semi-connected segments for a certain given time series P' and error bound δ'.

In the experiments, we use the original source codes of DisConnAlg and PipeOptConnAlg obtained from the authors of [14], [21], respectively. All algorithms are implemented with C ++. All experiments are performed on a laptop with a 2.4 GHz CPU and 12G memory. Our tests use 43 data sets selected from the UCR time series archive [11]. These data sets have been widely used for the evaluation of time series algorithms.

On testing, each algorithm is run 10 times for each data set and the average of the results and its standard deviation are reported. We set the error bound to be 2.5, 5 and 10 percent of the value range of testing data sets, respectively. We only display the results for the situation of $\delta = 2.5\%$ since other results are very similar.

Storage Costs, Time Costs and Memory Costs. The tested results on the number of constructed segments and storage costs are listed in Table 2 and Fig. 10. Let k be the number of segments generated from DisConnAlg and r_k be its storage cost. Then $r_k = 2k$, which stores the number of end points of the constructed segments. Table 2 confirms:

1) Although the methods of SemiOptConnAlg and PipeOptConnAlg are very different, they generate the same number of outputs on all tested data sets;
2) The number of semi-connected segments constructed by SemiOptConnAlg is within $[k, 2k-1]$, which is confirmed from Theorem 3.1. More specifically, the number of segments constructed by SemiOptConnAlg is about 134 percent of that constructed by DisConnAlg on the tested data sets;
3) As claimed in Property 3.3, the storage costs of SemiOptConnAlg are less than that of DisConnAlg and are bound in $[(r_k+2)/2, r_k]$ as shown in Fig. 10. The average storage cost for the outputs of SemiOptConnAlg is about 67 percent of that from DisConnAlg on 43 tested data sets.

Let $t(D)$, $t(S)$ and $t(P)$ be the time costs of DisConnAlg, SemiOptConnAlg and PipeOptConnAlg, respectively. As having been discussed in the previous section,

$$t(S) = t(DInS) + t(PInS),$$

where $t(DInS)$ is the time cost on generating disconnected segments by DisConnAlg and $t(PInS)$ is the time cost on

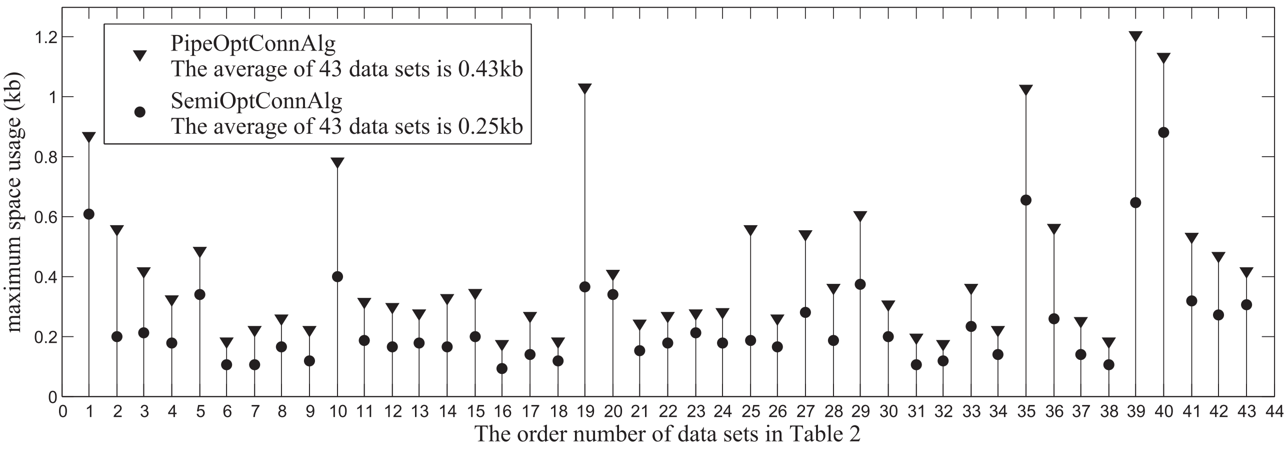


Fig. 11. Memory costs of SemiOptConnAlg and PipeOptConnAlg on 43 data sets.

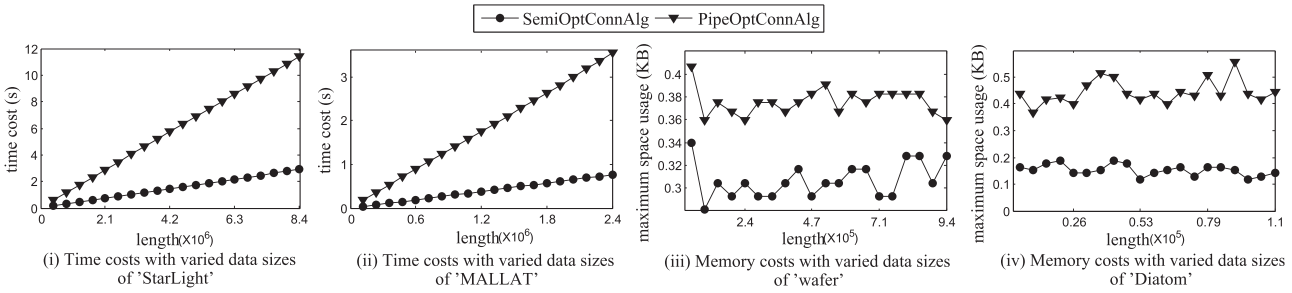


Fig. 12. Time and memory costs with varied length.

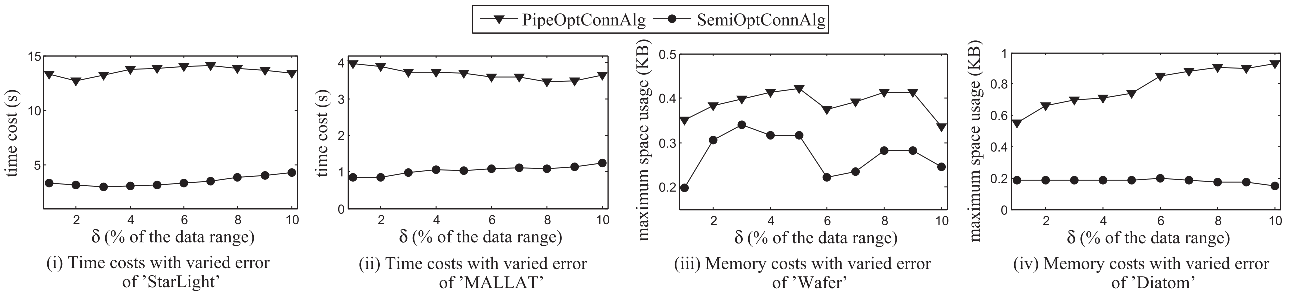


Fig. 13. Time and memory costs with varied error.

the post processing phase. All the results on time costs are listed in Table 2. These results have confirmed that

1) Property 3.4 and 3.5 are verified from all tested data sets;
2) The average time costs of SemiOptConnAlg and PipeOptConnAlg are about $202ms$ and $713ms$, respectively. SemiOptConnAlg is about 3 times faster than that of PipeOptConnAlg. The extra time consumption of PipeOptConnAlg comes from that (i) SemiOptConnAlg is designed in time domain, without constructing a polygon as in PipeOptConnAlg; (ii) SemiOptConnAlg can immediately output a segment from one extreme line.

We compare SemiOptConnAlg against PipeOptConnAlg on memory costs in Fig. 11. These results indicate that the average maximum memory cost of PipeOptConnAlg and SemiOptConnAlg is about 0.43 kb and 0.25 kb, respectively; The memory cost of PipeOptConnAlg can be 2.5 times larger than that by SemiOptConnAlg on a tested data set.

Scalability. On scalability tests, we designed two experimental strategies: (I) Data lengths impact on time and memory costs. We divided the testing data set into 20 fragments and computed the accumulated time cost and the maximum memory cost on each fragment. (II) Error bounds impact on time and memory costs. We computed the time and maximum memory costs on the varying error bounds from 1 to 10 percent of the testing data set range.

We only present the results on "StarLight", "MALLAT", "wafer" and "Diatom" for the sake of brevity. "StarLight" and "MALLAT" are the top two largest among the 43 data sets which include 8441901 and 2403626 data points, respectively. "wafer" and "Diatom" are the data sets which achieve the minimum (i.e., 1.2) and the maximum (i.e., 3) memory cost ratios between SemiOptConnAlg and PipeOptConnAlg among the 43 data sets. The tested results on strategies (I) and (II) are shown in Figs. 12 and 13, respectively. These results indicate:

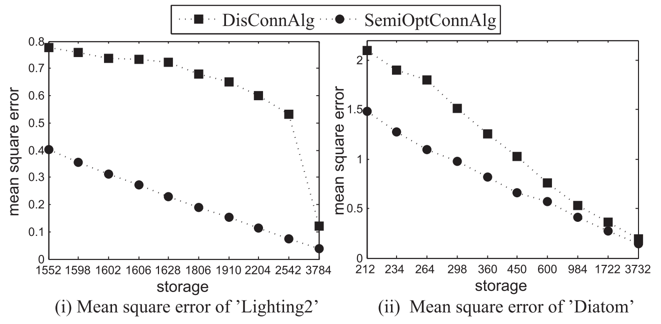


Fig. 14. L_2-error on the representations of DisConnAlg and SemiOptConnAlg.

1) With the data length increasing, the time cost of SemiOptConnAlg and PipeOptConnAlg are both increasing, but the variation tendency of SemiOptConnAlg is much slower than that of PipeOptConnAlg. The differences on their memory costs fluctuate around 0.32 kb, 0.38 kb, 0.15 kb, and 0.45 kb on "wafer" and "Diatom", respectively.
2) With the error bound increasing, the time costs of SemiOptConnAlg and PipeOptConnAlg are close to constant, but SemiOptConnAlg is about 3 times quicker than that of PipeOptConnAlg. The memory costs of SemiOptConnAlg are about 20 to 88 percent of that PipeOptConnAlg.

Representation Quality. The representation quality of SemiOptConnAlg is evaluated from two perspectives in the following: the minimized mean square error on the outputted segments and the accurate representation on ECG data.

Considering that many research results on PLA algorithms are on mean square error (L_2-error), we first illustrate the representation advantages of SemiOptConnAlg against DisConnAlg through comparing their L_2-error on the approximation of original data points under the fixed storage costs. This test strengthened and detailed the testing results in Figs. 11a and 11b of [14], specially for PipeOptConnAlg.

We adjust L_∞ error from 1 to 10 percent of the data set range separately to make SemiOptConnAlg and DisConnAlg have the same storage on the sets of constructed segments. We then compute the L_2 errors on their outputted segments. The results on "Lighting2" and "Diatom" are depicted in Fig. 14. For the same error bound in L_∞, it shows that: (1) Under the same storage costs, the L_2 errors of semi-connected segments are always smaller than those of disconnected segments. (2) The L_2 errors on semi-connected segments and disconnected segments are decreasing as the storage sizes increase, but the tendency variation of semi-connected is slower than disconnected. Similar test results were obtained for the remaining tested data sets.

The next test results are about the representation of ECG data picked up from MIT [24] which is depicted with the thick grey curve in Fig. 15i. We construct segments with

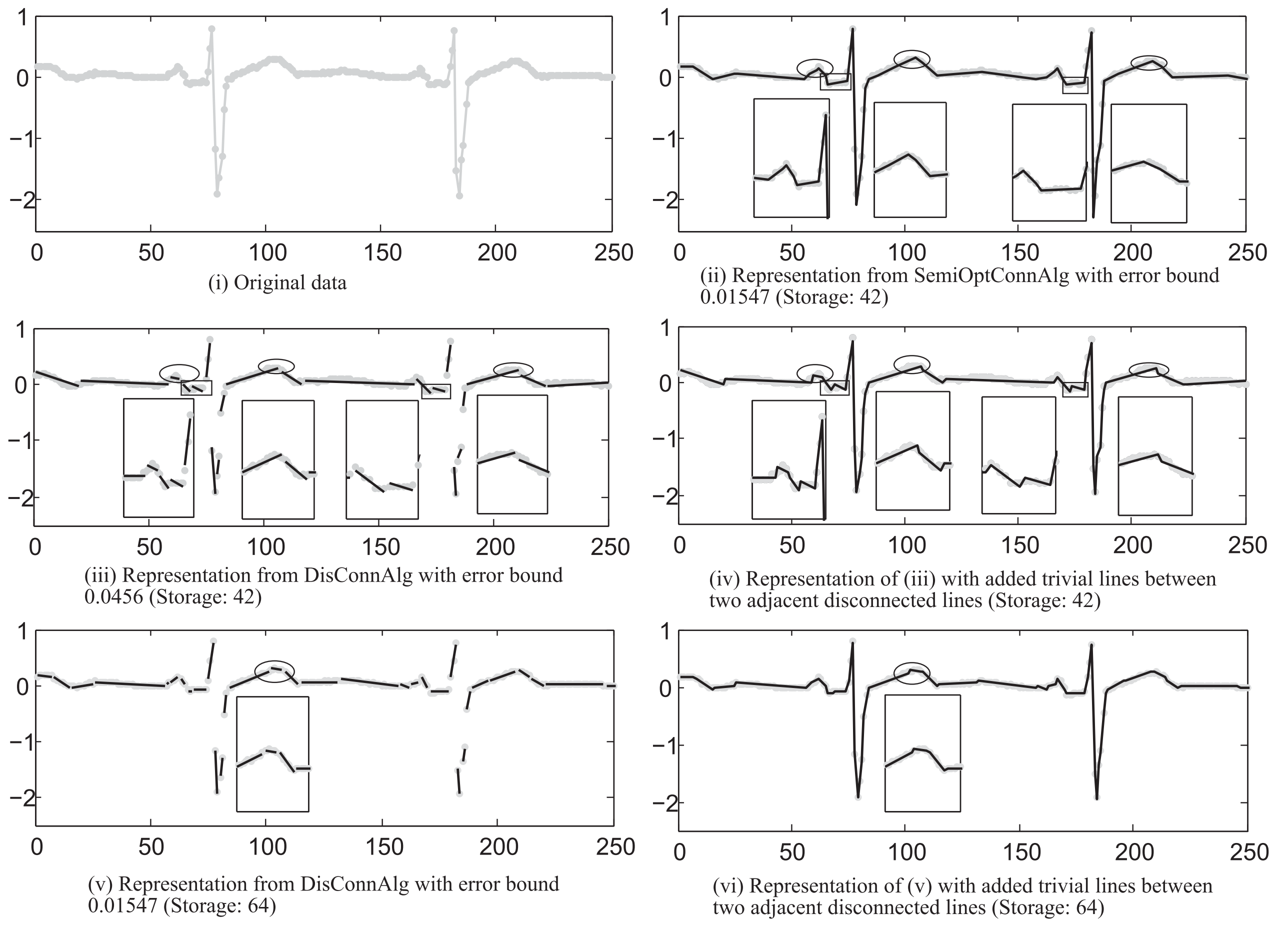


Fig. 15. Visual differences on ECG representations by DisConnAlg and SemiOptConnAlg.

a fixed storage cost or on a fixed error bound to represent the ECG data. In this regard, we set the error bounds of SemiOptConnAlg and DisConnAlg to be 0.01547 and obtained a storage of 42 and 62 respectively on the outputs (Figs. 15 ii and 15v). We then readjust the error bound of DisConnAlg to 0.0456 for achieving the storage of 42 (Fig. 15 iii). Figs. 15 iv and 15 vi are the “connected version” of Figs. 15 iii and 15v by connecting any two adjacent disconnected lines with a trivial line. Three advantages on the semi-connected representation are observed: (1) Under the same storage cost or error bound, the ECG curve can be represented with better precision and smoothness in semi-connected lines than that of disconnected lines; (2) For the tendencies marked by rectangles, semi-connected lines are more consistent with the corresponding original parts than that of disconnected ones; (3) For some important ECG features, the results by SemiOptConnAlg are more precise than that from DisConnAlg. For example, the peaks of T wave marked by ellipses are shifted away from the original locations when represented in disconnected segments, which can result in incorrect practical diagnostic results for patients.

5 Conclusion

In this article, we present a new linear time PLA algorithm to construct line segments with guaranteed max-error bound. We prove that SemiOptConnAlg can construct the minimum number of semi-connected segments. Extensive experiments on 43 data sets indicate that the proposed algorithm is very efficient and has better time and memory performance than PipeOptConnAlg. This obtained technique has been used in image processing with successful satisfaction in [17].

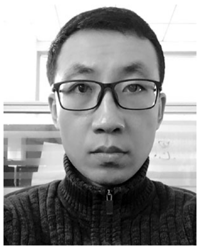
In [14], the authors proposed a new method that uses “mixed” segments to reduce storage further. As such, we have converted SemiOptConnAlg to generate “mixed segments” which achieved better test results. Our next research work will be focused on obtaining a more efficient way of computing mixed segments.

Acknowledgments

The authors would like to thank the authors of [14], [20], [21] for generously giving us their source code for the comparison tests in this article. We also thank the data providers of [11], [24] for the testing data sets. This work was partially supported by the Natural Science Foundation of China (No. 61572022), the Hebei Academy of Sciences Project (No. 161303), the Hebei “One Hundred Plan” Project (No. E2012100006), and the Ningbo eHealth Project (No. 2016C11024).

References

[1] U. Appel and A. V. Brandt, “Adaptive sequential segmentation of piecewise stationary time series,” *Inf. Sci.*, vol. 29, no. 1, pp. 27–56, 1983.
[2] R. Bellman, “On the approximation of curves by line segments using dynamic programming,” *Commun. ACM*, vol. 4, no. 6, 1961, Art. no. 284.
[3] M. D. Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer, 2008.
[4] T.-C. Fu, F. lai Chung, V. Ng, and R. Luk, “Evolutionary segmentation of financial time series into subsequences,” in *Proc. Congress Evol. Comput.*, 2001, vol. 1, pp. 426–430.
[5] M. Garofalakis and A. Kumar, “Deterministic wavelet thresholding for maximum-error metrics,” in *Proc. 23nd ACM SIGMOD-SIGACT- SIGART Symp. Princ. Database Syst.*, 2004, vol. 23, pp. 166–176.
[6] M. Garofalakis and A. Kumar, “Wavelet synopses for general error metrics,” *ACM Trans. Database Syst.*, vol. 30, no. 4, pp. 888–928, 2005.
[7] S. Hakimi and E. Schmeichel, “Fitting polygonal functions to a set of points in the plane,” *CVGIP: Graphical Models Image Process.*, vol. 53, no. 2, pp. 132–136, 1991.
[8] H. Imai and M. Iri, “An optimal algorithm for approximating a piecewise linear function,” *Inf. Process. Lett.*, vol. 9, no. 3, pp. 159–162, 1986.
[9] E. Keogh, S. Chu, D. Hart, and M. Pazzani, “An online algorithm for segmenting time series,” in *Proc. IEEE Int. Conf. Data Mining*, 2001, pp. 289–296.
[10] E. Keogh, S. Chu, D. Hart, and M. Pazzani, “Segmenting time series: A survey and novel approache,” *Data Mining in Time Series Databases*, vol. 57, pp. 1–21, 2003.
[11] H. Dau *et al.*, “The UCR time series classification archive,” 2019. [Online]. Available: http://www.tp-ontrol.hu/index.php/TP_Toolbox
[12] A. Koski, M. Juhola, and M. Meriste, “Syntactic recognition of ecg signals by attributed finite automata,” *Pattern Recognit.*, vol. 28, no. 12, pp. 1927–1940, 1995.
[13] X. Liu, Z. Lin, and H. Wang, “Novel online methods for time series segmentation,” *IEEE Trans. Knowl. Data Eng.*, vol. 20, no. 12, pp. 1616–1626, Dec. 2008.
[14] G. Luo *et al.*, “Piecewise linear approximation of streaming time series data with max-error guarantees,” in *Proc. 31st Int. Conf. Data Eng.*, 2015, pp. 173–184.
[15] J. O’Rourke, “An on-line algorithm for fitting straight lines between data ranges,” *Commun. ACM*, vol. 24, no. 9, pp. 574–578, 1981.
[16] C. Pang, Q. Zhang, X. Zhou, D. P. Hansen, S. Wang, and A. J. Maeder, “Computing unrestricted synopses under maximum error bound,” *Algorithmica*, vol. 65, no. 1, pp. 1–42, 2013.
[17] C. Pang, Y. Zhao, and T. Li, “Image scaling: How hard can it be?” Zhejiang University, Hangzhou, China, 2018.
[18] C. Perng, H. Wang, S. R. Zhang, and D. S. Parker, “Landmarks: A new model for similarity-based pattern querying in time series databases,” in *Proc. 16th Int. Conf. Data Eng.*, 2000, pp. 33–42.
[19] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita, “Improved histograms for selectivity estimation of range predicates,” *ACM SIGMOD Rec.*, vol. 25, no. 2, pp. 294–305, 1996.
[20] J. Qi, R. Zhang, K. Ramamohanarao, H. Wang, Z. Wen, and D. Wu, “Indexable online time series segmentation with error bound guarantee,” *World Wide Web*, vol. 18, no. 2, pp. 359–401, 2015.
[21] Q. Xie, C. Pang, X. Zhou, X. Zhang, and K. Deng, “Maximum error-bounded piecewise linear representation for online stream approximation,” *The VLDB J.*, vol. 23, no. 6, pp. 915–937, 2014.
[22] Z. Xu, R. Zhang, K. Ramamohanarao, and U. Parampalli, “An adaptive algorithm for online time series segmentation with error bound guarantee,” in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 192–203.
[23] H. Zhao, Z. Dong, T. Li, X. Wang, and C. Pang, “Segmenting time series with connected lines under maximum error bound,” *Inf. Sci.*, vol. 345, pp. 1–8, 2016.
[24] 2017. [Online]. Available: https://www.physionet.org/cgi-bin/atm/ATM
[25] S. Suri, “A linear time algorithm for minimum link paths inside a simple polygon,” *Comput. Vis. Graph. Image Process.*, vol. 34, no. 1, pp. 99–110, 1983.
[26] P. Egyed and R. Wenger, “Ordered stabbing of pairwise disjoint convex sets in linear time,” *Discr. Appl. Math.*, vol. 31, no. 2, pp. 133–140, 1991.
[27] L. Guibas, J. Hershberger, J. Mitchell, and J. Snoeyink, “Approximating polygons and subdivisions with minimum link paths,” in *Proc. 2nd Int. Symp. Algorithms*, 1991, pp. 151–162.
[28] P. Agarwal and K. Varadarajan, “Efficient algorithms for approximating polygonal chains,” *Discr. Comput. Geometry*, vol. 23, pp. 273–291, 2000.
[29] M. Abam, M. Berg, P. Hachenberger, and A. Zarei, “Streaming algorithms for line simplification,” in *Proc. 23rd Annu. Symp. Comput. Geometry*, 2007, vol. 23, pp. 175–183.
[30] M. A. Abam, S. Daneshpajouh, L. Deleuran, S. Ehsani, and M. Ghodsia, “Computing homotopic line simplification,” *Comput. Geometry: Theory Appl.*, vol. 47, pp. 728–739, 2014.



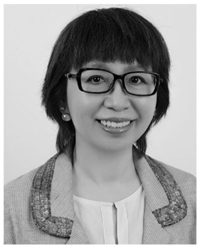
Huanyu Zhao received the MSc degree in applied mathematics from Hebei University, China, in 2009. He is currently an associate professor with the Institute of Applied Mathematics, Hebei Academy of Sciences, and School of Computer and Data Engineering, Zhejiang University (NIT), China. His current research interests include stream data compression and processing, machine learning, evolutionary computation, and computing algorithm.

Chaoyi Pang received the PhD degree from the University of Melbourne, Australia, in 1999. He is a senior member of ACM. After receiving the PhD degree, he worked in IT industrial as an IT engineer and consultant, in 1999–2002 and CSIRO as research scientist, in 2002–2014 in Australia. Currently, he is dean of the School of Computer and Data Engineering, Zhejiang University (NIT), China. He is a distinguished professor of Zhejiang University (NIT), China. His research interests include algorithm, stream data compression and processing, database theory, graph theory, and e-health.

Ramamohanarao Kotagiri has been with the University of Melbourne, Australia, since 1980. He served or serving on the editorial boards of the *Computer Journal*, the *IEEE Transactions on Knowledge and Data Engineering* and the *VLDB Journal*. He was the program co-chair for VLDB, SIGMOD, ICDE, and PAKDD conferences. He is a fellow of the Institute of Engineers Australia, a fellow of Australian Academy Technological Sciences and Engineering and a fellow of the Australian Academy of Science. He was awarded Distinguished Contribution Award, in 2009 by the Computing Research and Education Association of Australasia.

Christopher K. Pang is currently working toward the fourth year degree at The University of Queensland, Australia. He is currently enrolled in both the bachelor of engineering and bachelor of science. Within his degree, he has chosen to major in electronics and computer sciences.

Ke Deng received the PhD degree in computer science from The University of Queensland (UQ), Australia, in 2007 with focus on data and knowledge engineering. He is currently a lecturer in RMIT University, Australia. He had been a postdoctoral researcher in Huawei Noah Arks Research Lab, Hong Kong from 2013–2015. He had been a postdoctoral research fellow at CSIRO ICT center, in 2007 and an ARC Australian postdoctoral fellow during 2010–2012.



Jian Yang received the PhD degree from Australian National University, Australia, in 1995. She is a full professor with the Department of Computing, Macquarie University, Australia. She has published more than 200 scientific articles in top venues such the *IEEE Transactions on Knowledge and Data Engineering*, VLDB, ICDE, and ICDM. Her main research interests include big data analytics, process analytics, social networks, and service computing.

Tongliang Li received the PhD degree in computer science from Tianjin University, China, in 2014. He is currently an researcher with the School of Computer and Data Engineering, Zhejiang University (NIT), China, and institute of Applied Mathematics, the Hebei Academy of Sciences, China. His current research interests include stream data compression and processing, information privacy, and data mining.