The VLDB Journal
DOI 10.1007/s00778-014-0355-0

REGULAR PAPER

Maximum error-bounded Piecewise Linear Representation for online stream approximation

Qing Xie · Chaoyi Pang · Xiaofang Zhou · Xiangliang Zhang · Ke Deng

Received: 29 August 2013 / Accepted: 12 March 2014

Abstract Given a time series data stream, the generation of error-bounded Piecewise Linear Representation (error-bounded PLR) is to construct a number of consecutive line segments to approximate the stream, such that the approximation error does not exceed a prescribed error bound. In this work, we consider the error bound in L_∞ norm as approximation criterion, which constrains the approximation error on each corresponding data point, and aim on designing algorithms to generate the minimal number of segments. In the literature, the optimal approximation algorithms are effectively designed based on transformed space other than time-value space, while desirable optimal solutions based on original time domain (i.e., time-value space) are still lacked. In this article, we proposed two linear-time algorithms to construct error-bounded PLR for data stream based on time domain, which are named OptimalPLR and GreedyPLR, respectively. The OptimalPLR is an optimal algorithm that generates minimal number of line segments for the stream approximation, and the GreedyPLR is an alternative solution for the requirements of high efficiency and resource-constrained environment. In order to evaluate the superiority of OptimalPLR, we theoretically analyzed and compared OptimalPLR with the state-of-art optimal solution in transformed space, which also achieves linear complexity. We successfully proved the theoretical equivalence between time-value space and such transformed space, and also discovered the superiority of OptimalPLR on processing efficiency in practice. The extensive results of empirical evaluation support and demonstrate the effectiveness and efficiency of our proposed algorithms.

Keywords Stream approximation · Error bound · Piecewise Linear Representation

Q. Xie · X. Zhang (✉)
Division of CEMSE, KAUST, Thuwal, Saudi Arabia
e-mail: Xiangliang.Zhang@kaust.edu.sa

Q. Xie
e-mail: Qing.Xie@kaust.edu.sa

C. Pang
AEHRC, CSIRO, Brisbane, Australia
e-mail: Chaoyi.Pang@csiro.au

C. Pang
Zhejiang University (NIT), Ningbo, China

C. Pang
Hebei Academy of Sciences, Hebei, China

X. Zhou
School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane, Australia

X. Zhou
School of Computer Science and Technology, Soochow University, Suzhou, China
e-mail: zxf@itee.uq.edu.au

K. Deng
Huawei Noah's Ark Research Lab, Hong Kong, China
e-mail: deng.ke@huawei.com

1 Introduction

The use of line segments to represent a time series data stream, termed as piecewise linear representation (PLR), has been extensively studied for decades under different criteria (refer to [12,20] for general reviews). The idea of PLR is to represent a complicated wave-like data stream with a number of simple line segments, so that the streaming data can be efficiently archived, and a query on the stream can be approximately answered by a query on the line segments. Compared with the stream itself, the line segments constructed from PLR provide striking visual outlines of stream trends and can

Published online: 04 April 2014

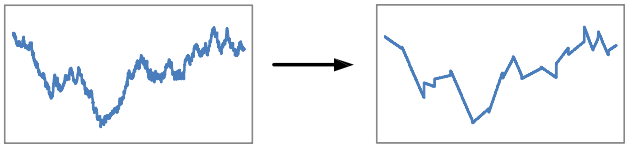

Fig. 1 An example of PLR on a time series data stream

be more efficiently processed and represented in the database (Fig. 1). These advantages make PLR the most popular representation technique for the data stream [12], and it has been widely applied to support date indexing [3,11,24], similarity search [26,30], and correlation analysis [28].

The intensive work on PLR research is the size-bounded PLR [3,5]. Its objective is to construct a prescribed number of line segments that minimize the approximation error under a specified metric, where L_2 norm (i.e., Euclidean distance) is mostly used. However, there are two main drawbacks on the size-bounded constraint and the use of L_2 norm for approximation: Firstly, the size-bounded constraint lacks the ability to generate error-guaranteed representations for streaming data since the stream is naturally unbounded in size; secondly, the use of L_2 norm leads to the inability of controlling the approximation error on individual stream data items. All these drawbacks severely impede further applications of the size-bounded representations. To alleviate these drawbacks, researchers have made efforts in constructing the representations with maximum approximate error constrained on each data point (L_∞ norm), which is termed as the (maximum) error-bounded representations.

In this article, we propose to investigate and design the optimal algorithm to generate the error-bounded PLR for data stream, which aims to construct the smallest number of line segments with approximation error constrained on each data point. Early in 1981, this problem was perfectly solved by an algorithm designed in a transformed space [18], which is termed as ParaOptimal in this paper. Recently, [4] studied this problem and provided an optimal algorithm SlideFilter based on time domain, which, however, cannot achieve the performance as ParaOptimal. In order to improve the work of [4] and complete the methodology, we consider exploring the optimal solution designed in original time domain, which achieves satisfactory efficiency as ParaOptimal and also provides essential and new sight on this classic problem.

Intuitively, the optimal results can be achieved by greedy mechanism, which is derived from the Theorem 1 in following sections. In order to adjust a line segment to approximate as many stream points as possible, the general idea is to determine the range of all feasible line segments, which is incrementally updated during the processing of consecutive sequence points. Whenever the current point cannot be approximated within error bound, a line segment can be determined.

Our proposed OptimalPLR algorithm is designed based on such general idea. It serves the same purpose, but methodologically differs from ParaOptimal in the process of constructing a segment. While ParaOptimal maintains a convex polygon in the specially designed slope-offset space, the OptimalPLR constructs a desired segment through maintaining two convex hulls in the time-value space (time domain). To solve the efficiency problem, OptimalPLR acts differently from SideFilter on the following three aspects: Firstly, it uses the *minimized convex hulls* to construct the new extreme slopes (minimum and maximum slopes) when a new stream point is added in; secondly, it maintains two minimized convex hulls progressively, that is, if the new extreme slopes are attainable from a point in the convex hull, all the earlier points in the convex hull are removed before the next round of updating; lastly, this algorithm only requires two points checking to decide whether the new extreme slopes can be made from a point in the convex hull and avoids extensive search on the whole convex hulls, which can be expensive in many situations.

In addition to the optimal algorithm OptimalPLR, we also propose an approximate algorithm GreedyPLR that can achieve near-optimal results but with even less computational cost. Our theoretical analysis shows that the GreedyPLR algorithm enables a lower processing time with $O(1)$ space cost for potentially lower compression ratio requirements and resource-constrained environment. In the process of generating a segment, the GreedyPLR algorithm uses the intersecting point of two extreme lines as the support point to swing up and down the slopes for the succeeding stream points. The extreme lines are the special lines with minimum and maximum slopes that represent the processed points. More importantly, the GreedyPLR inspires the mechanism of tuning trade-off between time efficiency and compression ratio, as the effectiveness of the GreedyPLR algorithm can be adjusted by using the intersecting point of proper extreme lines.

Since both of ParaOptimal and OptimalPLR are optimal solutions with linear-time complexity for error-bounded PLR problem, but derived from different spaces, we further propose to theoretically compare these two algorithms and the spaces they are based on, so as to provide deeper and more theoretical understanding for this problem. By setting up a mapping between the point and line in these two spaces, we creatively and theoretically proved the equivalence of these two spaces and further linked the two algorithms together, which explained the optimal solution from different views. However, due to the processing efficiency in different spaces, our approach shows better practicability than ParaOptimal, which is interpreted and demonstrated in Sect. 8.

Our experiments demonstrate that the OptimalPLR algorithm achieves the same optimal results or outperforms ParaOptimal and SlideFilter in both memory cost and construction time in all tested situations. In terms of efficiency,

OptimalPLR is often above 4 times faster than SlideFilter in general. In these situations, the memory cost of OptimalPLR algorithm can be 15 and 27 % less than SlideFilter and ParaOptimal, respectively, which results from the more practical convex hulls it maintains. The GreedyPLR algorithm largely reduces the time cost than the optimal approaches, which is similar with the SwingFilter, but it generally outperforms SwingFilter in terms of compression quality under all tested situations. All these results suggest that our proposed algorithms can efficiently support online stream processing.

Compared with the conventional approaches, our contributions in this work can be summarized as follows:

1. Design and implementation of two new algorithms: OptimalPLR and GreedyPLR.
2. Comprehensive analysis on the derivation of the proposed algorithms, including the superiority of the proposed algorithms over existing works.
3. Theoretical analysis of the relationship between slope-offset space and time-value space, as well as the comprehensive proof for the equivalence of the optimal solutions based on different spaces.
4. Demonstration of the effectiveness and practicability of the proposed algorithms on extensive synthetic and real-life data.

The rest of the paper is organized as follows: Sect. 2 discusses the important related works; Sect. 3 explains the basic terminologies, new concepts, and related essential results; Sect. 4 formulates the minimum and maximum possible slopes of a line segment based on coordinate rotation, and discusses the slope alteration on a new stream point; Sect. 5 advocates the optimization strategies for error-bounded PLR; Sect. 6 describes in details the two proposed algorithms OptimalPLR and GreedyPLR; Sect. 7 compares and links the optimal algorithms in two different spaces; Sect. 8 reports our experiment results; Sect. 9 concludes this paper.

2 Related works

The construction of error-bounded representations on data streams has been engaged in many real-world applications, such as continuous queries over data streams [14,17,28], sensor network management [23], and monitoring physiological data for surgery operations [21,29]. However, many contemporary error-bounded representations are based on either histogram (i.e., piecewise constant representation (PCR)) [5,14,17] or wavelet techniques [7–10,16,21,29].

For error-bounded PCR problems, Lazaridis et al. [14] provided an optimal algorithm to approximate sensor data stream. Gandhi et al. [6] proposed GAMPS that compressed multi-stream with guaranteed L_∞ error as well as a notable worst-case quality of approximation. They also extended the framework to address amnesic approximation and out-of-order approximation [5] using bucket merging. As the PCR uses constant values (i.e., horizontal line segments) in representing a wave-like stream, it fails to reflect the trends of streaming data. Regarded as a generalized version of error-bounded PCR, the error-bounded PLR usually achieves a greater compression ratio and has more superiority for advanced applications than PCR.

In comparison with wavelet-based methods, PLR is more visually comprehensive and convenient for stream queries. For example, the line segments generated by error-bounded PLR on a stream can denote the stream trends and be used *directly* for answering trend queries. Such queries are crucial in monitoring patients in intensive care as medical specialists [19] believe that more accurate and earlier notifications of adverse events can be predicated from the accumulated trends and variations of the physiological streams. Even though error-bounded wavelet representations such as [21,22] can be constructed very efficiently and effectively, the wavelet synopses may not be ready to answer stream trends directly.

The error-bounded PLR was studied by Buragohain et al. [2], Elmeleegy et al. [4], and Li et al. [15] more recently. With the name of piecewise linear histogram, efficient approximation algorithms for size-bounded and error-bounded PLR were provided in [2]. For error-bounded PLR, they proposed to construct the desired line segments by deriving the "thinnest bounding rectangle" through a linear scan of the stream points on the convex hull. In [4], the authors extended the result of [2] by presenting an effective new algorithm which is termed as SlideFilter. It is an optimal algorithm that generates the smallest number of line segments for a data stream, which employs a "sliding" mechanism that is optimized by restricting the slopes on the convex hull points. However, the time efficiency of the SlideFilter algorithm will be affected by the redundant convex points and its global scanning strategy. Li et al. proposed GDPLA to solve error-bounded PLR problem, which also considers the convex points. However, their solution still lacks optimal efficiency.

Furthermore, it seems that the research of [2,4,15] was unaware of the elegant work [18] in 1981, where O'Rourke proposed a linear-time optimal algorithm for error-bounded PLR problem. O'Rourke's approach, termed as ParaOptimal algorithm in this paper, derives each desired segment through constructing and incrementally maintaining a convex polygon in the slope-offset parameter space. Such innovative parameter space has been applied in the recent research for advanced applications [25,27]. We will mainly compare our approaches with these state-of-art solutions.

3 Preliminary

In this section, along with some notations and new concepts for the study of error-bounded PLR, we formally provide

Table 1 Notations

Symbol	Description
δ	Error bound for approximation (>0)
$S = \langle s_1, s_2, \ldots, s_k, \ldots \rangle$	A data stream
$s_i = (x_i, y_i)$	Data point s_i at time stamp x_i with value y_i
$S[i, j] = \langle s_i, s_{i+1}, \ldots, s_j \rangle$	A (stream) fragment from time x_i to x_j
$\underline{s_i} = (x_i, y_i - \delta)$	Data point with deleted tolerant error
$\overline{s_i} = (x_i, y_i + \delta)$	Data point with added tolerant error
seg$[i, j]$	A δ-representative line on time slot $[x_i, x_j]$
slp$[i, j]$	The slope of seg$[i, j]$
$\underline{\text{slp}}[i, j]$ or $\overline{\text{slp}}[i, j]$	The minimum or maximum slopes of all slp$[i, j]$
line(s_i, s_j)	Line (segment) that passes point s_i and s_j
line(ρ, s)	Line (segment) with slope ρ that passes point s
slope(s_i, s_j)	The slope of line(s_i, s_j)
$\underline{\text{cvx}}_k$, $\overline{\text{cvx}}_k$	Reduced convex hulls for $S[1, k]$.
	$\underline{\text{cvx}}$ / $\overline{\text{cvx}}$ bulge downward/upward

the problem definition and the objective addressed in this paper. We also present Theorem 1 which guarantees that the smallest number of line segments for the error-bounded PLR can be achieved by maximizing each representative line segment. The general notations used throughout this paper are summarized in Table 1.

Let $S = \langle s_1, s_2, \ldots, s_k, \ldots \rangle$ denote a data stream where each point $s_i = (x_i, y_i)$ designates the actual value y_i at time stamp x_i. We use $S[i, j] = \langle s_i, s_{i+1}, \ldots, s_j \rangle$ to denote the stream *fragment* on time slot $[x_i, x_j]$ $(i < j)$ and $|S[i, j]| = j - i + 1$ to denote the cardinality of $S[i, j]$.

As an approximation technique, PLR approximates S with line segments. A *line segment* on time slot $[x_i, x_j]$ is representable by the linear function $y = a \cdot x + b$ for $x \in [x_i, x_j]$ with two parameters: slope a and offset b. Since a line can be determined by its slope and a line point or two different line points alternatively, for the convenience of presentation, we also use line(ρ, s) to denote the line that passes point s with slope ρ, and line(s_i, s_j) to denote the line that passes the two points s_i and s_j. With this notation, we use slope(s_i, s_j) as the slope of line(s_i, s_j).

We define a stream fragment $S[i, j]$ $(i < j)$ as δ-*representable* (at time slot $[t_i, t_j]$) if there exists a line segment identified by

$$y'_h = a \cdot x_h + b,$$

such that $|y'_h - y_h| \leq \delta$ holds for each point $s_h = (x_h, y_h)$ of $S[i, j]$ $(i \leq h \leq j)$. In this situation, such line segment is defined as a δ-*representative* for fragment $S[i, j]$. (Notice that there can be more than one δ-representative.) For simplicity, we use seg$[i, j]$ to represent the δ-representative, and slp$[i, j]$ to denote the slope of seg$[i, j]$. Here, we use the maximum error metric L_∞ to guarantee the approximation quality at each stream data point. Furthermore, if fragment $S[i, j]$ is δ-representable but $S[i, j+1]$ is not, then fragment $S[i, j]$ is *maximally* δ-representable on time slot $[t_i, t_j]$[1] and accordingly its seg$[i, j]$ is called maximal δ-representative.

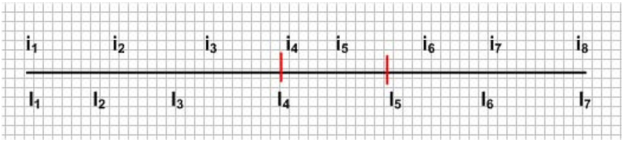

Fig. 2 The proof of Theorem 1: $\alpha = 4$

With these designations, the error-bounded Piecewise Linear Representation (error-bounded PLR) problem discussed in this paper is precisely defined as follows:

Definition 1 (Error-bounded PLR) Given a predefined error bound $\delta > 0$ and a data stream fragment $S[1, n] = \langle s_1, s_2, \ldots, s_n \rangle$, the (optimal) error-bounded PLR is to construct a (minimal) number of δ-representative line segments $\{\text{seg}[i_1, i_2\text{–}1], \text{seg}[i_2, i_3\text{–}1], \ldots, \text{seg}[i_k, i_{k+1}\text{–}1]\}$ to represent $S[1, n]$, where $i_1 = 1$ and $i_{k+1} - 1 = n$.

By the definition, we specify that this work focuses on generating disconnected line segments, i.e., the consecutive segments do not share same end points. Similar to *Theorem 1* of paper [14] and *Lemma 2* of paper [2] for the piecewise constant representation (PCR), the following theorem indicates that the optimal error-bounded PLR can be solved through computing maximal δ-representative line segments.

Theorem 1 *Given an error bound $\delta > 0$ and stream fragment $S[1, n]$, assume that seg$[i_j, i_{j+1} - 1]$ is a maximal δ-representative of $S[i_j, i_{j+1} - 1]$ on time slot $[i_j, i_{j+1} - 1]$ for $1 \leq j \leq k$ with $i_1 = 1$ and $i_{k+1} - 1 = n$. Then the error-bounded PLR on fragment $S[1, n]$ has at least k line segments.*

Proof Clearly, the claim is true for $k = 1$. For $k > 1$, assume that an optimal error-bounded PLR solution for fragment $S[1, n]$ is the set of segments seg$'[l_h, l_{h+1} - 1]$ for $1 \leq h \leq m$ where $l_1 = 1$, $l_{m+1} - 1 = n$ and $m < k$ (Fig. 2).

As claimed in the theorem, since each segment is a maximal δ-representative, both $i_1 = l_1 = 1$ and $l_2 \leq i_2$ hold. Let α be the index value such that $l_\alpha \leq i_\alpha$ and $l_{\alpha+1} > i_{\alpha+1}$ holds. Alternatively, since both $m < k$ and $l_{m+1} - 1 = n$ hold, $i_{m+1} < l_{m+1}$ holds. Hence, we have that α exists and $1 \leq \alpha \leq m$ is confirmed.

[1] It should be noted that $S[i-1, j]$ can be δ-representable even if $S[i, j]$ is maximally δ-representable.

Thus, we have $l_\alpha \le i_\alpha < i_{\alpha+1}-1 < l_{\alpha+1}-1$. It means that $S[i_\alpha, l_{\alpha+1}-1]$ is δ-representable and is contradictory to the hypothesis that seg$[i_\alpha, i_{\alpha+1}-1]$ is a maximal δ-representative line segment. Therefore, $k \le m$ holds, and the result is proven. □

With Theorem 1, it is inspired that the optimal error-bounded PLR results can be achieved by maximizing each δ-representative line segment, which is naturally the greedy mechanism. We can also conclude that the OptimalPLR algorithm we proposed, the SlideFilter [4], and ParaOptimal [18] are all able to generate optimal error-bounded PLR and thus are called optimal algorithms. These will be detailed in the following parts of this paper.

4 Extreme slopes of maximal δ-representative

Generally, there can exist many δ-representative segments with various slopes for a δ-representable stream fragment. The range of the candidates' slopes depends on the tolerant error δ and the stream itself. In this section, we first discuss the complete slope range of the δ-representative segments for a δ-representable stream fragment and then study the slope reductions and alterations when a new point is added to the stream fragment. Such analysis can provide the intuition of how to maximize the δ-representatives. Our discussion is based on the rotation of Cartesian coordinate system.

Some of the results in this section, such as Corollary 1 in Sect. 4.2, have been previously presented in papers [2,4] in varied forms. Here, besides providing more precise expressions to simplify the proof of those papers, we use these expressions to lead to our proposed algorithms. More significantly, we derive the novel reduction outcome (Eq. (6)) on which the results in Sect. 5 are based.

4.1 Slope rotation and extreme slopes

We start from the methods proposed by Lazaridis et al. [14] and Buragohain et al. [2] for error-bounded PCR. As a simplified version of error-bounded PLR, error-bounded PCR uses constant values rather than general line segments for the representation. Constant values can be regarded as a line with zero slope that is denoted by a horizontal line in a Cartesian coordinate plane as in Fig. 3a. Under a predefined error bound $\delta > 0$, the optimal error-bounded PCR aims at constructing the smallest number of representation $B = \{c_{i_1}, c_{i_2}, \ldots, c_{i_h}\}$ for a given stream fragment $S[1, n] = \langle s_1, s_2, \ldots, s_n \rangle$ such that

$$\begin{cases} 0 = i_0 < i_k < i_h = n & \text{for} \quad 0 < k < h, \\ |y_j - c_{i_k}| \le \delta & \text{for} \quad i_{k-1} < j \le i_k \text{ and } 1 \le k \le h. \end{cases}$$

That is, using constant value c_{i_k} represents y_j for $i_{k-1} < j \le i_k$ and $1 \le k \le h$.

To build B, the scheme of [2,14] greedily checks the points of the stream fragment in order and finds the maximally δ-representable stream fragment iteratively. Here, the stream fragment is only approximated by horizontal lines. To choose the maximally δ-representable fragment started from x_i, the scheme needs to find maximum time stamp x_j such that

$$\max_{i \le i_1 < i_2 \le j} |y_{i_1} - y_{i_2}| \le 2\delta.$$

In fact, the scheme constructs the longest horizontal rectangle with 2δ width starting from x_i to cover the maximal number of steam points. The intuition is depicted in Fig. 3a in general.

Extending the idea to error-bounded PLR, we have the following observation that can be proven easily according to the error bound definition.

Observation 1 *A stream fragment $S[i, j]$ is δ-representable if and only if there exists a parallelogram of 2δ width in the vertical direction such that no points of $S[i, j]$ are placed outside of the parallelogram.*

The intuition of this observation is illustrated in Fig. 3b. We will use this observation to study the feasible slopes of a line segment approximating a δ-representable stream fragment through rotating the Cartesian coordinate plane.

To simplify the discussion, we assume that $S[1, k-1] = \langle s_1, s_2, \ldots, s_{k-1} \rangle$ is δ-representable, and we use a line segment to represent it. For each point $s_i = (x_i, y_i)$ of $S[1, k-1]$ where $1 \le i < k$, its new coordinates $s'_i = (x'_i, y'_i)$, after having the axis rotated around the origin by an angle of $-\pi/2 < \theta < \pi/2$, must satisfy:

$$\begin{cases} x'_i = x_i \cos\theta + y_i \sin\theta, \\ y'_i = -x_i \sin\theta + y_i \cos\theta, \end{cases}$$

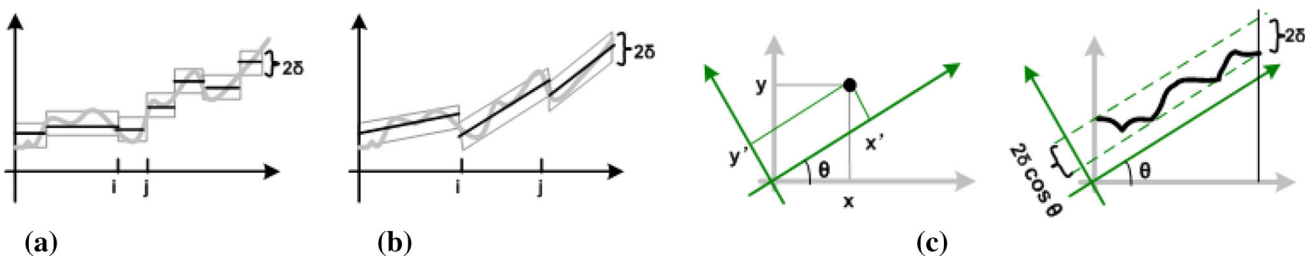

Fig. 3 Rotation examples. **a** Error-bounded PCR. **b** Error-bounded PLR. **c** Rotation

as illustrated in Fig. 3c. According to the observation, since fragment $S[1, k-1]$ is δ-representable, there exists $-\pi/2 < \theta < \pi/2$ such that for each pair of $\{i, j\}$,

$$|y'_i - y'_j| \leq 2\delta|\cos\theta|.$$

Since $-\pi/2 < \theta < \pi/2$, $|\cos\theta| \neq 0$ and

$$|y'_i - y'_j| = |[(-x_i \tan\theta + y_i) - (-x_j \tan\theta + y_j)]\cos\theta|$$

hold, we have $|(x_j - x_i)\tan\theta - (y_j - y_i)| \leq 2\delta$. By extending this formula, we have[2]:

$$\frac{(y_j - \delta) - (y_i + \delta)}{(x_j - x_i)} \leq \tan\theta \leq \frac{(y_j + \delta) - (y_i - \delta)}{(x_j - x_i)}. \quad (1)$$

In fact, θ is the intersection angle of x-axis and the parallelogram's non-vertical edge, which is also the inclination angle of a possible δ-representative for $S[1, k-1]$ (i.e., the centerline parallel to the parallelogram's non-vertical edge). In this sense, the $\tan\theta$ is the slope of the δ-representative line segment.

Since for each pair of $\{i, j\}$, the relationship of 1 must hold, we can derive the range of $\tan\theta$. Let $\underline{\text{slp}}[1, k-1]$ and $\overline{\text{slp}}[1, k-1]$ denote the minimum and maximum line slopes of the δ-representative for fragment $S[1, k-1]$, respectively, and we have

$$\begin{cases} \underline{\text{slp}}[1, k-1] = \max\limits_{1\leq i<j\leq k-1} \frac{(y_j-\delta)-(y_i+\delta)}{(x_j-x_i)} \\ \qquad\qquad = \frac{(y_c-\delta)-(y_a+\delta)}{(x_c-x_a)}, \\ \overline{\text{slp}}[1, k-1] = \min\limits_{1\leq i<j\leq k-1} \frac{(y_j+\delta)-(y_i-\delta)}{(x_j-x_i)} \\ \qquad\qquad = \frac{(y_d+\delta)-(y_b-\delta)}{(x_d-x_b)}, \end{cases} \quad (2)$$

Then, for a possible δ-representative line segment of $S[1, k-1]$, its slope $\text{slp}[1, k-1]$ satisfies:

$$\underline{\text{slp}}[1, k-1] \leq \text{slp}[1, k-1] \leq \overline{\text{slp}}[1, k-1]. \quad (3)$$

In fact, Eq. (3) denotes that the stream fragment $S[1, k-1]$ is δ-representable if and only if $\underline{\text{slp}}[1, k-1] \leq \overline{\text{slp}}[1, k-1]$. In the situations of $\underline{\text{slp}}[1, k-1] < \overline{\text{slp}}[1, k-1]$, $\text{slp}[1, k-1]$ can have many feasible values within the range $[\underline{\text{slp}}[1, k-1], \leq \overline{\text{slp}}[1, k-1]]$. Hence, we propose to discover the feasible range of the line slope in terms of $\underline{\text{slp}}[1, k-1]$ and $\overline{\text{slp}}[1, k-1]$.

Specifically, we also define the extreme points as those identify the δ-representatives with extreme slopes. Based on Eq. (2), we define:

$$\begin{cases} \{a, c\} = \arg\max\limits_{1\leq i<j\leq k-1} \frac{(y_j-\delta)-(y_i+\delta)}{(x_j-x_i)}, \quad (a < c) \\ \{b, d\} = \arg\min\limits_{1\leq i<j\leq k-1} \frac{(y_j+\delta)-(y_i-\delta)}{(x_j-x_i)}, \quad (b < d) \end{cases}$$

[2] Without loss of generality, we assume that $x_i < x_j$.

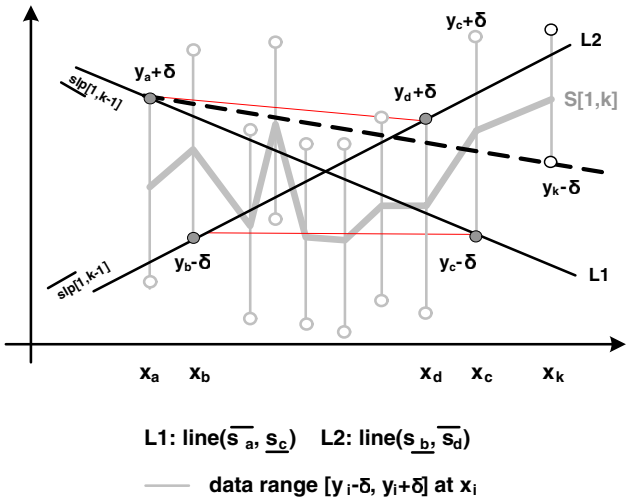

Fig. 4 Extreme slope evolution

where points s_a and s_b (s_c and s_d, respectively) are the rightmost (leftmost, respectively) points that satisfy $1 \leq a < c \leq k-1$ and $1 \leq b < d \leq k-1$. Assume $\underline{s_i} = (x_i, y_i - \delta)$ denotes data point with deleted tolerant error, and $\overline{s_i} = (x_i, y_i + \delta)$ denotes data point with added tolerant error, then $\overline{s_a}$, $\underline{s_c}$, $\underline{s_b}$, and $\overline{s_d}$ are the extreme points. As previously mentioned, we use line($\overline{s_a}$, $\underline{s_c}$) to denote the line that passes points $\overline{s_a}$ and $\underline{s_c}$, and use line($\underline{s_b}$, $\overline{s_d}$) to denote the line that passes points $\underline{s_b}$ and $\overline{s_d}$. In this way, line($\overline{s_a}$, $\underline{s_c}$) and line($\underline{s_b}$, $\overline{s_d}$) are the bounding lines with extreme slopes for all δ-representatives (Fig. 4).

According to Eqs. (2) and (3), we have the following lemma:

Lemma 1 *For each $1 \leq i \leq k-1$, data range $[y_i-\delta, y_i+\delta]$ at time stamp x_i is intersected or met by line($\overline{s_a}$, $\underline{s_c}$) and line($\underline{s_b}$, $\overline{s_d}$).*

The proof is straightforward: If any range $[y_i - \delta, y_i + \delta]$ is not intersected or met by one of these two lines, it will be contradictory with the definition of $\underline{\text{slp}}[1, k-1]$ or $\overline{\text{slp}}[1, k-1]$.

As depicted in Fig. 4, Lemma 1 means that each data range $[y_i - \delta, y_i + \delta]$, denoted by the thick vertical gray line at x_i, interacts (or meets) with both line($\overline{s_a}$, $\underline{s_c}$) (labeled by $L1$) and line($\underline{s_b}$, $\overline{s_d}$) (labeled by $L2$).

4.2 Slope evolution and reduction

Let $S[1, k]$ be the fragment created by the addition of a new point s_k at the end of fragment $S[1, k-1]$. As inspired by Theorem 1, we need to check if $S[1, k]$ is δ-representable greedily and resolve $\text{slp}[1, k]$, $\underline{\text{slp}}[1, k]$ and $\overline{\text{slp}}[1, k]$ to derive the potential maximal δ-representative. We will interpret how to simplify the process in three stages: Increment, Localization, and the Reduction of convex hull.

First, derived from Lemma 1, we have the following corollary to incrementally determine whether $S[1,k]$ is δ-representable when $S[1,k-1]$ is.

Corollary 1 *Suppose fragment $S[1,k-1]$ is δ-representable, fragment $S[1,k]$ is δ-representable if and only if the range $[y_k-\delta, y_k+\delta]$ at time x_k is not located below line$(\overline{s_a}, \underline{s_c})$ or above line$(\underline{s_b}, \overline{s_d})$.*

Proof For sufficiency, if the range $[y_k-\delta, y_k+\delta]$ at time x_k is not located below line$(\overline{s_a}, \underline{s_c})$ or above line$(\underline{s_b}, \overline{s_d})$, it is obvious that s_k can be approximated within error bound δ by at least one seg$[1,k-1]$, so $S[1,k]$ is δ-representable.

For necessity, given $S[1,k]$ is δ-representable, according to Lemma 1, data range $[y_k-\delta, y_k+\delta]$ must be intersected or met by the bounding lines of $S[1,k]$. Since $S[1,k-1]$ is always δ-representable, the bounding lines of $S[1,k]$ must be covered by those of $S[1,k-1]$, i.e., line$(\overline{s_a}, \underline{s_c})$ and line$(\underline{s_b}, \overline{s_d})$. If the range $[y_k-\delta, y_k+\delta]$ is located below line$(\overline{s_a}, \underline{s_c})$ or above line$(\underline{s_b}, \overline{s_d})$, then it will be also out of the bounding lines of $S[1,k]$, which conflicts with Lemma 1.

Combining the above two parts, the corollary is proven. □

Such corollary provides the method to quick check whether we should keep updating the extreme slopes for $S[1,k]$, or we can determine a fragment $S[1,k-1]$ as maximally δ-representable.

Assuming that $S[1,k]$ is verified as δ-representable, the next question is how to obtain $\underline{\text{slp}}[1,k]$ and $\overline{\text{slp}}[1,k]$. Computing them directly via Eq. (2) can be expensive in time cost as they require the complete computation of the intersection of slopes delineated by the equation. In the following, we will simplify the computation of $\underline{\text{slp}}[1,k]$ and $\overline{\text{slp}}[1,k]$ by incremental and localizing strategies in terms of $\underline{\text{slp}}[1,k-1]$ and $\overline{\text{slp}}[1,k-1]$.

4.2.1 Increment

We will refine Eq. (2) and express $\text{slp}[1,k]$ in terms of $\text{slp}[1,k-1]$. From Eq. (2), we have

$$\begin{cases} \underline{\text{slp}}[1,k] = \max_{1\le i<j\le k} \frac{(y_j-\delta)-(y_i+\delta)}{(x_j-x_i)}, \\ \overline{\text{slp}}[1,k] = \min_{1\le i<j\le k} \frac{(y_j+\delta)-(y_i-\delta)}{(x_j-x_i)}. \end{cases}$$

According to the definitions of $\underline{\text{slp}}[1,k-1]$ and $\overline{\text{slp}}[1,k-1]$, we have

$$\begin{cases} \underline{\text{slp}}[1,k] = \max\limits_{1\le i<k} \{\frac{(y_k-\delta)-(y_i+\delta)}{(x_k-x_i)}, \underline{\text{slp}}[1,k-1]\}, \\ \overline{\text{slp}}[1,k] = \min\limits_{1\le i<k} \{\frac{(y_k+\delta)-(y_i-\delta)}{(x_k-x_i)}, \overline{\text{slp}}[1,k-1]\}. \end{cases} \tag{4}$$

Equation (4) indicates that $\underline{\text{slp}}[1,k]$ and $\overline{\text{slp}}[1,k]$ can be derived by comparing point s_i with s_k $(i<k)$ rather than each pair of points s_i and s_j $(i<j\le k)$. Therefore, the extreme slopes can be more efficiently computed from Eq. (4) than from Eq. (2). However, based on the incremental strategy, we demonstrate that this process can be further simplified, according to the Lemma 1.

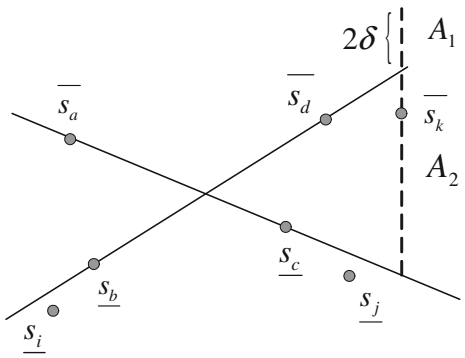

Fig. 5 Localization of slope reduction

4.2.2 Localization

With the availability of $\underline{\text{slp}}[1,k-1]$, $\overline{\text{slp}}[1,k-1]$ and the points $\{\overline{s_a}, \underline{s_c}, \underline{s_b}, \overline{s_d}\}$ defined in Sect. 4.1, we can further reduce the range of points needed to check for the extreme slopes. We take the update of $\overline{\text{slp}}[1,k]$ as an example to explain the reduction.

As exemplified in the Fig. 5, according to Corollary 1, if $S[1,k]$ is δ-representable, $\overline{s_k}$ must be within area A_1 or A_2 (as indicated by the dotted line, and the length of A_1 is 2δ). For $i\in[1,b)$, if $\overline{s_k}$ is in A_1 area, slope$(\underline{s_i}, \overline{s_k}) \ge \overline{\text{slp}}[1,k-1]$ holds; if $\overline{s_k}$ is in A_2 area, the range $[y_b-\delta, y_b+\delta]$ is above line$(\underline{s_i}, \overline{s_k})$. For $i\in(c,k-1]$, the range $[y_c-\delta, y_c+\delta]$ is always above line$(\underline{s_i}, \overline{s_k})$. According to the Lemma 1 and the definition of extreme slopes, if $S[1,k]$ is δ-representable, there is no need to check the points before s_b and after s_c. The similar conclusion can be made on the case of $\underline{\text{slp}}[1,k]$ update. Thus, Eq. (4) can be rewritten into

$$\begin{cases} \underline{\text{slp}}[1,k] = \max\limits_{a\le i\le d} \{\frac{(y_k-\delta)-(y_i+\delta)}{(x_k-x_i)}, \underline{\text{slp}}[1,k-1]\}, \\ \overline{\text{slp}}[1,k] = \min\limits_{b\le i\le c} \{\frac{(y_k+\delta)-(y_i-\delta)}{(x_k-x_i)}, \overline{\text{slp}}[1,k-1]\}. \end{cases} \tag{5}$$

4.2.3 Convex Reduction

In the following, based on the definition of the extreme lines and Lemma 1, we further indicate that the computation can be constrained to those localized points on the convex hulls.

For stream fragment $S[1,k-1]$, let $\overline{\text{cvx}}_{k-1}$ denote the set of convex points of the sequence $\{\underline{s_b}, \underline{s_{b+1}}, \ldots, \underline{s_c}\}$, which are points with deleted tolerant error. Here, $\overline{\text{cvx}}_{k-1}$ bulges upward and is depicted by a sequence of point in the ascent time stamp order. For example, $\overline{\text{cvx}}_{k-1} = \langle \underline{s_b}, \underline{s_i}, \underline{s_j}, \underline{s_c}\rangle$ in

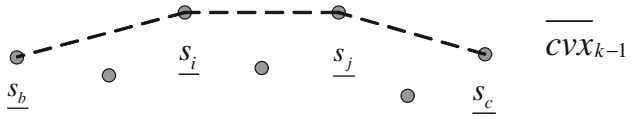

Fig. 6 Example of the convex hull points $\overline{\text{cvx}}$

Fig. 6. Similarly, we define $\underline{\text{cvx}}_{k-1}$ to be the points of the convex hull of $\{\overline{s_a}, \overline{s_{a+1}}, \ldots, \overline{s_d}\}$ that bulges downward. With the notations of $\underline{\text{cvx}}_{k-1}$ and $\overline{\text{cvx}}_{k-1}$, we have the following lemma.

Lemma 2 *If* $\underline{slp}[1,k] = slope(\overline{s_i}, \underline{s_k})$ *and* $x_a \le x_i \le x_d$, *then* $\overline{s_i}$ *is in* $\underline{cvx}_{k-1}$. *Similarly, if* $\overline{slp}[1,k] = slope(\underline{s_j}, \overline{s_k})$ *and* $x_b \le x_j \le x_c$, *then* $\underline{s_j}$ *is in* $\overline{cvx}_{k-1}$.

Proof We only prove the case of $\underline{\text{slp}}[1,k]$. According to the definition of convex hull, if $\overline{s_i}$ is not in the convex hull, line$(\overline{s_i}, \underline{s_k})$ must go inside the convex hull, and there is at least one point $\overline{s_h}$ for $x_a \le x_h \le x_d$ below line$(\overline{s_i}, \underline{s_k})$. If $\underline{\text{slp}}[1,k] = \text{slope}(\overline{s_i}, \underline{s_k})$, it will be contradictory with Lemma 1 and Corollary 1, so $\overline{s_i}$ is in $\underline{\text{cvx}}_{k-1}$. □

Lemma 2 implies that Eq. (5) can be rewritten into

$$\begin{cases} \underline{\text{slp}}[1,k] = \max\limits_{\overline{s_i} \in \underline{\text{CVX}}_{k-1}} \{\frac{(y_k-\delta)-(y_i+\delta)}{(x_k-x_i)}, \underline{\text{slp}}[1,k-1]\}, \\ \overline{\text{slp}}[1,k] = \min\limits_{\underline{s_j} \in \overline{\text{CVX}}_{k-1}} \{\frac{(y_k+\delta)-(y_j-\delta)}{(x_k-x_j)}, \overline{\text{slp}}[1,k-1]\}. \end{cases} \tag{6}$$

Clearly, $\underline{\text{slp}}[1,k]$ and $\overline{\text{slp}}[1,k]$ can be more efficiently obtained from Eq. (6) than from Eq. (5) as the number of points in the convex hulls can be significantly smaller than the total number of data points in the relevant intervals.

5 Optimization strategies

In this section, based on the previous discussion, we will provide some useful theorems, which derive the design of the optimal algorithm for error-bounded PLR generation. The update of extreme slopes and the convex hulls are the major points we will focus on. In the following, we will study the maintenance of $\underline{\text{cvx}}_{k-1}$ and $\overline{\text{cvx}}_{k-1}$ and the derivation of $\underline{\text{slp}}[1,k]$ and $\overline{\text{slp}}[1,k]$ based on Eq. (6). The results are demonstrated in Theorem 2. For simplicity, we will only discuss the results for the update of $\underline{\text{cvx}}_k$ and $\underline{\text{slp}}[1,k]$ as the analogous results exist for $\overline{\text{cvx}}_k$ and $\overline{\text{slp}}[1,k]$.

5.1 Computing extreme slopes

In the process of computing $\underline{\text{slp}}[1,k]$, we first decide whether $\underline{\text{slp}}[1,k] > \underline{\text{slp}}[1,k-1]$ and whether the bounding lines with extreme slopes need to update. Theorem 2 suggests that only the first and last points of the convex hulls need to be used to determine further processing. If the extreme slope $\underline{\text{slp}}[1,k]$ needs to update from $\underline{\text{slp}}[1,k-1]$, Theorem 2 also concludes that the computation can be incrementally performed and the time cost for computing $\underline{\text{slp}}[1,k]$ is proportional to the number of removed points from $\underline{\text{cvx}}_{k-1}$.

Theorem 2 *Suppose* $\underline{cvx}_{k-1} = \langle \overline{s_{i_1}}, \ldots, \overline{s_{i_h}} \rangle$, *here* $\overline{s_{i_1}} = \overline{s_a}$ *and* $\overline{s_{i_h}} = \overline{s_d}$, *then the following results hold for* $\underline{slp}[1,k]$ *and* $\underline{cvx}_k$.

(1) *If* $slope(\overline{s_{i_1}}, \underline{s_k}) \le \underline{slp}[1,k-1]$ *then* $\underline{slp}[1,k] = \underline{slp}[1,k-1]$ *holds.*
(2) *If* $slope(\overline{s_{i_1}}, \underline{s_k}) > \underline{slp}[1,k-1]$ *then* $\underline{slp}[1,k] = \max_e \frac{(y_k-\delta)-(y_{i_e}+\delta)}{(x_k-x_{i_e})}$ *where* $1 \le e \le h$, *and* $\overline{s_{i_m}} \notin \underline{cvx}_k$ *for* $1 \le m < e$.

Proof *The proof of claim (1):* If $\text{slope}(\overline{s_{i_1}}, \underline{s_k}) \le \underline{\text{slp}}[1,k-1]$, then point $\underline{s_k} = (x_k, y_k - \delta)$ is either below or on the line of line$(\overline{s_a}, \underline{s_c})$. Again, under the assumption, each point $\overline{s_{i_e}} \in \underline{\text{cvx}}_{k-1}$ is not below the line of line$(\overline{s_a}, \underline{s_c})$. As a result,

$$\max_{\overline{s_i} \in \underline{\text{CVX}}_{k-1}} \{\frac{(y_k-\delta)-(y_i+\delta)}{(x_k-x_i)}\} \le \underline{\text{slp}}[1,k-1]$$

holds, which leads to $\underline{\text{slp}}[1,k] = \underline{\text{slp}}[1,k-1]$ from Eq. (6).

The proof of claim (2): Clearly, if $\text{slope}(\overline{s_{i_1}}, \underline{s_k}) > \underline{\text{slp}}[1,k-1]$ then point $\underline{s_k} = (x_k, y_k - \delta)$ is above the line of line$(\overline{s_a}, \underline{s_c})$, and the new extreme slope can be derived from the points on the convex hull. From Eq. (6), we have $\underline{\text{slp}}[1,k] = \max_e \frac{(y_k-\delta)-(y_{i_e}+\delta)}{(x_k-x_{i_e})}$ where $1 \le e \le h$. Moreover, since we confirm the new extreme line is determined by $\overline{s_{i_e}}$ and $\underline{s_k}$, s_{i_e} will replace as new s_a, so all the points on the convex hull before $\overline{s_{i_e}}$ will be removed from the new convex hull $\underline{\text{cvx}}_k$. □

In fact, from the theorem and its proof, we can conclude that when we determine that the extreme slope should be updated, the new extreme slope is actually the slope of tangent line from $\underline{s_k}$ to the convex hull $\underline{\text{cvx}}_{k-1}$.

5.2 Updating convex hulls

If the extreme slopes are not updated when processing the new point, then the convex hulls keep unchanged. Or else if we have updated the extreme slopes after adding the new stream point, the convex hulls should be also updated. The maintenance of convex hull is formed by two parts: *point deletion* and *point addition*. We also take the updating of $\underline{\text{slp}}[1,k]$ as an example to explain the process.

The *point deletion* part has been previously mentioned in Theorem 2(2), that is, if $\underline{\text{slp}}[1,k] = \text{slope}(\overline{s_i}, \underline{s_k})$ for $\overline{s_i} \in \underline{\text{cvx}}_{k-1}$, then the earlier points of $\underline{\text{cvx}}_{k-1}$ before $\overline{s_i}$ are NOT in $\underline{\text{cvx}}_k$.

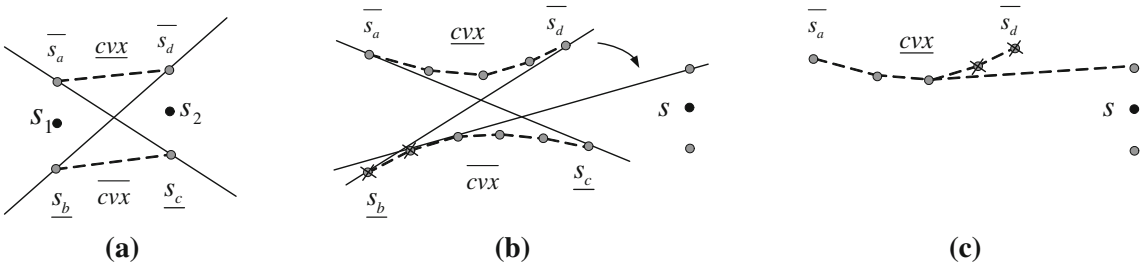

Fig. 7 Optimal algorithm: **a** Initialization; **b** Updating of extreme slopes; **c** Updating of convex hull

For the *point addition* part, we have

$$\overline{\text{cvx}}_k \subseteq \overline{\text{cvx}}'_{k-1} \cup \{\underline{s_k}\},$$

where $\overline{\text{cvx}}'_{k-1}$ denotes the updated $\overline{\text{cvx}}_{k-1}$ after removing those earlier points in *point deletion*. If $\underline{s_k}$ determines the new extreme slope, it should also be added into $\overline{\text{cvx}}_k$, since it will replace as new $\underline{s_c}$ as defined before. After adding $\underline{s_k}$ to the tail of $\overline{\text{cvx}}'_{k-1}$, some previous convex points may have to be deleted in order to keep the convex characteristic. Such process can be derived from the triangle check technique in [1], which will be explained later.

6 Error-bounded PLR algorithms

In this section, we present two linear-time algorithms, OptimalPLR and GreedyPLR, for the error-bounded PLR problem. Let us assume that the given stream fragment has n points, and the error bound for each data point is δ. The OptimalPLR algorithm outputs a set of maximal δ-representative line segments with a minimized space cost upper bounded by n and achieves the minimized number of segments. The GreedyPLR algorithm can benefit the resource-constrained environment, and construct a set of δ-representatives, which individually may be not maximal δ-representative, with constant space cost. For the convenience of algorithm presentation, we use $\underline{\rho}$ or $\overline{\rho}$ to represent the extreme slopes during processing.

In general, the strategy used in these two algorithms is to progressively propagate the extreme slopes $\underline{\text{slp}}[1, k]$ and $\overline{\text{slp}}[1, k]$ upon new stream points in the process of generating δ-representative (or maximal δ-representative) segments.

6.1 Optimal algorithm

Given a stream fragment $S[1, n]$, the OptimalPLR algorithm generates maximal δ-representative line segments starting from x_1 successively. In the process of generating the maximal δ-representative segment from x_i (we assume $i = 1$ without the loss of generality), the OptimalPLR algorithm needs to maintain $\underline{\text{cvx}}_{k-1}$, $\overline{\text{cvx}}_{k-1}$, $\underline{\rho}$, and $\overline{\rho}$ to compute $\underline{\text{slp}}[1, k]$ and $\overline{\text{slp}}[1, k]$ as indicated in Sect. 5. In this part, we first describe the OptimalPLR algorithm and then discuss the time and space complexities of the OptimalPLR algorithm, which are compared with existing works.

6.1.1 Description of OptimalPLR

The general steps of OptimalPLR are interpreted in Fig. 7, and we depict the procedure for constructing a maximal δ-representative line segment in Algorithm 1. Referring to the figure and the algorithm, we describe the OptimalPLR in details.

Initialization Primely, we initialize the extreme lines and the convex hulls by the first two stream points s_1 and s_2 by setting $\overline{\text{cvx}} = \langle \underline{s_b}, \underline{s_c} \rangle$ and $\underline{\text{cvx}} = \langle \overline{s_a}, \overline{s_d} \rangle \cdot (a = b = 1, c = d = 2)$

Extreme slope updating As stated in Sect. 5, the extreme lines and the convex hulls may need to be updated when a new point s is read in. The condition of Line (8) implies that point s is in the current segment.

Taking the update of $\overline{\rho}$ as an example, since the approximate value for point s is within the range of $[\underline{s}, \overline{s}]$, if $\overline{s}$ is NOT under the extreme line line$(\underline{s_b}, \overline{s_d})$, the maximum slope will not be updated and the extreme line line$(\underline{s_b}, \overline{s_d})$ will be used as the new extreme line. Otherwise, the maximum slope should be reduced. As exemplified in Fig. 7b, the strategy is to find the point q of $\overline{\text{cvx}}$ that minimizes slope$(q, \overline{s})$, which can be found by the tangent line of the convex hull from $\overline{s}$. The new extreme line is then defined as line$(q, \overline{s})$, and the new $\overline{\text{cvx}}$ is updated from the old one by removing the points before q.

Convex hull updating After updating the extreme line, $\overline{s}$ should be merged into the upper convex hull $\underline{\text{cvx}}$. Figure 7c depicts the merging strategy. After inserting $\overline{s}$ into the tail of $\underline{\text{cvx}}$, the triangle check [1] needs to be carried out to maintain the convex characteristic. It starts by 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 (refer to [1] for the details of convex hull algorithm).

Algorithm 1: Optimal procedure

Input: **S**: stream fragment starts from x_1; δ: the specified error bound
Output: A maximal δ-representative line segment starts from x_1

1 % **Initialization**
2 $s_1 = (x_1, y_1)$; $s_2 = (x_2, y_2)$;
3 $\overline{s_a} = (x_1, y_1 + \delta)$; $\underline{s_c} = (x_2, y_2 - \delta)$;
4 $\underline{s_b} = (x_1, y_1 - \delta)$; $\overline{s_d} = (x_2, y_2 + \delta)$;
5 $\underline{\rho} = \text{slope}(\overline{s_a}, \underline{s_c})$; $\overline{\rho} = \text{slope}(\underline{s_b}, \overline{s_d})$;
6 $\underline{\text{cvx}} = \langle \overline{s_a}, \overline{s_d} \rangle$; $\overline{\text{cvx}} = \langle \underline{s_b}, \underline{s_c} \rangle$;
7 % **Processing**
8 **while** *s is NOT outside the two lines, line($\overline{s_a}$, $\underline{s_c}$) and line($\underline{s_b}$, $\overline{s_d}$), more than δ* **do**
9 % Maintain $\underline{\rho}$, $\overline{\rho}$, $\underline{\text{cvx}}$ and $\overline{\text{cvx}}$
10 **if** $y + \delta < \overline{\rho}(x - x_b) - y_b$ **then**
11 find the point q of $\overline{\text{cvx}}$ that minimizes slope$(q, \overline{s})$;
12 let $\underline{s_b}$ be q and $\overline{s_d}$ be $\overline{s}$;
13 delete all the points of $\overline{\text{cvx}}$ prior to point q;
14 $\overline{\rho} = \text{slope}(\underline{s_b}, \overline{s})$;
15 insert $\overline{s}$ to the tail of $\underline{\text{cvx}}$, and update $\underline{\text{cvx}}$ by triangle check([1]);
16 **end**
17 **if** $y - \delta > \underline{\rho}(x - x_a) - y_a$ **then**
18 find the point q of $\underline{\text{cvx}}$ that maximizes slope$(q, \underline{s})$;
19 let $\overline{s_a}$ be q and $\underline{s_c}$ be $\underline{s}$;
20 delete all the points of $\underline{\text{cvx}}$ prior to point q;
21 $\underline{\rho} = \text{slope}(\overline{s_a}, \underline{s})$;
22 insert $\underline{s}$ to the tail of $\overline{\text{cvx}}$, and update $\overline{\text{cvx}}$ by triangle check;
23 **end**
24 **end**
25 Let $s_o = (x_o, y_o)$ be the intersection of line$(\overline{s_a}, \underline{s_c})$ and line$(\underline{s_b}, \overline{s_d})$;
26 $\rho = (\underline{\rho} + \overline{\rho})/2$;
27 **return** *a line segment: pass point $s_o = (x_o, y_o)$ with slope ρ*

The correctness of OptimalPLR algorithm is evidenced from both Theorem 1 and the derivation of Eq. (6) as the deduction process preserves the maximum range of slopes for δ-representative segments.

6.1.2 Complexity of OptimalPLR

In the following, we discuss the time and space complexity of the OptimalPLR algorithm.

Time complexity: To show that the time complexity of the OptimalPLR algorithm is $O(n)$ for stream fragment $S[1, n]$, it is sufficient to show the time complexity of Algorithm 1 is $O(k)$ for maximally δ-representable fragment $S[1, k]$.

Clearly, the iteration times of while loop is bounded by k for fragment $S[1, k]$. In each loop, the extreme slopes ($\underline{\text{slp}}$ and $\overline{\text{slp}}$) and convex hulls ($\underline{\text{cvx}}$ and $\overline{\text{cvx}}$) need to be updated for the newly inserted stream point. As indicated in Line (10–15), the costs of updating extreme slopes are dominated by the costs of updating convex hulls. In the process of updating convex hulls, each convex hull ($\underline{\text{cvx}}$ or $\overline{\text{cvx}}$) needs to be maintained by deleting some earlier stream points from it (e.g., Line (13)) and/or inserting a recent stream point into it (e.g., Line (15)). Once a point is deleted from $\underline{\text{cvx}}$ (or $\overline{\text{cvx}}$), the point will not be inserted back into $\underline{\text{cvx}}$ (or $\overline{\text{cvx}}$). Therefore, the total costs for maintaining $\underline{\text{cvx}}$ (or $\overline{\text{cvx}}$) in the process of constructing the segment are bounded by $2k$. Thus, the time cost of Algorithm 1 on fragment $S[1, k]$ is bounded by $(2k + 2k) + ck = (4 + c)k$ where c is a constant number that summarizes other costs in the loop. We conclude that the time complexity of the OptimalPLR algorithm is $O(n)$.

Space complexity: Since each early obtained segment is not used for latter computation and can be output directly, the space cost of the OptimalPLR algorithm on stream fragment $S[1, n]$ is proportional to the space cost on generating a maximal δ-representative segment of $S[1, n]$.

During the process of generating a segment, we only need to maintain cvx plus a constant number of points such as ρ, at each while loop of Algorithm 1. Therefore, the space complexity of the OptimalPLR algorithm is $O(n_{cx})$, where n_{cx} is the size of maximum cvx. Let n_{sg} denote the maximum number of stream points in the derived maximal δ-representative segments of $S[1, n]$. Then, we have $n_{cx} \leq n_{sg} \leq n$ holds. Therefore, the space cost of the OptimalPLR algorithm is also bounded by $O(n)$.

It should also be noted that $n_{cx} \leq m_H$ where m_H is the number of convex points from paper [4] describing the time/space costs on deriving $\underline{\text{slp}}[1, k]$ (or $\overline{\text{slp}}[1, k]$) from $\underline{\text{slp}}[1, k-1]$ (or $\overline{\text{slp}}[1, k-1]$) by SlideFilter algorithm. In addition to the higher space cost of SlideFilter algorithm, this fact also indicates that our algorithm is more efficient, due to the different strategies of updating extreme lines in local scanning. When n is very close to n_{cx}, e.g., when the stream forms a flat concave shape, the number convex points of the SlideFilter algorithm can be much higher, which will affect the processing performance.

In summary, we have the following major result for the OptimalPLR algorithm.

Theorem 3 *Given a stream fragment $S[1, n]$ and the error bound δ, the OptimalPLR algorithm is an optimal algorithm for error-bounded PLR with $O(n)$ time and $O(n_{cx})$ space complexities where $n_{cx} \leq n$.*

6.2 Greedy algorithm

For the GreedyPLR algorithm, the procedure of constructing a δ-representative line segment is depicted in Algorithm 2. The GreedyPLR algorithm is structurally similar to SwingFilter but differs at using different support point to swing up and down. For example, in the process of generating the first line segment for fragment $S[1, n]$, the SwingFilter algorithm always uses the first stream point s_1 as its support point, while the GreedyPLR algorithm uses the intersecting point

Algorithm 2: Greedy procedure

Input: **S**: stream fragment starts from x_1; δ: the specified error bound
Output: A δ-representative line segment starts from x_1

```
% Initialization
s_1 = (x_1, y_1); s_2 = (x_2, y_2);
ρ_ = slope(s̄_1, s_2); ρ̄ = slope(s_1, s̄_2);
Let s_o = (x_o, y_o) be the intersection of line(s̄_1, s_2) and line(s_1, s̄_2);
% Processing
while s is NOT outside the two lines, line(ρ_, s_o) and line(ρ̄, s_o), more than δ do
    % Maintain ρ_ and ρ̄
    if |y − ρ_(x − x_o) − y_o| > δ then
        replace ρ_ with slope(s_o, s);
    end
    if |y − ρ̄(x − x_o) − y_o| > δ then
        replace ρ̄ with slope(s_o, s̄);
    end
end
ρ = (ρ_ + ρ̄)/2;
return a line segment: pass point s_o = (x_o, y_o) with slope ρ
```

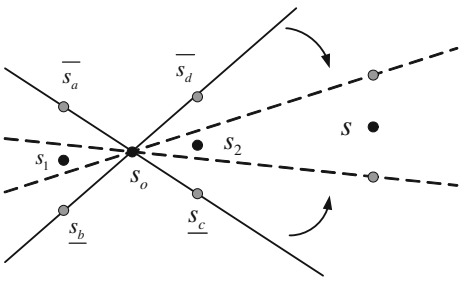

Fig. 8 Description of GreedyPLR algorithm

of line($\overline{s_a}$, $\underline{s_c}$) and line($\underline{s_b}$, $\overline{s_d}$) as its support point (i.e., point s_o in Fig. 8). It should be noted that a support point is not changed during the segment generation.

As interpreted in Fig. 8, the grey points stand for the bounding points with maximum error, and black points are original stream points. At the beginning of the algorithm, we initialize the extreme lines using $\overline{s_a} = \overline{s_1}$, $\underline{s_c} = \underline{s_2}$, $\underline{s_b} = \underline{s_1}$, and $\overline{s_d} = \overline{s_2}$. The GreedyPLR then gradually maintains and updates the extreme slopes. When a new point s is read in, the new constraints for the extreme lines are determined by the line(s_o, $\underline{s}$) and line(s_o, $\overline{s}$). The updated extreme slopes should be in the range of the intersection of the existing range and the new constraints. If the intersection is not empty, replace the extreme lines to be the bounding lines of the intersection. Otherwise, a new segment can be determined by the support point and the existing extreme slope range, and the algorithm makes initialization for new segmentation. The procedure continues until the stream ends.

Apparently, there is no much difference between GreedyPLR and SwingFilter except the support point, but the more significant benefit of GreedyPLR is from the choice of support point for extreme lines to swing, we can derive the following conclusion intuitively. If more stream points are considered to derive the support point, the approximation quality can be improved in terms of approximation ratio, with the trade-off in processing time. This is also the inherent concept of optimal solution OptimalPLR. From this point of view, SwingFilter and OptimalPLR can be viewed as extreme situations of GreedyPLR, that is, SwingFilter chooses the support point as the first stream point, while OptimalPLR determines the support point by maximum number of stream points. With this conclusion, we may modify the GreedyPLR to improve the accuracy and keep constant space cost at the same time. In general, let us assume that the bounding lines are constructed from the first $h-1$ stream points and s_o is the intersecting point of the two extreme lines. We could choose the largest h such that $((\underline{\text{cvx}}_h > 2) \vee (\overline{\text{cvx}}_h > 2)) \wedge (\underline{\text{cvx}}_{h-1} = \overline{\text{cvx}}_{h-1} = 2)$ holds. However, the benefit of GreedyPLR is that we can adjust trade-off between efficiency and effectiveness if necessary. Since the GreedyPLR algorithm has a constant space cost, it is suitable for multi-stream process with limited buffer resource.

6.3 Discussions of PLR algorithms

In this section, we will discuss some important issues of our error-bounded PLR algorithms, including the case of generating connected line segments, and the application for high-dimensional data.

6.3.1 Generation of connected line segments

Generally speaking, all our discussion is based on the hypothesis of generating disconnected line segments for error-bounded PLR, and it guarantees that the proposed algorithm can produce optimal results (minimized number of line segments). However, in some cases, the connected line segments are desired, and we are still interested to discuss whether our solution can be applied to support such cases, although we may not obtain optimal results. For example, the sensor networks may require that each sensor node only reports sampled data values, and the ignored data can be estimated and derived from the recorded values. In such situation, the connected line segments are more appropriate.

We still follow the original idea that derives our optimal strategy. For the case of generating connected line segments, the basic idea is to share the end points between the consecutive line segments, and therefore, we are inspired to use the end point of the previous line segment as the start point of the current one, so that our strategy of PLR generation can still work.

More specifically, as denoted in Fig. 9, the black point is the original stream point, the gray ones are those bounding with maximum added or reduced error, and the current

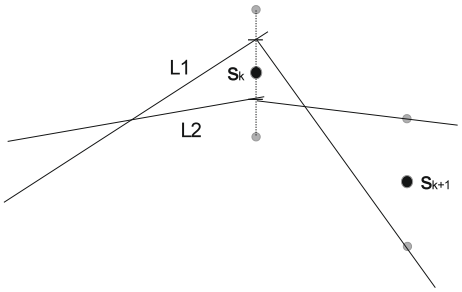

Fig. 9 Extension of OptimalPLR for generation of connected line segments

bounding lines with extreme slopes are L_1 and L_2, respectively. When processing point s_{k+1}, we find that it cannot be approximated by current line segment. Different from the approach of OptimalPLR, we first get the intersection of data range $[y_k - \delta, y_k + \delta]$ at time stamp x_k with the bounding lines. Without the loss of generality, we can view the intersection as the range of a data point with tolerant error, so it can be applied to initialize the extreme slopes of the new line segment together with $\overline{s_{k+1}}$ and $\underline{s_{k+1}}$, as depicted in Fig. 9. As to the further processing, we can still follow the approach of OptimalPLR.

It is obvious that such strategy can produce connected line segments and guarantee that the approximation error for each point can be restricted within error bound. However, it cannot promise the optimal result, i.e., the minimized number of segments. As to the discussion of the optimal solution for producing connected line segments, it will be out of the scope of this work.

6.3.2 Extension for high-dimensional data

High-dimensional data are popular in streaming data nowadays, so we also concern whether our solution can work on high-dimensional data and how we can extend our algorithm effectively.

Since our aim was to generate error-bounded PLR for data stream, it is essential how we define the error bound. Different from the case that the stream point value is a single number, for high-dimensional data, we have to consider whether the error bound is defined on each dimension, or on the data point in high-dimensional space.

Case 1: error bound defined on each dimension.

If the error bound is same for each dimension, it will be easy to extend. We can simply treat the data values of each dimension as a data stream and carry out the original OptimalPLR algorithm to generate line segments. To combine the segments of different dimensions together, when we find the first maximal δ-representative for any dimension, we stop and determine a maximally representable segment, and then start a new segmentation. Such strategy is also applied in [4].

Case 2: error bound defined on high-dimensional point.

If the error bound is defined in high-dimensional space, it will be more complicated. We will only discuss the data stream in 3D space, in which each stream point at time stamp x_k is identified by $\{y_k, z_k\}$. In this case, the error bound δ will be defined as follows: For each stream point s_k, the Euclidean distance between original point and the approximate point satisfies $\sqrt{(y'_k - y_k)^2 + (z'_k - z_k)^2} \leq \delta$. From geometrical view, the error bound describes a circle with radius δ around the point s_k, so the OptimalPLR cannot be applied for this situation directly. However, we can still find an approximated way to embed OptimalPLR.

First we have the following relationship:

$$\sqrt{2|y'_k - y_k||z'_k - z_k|} \leq \sqrt{(y'_k - y_k)^2 + (z'_k - z_k)^2} \leq \delta.$$

If we define $|y'_k - y_k| \leq \delta \Big/ \sqrt{2}$ and $|z'_k - z_k| \leq \delta \Big/ \sqrt{2}$, then it can be guaranteed that the above relationship always holds. In this way, instead of dealing with the error bound in high-dimensional space, we can define alternative error bound in each dimension, and the solution can be designed as Case 1.

It should be noticed that such strategy cannot guarantee optimal results (minimized number of segments), since we are not maintaining all feasible line segments. However, it can be executed efficiently and promise the bounded approximation error in high-dimensional space.

7 Equivalence of two spaces

Although our proposed algorithm generates the optimal results with linear complexity, the previous work of [18] also obtains similar approximation quality, which designs the algorithm in a special slope-offset parameter space. As a result, we are motivated to compare our time-domain-based approach with the ParaOptimal algorithm designed in [18] theoretically. We propose to analyze the relationship and difference between the time-value space (or stream space) and the parameter space, which is able to provide a deeper understanding for the optimal algorithm.

7.1 ParaOptimal algorithm

First of all, we briefly elaborate the ParaOptimal algorithm to indicate how it works to create piecewise linear representation.

Similarly, the data stream is a sequence of data points which are numerical numbers, and we denote it as $S =$

$\langle s_1, s_2, \ldots, s_n \rangle$. Each data point s_i consists of the value y_i and the time stamp x_i. The approximate value of s_i can be determined by $y_i' = a \cdot x_i + b$, where the a and b, respectively, stand for the slope and the offset of the line segment approximating s_i. The error bound δ is defined to restrict the approximate error on single point, i.e., $\forall i \in [1, n], |y_i' - y_i| \leq \delta$.

The ParaOptimal is a greedy and incremental algorithm in which the line segments are generated and adjusted to approximate as many points as possible. The basic idea of ParaOptimal is to determine the line function by maintaining the feasible region of all line candidates in the parameter space. More specifically, for a point s_i, the approximate value should be within the error bound, that is,

$$y_i - \delta \leq a \cdot x_i + b \leq y_i + \delta.$$

It means any pair of parameters $\{a, b\}$ that meets the above inequalities can identify a candidate line function, which composes the solution set. If more data points are approximated, the solution set of line functions approximating all data points will be the intersection of all solution sets for individual points.

Considering the parameter pair in the parameter space (Fig. 10), each pair can be viewed as a point, and the error bound provides linear constraints on the parameters. The ParaOptimal algorithm runs in this way: Each new point provides two linear constraints; the algorithm incrementally updates the feasible region of candidate line parameters by intersecting the feasible region with the solution region of new linear constraints; a new line segment is determined if the feasible region is empty, and then, the algorithm initializes the feasible region for a new segmentation. Figure 10 exemplifies the feasible region update in parameter space.

7.1.1 Theoretical preparation

Primely, we provide some basic properties and lemmas for the further description.

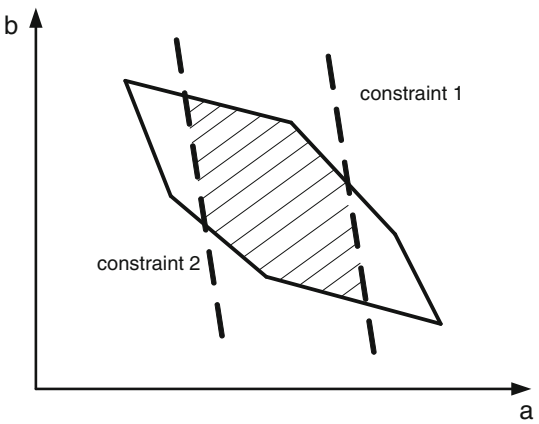

Fig. 10 Feasible region update in parameter space

Lemma 3 *For the candidate line parameters, any edge of the feasible region belongs to a boundary of a linear constraint.*

Proof Since the feasible region is the intersection of all the solution regions for all linear constraints, it is obvious that the boundaries of the feasible region are composed of the boundaries of linear constraints, so any edge is part of a boundary of a linear constraint. □

Corollary 2 *The feasible region of the candidate line parameters is a convex polygon.*

Proof Given an edge of the feasible region, it is part of a boundary of certain linear constraint. Since the feasible region is the intersection of all solution regions of linear constraints, it must be the subset of the solution region of corresponding linear constraint, so all the feasible region lies on one side of the given edge. According to the convex definition, the feasible region is a convex polygon. □

Lemma 4 *For the boundary of linear constraints provided by error bound, the slopes are less than zero, and decreasing monotonously as the order of data points.*

Proof Given an upper error bound of point s_i, the linear constraint is identified by

$$a \cdot x_i + b \leq y_i + \delta,$$

so the slope of the linear constraint boundary is $-x_i$. Since $x_i > 0$ always holds, the slope is less than zero.

For any two points s_i and s_j, if $x_i < x_j$, then the slope of the constraint boundary corresponding to s_i is greater than that of s_j. □

Corollary 3 *The left most point of the feasible region is the highest point; the right most point of the feasible region is the lowest point.*

Proof Given a point on the feasible region boundary, since all the edge slopes are less than zero, it must be lower than the boundary point to its left. The rest can be done in the same manner, and we will have the point is lower than the leftmost point of the feasible region. So, the leftmost point is the highest point. Similarly, we can have the rightmost point of the feasible region is the lowest point. □

Now, we describe the ParaOptimal algorithm step by step in the following parts.

7.1.2 Initialization

In the parameter space, the error bound of a data point will provide two linear constraints for the candidate line parameters, which compose a pair of parallels. When the segmentation starts, to restrict the feasible region with bounded size, the algorithm considers the first two points for initialization and generates the feasible region. Figure 11 demonstrates this process.

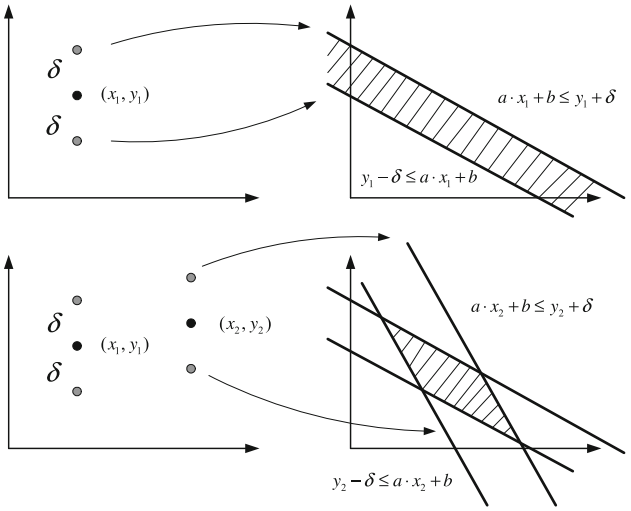

Fig. 11 Initialization of feasible region

7.1.3 Feasible region update

When the algorithm processes a new data point, it updates the current feasible region according to the new linear constraints and checks whether it is possible to process more points, or else a line segment can be determined. For simplification, we only discuss the linear constraint by the upper error bound.

Assume we process a new data point s_i, and the linear constraint provided by the upper error bound is

$$a \cdot x_i + b \le y_i + \delta.$$

Since x_i is always greater than zero, the solution area for the above linear constraint is the half space to the left of the boundary, which is identified by linear function $a \cdot x_i + b = y_i + \delta$. As exemplified in Fig. 13 in the clockwise order, we divide the edge points of feasible region into $\langle \overline{p}_1, \overline{p}_2, \ldots, \overline{p}_t \rangle$ and $\langle \underline{p}_1, \underline{p}_2, \ldots, \underline{p}_s \rangle$ by the leftmost point p_l and rightmost point p_r. Each point is a 2-tuple $\{a, b\}$ in parameter space.

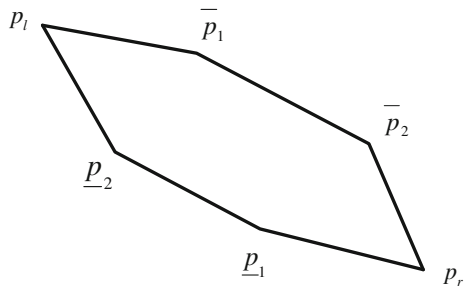

Fig. 13 Points definition of feasible region

The feasible region update can be categorized into three cases (Fig. 12).

Case 1: $p_r.a \cdot x_i + p_r.b < y_i + \delta$. In this case, the whole feasible region satisfies the linear constraint, so the updated feasible region will remain the same and no extra processing is needed.

Case 2: $p_l.a \cdot x_i + p_l.b \le y_i + \delta \le p_r.a \cdot x_i + p_r.b$. In this case, the solution area of the linear constraint will intersect with the current feasible region, and the updated feasible region should be determined. Since we discuss the linear constraint from the upper error bound, we can first determine the new p_r for the new feasible region. We start from the first point of the sequence $\langle \underline{p}_1, \underline{p}_2, \ldots, \underline{p}_s \rangle$, and check backwards until the current point $\underline{p}_i$ lies on the right side of the boundary of linear constraint, that is, $\underline{p}_i.a \cdot x_i + \underline{p}_i.b \le y_i + \delta$. Then, we calculate the intersection point generated by the boundary line of linear constraint, and the line identified by point $\underline{p}_i$ and point $\underline{p}_{i-1}$. (For $\underline{p}_1$, $\underline{p}_{i-1}$ is p_r.) We set the intersection point as new p_r. Then, the sequence of $\langle \overline{p}_i \rangle$ should be updated similarly as the previous procedure did, and then, the feasible region can be updated according to the new boundary points.

Case 3: $p_l.a \cdot x_i + p_l.b > y_i + \delta$. In this case, the whole feasible region will fall outside the solution area of linear constraint, so the updated feasible region will be empty. The

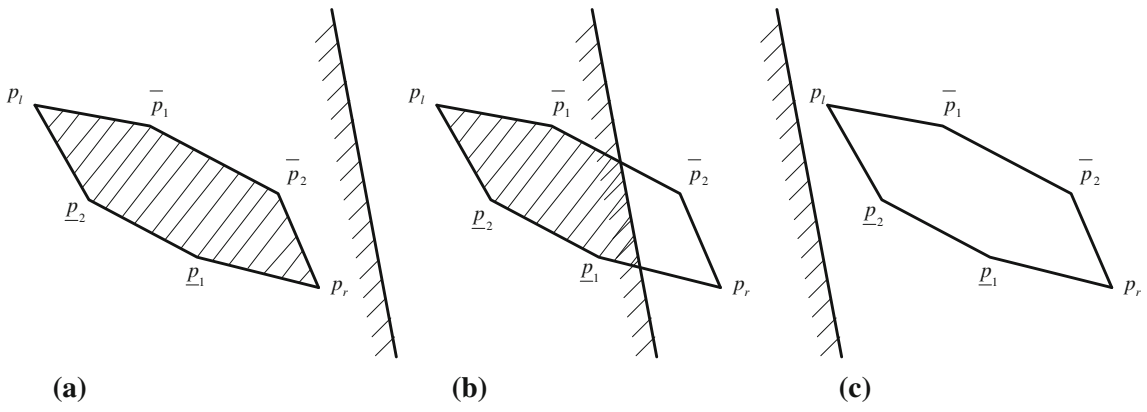

Fig. 12 Three cases for feasible region update. **a** Case 1. **b** Case 2. **c** Case 3

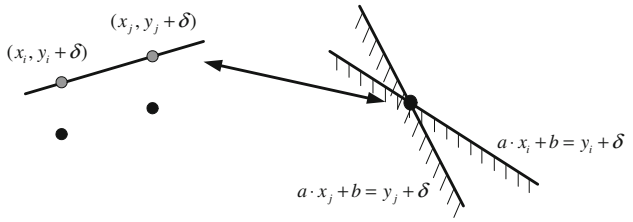

Fig. 14 Mapping between two spaces

line segment can be determined, and new segmentation can be initialized.

After the feasible region update, if the new feasible region is empty, we can randomly choose a point in the old feasible region and apply the line segment identified by the chosen parameter to approximate the streaming points until the current point. The current point will be applied to initialize a new segmentation. Otherwise, the feasible region is not empty, and the algorithm will continue reading points and updating feasible region. All the procedures will work until the data stream ends.

7.2 Equivalence between two algorithms

In this part, we will formally compare the OptimalPLR algorithm and ParaOptimal algorithm and discuss the equivalence between the two spaces and these two algorithms.

7.2.1 Mapping of two space

First, we put our effort to discover the relationship between the stream point space and line parameter space, namely S-space and P-space. We have the following theorem:

Theorem 4 *The line function identified by two points in S-space, is the intersection point of two lines in P-space, which are corresponding to the linear constraints brought by related points in S-space. Vise versa.*

Proof As denoted in Fig. 14, consider the stream points s_i and s_j, and for the line function to approximate the data points, the function parameters must satisfy the following inequality set for upper error bound:

$$\begin{cases} a \cdot x_i + b \leq y_i + \delta \\ a \cdot x_j + b \leq y_j + \delta \end{cases}$$

Therefore, the boundary lines for corresponding linear constraints in P-space are $a \cdot x_i + b = y_i + \delta$ and $a \cdot x_j + b = y_j + \delta$, respectively. In this way, we can calculate the intersection point of these two lines by combining and solving the above equations.

Notice that in S-space, the line identified by the upper error bound points of s_i and s_j is the line crossing $(x_i, y_i + \delta)$ and $(x_j, y_j + \delta)$. So, the line function parameters are just mapped to the intersection point in the P-space as described above.

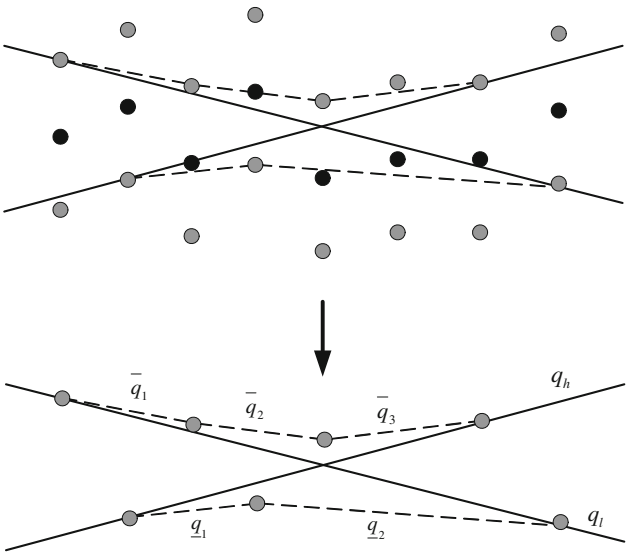

Fig. 15 Segment definition of convex hull in S-space

With the same argument, we claim that the line function identified by two points in P-space is the intersection point of corresponding lines in S-space. □

7.2.2 Equivalence discussion

Based on Theorem 4, we can formally establish the equivalence between OptimalPLR and ParaOptimal algorithms.

For OptimalPLR, we model the convex hull which constrains the line candidates in the following way. As depicted in Fig. 15, the segment sequence composing the upper convex hull is denoted as $\langle \overline{q}_1, \overline{q}_2, \ldots, \overline{q}_s \rangle$, and the sequence composing the lower convex hull is $\langle \underline{q}_1, \underline{q}_2, \ldots, \underline{q}_t \rangle$. The line segments with maximum slope and minimum slope are q_h and q_l, respectively.

Define $p \sim q$ as the mapping relationship between point p and segment q, and the following theorem illustrates the equivalence between OptimalPLR and ParaOptimal in different spaces.

Theorem 5 *The points on the feasible region in P-space are one-to-one mapped with those segments composing the convex hull in S-space. More specifically,* $\overline{p}_i \sim \overline{q}_i$, $\underline{p}_i \sim \underline{q}_i$, $p_r \sim q_h$ *and* $p_l \sim q_l$.

We apply mathematical induction method to prove the theorem.

Proof **Step 1**: For the first two points of initialization, the theorem holds obviously (Fig. 16).

Step 2: Assume for the first k points, the theorem holds.

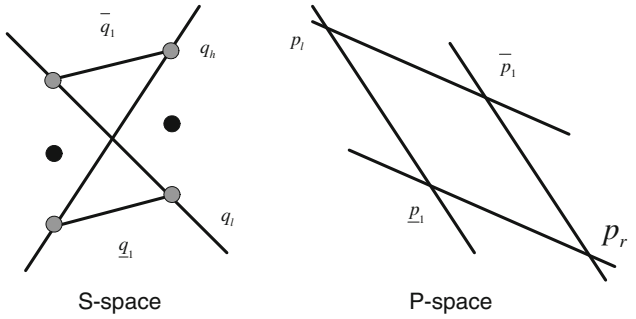

Fig. 16 Mapping for the first two points

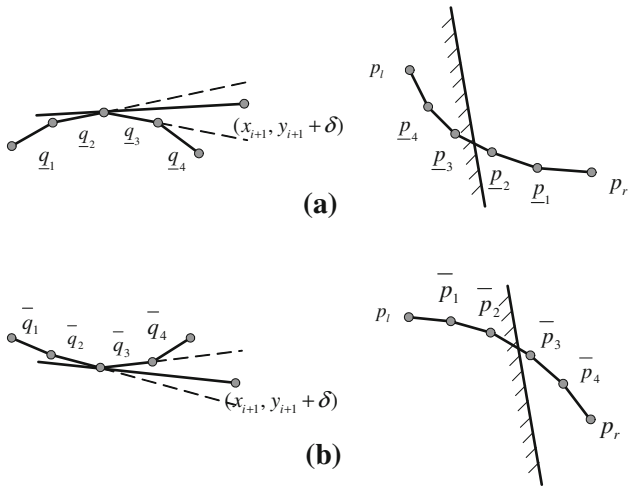

Fig. 17 Mapping for k+1 points

Step 3: Now, we discuss when processing the $k+1$ point and still only consider the upper error bound of s_{k+1}. The case of lower error bound can be proved with similar argument.

- **Case 1**: $q_h.a \cdot x_{i+1} + q_h.b < y_{i+1} + \delta$. In this case, the segment indicating the maximum slope q_h will not change. Since $q_h \sim p_r$, Case 1 describes the same situation as the first case in Fig. 12.
- **Case 2**: $q_l.a \cdot x_{i+1} + q_l.b \leq y_{i+1} + \delta \leq q_h.a \cdot xi + 1 + q_h.b$. In this case, q_h should be adjusted. According to OptimalPLR, the new q_h is the tangent line from the point $(x_{i+1}, y_{i+1} + \delta)$ toward the lower convex hull. As described in Fig. 17a, the point of tangency is the intersection point of $\underline{q}_2$ and $\underline{q}_3$, one of which is out of the upper error bound, and the other of which is within the upper error bound. According to Theorem 4, the intersection point is the segment connecting $\underline{p}_2$ and $\underline{p}_3$. Therefore, the new q_h is mapped with new p_r. For the lower convex hull, $\underline{q}_1$ and $\underline{q}_2$ should be deleted, and also for the feasible region in the P-space, $\underline{p}_1$ and $\underline{p}_2$ are deleted. So, the rest of $\underline{q}_i$ and $\underline{p}_i$ is still mapped.
  Furthermore, the upper convex hull should also change according to the new q_h. The OptimalPLR proposes to find the tangent line from point $(x_{i+1}, y_{i+1} + \delta)$ toward the upper convex hull. Similar as lower convex hull, we can find the tangency point, and $\overline{q}_3$ and $\overline{q}_4$ are deleted. Accordingly, in P-space, $\overline{p}_3$ and $\overline{p}_4$ are deleted, and the rest of $\overline{q}_i$ and $\overline{p}_i$ is still mapped.
- **Case 3**: $q_l.a \cdot x_{i+1} + q_l.b > y_{i+1} + \delta$. In this case, the convex hull to indicate the possible line segment candidates is empty, which is exactly same as the third case in Fig. 12.

Based on the above analysis, we prove that ParaOptimal and OptimalPLR are theoretically equivalent. □

The theoretical meaning above is that it guarantees that the two methods are basically the same, which reveals the inherent reason deriving the optimal approximation with linear complexity and also provides the understanding of the algorithm from different view. However, the two algorithms may have different performance in practice, due to the different ways of recording intermediate data during the processing, which results from the space they are based on. Such difference may affect the processing efficiency in both memory and time cost under certain situation. We will further analyze and discuss this issue with the experimental results in the next section.

8 Experimental evaluation

In this section, we present the experimental evaluation of our proposed OptimalPLR and GreedyPLR algorithms to illustrate their performance on data approximation, by a wide range of synthetic and real datasets. We compare the results to those achieved by SwingFilter and SlideFilter of [4], and ParaOptimal of [18]. All the algorithms are implemented in C++, and all the experiments are performed on a PC with CPU of Intel Core 2.50 GHz and 4 GB memory.

Since the error-bounded PLR is a representation to approximate data stream, which is also a kind of compression, we propose to evaluate the performance of the algorithms by the compression quality (the number of segments generated) and the time efficiency (construction time). We choose the number of segments to indicate the compression quality, because the approximation results are usually achieved as line segments in database (i.e., the end points and the slope); on the other hand, our algorithm produces disconnected line segments, so it is convenient to evaluate the compression quality by the segment number.

We will first verify the effectiveness of our proposed algorithms by conducting the PLR generation on all datasets, which cover diverse domains with different characteristics. Then, by tuning the synthetic data, we test the effect of different factors on the compression quality and the time efficiency, including scalability analysis. Finally, we discuss the

trade-off issues for GreedyPLR and provide some severe situations to demonstrate the practicability and efficiency of OptimalPLR algorithm over other algorithms.

8.1 Dataset description

The experiments are conducted on both real and synthetic datasets. We choose the UCR Time Series Archive [13] as real dataset, since it contains a large number of real data with diverse characteristics, and widely applied for evaluation in time series problems. We are able to make more valid and convincing conclusion about the algorithm effectiveness based on the performance on such data collection. However, the dataset consists of time series with short length, so we concatenate the short sequences together to simulate the long-streaming data.

The synthetic dataset contains two kinds of data, with distribution following *Zipf* and *Random* respectively, as exemplified in Fig. 18a, b. *Zipf* contains data values following zipf distribution with a zipf parameter of 2.0, i.e., a highly skewed distribution. *Random* follows the random model, in which each point value can be lower or higher than that of the previous data point. That is, $y_{i+1} = y_i + \varepsilon$ and $\varepsilon \sim N(0, 1)$. The synthetic data can simulate the regularly specified data and totally random data, and we are able to tune the parameter of the distribution to control the data characteristics for different experimental tests.

8.2 Overall comparison on different datasets

We first carry out the PLR algorithms on all datasets and record the number of segments generated and the construction time. For the data streams of real dataset, we set the error bound as the 5 % of the value range of the stream, for the sake of fairness. For synthetic dataset, we set the error bound of *Random* data as 2, and the error bound of *Zipf* data is 20. The overall comparison results are listed in Table 2.

From the table, we can conclude that the extensive experimental results highly support the viewpoints and the theoretical results in previous sections. OptimalPLR, ParaOptimal, and SlideFilter are all optimal algorithms and achieve similar approximation performance. Overall speaking, three algorithms can all generate the same number of segments, while generally OptimalPLR can have better time efficiency, especially when the data stream is long and the compression ratio is high, e.g., the **StarLight** stream and **Symbols** stream. SwingFilter has the highest time efficiency, with the sacrifice of lowest compression ratio. GreedyPLR algorithm stands at middle position and shows the trade-off between time efficiency and compression quality, and better performance than SwingFilter, which matches our previous discussion.

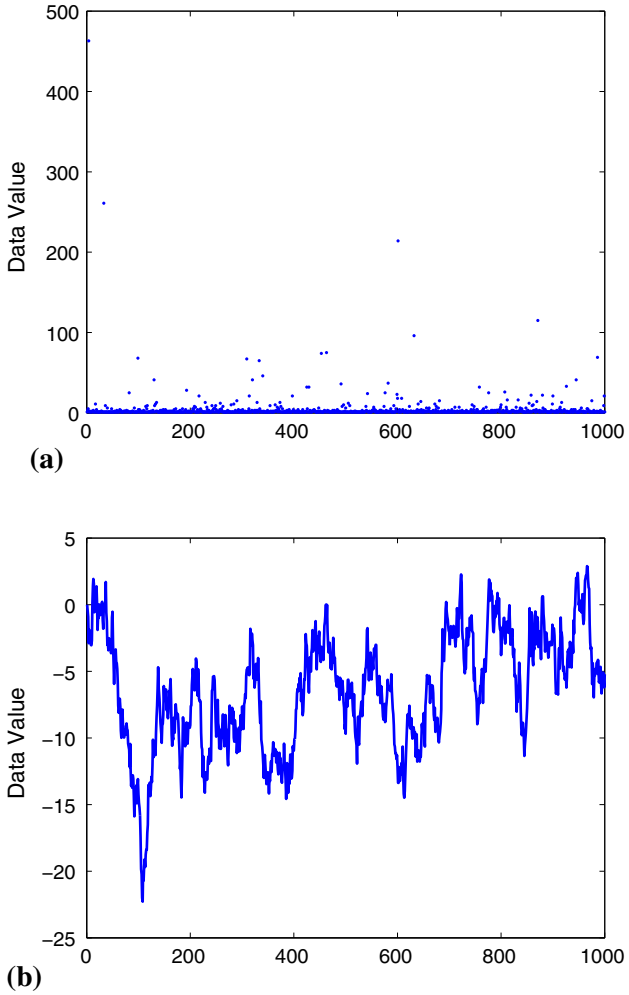

Fig. 18 Examples of synthetic datasets. **a** Zipf. **b** Random

8.3 Effect of different factors

Based on the general results above, in this part, we will change and tune some important parameters and discover the effect of different factors on the approximation performance of different algorithms. For the convenience of parameter tuning, we choose *Random* data for testing.

8.3.1 Effect of error bound

In this experimental evaluation, we choose the data sequence from the dataset with fixed length of 10^6 and carry out different error-bounded PLR algorithms on it with varied error bounds. The number of segments and the construction time based on different error bounds are recorded in Figs. 19 and 20, respectively.

It is depicted from Fig. 19 that the number of segments decreases as the error bound is enlarged, because each line segment can potentially approximate more stream points if

Table 2 Comparison of different algorithms

Data	Length	Number of segment			Processing time (ms)				
		SwingFilter	GreedyPLR	Optimal results[a]	SwingFilter	SlideFilter	ParaOptimal	GreedyPLR	OptimalPLR
50words	121,500	4,536	4,386	3,808	3	66	57	6	49
Adiac	68,640	2,178	2,139	1,830	2	34	34	3	29
Beef	14,100	153	149	127	<1	6	6	1	5
CBF	115,200	29,547	25,585	23,475	3	37	48	17	40
Chlorine	77,522	6,084	5,693	4,270	1	29	27	6	26
CinC_ECG	65,560	233	231	186	2	26	19	1	19
Coffee	8,008	409	392	332	<1	3	3	1	6
Cricket_X	117,000	2,605	2,469	2,064	2	45	38	5	37
Cricket_Y	117,000	2,607	2,465	2,015	3	46	37	5	37
Cricket_Z	117,000	2,582	2,472	2,053	2	47	41	5	41
Diatom	105,570	1,767	1,757	1,497	3	54	48	4	59
ECG200	9,600	892	809	732	<1	3	4	1	4
ECGFiveDays	117,096	4,846	4,622	4,236	3	50	48	6	47
FaceFour	30,800	2,306	2,163	1,876	1	12	12	3	11
FacesUCR	26,200	3,271	3,033	2,674	1	10	11	6	11
Fish	81,025	1,530	1,507	1,276	1	42	38	3	34
Gun_Point	22,500	676	674	652	<1	11	10	1	11
Haptics	169,260	1,358	1,299	1,025	3	75	56	4	55
InlineSkate	188,200	618	609	504	5	74	52	4	55
ItalyPower	24,696	5,307	4,872	4,444	1	10	11	4	10
Lighting2	38,220	433	403	339	1	13	10	2	11
Lighting7	22,330	472	435	357	<1	8	7	1	7
MALLAT	56,320	1,439	1,401	1,176	1	36	26	3	23
MedicalImages	75,240	3,553	3,392	2,977	2	34	34	4	30
Motestrain	105,168	4,905	4,713	4,011	3	45	44	6	39
OliveOil	17,100	506	480	397	<1	9	7	1	7
OSULeaf	85,400	2,943	2,904	2,598	2	37	37	5	37
RobotSurface	42,070	8,099	7,379	6,870	4	15	19	6	16
RobotSurfaceII	61,945	13,055	11,973	11,229	14	31	35	18	30
StarLight	1,024,000	4,250	4,236	3,670	18	**840**	**398**	22	**384**
SwedishLeaf	64,000	4,318	4,127	3,652	2	26	29	4	26
Symbols	396,010	6,408	6,285	5,338	7	**191**	**158**	11	**148**
Synthetic_Control	18,000	6,928	6,413	6,300	1	7	8	5	6
Trace	27,500	431	424	369	1	12	8	1	10
Two_Patterns	128,000	31,521	28,818	27,954	3	44	72	29	58
TwoLeadECG	93,398	6,610	6,287	5,642	2	37	40	7	37
uWave_X	282,240	5,360	5,296	4,496	9	121	113	8	109
uWave_Y	282,240	3,675	3,640	2,946	5	127	117	8	110
uWave_Z	282,240	5,497	5,409	4,556	6	125	122	9	107
Wafer	152,000	4,342	4,166	3,902	3	56	75	6	70
WordsSynonyms	72,090	2,660	2,583	2,262	1	38	33	3	44
Yoga	127800	2,797	2,764	2,317	3	59	54	5	54
Zipf	100000	3,413	3,287	2,810	3	35	31	5	32
Random	100,000	5,789	5,502	4,259	3	36	36	6	37

[a] The optimal results are same for three optimal algorithms: SlideFilter, ParaOptimal, and OptimalPLR

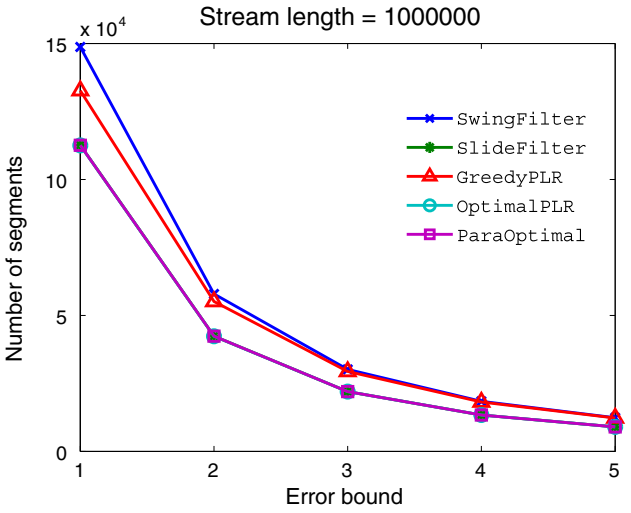

Fig. 19 Effect of error bound on compression ratio (the results of SlideFilter, OptimalPLR, and ParaOptimal are overlapped)

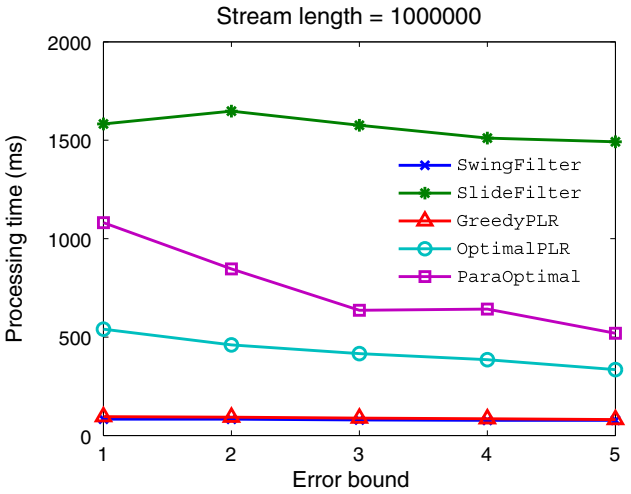

Fig. 20 Effect of error bound on time efficiency

a larger error bound is provided. Since SlideFilter, OptimalPLR, and ParaOptimal can all construct the optimal number of segments for a stream, their results are overlapped, which outperform the results of SwingFilter and GreedyPLR. The results indicate that GreedyPLR achieves better compression quality than SwingFilter on the tested data set with 4–4 % improved compression ratio, which results from the different support point of GreedyPLR, and the optimal solution exceeds the greedy solutions by about 15 % in compression quality.

However, the improvement on compression quality will sacrifice the processing efficiency. Figure 20 demonstrates the segmentation construction time on the data stream, which indicates the trade-off between compression quality and time efficiency. It shows that the time costs have little variation on different error bounds. This fact indicates that the time efficiency of the algorithms is dominated by the number of stream points processed rather than the error bound. In the figure, the SwingFilter and GreedyPLR achieve the best performance due to the simple strategy of extreme lines maintenance. OptimalPLR and ParaOptimal have similar performance and are significantly faster than SlideFilter. The OptimalPLR can be 1.5 and 4 times faster than ParaOptimal and SlideFilter, respectively, in all situations, which illustrates the theoretical analysis results on the algorithm complexity in previous sections. As to the difference between OptimalPLR and ParaOptimal, we will further analyze in the following evaluation.

8.3.2 Effect of data variation

Besides the error bound, the characteristics of stream fluctuation will also affect the approximation quality. In this part, we will change the variation of the test data to see its effect, by tuning the standard deviation of the normal distribution, based on which the random data are generated. We tune the standard deviation from 1 to 5 and generate random stream of length 10^6. The results to the varied standard deviation are recorded in Figs. 21 and 22, respectively.

Figure 21 shows that as the increase in standard deviation, all algorithms will generate more line segments to approximate the stream, because the larger standard deviation will result in more severe fluctuation in the stream, which will reduce the capability of a line segment approximating more stream points within error bound. As expected, the results of SlideFilter, OptimalPLR, and ParaOptimal are overlapped, and always outperform the results of SwingFilter and GreedyPLR.

The results for time efficiency in Fig. 22 follow the similar trend as in the test of error bound effect. Although the

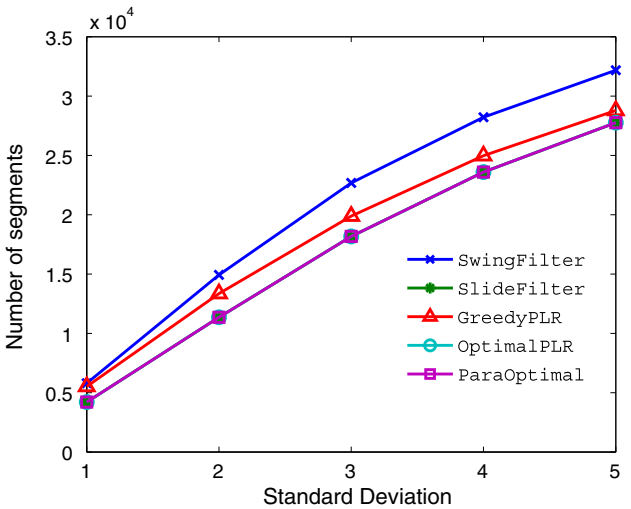

Fig. 21 Effect of data variation on compression ratio (the results of SlideFilter, OptimalPLR, and ParaOptimal are overlapped)

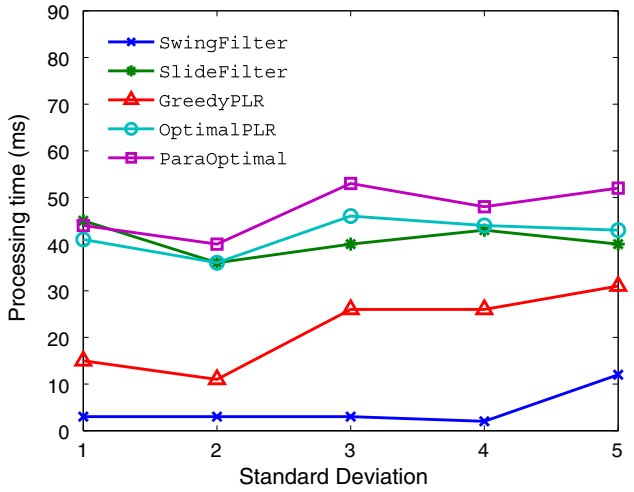

Fig. 22 Effect of data variation on time efficiency

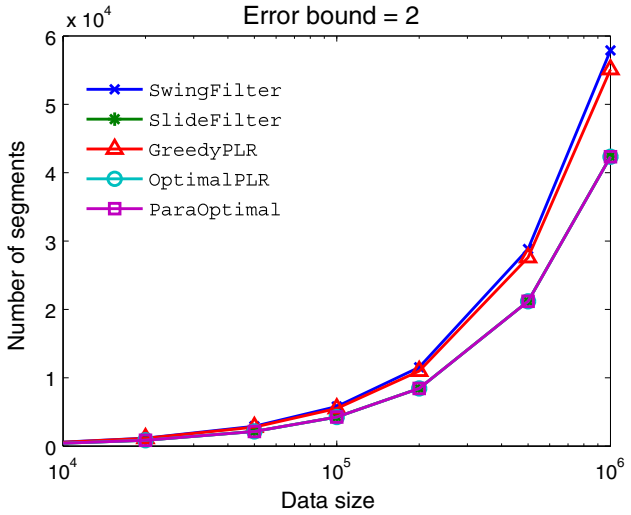

Fig. 23 Effect of stream length on compression ratio (the results of SlideFilter, OptimalPLR, and ParaOptimal are overlapped)

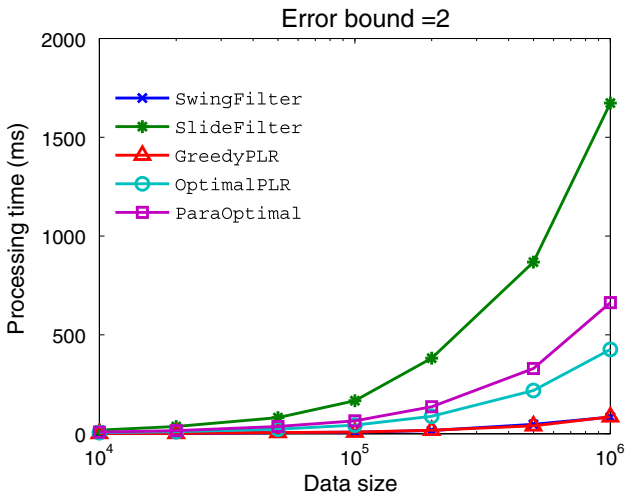

Fig. 24 Effect of stream length on time efficiency

changed standard deviation has affected the stream fluctuation, the construction time still varies little given different deviations. This test further verifies that the time cost is related mainly with the length of the stream. Again the SwingFilter achieves best performance in time efficiency, and the time costs of three optimal algorithms stay at the same level.

8.3.3 Scalability evaluation

To test the performance trends of different algorithms on long data streams, we gradually increase the length of test data sequences and apply different algorithms to generate segments on these sequences with the fixed error bound. The results of the number of generated segments and construction time on different sequence length are recorded and demonstrated in Figs. 23 and 24, respectively. To produce curves of better view, we set the data size axis in logarithmic mode.

Figure 23 demonstrates the changing trends of the synopses size under the increased sequence length with a fixed error bound. It shows that GreedyPLR has better compression quality on various data sizes than that of SwingFilter; however, the superiority is not significant. SlideFilter, OptimalPLR, and ParaOptimal still show similar performance and achieve 15–25 % superiority over those achieved by GreedyPLR and SwingFilter. It should be noticed that the results for the data size less than 10^5 are not well distinguished due to the wide range of sequence length.

The comparison on time efficiency can better emphasize the superiority of proposed algorithms. Figure 24 shows that the running time of our proposed algorithms scales well with larger data sizes, while SlideFilter has worse performance when sequence size increases. In these tests, both proposed algorithms can process 10^6 data points in less than 600 ms on the *Random* data set. The OptimalPLR algorithm is often above 4 times faster than the SlideFilter algorithm, which can illustrate the theoretical analysis of the algorithm complexity.

8.4 Discussion of proposed algorithms

In this section, we will focus on the proposed algorithms and discuss some issues mentioned in Sect. 6, which will provide alternative choice to apply the algorithms in real applications, and also further demonstrate the robustness and practicability of our proposed algorithms.

8.4.1 Trade-off issues for GreedyPLR

In Sect. 6.2, we have indicated that the performance of GreedyPLR algorithm can be adjusted by choosing different

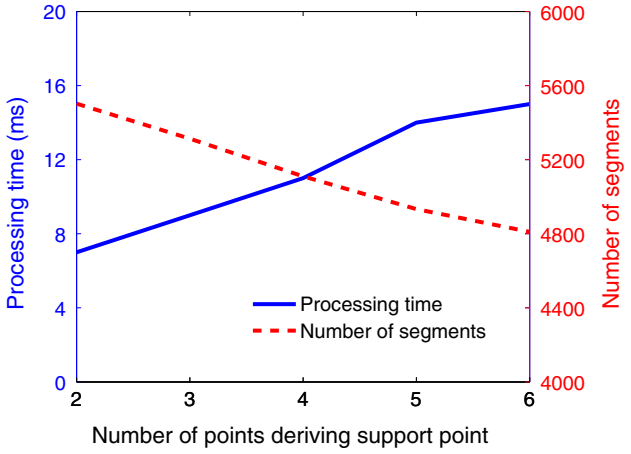

Fig. 25 Trade-off discussion for GreedyPLR

support point for extreme lines to swing. Generally speaking, if the support point is derived from more initial points, the number of line segments will decrease and the construction time will increase accordingly. In this part, we will focus on this trade-off issue and study the performance change based on the different choice of support point. The results are denoted in Fig. 25.

As denoted in the figure, when we use more initial points to derive the support point, the compression ratio will be significantly improved, under the sacrifice of the time efficiency. As we can expect, the performance will finally converge to the optimal results. Such results inspire that we can to certain degree tune the construction time and the compression ratio according to specific approximation criteria, which provides some freedom to adjust the approximation results to meed different requirements.

8.4.2 Space and time tests on severe situations

In the Sect. 6.1.2, we indicated that our algorithm is more efficient than SlideFilter. However, the above evaluation results cannot emphasize the practical value of OptimalPLR algorithm, especially given the theoretical analysis that OptimalPLR and ParaOptimal are equivalent. In the following, we will demonstrate the superiority of the OptimalPLR algorithm with severe situation where vast amount of local convex points exists, which can also explain the previous results that the time efficiency of OptimalPLR is better than ParaOptimal.

To simulate this situation, we generate the stream by function $y = x^{1.01}$ where x goes from 1 to 50,000. Such stream forms a concave shape with gradually increased values. We control the error bound to enable the stream approximated by single segment, in which case the size of convex hulls will always increase. By this example, we can magnify and emphasize the performance distinction of different algorithms given by the effect of convex hull size. Figure 26 demonstrates the time and space costs of SlideFilter, ParaOptimal, and OptimalPLR in the process of generating the line segment. It indicates that as the increase in convex hull size, the time and space consumptions of OptimalPLR can be 50 and 70 % of ParaOptimal. The OptimalPLR can be much faster than SlideFilter. In this situation, the total running times of OptimalPLR and SlideFilter are 33ms and 235 sec, respectively. The performance of SlideFilter results from the whole scanning on the convex hull for extreme line updating, and the redundant points on convex hulls, which will degrade the efficiency when there are abundant number of points on the convex hull. The different performance further denotes that OptimalPLR applies better strategy of convex hull maintenance.

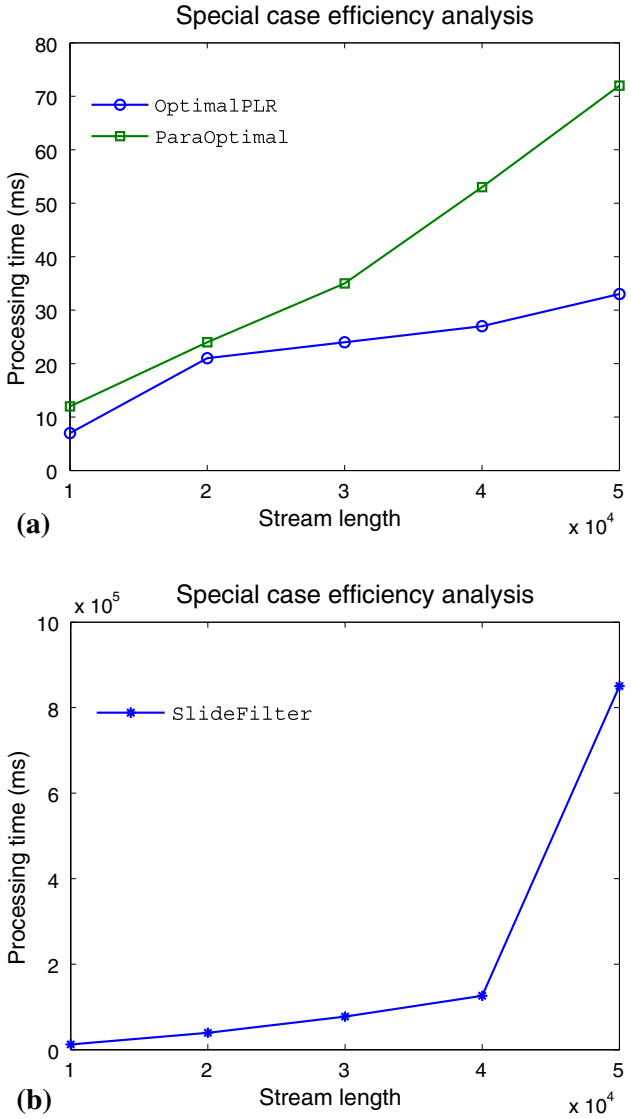

Fig. 26 Time costs for the severe situation. **a** Time cost of OptimalPLR and ParaOptimal. **b** Time cost of SlideFilter

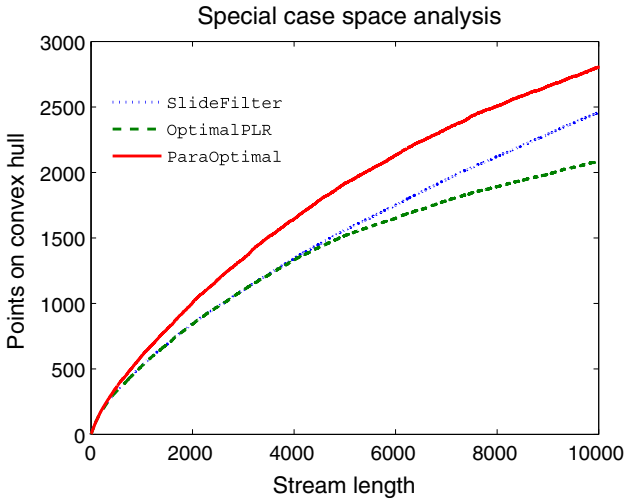

Fig. 27 Space costs for the severe situation

Comparison between OptimalPLR and ParaOptimal

Finally, we analyze and compare the two optimal algorithms, OptimalPLR and ParaOptimal. Along with the above experimental tests, we have performed many other tests that are especially designed for the comparison of ParaOptimal and OptimalPLR. Even through OptimalPLR and ParaOptimal are both linear-time algorithms, and theoretically equivalent, all the test results indicate that OptimalPLR and ParaOptimal may perform differently in practice, and OptimalPLR can generally take less time and space costs. With a detailed analysis on the tested results, we conclude that this fact is caused by the different intermediate data recorded during the process of generating PLR.

The major difference between OptimalPLR and ParaOptimal is that they maintain convex hulls in different spaces. ParaOptimal algorithm keeps the convex feasible region in parameter space, in which each convex point recorded is actually the intersection point of two linear functions, so there will be certain error accumulation and propagation for the value of the convex point (e.g., error caused by division calculation). When a new point is read in, the new linear constraint may intersect the feasible region at a convex point. However, due to the error propagation mentioned above, the intersection point may fail to be identified at the convex point. As a result, such "intersection" point will be added into convex hull and cause redundancy in space cost and slow down execution time.

On the contrary, since the convex points of OptimalPLR are the original stream points varied with error bound δ, the maintenance of convex hull and the calculation of the extreme slopes in OptimalPLR only involve original data value and would not cause significant redundancy as that of ParaOptimal. Figure 27 demonstrates that the points on the convex hull maintained by ParaOptimal are always much more than those by OptimalPLR (up to 33 % in the tested case), which results from the significant redundancy on the points of convex hull.

9 Conclusions

In this paper, continuing the work of [2,4,18], we investigated and made deep analysis into the significant piecewise linear representation (PLR) for online stream approximation under a given error bound in L_∞ norm. Based on the theoretical analysis, highly practical algorithms in time domain are proposed to achieve the optimal results with linear-time and space complexity. We also investigated the relationship between the time-value space and the parameter space, which provides complete and better theoretical understanding for this classic problem. From the practical aspect, in all experimental evaluations, our proposed optimal algorithm can generally achieve the same or higher time efficiency than the state-of-art solutions SlideFilter [4] and ParaOptimal [18]. The superiority of our solution in the time and space costs can be further magnified under severe situation, which indicates our proposed solution is more robust and practical for real applications. Thus, we conclude that the proposed algorithms can effectively solve the PLR problem from time domain with high efficiency and provide essential solutions for this classic problem.

Acknowledgments This research is partially supported by Natural Science Foundation of China (Grant No.61232006) and the Australian Research Council (Grant No. DP140103171 and DP130103051).

References

1. Berg, M.D., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry Algorithms and Applications. Springer, Berlin (2008)
2. Buragohain, C., Shrivastava, N., Suri, S.: Space efficient streaming algorithms for the maximum error histogram. In: Proceedings of the 23rd International Conference on Data, Engineering, pp. 1026–1035 (2007)
3. Chen, Q., Chen, L., Lian, X., Liu, Y., Yu, J.X.: Indexable pla for efficient similarity search. In: Proceedings of the 33rd International Conference on Very large Data Bases, pp. 435–446 (2007)
4. Elmeleegy, H., Elmagarmid, A.K., Cecchet, E., Aref, W.G., Zwaenepoel, W.: Online piece-wise linear approximation of numerical streams with precision guarantees. Proc. VLDB Endow. **2**, 145–156 (2009)
5. Gandhi, S., Foschini, L., Suri, S.: Space-efficient online approximation of time series data: Streams, amnesia, and out-of-order. In: Proceedings of IEEE 26th International Conference on Data, Engineering, pp. 924–935 (2010)
6. Gandhi, S., Nath, S., Suri, S., Liu, J.: Gamps: compressing multi sensor data by grouping and amplitude scaling. In: Proceedings of the ACM SIGMOD International Conference on Management of data, pp. 771–784 (2009)
7. Garofalakis, M., Kumar, A.: Deterministic wavelet thresholding for maximum-error metrics. In: Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 166–176 (2004)

8. Guha, S., Harb, B.: Approximation algorithms for wavelet transform coding of data streams. IEEE Trans. Inf. Theory **54**, 811–830 (2008)
9. Guha, S., Shim, K.: A note on linear time algorithms for maximum error histograms. IEEE Trans. Knowl. Data. Eng. **19**, 993–997 (2007)
10. Karras, P., Sacharidis, D., Mamoulis, N.: Exploiting duality in summarization with deterministic guarantees. In: Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 380–389 (2007)
11. Keogh, E., Chakrabarti, K., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 151–162 (2001)
12. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: Proceedings of the 1st IEEE International Conference on Data Mining, pp. 289–296 (2001)
13. Keogh, E., Zhu, Q., Hu, B., Hao, Y., Xi, X., Wei, L., Ratanamahatana, C.A.: The ucr time series classification/clustering homepage. www.cs.ucr.edu/eamonn/time_series_data/ (2011)
14. Lazaridis, I., Mehrota, S.: Capturing sensor-generated time series with quality guarantees. In: Proceedings of the 19th International Conference on Data, Engineering, pp. 429–440 (2003)
15. Li, G., Li, J., Gao, H.: ε-approximation to data streams in sensor networks. In: Proceedings of IEEE INFOCOM, pp. 1663–1671 (2013)
16. Matias, Y., Urieli, D.: Optimal workload-based weighted wavelet synopses. In: Database Theory—ICDT 2005, pp. 368–382. Springer, Berlin (2005)
17. Olston, C., Jiang, J., Widom, J.: Adaptive filters for continuous queries over distributed data streams. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 563–574 (2003)
18. O'Rourke, J.: An on-line algorithm for fitting straight lines between data ranges. Commun. ACM **24**(9), 574–578 (1981)
19. Paix, A.D., Williamson, J.A., Runciman, W.B.: Crisis management during anaesthesia: difficult intubation. Qual. Saf. Health Care (2005)
20. Palpanas, T., Vlachos, M., Keogh, E.: Online amnesic approximation of streaming time series. In: Proceedings of the 20th International Conference on Data, Engineering, pp. 339–349 (2004)
21. Pang, C., Zhang, Q., Hansen, D., Maeder, A.: Unrestricted wavelet synopses under maximum error bound. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pp. 732–743 (2009)
22. Pang, C., Zhang, Q., Zhou, X., Hansen, D., Wang, S., Maeder, A.: Computing unrestricted synopses under maximum error bound. Algorithmica **65**, 1–42 (2013)
23. Sathe, S., Papaioannou, T.G., Jeung, H., Aberer, K.: A survey of model-based sensor data acquisition and management. In: Managing and Mining Sensor Data, pp. 9–50 Springer (2013)
24. Shatkay, H., Zdonik, S.B.: Approximate queries and representations for large data sequences. In: Proceedings of the 12th International Conference on Data, Engineering, pp. 536–545 (1996)
25. Soroush, E., Wu, K., Pei, J.: Fast and quality-guaranteed data streaming in resource-constrained sensor networks. In: Proceedings of the 9th ACM International Symposium on Mobile ad Hoc Networking and Computing, pp. 391–400 (2008)
26. Wu, H., Salzberg, B., Zhang, D.: Online event-driven subsequence matching over financial data streams. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 23–34 (2004)
27. Xu, Z., Zhang, R., Kotagiri, R., Parampalli, U.: An adaptive algorithm for online time series segmentation with error bound guarantee. In: Proceedings of the 15th International Conference on Extending Database Technology, pp. 192–203 (2012)
28. Yu, L., Li, J., Gao, H., Fang, X.: Enabling ϵ-approximate querying in sensor networks. Proc. VLDB Endow. **2**(1), 169–180 (2009)
29. Zhang, Q., Pang, C., Hansen, D.: On multidimensional wavelet synopses for maximum error bounds. In: Proceedings of 14th International Conference on Database Systems for Advanced Applications, pp. 646–661 (2009)
30. Zhou, M., Wong, M.H.: A segment-wise time warping method for time scaling searching. Inf. Sci. **173**, 227–254 (2005)