Soft Computing (2020) 24:10535–10560
https://doi.org/10.1007/s00500-019-04562-7

METHODOLOGIES AND APPLICATION

A novel multi-resolution representation for time series sensor data analysis

Yupeng Hu[1] · Cun Ji[2] · Qingke Zhang[2] · Lin Chen[3] · Peng Zhan[3] · Xueqing Li[3]

Published online: 4 December 2019

Abstract
The evolution of IoT has increased the popularity of all types of sensing devices in a variety of industrial fields and has resulted in enormous growth in the volume of sensor data. Considering the high volume and dimensionality of sensor data, the ability to perform in-depth data analysis and data mining tasks directly on the raw time series sensor data is limited. To solve this problem, we propose a novel dimensional reduction and multi-resolution representation approach for time series sensor data. This approach utilizes an appropriate number of important data points (IDPs) within a certain time series sensor data to produce a corresponding multi-resolution piecewise linear representation (MPLR), called MPLR-IDP. The results of the theoretical analyses and experiments show that MPLR-IDP can reduce the dimensionality while maintaining the important characteristics of time series data. MPLR-IDP can represent the data in a more flexible way to meet diverse needs of different users.

Keywords Internet of things · Time series · Piecewise linear representation · Multi-resolution representation

Abbreviations

TS_n	A time series with length n
PLR	Piecewise linear representation
MPLR	Multi-resolution PLR
BMPLR	The basic multi-resolution PLR
EMPLR	The extended multi-resolution PLR
PIPs	Perceptually important points
TPs	Turning points
IDPs	Important data points
$TSRS$s	Time series representation standards
Num_{seg}	The user-specified number of segments
TFE	The user-specified fitting error of entire time series
MFE_{seg}	The user-specified maximum fitting error of segment
ARI	Adaptive representation index
SB-Tree	Specialized binary tree index
OBST	The optimal binary search tree
LI	Linear interpolation
LR	Linear regression
$seg<x, y>$	$Segment$ object from v_x to v_y
$es_{<x,y>}$	The fitting error of $seg<x, y>$
BMPLR	Basic multi-resolution PLR
EMPLR	Extended multi-resolution PLR
DS_m	The time series dataset with m time series
MN_{TP}	The maximum number of TPs
DCR	Data compression ratio
TSC	Time series classification
ST	Shapelet transformation

Communicated by V. Loia.

✉ Xueqing Li
xqli@sdu.edu.cn

Yupeng Hu
huyupeng@sdu.edu.cn

Cun Ji
jicun@sdnu.edu.cn

Qingke Zhang
tsingke@sdnu.edu.cn

Lin Chen
chenlin@sdu.edu.cn

Peng Zhan
zhanpeng@sdu.edu.cn

[1] School of Computer Science and Technology, Shandong University, Tsingtao 266000, China

[2] School of Information Science and Engineering, Shandong Normal University, Jinan 250358, China

[3] School of Software, Shandong University, Jinan 250101, China

TDS	Time series training dataset
$SubTS$	All the time series subsequences set
FSS	The final shapelets set

1 Introduction

With the evolution of Internet of Things (IoT), billions of connecting devices in widespread fields, including business (Lomet et al. 2008), finance (Wan and Si 2017), medicine (Doerr et al. 2016), etc., have already produced more than 1000 Exabytes sensor data annually, whose scale will increase to an unprecedented level Yin and Kaynak (2015). Considering that all the types of sensor data are collected based on their time-sequence characteristics, multiple sensor data analyses and mining research operations can be performed in the form of time series.

A time series with length n can be expressed as $TS_n = (p_1, p_2, \ldots, p_i, \ldots, p_n)$, where element $p_i = (v_i, t_i)$ indicates that the recorded value is v_i at time t_i, where the time is strictly increasing. For example, a part of the revolution sensor data of the momentum wheel (MW) in the DFH satellite provided by the China Academy of Space Technology (referred to as MW in what follows) collected in their original time-sequence has been represented as a time series and shown in Fig. 1a.

However, the time series data analysis perspective does not reduce the amount of sensor data that must be analyzed; the data are still large, and the capacity to perform in-depth data analysis and data mining tasks directly on the raw time series data is limited. Instead, a representation of the time series, which results in an approximate presentation of the raw data, should be created as the first step to reduce the costs of data storage, transmission, and processing.

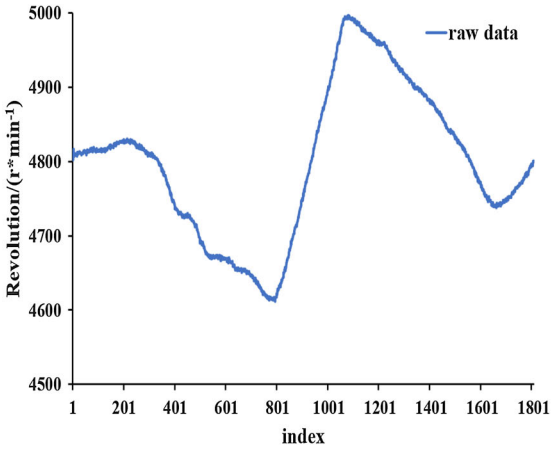
Scholars have expended considerable efforts to Scholars have expended considerable efforts to propose several effective time series representation algorithms, including discrete Fourier transform (DFT) (Agrawal et al. 1993), discrete wavelet transform (DWT) (Chan and Fu 1999), singular value decomposition (SVD) (Korn et al. 1997) and piecewise linear representation (PLR) (Keogh and Smyth 1997) also called piecewise linear approximation (PLA) (Shatkay and Zdonik 1996a). Of these, PLR has been one of the most widely used algorithms, which divides a time series into segments and then uses a linear function to approximate each segment. Compared with other methods, PLR has a lower index dimension, fosters higher calculation speeds and has the ability to support efficient similarity searches. In addition, PLR is more consistent with human visual experience.

According to the above analysis, the part of MW data in Fig. 1a could be represented by PLR. The corresponding result is shown in Fig. 1b. In this figure, the entire time series sequence is represented by four straight lines. The figure also shows the corresponding fitting error of the time series and the maximum fitting error of segments. Although a certain approximation error exists in the above PLR, the basic trends of the entire time series are fully expressed. Furthermore, considering only four segments (five original data points) were used to represent the entire sequence, which consists of 1801 data points, these results demonstrate the superiority of PLR in time series representation and dimensionality reduction.

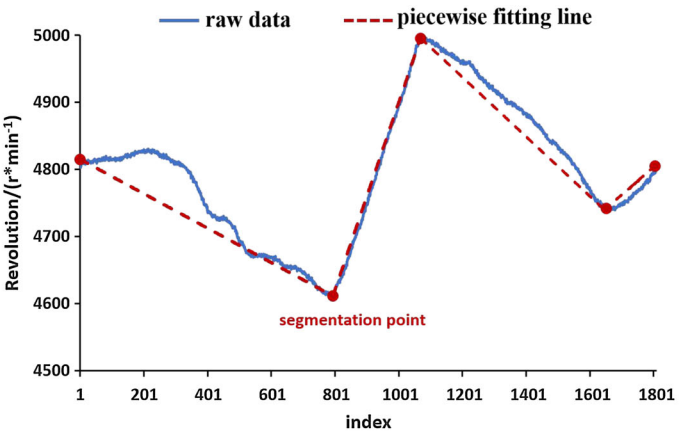
Therefore, when we wish to conduct some kinds of in-depth data mining based on time series sensor data, PLR can be utilized as a preprocessing tool for the subsequent correlative research. In the literature, traditional PLR methods, such as those proposed by Qu et al. (1998), Keogh and Pazzani (1998), Park et al. (1999), produce a series of appropriate linear segments to represent the original time series based on user-specified correlative requirements (the number of segments or the fitting error limits). Other PLR methods, such as those introduced by Perng et al. (2000), Pratt and Fink (2002), utilize a series of appropriate special points in the original time series (e.g., local extreme points, edge points, etc.) to segment the raw sequence into several subsequences and approximate each subsequence by linear interpolation. The overall representation effect can be controlled by similar requirements as mentioned above.

Although all the traditional PLR methods mentioned above support a piecewise linear representation of the time series, the corresponding result cannot be adjusted dynamically as users' needs change. It is quite common to perform a PLR operation repeatedly according to different requirements as a preprocessing tool for multiple follow-up studies. After completing the PLR on MW as shown in Fig. 1b, if users are to reconsider and perform PLR on the same time series with eight data points (i.e., to linearly approximate the entire sequence with seven segments) or apply a new PLR operation with the fitting error limitation, for example, the total fitting error of the time series is required to be no more than 29,000. The above two new PLR results are shown in Fig. 1c and d, respectively.

Accordingly, for a given time series, if users want to obtain a series of PLR results based on varying requirements, traditional PLR methods have to perform the similar operations repeatedly since they can only generate the one PLR result based on one requirement once. Such a processing strategy inevitably incurs unnecessary computing costs; in other words, few PLR methods dynamically return PLR results suitable for users' changing needs; and a method that does dynamically reflect the varying needs of users, referred to as a "multi-resolution representation," has become an important and popular subject in time series data analysis. Moreover, a comparison of the two PLR results in Fig. 1b and c shows that only three new points are added to refine the original linear fitting results. The fitting error of the entire time series

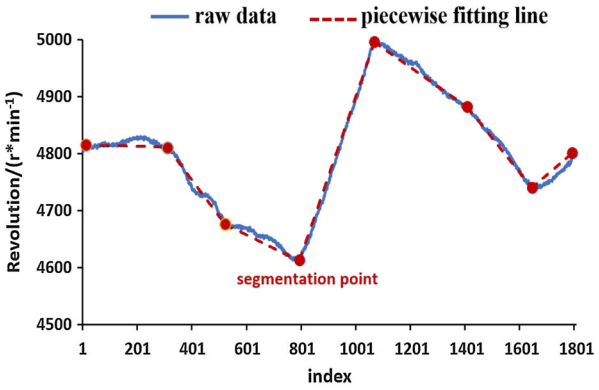


(a) Part of time series sonsor data of MW



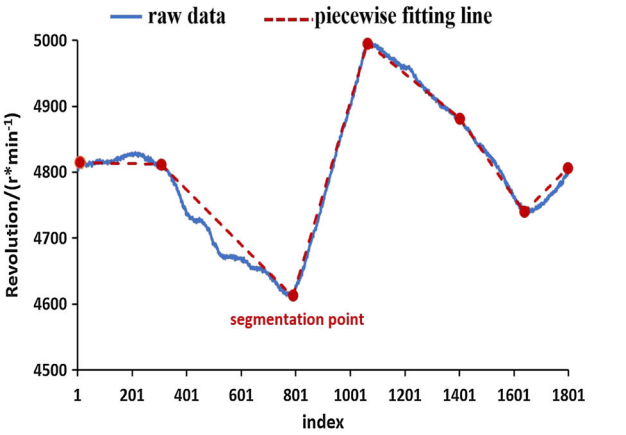
The corresponding PLR info of MW		
Number of segments	Total Fitting Error	Maximum Fitting error of segments
4	39389.26	22391.91

(b) PLR on MW with 4 segments



The corresponding PLR info of MW		
Number of segments	Total Fitting Error	Maximum Fitting error of segments
7	12354.99	2319.6

(c) PLR on MW with 7 segments



The corresponding PLR info of MW		
Number of segments	Total Fitting Error	Maximum Fitting error of segments
6	28435.39	12640.71

(d) PLR on MW with the fitting error limitation(29000)

Fig. 1 The overview of MPLR

decreases sharply, by 68.6%, while the corresponding maximum fitting error of segments declines even more, by 89.6%. These results also demonstrate the necessity and effectiveness of multi-resolution PLR (MPLR).

According to the above analysis, the ability to generate a series of MPLR results is determined primarily by the three time series representation standards ($TSRS$)s, which have also been used for time series piecewise linear segmentation by Keogh et al. (2001). According to these three $TSRS$s, the corresponding MPLR for TS_n are described as follows.

1. Given a user-specified number of segments k, expressed as $Num_{seg} = k$, the appropriate k linear segments would be produced to complete the PLR for TS_n.
2. Given a user-specified fitting error ε of the entire TS_n, expressed as $TFE = \varepsilon$, the appropriate PLR for TS_n

would be produced; this process would currently ensure that the total fitting error on the entire TS_n did not exceed ε.
3. Given a user-specified maximum fitting error η of the segments in TS_n, expressed as $MFE_{seg} = \eta$, the appropriate PLR for TS_n would be produced; this process would currently ensure that the fitting error for any segment in TS_n did not exceed η.

In view of the above situation, Fu et al. (2008) proposed a multi-resolution representation approach for financial time series based on perceptually important points (PIPs) selection and a specialized binary tree (SB-Tree) index, which is named MPLR-PIP in this paper. MPLR-PIP can present a financial time series at different levels of detail based on the importance of PIPs. Considering the drawback in PIP selection, Si and Yin (2013) proposed a novel multi-resolution PLR method for financial time series based on the turning points (TPs) in accordance with their contribution to preserve the trends and shape of financial time series, which is named MPLR-TP in this paper. Moreover, an optimal binary search tree (OBST) index was also devised for not only storing all these selected TPs, but also supporting reduced-sample time series reconstruction. Furthermore, both of these research results could also be utilized for technical pattern retrieving in reduced-sample financial time series.

To the best of our knowledge, the above two PLR methods, especially the TPs selection proposed by Si and Yin (2013), have made important contributions to the multi-resolution representation for financial time series in preserving the temporal features. The beginning or ending of a crucial transaction period could be clearly identified by TPs to guide users toward the best time for trading operations. In addition, stock movement trends can also be represented by TPs to form the corresponding reduced-sample models for achieving high performance on technical pattern matching (head and shoulder, double tops, etc.). However, the above two MPLR methods may not be directly applicable in our sensor data analysis scenario. Although the above two MPLR methods do provide a series of MPLR results which meet the $Num_{seg} = k$ requirements by selecting the corresponding $k + 1$ PIPs or TPs from their own index structures, respectively, in traditional time series sensor data mining research, a given time series would also be required to produce the corresponding MPLR results in accordance with the user-specified TFE or MFE_{seg}. Since MPLR-PIP records only the corresponding selection order and the fitting error of single point in each PIP selection and MPLR-TP stores only the selection order in each TP evaluation, in order to produce the ideal MPLR results which satisfy the corresponding TFE or MFE_{seg} requirements, the above two methods would need to add the selected data points (PIPs or TPs) individually and then calculate repeatedly until the corresponding $TSRS$s were met, which would undoubtedly increase the corresponding time overhead.

Motivated by the above analysis, we propose a novel multi-resolution piecewise linear representation based on important data points called MPLR-IDP for short. For a given time series, MPLR-IDP selects these data points as IDPs in accordance with their importance in global linear fitting. Meanwhile, all types of representative information along with each IDP selection would be stored into the adaptive representation index (ARI). After ARI construction is complete, a large number of MPLR based on different user-specified requirements (Num_{seg},TFE and MFE_{seg}) could be implemented efficiently. Our contributions in this paper could be summarized as follows.

1. We propose a novel multi-resolution piecewise linear representation based on important data points (MPLR-IDP), which provides a more accurate representation than the highly cited baseline methods (MPLR-PIP and MPLR-TP).
2. With the help of ARI, MPLR-IDP could complete the corresponding representation according to the diverse needs of users efficiently. Moreover, after a certain MPLR result has been achieved, all the relevant representation information (Num_{seg}, TFE, MFE_{seg}) would be obtained automatically without unnecessary repetitive calculations.
3. MPLR-IDP can also be utilized as a useful preprocessing tool for follow-up data mining tasks. A novel acceleration strategy for time series classification by MPLR-IDP will be illustrated in Sect. 6.

The remainder of this paper is organized as follows. Section 2 summarizes the existing related works. Section 3 describes some preliminaries. In Sect. 4, the multi-resolution piecewise linear representation algorithm based on important data points (MPLR-IDP) is described in detail. Section 5 presents the experimental results and corresponding analyses. A novel acceleration strategy for time series classification by MPLR-IDP is illustrated in Sect. 6. Finally, we conclude our paper and discuss future work in Sect. 7.

2 Related work

The piecewise linear representation (PLR), which is also called piecewise linear approximation (PLA), is a simple and intuitive feature representation method introduced by Keogh and Smyth (1997), Shatkay and Zdonik (1996a) and Keogh and Pazzani (1998).

Considering that PLR methods approximate a time series with several straight lines, there are two major approaches to

find these fitting lines, introduced by Keogh et al. (2001) and Fu (2011), respectively.

- *Linear interpolation (LI)* The approximating line for TS_n is the straight line connected from v_1 to v_n, which can be completed in a constant time.
- *Linear regression (LR)* The approximating straight line for TS_n is the best fitting line, which can be obtained by using the least squares strategy proposed by Shatkay and Zdonik (1996b). Depending on the length of the sequence, LR can be completed in linear time.

In this paper, we focus on describing PLR by important data points and on flexible representation by multi-resolution index structure; thus, either LI or LR can be utilized as the approximation technique in MPLR-IDP. To display the multi-resolution data representation more clearly, we selected LI in the subsequent MPLR-IDP algorithm description and experimental analysis. Before formally introducing MPLR-IDP, we briefly review two compelling MPLR methods.

1. MPLR-PIP proposed by Fu et al. (2008)

Given the time series TS with n data points, expressed as $TS_n = \{p_1, p_2, \ldots, p_n\}$. MPLR-PIP selects the first data point p_1 and the last data point p_n as two initial PIPs. Then MPLR-PIP would iteratively select points in TS that has the current maximum vertical distance (MVD) as PIPs. Meanwhile, the specialized binary tree (SB-Tree) stores these PIPs in accordance with their selected order. The PIP selection process continues until all the points in TS_n have been stored in the SB-Tree. With the help of SB-tree, TS_n could be represented at different levels of detail, while achieving the multi-resolution dimensionality reduction for technical pattern matching.

2. MPLR-TP proposed by Si and Yin (2013)

Given the time series TS_n with n data points. In the first place, all the TPs in TS_n would be selected to form the corresponding TP set denoted as set_{tp}, whose number may be smaller than n in some cases. Then MPLR-TP would sort all the TPs in set_{tp} from high to low based on their degree of importance. Although, MVD (as in MPLR-PIP) can be utilized as the importance evaluation criterion, six supplementary criteria can also be used for TP evaluation in MPLR-TP. After all the TPs have been sorted orderly, each TP in set_{tp} will be inserted into the most appropriate location of the optimal binary search tree (OBST), based on the continuously updated root table. With the help of OBST, TS_n can be represented in different MPLR forms, which is suitable for achieving the technical pattern matching in reduced-sample time series. More importantly, compared with the SB-tree, the more important TP in set_{tp} is much closer to the root of OBST, which can also retrieve the original time order for all the selected TPs through a depth-first traversal strategy.

By comprehensively analyzing the feature point assessment strategies in the above two methods, a novel feature point selection for time series sensor data is proposed in Sect. 3. Moreover, to ensure the efficiency of multi-resolution representation for sensing time series under different $TSRS$s (Num_{seg}, TFE, MFE_{seg}), a novel adaptive representation index (ARI) is illustrated in detail in Sect. 4.

3 Preliminaries

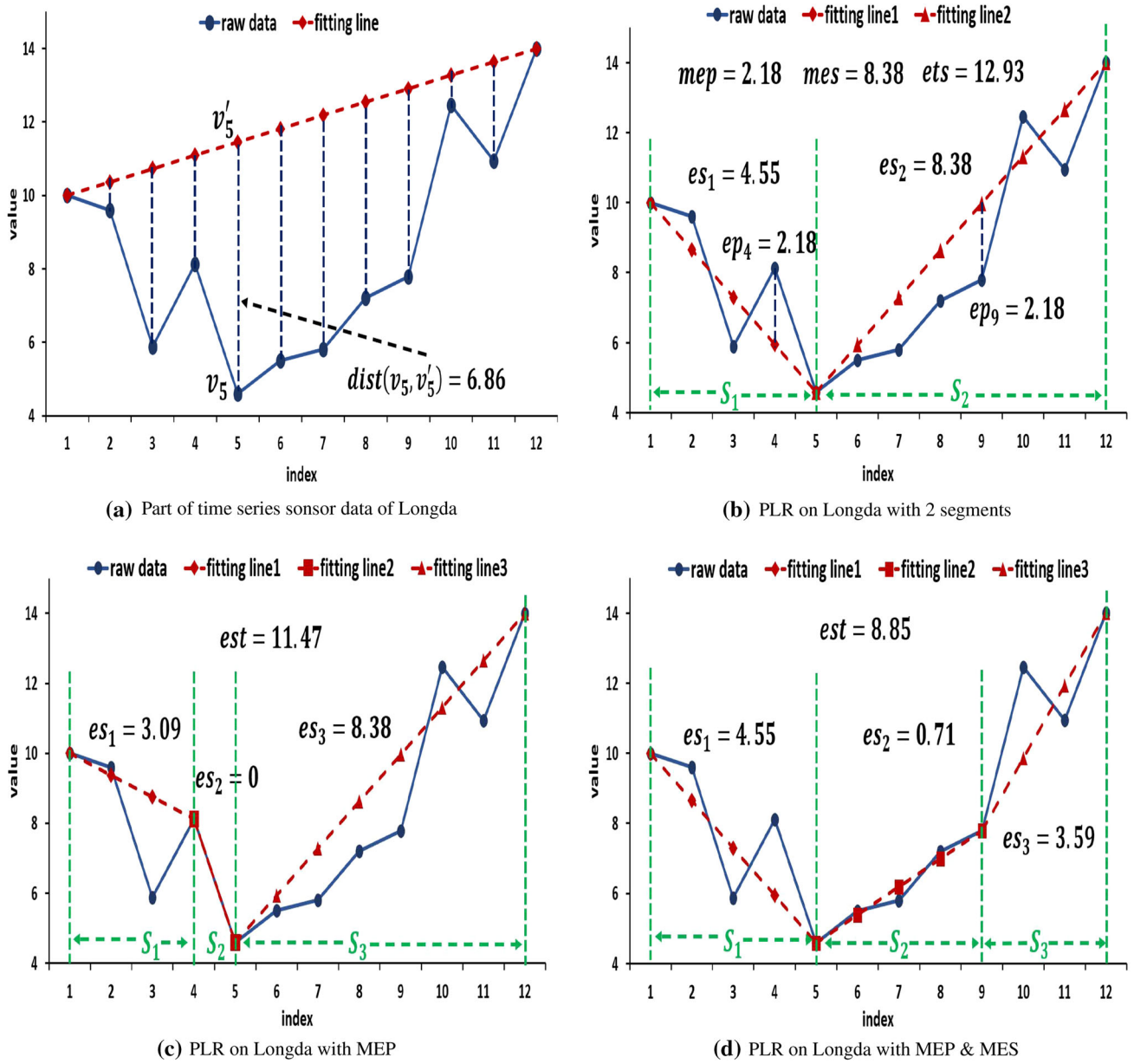
In this section, six basic definitions of PLR would be given in the first place, then the formal definition of IDP is proposed, and the related characteristics of the above definitions are analyzed subsequently.

Definition 1 (TS_n) For a time series TS with n data points expressed as $TS_n = \{p_1, p_2, \ldots, p_i, p_{i+1}, \ldots, p_n\}$, where $1 \leq i \leq n$, element $p_i = (v_i, t_i)$ indicates that the recorded value v_i arrives at the distinct timestamp t_i. Considering the time order in TS_n is obvious ($i < i + 1$), which can be simplified as $TS_n = \{v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n\}$.

According to the above definition, part of the time series from the cold storage facility in Longda Foodstuff Group Co., Ltd. (referred to as Longda in what follows), is shown in Fig. 2a. The time series can be expressed as $TS_{12} = \{v_1, v_2, \ldots, v_i, \ldots, v_{12}\}$. Based on the line interpolation (LI) introduction in Sect. 2, TS_{12} is represented by the straight line from v_1 to v_{12}, denoted as $TS'_{12} = \{v_1, v'_2, \ldots, v'_i, \ldots, v_{12}\}$ as shown in Fig. 2a. The fitting error of v_i in TS_{12} is the distance between v_i and its corresponding points v'_i in TS'_{12}, which are denoted as $dist(v_i, v'_i)$.

Three types of $dist(v_i, v'_i)$ can be used to measure the fitting error of single points: Euclidean distance (ED), perpendicular distance (PD) and vertical distance (VD) introduced in Fu et al. (2008). Although the above three methods have been widely used in various PLR algorithms, considering VD is equivalent to orthogonal distance in generating important points, as mentioned in Zhou and Li (2008) and the VD calculation is more efficient than the other two methods, in this paper, we adopt VD to calculate the fitting error of single point, abbreviated as EP. The definition of EP is described as follows.

Definition 2 (EP) For time series TS_n, whose fitting line can be expressed as $TS'_n = \{v_1, v'_2, \ldots, v'_i, \ldots, v_n\}$. The fitting error of single point v_i is denoted as ep_i and calculated by Eq. (1)



(a) Part of time series sonsor data of Longda

(b) PLR on Longda with 2 segments

(c) PLR on Longda with MEP

(d) PLR on Longda with MEP & MES

Fig. 2 IDP selection

$$ep_i = \sqrt{(v_i - v'_i)^2} \tag{1}$$

According to Definition 2, all ep_is in TS_{12} can be obtained by Eq. (1). If we want to continue to perform PLR on TS_{12} according to the MPLR-PIP introduced in Sect. 2, the point v_5 with the maximum EP ($ep_5 = 6.68$) in TS_{12} could be selected to subdivide TS_{12} into two consecutive subsegments $TS_{12} = \{S_1, S_2\}$, where $S_1 = \{v_1, v_2, \ldots, v_5\}$, $S_2 = \{v_5, v_6, \ldots, v_{12}\}$, then LI could also be used to produce the corresponding linear representation for each subsegment, respectively, in Fig. 2b. Based on the above data results, the definition of the maximum EP (MEP), the definition of the fitting error of segment (ES), the definition of the maximum fitting error of segment (MES) and the definition of the fitting error of TS (ETS) are given below.

Definition 3 (*MEP*) For segment S_j in time series TS_n expressed as $S_j = \{v_k, v_{k+1}, \ldots, v_m, \ldots, v_l\}$, where $k \leq m \leq l$. The maximum fitting error of single point in S_j is denoted as mep_j and obtained by Eq. (2).

$$mep_j = \max\{ep_k, ep_{k+1}, \ldots, ep_l\} \tag{2}$$

Definition 4 (*ES*) For segment S_j in time series TS_n expressed as $S_j = \{v_k, v_{k+1}, \ldots, v_m, \ldots, v_l\}$, where $k \leq m \leq l$.

The fitting error of S_j can be denoted as es_j and calculated by Eq. (3).

$$es_j = \sum_{j=k}^{l} ep_j \tag{3}$$

Definition 5 (*MES*) For time series TS_n, which has been subdivided into L segments, expressed as $TS_n = \{S_1, S_2, \ldots, S_i, \ldots, S_L\}$, where $1 \leq i \leq L$. The maximum fitting error of segment is denoted as mes and calculated by Eq. (4).

$$mes = \max\{es_1, es_2, \ldots, es_L\} \tag{4}$$

Definition 6 (*ETS*) For time series TS_n, which has been subdivided into L segments, expressed as $TS_n = \{S_1, S_2, \ldots, S_i, \ldots, S_L\}$, where $1 \leq i \leq L$. The fitting error of TS_n can be denoted as ets and calculated by Eq. (5).

$$ets = \sum_{i=1}^{m} es_i \tag{5}$$

According to the above analysis and definitions, if we select the corresponding points based on MEP only, we may encounter confusion in some cases. When we decide to further perform PLR operation as shown in Fig. 2b, we will find that v_4 in S_1 and v_9 in S_2 have the same EP (2.18); thus, it is unclear which one should be selected to continue PLR on TS_{12}.

Based on the above introduction on MPLR, the selection order of the specific point is reflected in the main trend reconstruction for the entire TS_n; in other words, a more important point can reduce the current ETS in PLR. Comparing the two different PLR results based on v_4 and v_9, shown in Fig. 2c and d, respectively, we can find that because the MES (8.38) segment S_2 in Fig. 2b is subdivided into two consecutive subsegments by MEP point v_9, the corresponding ETS in Fig. 2d will also be greatly reduced. Accordingly, MEP point v_9, which is in the MES segment, should be selected to subdivide S_2 into two subsegments for subsequent PLR.

Through the above analysis, suppose that S_i is the current MES segment in TS_n; then, the MEP point in S_i should be selected as the important data point (IDP) for subsequent PLR. IDPs, which could reduce the current ETS more effectively, should be selected orderly for the subsequent multi-resolution PLR. The definition of IDPs can be described as follows.

Definition 7 (*IDP*) For a time series $TS_n = \{v_1, v_2, \ldots, v_i, \ldots, v_n\}$, where $1 \leq i \leq n$. The point, which can be selected as an IDP, needs to satisfy one of the following conditions.

1. The first point v_1 and the last point v_n should be selected as two initial IDPs.
2. For TS_n, with m segments, $Seg_m = \{S_1, \ldots, S_j, \ldots, S_m\}$, where $1 \leq j \leq m$.
   $\exists$ point $v_i \in S_j$ satisfies Eq. (6) that should be selected as an IDP immediately.

$$\{(v_i, S_j) \mid S_j \in Seg_m, Mes = es_j, Mep_j = ep_i\} \tag{6}$$

Based on Definition 7, the most appropriate point in the current linear approximation would be selected as an IDP to aid in the ETS reduction in the next PLR operation. There is still one thing should be concerned if more than one point in different segments ($v_x \in S_k$,$v_y \in S_l$) meet Definition 7. (2) Concurrently, the earlier point should be selected as the IDP in such a situation, i.e., if the positional relationship of v_x and v_y in TS_n is expressed as $1 \leq x < y \leq n$, v_x should be selected as the current IDP.

According to the above definitions, the entire time series can be represented dynamically based on different $TSRS$s to meet the diverse needs of users. To achieve such a representation more flexibly and efficiently, the adaptive representation index (ARI) for MPLR is introduced in Sect. 4.

4 Algorithm

In this section, we will describe our multi-resolution piecewise linear representation based on important data points (MPLR-IDP) in details.

4.1 IDP selection and index construction

According to the above analysis, IDPs are selected and inserted into the index in each PLR iteration. The key to multi-resolution representation implementation lies in the construction of a reasonable index structure that not only preserves the priorities of these selected IDPs effectively, but also ensures efficient retrieval of the IDPs. Therefore, we propose the adaptive representation index (ARI) in this paper, which can be divided into the following two parts.

1. ARI-List

ARI-List is a priority list, which records all the IDPs based on their selection order and incrementally updates the current ets and mes when a new IDP is inserted into the list. With the help of ARI-List, the basic MPLR based on one of the $TSRS$s can be implemented efficiently.

Table 1 Attributes of *ListNode*

Attributes	Description
$index$	The original position of the point
$value$	The value of the point
$rank$	The order in which the point is selected
es_{L}	The fitting error of the left segment when the point has been selected
es_{R}	The fitting error of the right segment when the point has been selected
mes	The current MES when the point has been selected
ets	The current ETS when the point has been selected

Table 2 Attributes of *TreeNode*

Attributes	Description
$index$	The original position of the point
$value$	The value of the point
$rank$	The order in which the point is selected
es_{L}	The fitting error of the left segment when the point has been selected
es_{R}	The fitting error of the right segment when the point has been selected
$child_{\mathrm{L}}$	The left child of this $TreeNode$ in the index tree
$child_{\mathrm{R}}$	The right child of this $TreeNode$ in the index tree

2. ARI-Tree

ARI-Tree, which is similar to a binary tree, is constructed to store IDPs and their left and right subsegments fitting errors for local feature retrieval. With the help of ARI-Tree, the extended MPLR based on the combination of multiple $TSRS$s can be implemented efficiently.

Accordingly, some objects will be defined in advance, and then a heuristic instance of ARI construction will be described later.

Objects of ListNode In our algorithm, all IDPs should be defined as $ListNode$ objects before inserted into ARI-List. A $ListNode$ object is expressed as $ListNode = \{index; value; rank; es_{\mathrm{L}}; es_{\mathrm{R}}; mes; ets\}$. The corresponding attributes of $ListNode$ are described in Table 1.

Objects of TreeNode Similarly, all IDPs should be defined as $TreeNode$ objects before inserted into ARI-Tree. A $TreeNode$ object is expressed as $TreeNode = \{index; value; rank; es_{\mathrm{L}}; es_{\mathrm{R}}; child_{\mathrm{L}}; child_{\mathrm{R}}\}$. The corresponding attributes of $TreeNode$ are described in Table 2.

Objects of Segment *Segment* records the details of segment S_i in TS_n and is expressed as $Segment = \{p_{\mathrm{b}}, p_{\mathrm{e}}, p_{\max}, value, sumdist, dist_{\max}\}$. The corresponding attributes of *Segment* are described in Table 3.

Table 3 Attributes of *Segment*

Attributes	Description
p_{b}	The beginning point of the segment
p_{e}	The ending point of the segment
$p_{\max}$	The index of the MEP point
$value$	The value of the MEP point
$sumdist$	The fitting error of the segment
$dist_{\max}$	The MEP of the segment

Based on the above introduction, note that the first and last point (v_1 and v_n) of TS_n are not defined as $TreeNode$ objects and inserted into ARI-Tree, for the following two reasons.

For one thing, since ARI-tree is built for retrieving local features, the ARI-Tree construction begins at the 3rd selected IDP, which could not only reduce the total storage overhead, but also lower the height of ARI-Tree to improve the corresponding searching efficiency. For another, although the above two IDPs are not stored in the ARI-Tree, they are created as the first two $ListNode$s and inserted into ARI-List; thus, their relevant information can be efficiently retrieved from ARI-List to ensure the integrity of the corresponding PLR result.

To illustrate the construction process more clearly, we use a part of time series in Longda (with 53 data points) as an example. The step-by-step processes of IDP selection and ARI construction on Longda are shown in Fig. 3, which could be divided into the following five steps.

Step 1 According to Definition 7 of IDP, the first point v_1 and the last point v_{53} in Longda are initially selected as the first two IDPs (IDP1 and IDP2). Then a new *Segment* object $seg{<}1, 53{>}$ (from v_1 and v_{53}) is created, denoted as dotted line in Fig. 3a, and all the attributes of these two IDPs can be completely calculated based on Eqs. (2), (3), (4) and (5) in Sect. 3. In Fig. 3a, the fitting error of $seg{<}1, 53{>}$, denoted as $es_{<1,53>}$ for brevity, is 859.9, and the $dist_{\max}$ and $p_{\max}$ of $seg{<}1, 53{>}$ are 57.2 and v_{31}, respectively. Since there is only one *Segment* object in current PLR result, both mes and ets are 859.9. Subsequently, the current $seg{<}1, 53{>}$ can be expressed as $seg{<}1, 53{>} = \{1, 53, 31, 399.5, 895.9, 57.2\}$; these are the attributes of p_{b}, p_{e}, $p_{\max}$, $value$, $sumdist$ and $dist_{\max}$, respectively. Then, $seg{<}1, 53{>}$ can be inserted into the priority queue $segList$, which is created for storing *Segment* objects based on the descending order of $Segment.sumdist$. Furthermore, according to the $ListNode$ description in Table 1, two new $ListNode$ objects are created based on IDP1 and IDP2, named LN_1 and LN_2, respectively, whose $rank$s are consistent with the selection order of IDPs as shown in Fig. 3a. Taking LN_2 as an example, LN_2 can be expressed

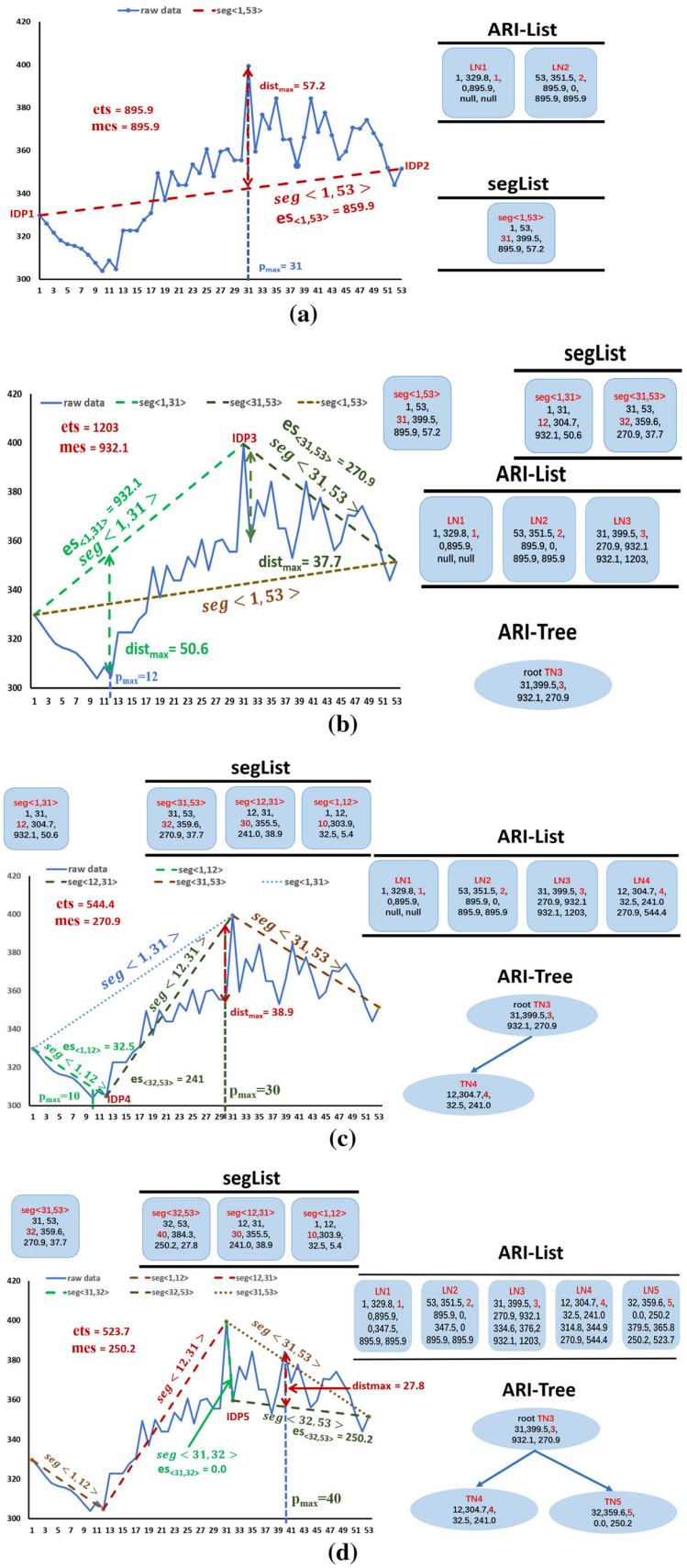


Fig. 3 The major steps in ARI construction

as $LN_2 = \{53, 351.5, 2, 895.9, 0, 895.9, 895.9\}$, which are the attributes of $index$, $value$, $rank$, es_{L}, es_{R}, mes, ets of LN_2, respectively.

Step 2 $seg{<}1, 53{>}$ is removed from $segList$ and $p_{\max}$ point (v_{31}) in $seg{<}1, 53{>}$ should be selected as the 3rd IDP named IDP3. Subsequently, $seg{<}1, 53{>}$ is changed into two *Segment* objects based on IDP3, which are $seg{<}1, 31{>}$ and $seg{<}31, 53{>}$. Similarly, not only $p_{\max}$ and $dist_{\max}$ of these two new *Segment*s but also $es_{<1,31>}$ and $es_{<31,53>}$ can be completely obtained from the above mentioned equations. As shown in Fig. 3b, $es_{<1,31>}$ (932.1) is much larger than $es_{<31,53>}$ (207.9); consequently, the current mes is 932.1 and ets is 1203 (932.1 + 207.9). Subsequently, $seg{<}1, 31{>}$ and $seg{<}31, 53{>}$ are added into $segList$ in accordance with the descending order of their own $sumdist$ attributes. Thereafter, according to the previous definitions, two new objects of $ListNode$ and $TreeNode$ are created based on IDP3, named LN_3 and TN_3, whose $rank$s are consistent with the selection order of IDP3. Similar to LN_2, LN_3 is created and inserted at the end of ARI-List (according to the ascending order of $rank$). More importantly, the $root$ of ARI-Tree can be created based on $TN3$, in other words, ARI-Tree is established staring at IDP3, which reduces the total storage overhead and improves the corresponding retrieval efficiency. As shown in Fig. 3b, taking $TN3$ as an example, $TN3$ could be expressed as $root = \{31, 399.5, 3, 932.1, 270.9, null, null\}$, which denotes $index$, $value$, $rank$, es_{L}, es_{R}, $child_{\mathrm{L}}$, $child_{\mathrm{R}}$ of $TN3$, respectively.

Step 3 The *Segment* object ($seg{<}1, 31{>}$) currently first in $segList$ should be popped out; then, the $p_{\max}$ point (v_{12}) in $seg{<}1, 31{>}$ is selected as the 4th IDP, named IDP4. Subsequently, $seg{<}1, 31{>}$ is transformed into two *Segment* objects: $seg{<}1, 12{>}$ and $seg{<}12, 31{>}$. Subsequently, $es_{<1,12>}$ (32.5) and $es_{<12,31>}$ (241.0) can be completely obtained. Furthermore, the global fitting error (ets) in the current PLR result can be calculated efficiently by incremental updating. Considering that the current change of the result is the transformation from the original $seg{<}1, 31{>}$ to the two new *Segment*s ($seg{<}1, 12{>}$ and $seg{<}12, 31{>}$) in Fig. 3c. Without loss of generality, the ets variation between LN_i and LN_{i-1} is expressed as $evar_i$, the fitting error of the original *Segment* in LN_{i-1} is denoted as es_{i-1}, the fitting errors of the two new *Segment*s are denoted as es_{iL} and es_{iR}, respectively, where the current $evar_i$ and ets_i can be calculated by Eqs. (7) and (8), respectively.

$$evar_i = es_{i-1} - (es_{iL} + es_{iR}) \tag{7}$$

$$ets_i = ets_{i-1} - evar_i \tag{8}$$

With the help of the above two equations, in Fig. 3c, $evar_4$ can be calculated as $evar_4 = es_3 - (es_{4L} + es_{4R}) = 932.1 - (32.5 + 241) = 658.6$ and the current ets_4 is obtained as $ets_4 = ets_3 - evar_4 = 1203 - 658.6 = 544.4$. Based on the above results, it is obvious that the current ets has been sharply reduced (more than 50%) by the addition of IDP4 (v_{12}) for PLR. Besides, the current mes can be found efficiently by choosing the relatively larger fitting error in $seg{<}12, 31{>}$ ($es_{<12,31>} = 240.0$) and $seg{<}31, 53{>}$ ($es_{<31,53>} = 270.9$), with the largest fitting error in the current $segList$.

Thereafter, two new objects of $ListNode$ and $TreeNode$ are created based on IDP4, named LN_4 and TN_4, respectively. Similar to LN_3, LN_4 can be inserted into ARI-List consistent with its ascending $rank$ order. Besides, TN_4 is inserted into ARI-Tree based on its $TreeNode.index$. If the $index$ of the new $TreeNode$ object is smaller than the $index$ of the $root$, the new object is inserted into the left child of the $root$; otherwise, it is inserted into the right child of the $root$. However, if the corresponding $child_{\mathrm{L}}$ or $child_{\mathrm{R}}$ of the $root$ is not $null$, the process continues to look downwards for the appropriate position into which to insert the new $TreeNode$ object. As shown in Fig. 3c, $TN_4 = \{12, 304.8, 4, 32.5, 240.0, null, null\}$, which are the attributes of $index$, $value$, $rank$, es_{L}, es_{R}, $child_{\mathrm{L}}$, $child_{\mathrm{R}}$, respectively, should be inserted into ARI-Tree as the left child of the $root$, (i.e., $root.child_{\mathrm{L}} = TN_4$). Finally, $seg{<}1, 12{>}$ and $seg{<}12, 31{>}$ are pushed into $segList$ in accordance with the descending order of their own $sumdist$ attributes in Fig. 3c.

Step 4 The current first *Segment* object ($seg{<}31, 53{>}$) in $segList$ should be popped out and the $p_{\max}$ point (v_{32}) in $seg{<}31, 53{>}$ is selected as the 5th IDP named IDP5. Subsequently, $seg{<}31, 53{>}$ is transformed into two *Segment* objects: $seg{<}31, 32{>}$ and $seg{<}32, 53{>}$. Similar to Step 3, $es_{<31,32>}$ (0.0) and $es_{<32,53>}$ (250.2) can be completely obtained, and the current ets (523.7) can also be calculated efficiently based on Eqs. (7) and (8). Moreover, as shown in Fig. 3d, the current mes is obtained by choosing the relatively larger fitting error in $seg{<}32, 53{>}$ ($es_{<32,53>} = 250.2$) and $seg{<}12, 31{>}$ ($es_{<12,31>} = 241.0$), with the largest fitting error in the current $segList$.

Thereafter, two new objects of $ListNode$ and $TreeNode$ are created based on IDP5, named LN_5 and TN_5, respectively. Similar to LN_4 and TN_4, LN_5 is inserted into the end of ARI-List consistent with the ascending $rank$ order, and TN_5 is inserted into ARI-Tree based on its own $index$. In Fig. 3d, TN_5 is inserted into ARI-Tree as the right child of the $root$ (i.e., $root.child_{\mathrm{R}} = TN_5$). Finally, $seg{<}32, 53{>}$ is pushed into $segList$; however, $seg{<}31, 32{>}$ does not perform the same operation because $es_{<31,32>}$ is 0.0, which means that $seg{<}31, 32{>}$ cannot be refined again.

Algorithm 1 The Construction of ARI

Input: time series: $TS_n = \{v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n\}$

Output: ARI $index$, which contains ARI-List and the $root$ of ARI-Tree

1: $ListNode$ [] ARI-List = new $ListNode$[n]; // Create ARI-List as a global array
2: $segList$ = new $Segment$[n]; // Create $segList$ as a global priority queue
3: initARI(TS_n); // Invoke initARI function in Algorithm 2 for initialization
4: $i = 4$; // Follow up from the searching for the 4th IDP
5: **while** $segList$ is not empty **do**
6: seg= $segList$.pop();
7: $IDP = seg.p_{\max}$;
8: $segL$ = createSeg(TS_n, $seg.p_b$, IDP); // Create new $Segment$ object $segL$
9: $segR$ = createSeg(TS_n, IDP, $seg.p_e$);
10: $value = TS[seg.p_{\max}]$;
11: $es_L = segL.sumdist$;
12: $es_R = segR.sumdist$;
13: mes = getMAX($segList$, es_L, es_R); // Invoke getMAX function in Algorithm 3 for getting current mes
14: ets = getETS(seg, es_L, es_R); // Invoke getETS function in Algorithm 4 for getting current ets
15: LN_i = new ListNode($seg.p_{\max}$, $value$, i, es_L, es_L, mes, ets); // Create new $ListNode$ LN_i
16: ARI-List.add(LN_i); // Insert LN_i into the end of ARI-List
17: TN_i = new TreeNode($seg.p_{\max}$, $value$, i, es_L, es_L, null, null); // Create new $TreeNode$ TN_i
18: insertTreeNode($root$,TN_i); // Invoke insertTreeNode function in Algorithm 5 to insert TN_i into ARI-Tree
19: **if** $seg_L.sumdist \neq 0$ **then**
20: $segList$.push(seg_L)
21: **end if**
22: **if** $seg_R.sumdist \neq 0$ **then**
23: $segList$.push(seg_R)
24: **end if**
25: i++;
26: **end while**
27: **return** $index$

Similar steps are repeated until $segList$ is empty, which means that all the points in Longda have been processed completely and ARI is also constructed simultaneously. As in the above step-by-step description, the overview of ARI construction is shown in Fig. 4, and the main algorithm for ARI construction is shown in Algorithm 1.

Algorithm 2 The initialization of ARI

Input: time series: $TS_n = \{v_1, v_2, \ldots, v_i, v_{i+1}, \ldots, v_n\}$

Output: $segList$

1: seg = createSeg(TS_n, 1, n); // Create new $Segment$ object seg
2: LN_1 = new ListNode(1, v_1, 1, 0, $seg.sumdist$, null, null); // Create $ListNode$ LN_1
3: LN_2 = new ListNode(n, v_n, 2, $seg.sumdist$, 0, $seg.sumdist$, $seg.sumdist$);
4: ARI-List.add(LN_1); // Insert LN_1 into the end of ARI-List
5: ARI-List.add(LN_2);
6: $segL$ = createSeg(TS_n, 1, $seg.p_{\max}$);
7: $segR$ = createSeg(TS_n, $seg.p_{\max}$, n);
8: $value = TS[seg.p_{\max}]$; // Get the $value$ of the $p_{\max}$ point
9: $es_L = seg1.sumdist$;
10: $es_R = seg2.sumdist$;
11: mes = Max($seg1.sumdist$,$seg2.sumdist$); // Get the current mes
12: $ets = seg1.sumdist + seg2.sumdist$; // Get the current ets
13: LN_3 = new ListNode($seg.p_{\max}$, $value$, 3, es_L, es_L, mes, ets);
14: ARI-List.add(LN_3);
15: $root = TN_3$ = new TreeNode($seg.p_{\max}$, $value$, 3, es_L, es_L, null, null); // Create $TreeNode$ TN_3
16: **if** $seg_L.sumdist \neq 0$ **then**
17: $segList$.push(seg_L)
18: **end if**
19: **if** $seg_R.sumdist \neq 0$ **then**
20: $segList$.push(seg_R)
21: **end if**
22: **return** $segList$

The basic idea underlying Algorithm 1 is completely consistent with the previous description of ARI construction shown in Fig. 3. The initialization function (initARI) is invoked to create the first three $ListNode$ objects for ARI-List and one $TreeNode$ object for ARI-Tree in ③ of

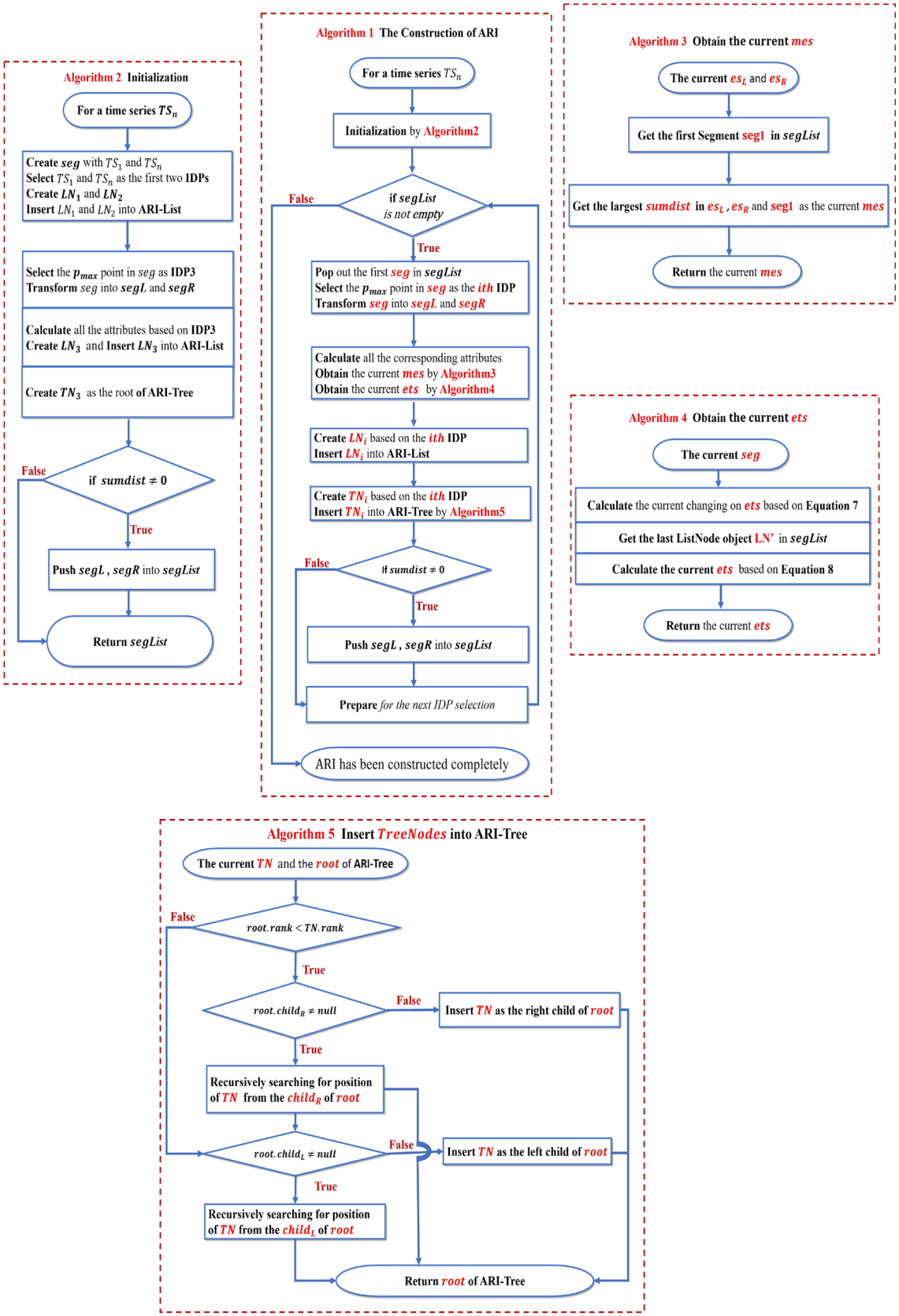


Fig. 4 Overview of ARI construction

Algorithm 3 Obtain mes

Input: PrioList $segList$, es_{L}, es_{R}

Output: the current mes

1: $seg1$ = $segList$.get(0); // Get the first $Segment$ object $seg1$ in $segList$, whose mes is the largest in current $segList$
2: mes = Max($seg1.sumdist$, es_{L}, es_{R});
3: **return** mes

Algorithm 4 Obtain ets

Input: seg, es_{L}, es_{R}

Output: the current ets

1: $evar = seg.sumdist$- ($es_{\mathrm{L}} + es_{\mathrm{R}}$); // Calculate the current changing on ets according to Eq. (7)
2: end = ARI-List.length-1;
3: LN' = ARI-List.get(end); // Get the last $ListNode$ object LN' in ARI-LIst
4: ets = $LN'.ets$ - $evar$; // Calculate the current ets according to Eq. (8)
5: **return** ets

Algorithm 5 Insert $TreeNode$ into ARI-Tree

Input: TN, $root$ of ARI-Tree

Output: $root$

1: **if** $root.rank < TN.rank$ **then**
2: **if** $root.child_{\mathrm{R}} \neq$ null **then**
3: insertTreeNode($root.child_{\mathrm{R}}$,TN); // Recursively searching for the position of TN from the right child ($child_{\mathrm{R}}$) of $root$
4: **else**
5: $root.child_{\mathrm{R}} = TN$; // insert TN as the right child of $root$
6: **end if**
7: **else**
8: **if** $root.child_{\mathrm{L}} \neq$ null **then**
9: insertTreeNode($root.child_{\mathrm{L}}$,TN); // Recursively searching for the position of TN from the left child ($child_{\mathrm{L}}$) of $root$
10: **else**
11: $root.child_{\mathrm{L}} = TN$;
12: **end if**
13: **end if**
14: **return** $root$;

Algorithm 1. The specific initARI function, shown in Algorithm 2, performs the following three main operations.

1. Create the first two $ListNode$ objects (LN_1, LN_2) and insert them into ARI-List, corresponding to the previous description in Fig. 3a.
2. Select the $p_{\max}$ point of seg as the 3rd IDP (IDP3) and transform seg into two $Segment$ objects (seg_{L}, seg_{R}). Subsequently, the relatively larger $sumdist$ from $segL$ and $segR$ is assigned to mes. The current ets is the sum of the two $sumdist$s from $segL$ and $segR$. According to the above information, the 3rd $ListNode$ (LN_3) is created by the $ListNode$ constructor in ⑬ of Algorithm 2. The $root$ of ARI-Tree is also created, expressed as TN_3, in ⑮ of Algorithm 2. The specific process of LN_3 and TN_3 creation corresponds to the previous description in Fig. 3b.
3. From ⑯ to ㉑ of Algorithm 2, the $sumdist$s of the above two $Segment$ objects (seg_{L}, seg_{R}) would be judged. When the $sumdist$ of the seg_{L} (seg_{R}) is 0, the corresponding seg_{L} (seg_{R}) object will not be processed by any subsequent operations; otherwise, it will be pushed into $segList$ according to the descending order of $sumdist$ for the 4th IDP selection.

After the initialization process, Algorithm 1 continues to build ARI from the 4th IDP selection. Therefore, the temporary variable i will be assigned a value of 4 in ④ of Algorithm 1. The specific operations from ⑤ to ㉖ in Algorithm 1 are in the while loop, which repeats the following five steps until $segList$ is empty.

1. The first $Segment$ object seg in $segList$ should be popped out. The $p_{\max}$ point of seg is selected as the current IDP, and the original seg (from p_{b} to p_{e}) is changed into two $Segment$ objects: seg_{L} (from p_{b} to $p_{\max}$), seg_{R} (from $p_{\max}$ to p_{e}). Subsequently, seg_{L} and seg_{R} are created completely from ⑧ to ⑨ of Algorithm 1.
2. To create a new $ListNode$ object, named LN_i for short, in the ith loop, the current mes is obtained by the getMAX function in ⑬ of Algorithm 1, which is defined in Algorithm 3. Moreover, the current ets is efficiently obtained by calling the getETS function in ⑭ of Algorithm 1 efficiently, which is defined in Algorithm 4.
3. After the above calculation processes have completed, the new LN_i in the ith loop is created by $ListNode$ constructor in ⑮ of Algorithm 1 and added directly to the end of ARI-List. In general, the above process of creating LN_i and adding it into ARI-List in the ith loop

corresponds to the previous detailed descriptions of LN_4 and LN_5 in Fig. 3c and d, respectively.

4. Similarly, the new TN_i in the ith loop is also created by the $TreeNode$ constructor in ⑰ of Algorithm 1. However, the insertion process of TN_i is different from that of LN_i. The insertTreeNode function, invoked in ⑱ of Algorithm 1, is a recursive function defined in Algorithm 5. The specific creation and subsequent insertion of TN_i in ith loop correspond to the previous description of TN_4 and TN_5 in Fig. 3c and d, respectively.
5. Finally, if the $sumdist$s of seg_{L} and seg_{R} are not 0, they will be pushed into $segList$ to prepare for the $(i+1)$th IDP selection.

When Algorithm 1 based on TS_n is completed, the ARI is also established concurrently, which can be utilized for dynamic multi-resolution PLR under different $TDRS$s.

4.2 The multi-resolution representation

With the help of ARI, MPLR-IDP provides a more flexible piecewise linear representation (PLR) based on different $TSRS$s to meet the diverse needs of users. MPLR-IDP can also be used as a preprocessing tool for subsequent time series research. Accordingly, the multi-resolution representation strategy of MPLR-IDP can be divided into two main categories: the basic multi-resolution PLR (BMPLR) and the extended multi-resolution PLR (EMPLR).

According to the above introduction in Sect. 1, there are three time series representation standards ($TSRS$)s (Num_{seg}, TFE and MFE_{seg}) for multi-resolution PLR. For a certain TS_n, BMPLR only provides adaptive MPLR in accordance with anyone of the $TSRS$s efficiently, but also obtains the relevant information for the other two $TSRS$s automatically without unnecessary repeated calculations. To the best of our knowledge, BMPLR is able to meet the diverse requirements of MPLR. However, in an actual application scenario, when a certain PLR result has been obtained by BMPLR, users may tend to make corresponding local adjustments to it, based on their own needs. Consequently, we propose a more flexible multi-resolution representation strategy, i.e., the extended multi-resolution PLR (EMPLR). EMPLR not only supports local representation adjustments based on the completed PLR result produced by BMPLR, but also permits corresponding local adjustments in the form of multi-parameter combinations; in other words, the relevant users can simultaneously utilize two time series representation standards in $TSRS$s together to achieve the corresponding local adjustments.

To make MPLR more clearly, the ARI mentioned in Sect. 4.1 can be taken as an example for BMPLR and EMPLR. Due to space limitations, the portion of the ARI, containing the first ten $ListNode$s and the first eight $TreeNode$s, is shown in Fig. 5.

1. The basic multi-resolution PLR

With the help of ARI, BMPLR can provide a series of PLR results on Longda under one of $TSRS$s (Num_{seg}, TFE, MFE_{seg}). Without loss of generality, we choose $Num_{seg} = 4$ to illustrate the main processes of BMPLR, which involves the following three steps.

- Select $ListNode$s from ARI-List to form a $ListNode$ set (LNS). The PLR result with $Num_{seg} = 4$ could be converted to select the first five $ListNode$s in ARI-List to form LNS, as shown in Fig. 6a. After getting the LNS, the current TFE and MFE_{seg} of Longda are obtained from $LN_5.ets$ (523.7) and $LN_5.mes$ (250.2) efficiently.
- Resort the above LNS. Since these selected $ListNode$s in LNS are sorted by the ascending order of $rank$, in order to get the corresponding PLR result, we need to resort these $ListNode$s according to their own $index$ attributes (their original order in Longda). As shown in Fig. 6b, the five $ListNode$s have been resorted in ascending $index$ order.
- Obtain the specific PLR result. After the sorting operation is completed, each two adjacent $ListNode$s in LNS could form a corresponding $Segment$ object. For instance, LN_4 and LN_3 in Fig. 6b produce $seg{<}12, 31{>}$. Eventually, the above five $ListNode$s produce four different $Segment$ objects to complete the final PLR result according to $Num_{seg} = 4$.

Furthermore, there is still one thing should be mentioned that the corresponding fitting error of each segment (es) in the current PLR result can also be obtained efficiently from the es_{L} or es_{R} of the two adjacent $ListNode$s. For example, the es_{R} (241.0) of LN_4, whose $rank(4)$ is larger than $LN_3.rank$ (3), should be selected as the current fitting error of $seg{<}12, 31{>}$ in Fig. 6c. In the same way, the fitting error of other $Segment$s in the current PLR could be obtained comprehensively.

According to the PLR result in Fig. 6, no matter which of $TSRS$s is used for BMPLR, the corresponding Num_{seg}, TFE and MFE_{seg} in the current PLR result, can be obtained completely. In other words, MPLR-IDP reflects the relevant characteristics of the current PLR from different perspectives.

2. The extended multi-resolution PLR

Based on the above introduction, with the help of ARI, EMPLR allows users to further perform local adjustments

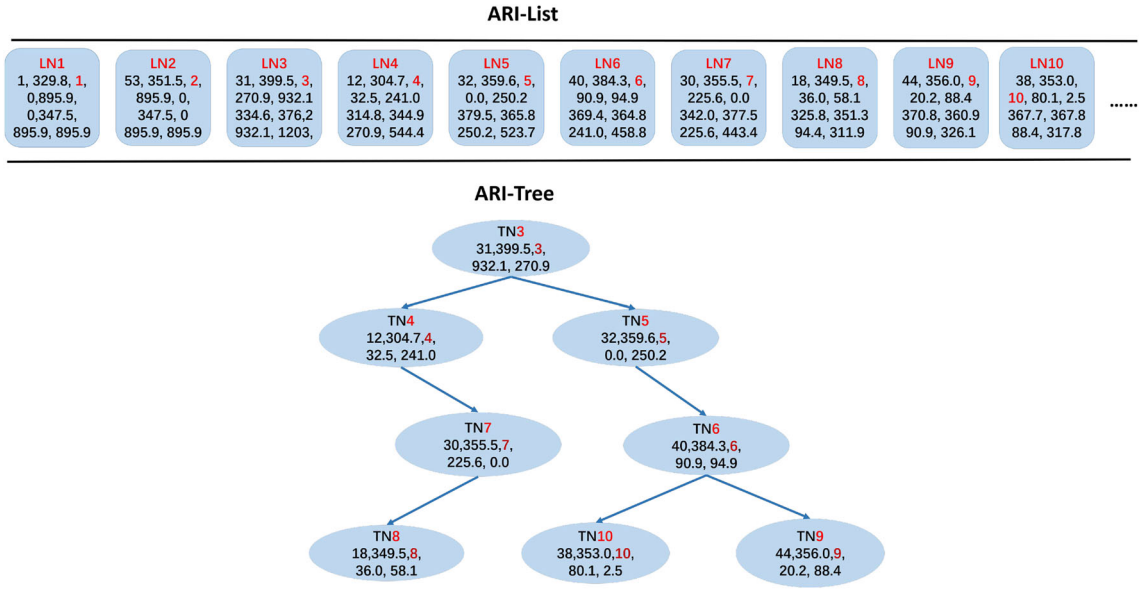


Fig. 5 Part of ARI on Longda

on the specific PLR result. Moreover, EMPLR also permits corresponding local adjustments in multi-parameter combination. We will continue to make corresponding local adjustments on the previous PLR result of BMPLR as shown in Fig. 6.

In Fig. 7a, suppose that a user wants to transform the current S_2 ($seg{<}12, 31{>}$) into S_2', which is composed of more than two $Segment$ objects, while concurrently limiting the fitting error to less than 100. This local adjustment on S_2 can be achieved by EMPLR, which contains the following two main steps.

- Determine the search range in ARI-Tree. In Fig. 7a, the beginning and the ending points of S_2 are v_{12} ($TN_3.index$) and v_{31} ($TN_4.index$), respectively, and the fitting error of S_2 is 241.0. To adjust S_2 to S_2', more than two $TreeNode$s, whose $index$ is in the range of 12 to 31 and $rank$s are larger than 4, need to be searched in ARI-Tree. Accordingly, the corresponding $TreeNode$s, which are stored in ARI-Tree as the right children of TN_4, should be selected to complete the local adjustment.
- Searching for $TreeNode$s in ARI-Tree. According to the above analysis, these qualifying $TreeNode$s can be found in ARI-Tree, which is equivalent to an efficient binary search. In Fig. 7b, when TN_7 is selected, the original S_2 changes into S_2', which is composed of $seg{<}12, 30{>}$ and $seg{<}30, 31{>}$. Similar to BMPLR, the fitting errors of $seg{<}12, 30{>}$ and $seg{<}30, 31{>}$ can also be obtained from the es_{L} or es_{R} of two adjacent $TreeNode$s. Due to $TN_7.rank$ is larger than $TN_4.rank$, es_{L} of TN_7 (225.6) should be selected as $es_{<12,30>}$ of $seg{<}12, 30{>}$. Analogously, es_{R} of TN_7 (0.0) should be selected as $es_{<30,31>}$ of $seg{<}30, 31{>}$. The current fitting error of S_2' is 225.6, which is larger than 100. Subsequently, TN_8 is selected in Fig. 7c and the updated S_2' is composed of three $Segment$s: $seg{<}12, 18{>}$, $seg{<}18, 30{>}$ and $seg{<}30, 31{>}$. The corresponding fitting errors of these three $Segment$s in S_2' are $es_{<12,18>}$ (36.0), $es_{<18,30>}$ (58.1) and $es_{<30,31>}$ (0.0). At this time the current fitting error of S_2' is 94.1($58.1 + 36.0 + 0.0$), which is less than 100; thus, the local adjustment has been completely achieved.

4.3 Performance analysis of MPLR-IDP

The cost of the representation depends on the time complexity of the PLR algorithm. The time complexities of several highly cited PLR algorithms up to date can be compared. For example, the time complexity of the traditional PLR top-down method is $O(n^2)$, and the time complexity of the traditional PLR sliding window method is $O(l{*}n)$, where l is the number of piecewise linear segments and n is the length of time series. The time complexity of PLR-PIP is $O(n^2)$, and the time complexity of PLR-TP is $O(k * n)$, where k is the number of TPs in the time series.

For MPLR-IDP on a time series with length n (TS_n), the time complexity evaluation is divided into two main aspects.

1. The time complexity for IDP selection and ARI index construction. To select a proper data point as an IDP in TS_n requires $O(\log(n))$ on average and is never worse than $O(n)$; the selected IDP would be created as a $ListNode$ and $TreeNode$ and inserted into ARI-List and ARI-Tree, respectively. The corresponding creation

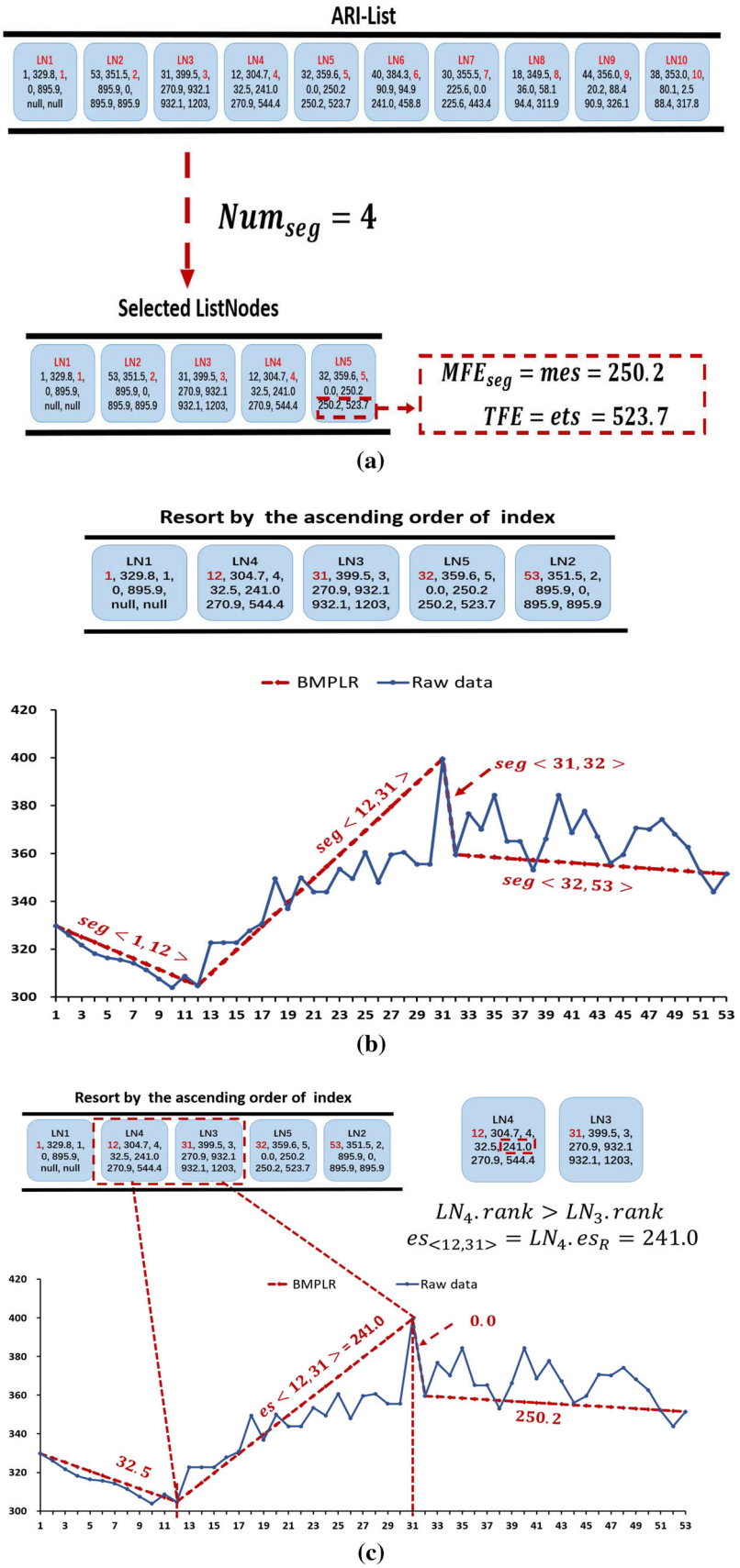


Fig. 6 The main steps of BMPLR

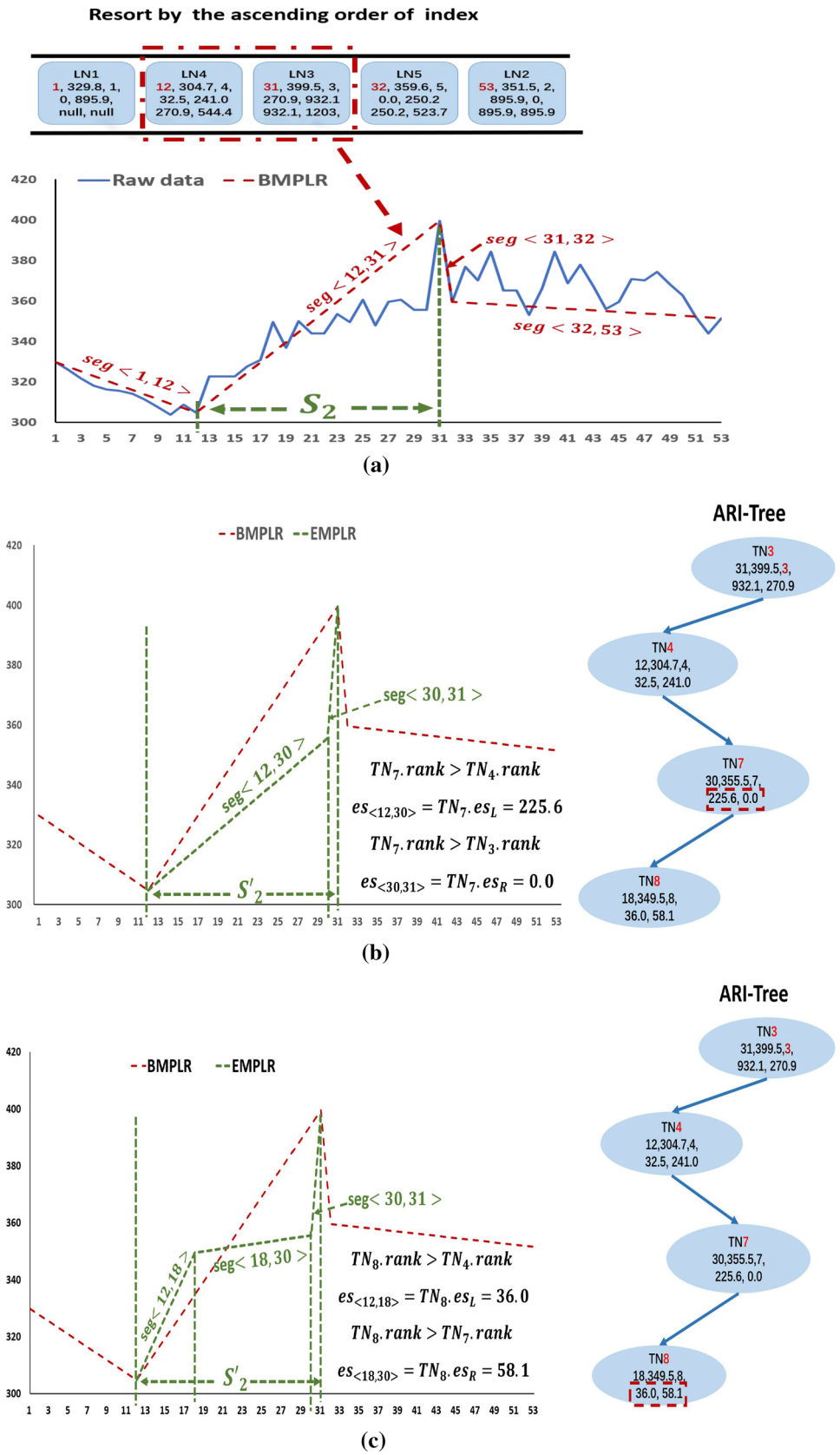


Fig. 7 The main steps of EMPLR

Table 4 The description of datasets

Name	Size	Length	Specification
Worms	258	900	The movement of worms for genetics
Phoneme	2110	1024	The audio segments from Google Translate
Haptics	463	1092	The collected passgraph from graphical authentication system
CinCECGtorso	1420	1639	ECG data for multiple torso-surface sites
InlineSkate	650	1882	EMG data for inline speed skating testing
HSI	5	4481	The historical data of Hang Seng Index from January 3, 2000, to March 1, 2018
Longda	2096	5182	The monitoring data from the cold storage facility of Longda
MW	2735	7735	The revolution data of the momentum wheel in DFH satellite

and insertion operations of $ListNode$ complete in unit time; in other words, the time complexity to add a new node in ARI-List is $O(1)$. The insertion of $TreeNode$ into ARI-Tree is a recursive process based on a binary search, whose time complexity is $O(\log(n))$. So the overall time complexity for the above process is $O(n * (\log(n))^2)$.

2. The time complexity for the multi-resolution PLR on TS_n. According to the above introduction in Sect. 4.2, the basic MPLR completes in two steps. In the first step, the corresponding $ListNode$ sequence is obtained by scanning ARI-List and the time complexity is no worse than $O(n)$. In the second step, the above sequence is reordered based on the nodes' $index$ attribute. The time complexity of this operation is no worse than $O(n * \log(n))$. Therefore, the overall time complexity for BMPLR is $O(n * \log(n))$. Similarly, the extended MPLR can be completed by adding an additional step to BMPLR, (i.e., searching for a certain number of $TreeNode$s in ARI-Tree); thus, the overall time complexity of EMPLR can be approximated as $O(n * \log(n))$.

Through the above analysis, the overall time complexity of MPLR-IDP can be approximated as $O(n * (\log(n))^2)$ and the overall space complexity of MPLR-IDP is no less than $O(n)$. In addition, although MPLR-IDP can complete the corresponding representation and obtain all the relevant representation information (Num_{seg}, TFE, MFE_{seg}) efficiently, each $ListNode$/$TreeNode$ stored in ARI-List/ARI-Tree including seven attributes internally, which results in a certain storage overhead. Therefore, in an actual application scenario, we can complete the corresponding ARI construction according to the objective needs of MPLR; in other words, when only BMPLR (EMPLR) is needed, we can construct ARI-List (ARI-Tree) merely instead of completing the entire ARI construction to satisfy the MPLR requirements while reducing the corresponding storage overhead.

Table 5 The description of experimental environment

Name	Configuration
CPU	Intel® Core™ i7-7700 @ 3.60 GHZ
Memory	Kingston HyperX DDR4 16 GB
Hard disk	WD Blue 3D NAND SSD 500 GB
Operating system	Ubuntu 16.10
Programming language	Java with JDK 1.8.0_77
Database	Oracle 11g (11.2)

5 Experiments and analyses

In this section, we evaluate the performance of our algorithm MPLR-IDP and compare it with the two highly cited multi-resolution PLR algorithms mentioned in Sect. 2. To complete the following experiments, some typical time series datasets, including five open source time series datasets provided by Chen et al. (2015), the Hang Seng Index (HSI) from January 3, 2000, to March 1, 2018, from the Yahoo web site, the monitoring data of Longda and the revolution data of MW in the DFH satellite are selected for corresponding comparison experiments. The descriptions of these datasets and the experimental environment are shown in Table 4 and Table 5, respectively.

Considering that MPLR-IDP and two baseline multi-resolution PLR methods (MPLR-PIP and MPLR-TP) have their own index structures, in our experiments, we not only pay close attention to the efficiency of MPLR but are also concerned with the fitting precision of the corresponding methods. Accordingly, our experiments are divided into the following two subsections.

5.1 Comparison experiments based on operating efficiency

In this subsection, the above three methods (MPLR-IDP, MPLR-PIP and MPLR-TP) each initially establish their own

indexes (ARI, SB-Tree and OBST, respectively) on the above eight datasets. Then, the three different index structures are utilized for the corresponding MPLR. Therefore, this efficiency evaluation is conducted in terms of index construction and multi-resolution representation.

Runtime for index construction

Although each time series sequence in any given dataset has the same length, there are differences among the various sequences. Therefore, to obtain a relatively accurate index construction time for a certain dataset, we execute a large number of repetitive experiments on each dataset. The corresponding experimental settings are given first. Suppose the specific dataset DS with size m is expressed as DS_m, and the ith time series sequence in DS_m is expressed as TS_i, where $1 \leq i \leq m$. Accordingly, the runtimes on all DS_ms (mentioned in Table 4) are calculated completely.

The overall runtime results required to build the above three index structures are shown in Table 6, where the eight datasets have been sorted in ascending order by $Length$ attribute. For example, $Worms(900)$ denotes the dataset $Worms$, in which each time series sequence has 900 data points. In addition, Num in each column denotes the final number of points in each index after the corresponding index has been completely constructed; $Runtime$ denotes the average running time on each time series sequence in a specific dataset. Taking HSI (line 6) as an example, the total number of data points is 4481, the number of TPs in HSI is 2221, and the runtime for OBST and ARI construction is 21.75 ms and 49.73 ms, respectively.

From the results in Table 6, we can see that the runtimes for SB-Tree and ARI construction by MPLR-PIP and MPLR-IDP gradually rise as the number of data points increases. Similarly, the runtimes for OBST construction by MPLR-TP also grow as the number of TPs increases. The difference between the above can be clearly demonstrated based on the runtime comparison between HSI (line 6) and $Longda$ (line 7). As the number of data points grows, so do the corresponding runtimes of MPLR-PIP and MPLR-IDP. However, because the number of TPs is basically the same, the runtime of MPLR-TP remains almost constant

Compared with the above three methods, due to the relatively small number of TPs involved OBST construction, the overall processing time of MPLR-TP is much smaller than those of the other methods; correspondingly, the reason why MPLR-IDP has a longer runtime than the other two methods is that all the $rank$, est, mes, $dist_{\max}$, etc., in each IDP selection are saved completely and inserted in ARI-List and ARI-Tree as $ListNode$ and $TreeNode$ objects, respectively. These operations result in a greater time overhead compared to MPLR-PIP and MPLR-TP.

Runtime for multi-resolution representation

After the above three index structures have been constructed completely, a large number of multi-resolution representation could be implemented by these index structures according to different $TSRS$s. Compared to MPLR-IDP and MPLR-PIP, MPLR-TP provides a relatively small range of MPLR. For example, based on the result of HSI in Table 6, MPLR-IDP and MPLR-PIP can produce multi-resolution representation results based on the 4481 IDPs/PIPs stored in ARI/SB-Tree. MPLR-TP can provide similar results based on the 2221 TPs stored in OBST.

Considering the above situation, the comparison experiments on the runtime of multi-resolution representation are carried out as follows.

- First, for fairness, MPLR-IDP, MPLR-PIP and MPLR-TP should provide corresponding representation results based on the same number of points stored in ARI, SB-Tree and OBST, respectively. In other words, it is necessary for the above three methods to compare the running time in the case of, respectively, generating the final PLR results with the same number of segments (named as Num_{seg}). Given that the maximum number of TPs stored into OBST, denoted as MN_{TP} for simplicity, Num_{seg} is uniformly set to $MN_{\mathrm{TP}} + 1$. Taking HSI as an example, Num_{seg} on HSI would be set to 2222.
- Second, without loss of generality, given that the dataset DS contains n time series TS_i ($1 \leq i \leq n$), for each time series TS_i, the corresponding MPLR results including the fitting error of entire time series ets and the maximum fitting error of segments mes should be generated by the above three methods, respectively, based on the same Num_{seg}. The running time of producing the corresponding result on TS_i of DS is the cumulative time of performing 100 representation operations continuously.
- Finally, the total runtime for multi-resolution representation on DS can be obtained by accumulating the PLR runtime on each TS_j of DS. Accordingly, the runtime results on the above eight datasets for MPLR based on three different index structures are shown in Table 7.

From the results in Table 7, the runtimes for MPLR by the three MPLR methods gradually rise as Num_{seg} increases. In general, MPLR-TP is slightly faster than MPLR-PIP, which can be attributed primarily to the corresponding OBST structure in MPLR-TP; in other words, all the TPs in OBST can be obtained in order by a complete depth-first traversal search without repetitive sort operations. However, the PLR runtime by MPLR-IDP on different dataset is shorter than the other two methods. According to the introduction in Sect. 4.2, ARI index has already stored all the required types of representation information along with each IDP selection.

Table 6 Runtime for index construction (in ms)

Dataset	MPLR-PIP with SB-Tree		MPLR-TP with OBST		MPLR-IDP with ARI	
	Num	*Runtime*	*Num*	*Runtime*	*Num*	*Runtime*
Worms (900)	900	5.58	157	4.03	900	6.81
Phoneme (1024)	1024	6.04	306	5.67	1024	7.02
Haptics (1092)	1092	6.11	327	5.71	1092	7.06
CinCECGtorso (1639)	1639	7.78	430	6.61	1639	9.36
InlineSkate (1882)	1882	8.24	1063	7.86	1882	10.12
HSI (4481)	4481	38.03	2221	21.75	4481	49.73
Longda (5182)	5182	49.83	2217	21.73	5182	60.82
MW (7735)	7735	92.79	4566	51.92	7735	113.73

Table 7 Runtime for MPLR with Num_{seg} (in ms)

Dataset	MPLR-PIP with SB-Tree		MPLR-TP with OBST		MPLR-IDP with ARI	
	Num_{seg}	*Runtime*	Num_{seg}	*Runtime*	Num_{seg}	*Runtime*
Worms (900)	158	6.83	158	6.78	158	1.46
Phoneme (1024)	307	8.01	307	7.89	307	1.68
Haptics (1092)	328	8.11	328	8.13	328	1.71
CinCECGtorso (1639)	431	11.73	431	11.67	431	2.19
InlineSkate (1882)	1064	21.68	1064	20.99	1064	8.45
HSI (4481)	2222	34.29	2222	33.87	2222	17.32
Longda (5182)	2218	34.12	2218	33.92	2218	17.25
MW (7735)	4567	51.33	4567	51.11	4567	29.87

Therefore, the relevant information in a specific MPLR can be obtained efficiently by searching for the corresponding $ListNode$s and $TreeNode$s in ARI. Taking the MPLR with $num_{seg} = 2222$ as an example, the corresponding ets and mes can be obtained immediately by searching the 2223rd $ListNode$ in ARI-List; in other words, the corresponding time expenditure for MPLR-IDP is mainly consumed in sorting the corresponding $ListNode$s in ascending $index$ order. Furthermore, considering that the number of specific nodes is determined in advance, the direct insertion sort algorithm could be used efficiently for the corresponding MPLR.

However, SB-Tree stores only $rank$ and $dist_{\max}$ with each PIP selection, while OBST stores only $rank$ in each TP evaluation; therefore, the corresponding ets and mes of MPLR are obtainable only through repeated calculations. Consequently, the MPLR produced by MPLR-IDP is more efficient than those of the other two methods.

5.2 Comparison experiments based on fitting precision

In this section, the corresponding fitting errors of PLR (ets, mes) by the above three methods are analyzed comparatively. To obtain a relatively accurate fitting error, the overall experimental setting is consistent with the preceding experiments on index construction runtimes. Based on the introduction in Sect. 3, the specific number of points in the corresponding index structures for PLR is equivalent to PLR with Num_{seg}. For example, if we select K IDPs for PLR, there will be $K-1$ segments in the corresponding representation result ($Num_{seg} = K-1$). Therefore, a series of MPLR results on a certain dataset could be implemented by varying the number of points (IDPs, PIPs, TPs), which can also be denoted as the data compression ratio (DCR). $\mathrm{DCR_{IDP}}$ is the ratio of the current specific number of IDPs to the total number of points in the entire time series. The corresponding equation is as follows.

$$\mathrm{DCR_{IDP}} = \frac{\mathrm{IDP_{num}}}{\mathrm{point_{sum}}} \tag{9}$$

In Eq. (9), $\mathrm{IDP_{num}}$ denotes the number of IDPs in accordance with the ascending selection order from ARI-List. $point_{sum}$ denotes the total number of data points in the entire time series. Similarly, $\mathrm{DCR_{PIP}}$ and $\mathrm{DCR_{TP}}$ are defined in the following equations.

$$\mathrm{DCR}_{\mathrm{PIP}} = \frac{\mathrm{PIP}_{\mathrm{num}}}{\mathrm{point}_{\mathrm{sum}}} \tag{10}$$

$$\mathrm{DCR}_{\mathrm{TP}} = \frac{\mathrm{TP}_{\mathrm{num}} + 2}{\mathrm{point}_{\mathrm{sum}}} \tag{11}$$

According to the analysis in Sect. 5.1, when using MPLR-TP on a certain dataset, the beginning and ending point of time series should be selected automatically. Therefore, the number of TPs in Eq. (11) should be expressed as $\mathrm{TP}_{\mathrm{num}} + 2$.

After completing the above experimental settings, to ensure the effectiveness of *ets* and *mes* obtained from the above three MPLR methods, the corresponding representation on a certain dataset should be required to have the same DCR ($\mathrm{DCR}_{\mathrm{IDP}}$, $\mathrm{DCR}_{\mathrm{PIP}}$, $\mathrm{DCR}_{\mathrm{TP}}$) from MPLR-IDP, MPLR-PIP and MPLR-TP, respectively. The overall representation results from the above four datasets are shown in Table 8.

As shown by the results in Table 8, the *ets* and *mes* of MPLR by MPLR-PIP and MPLR-TP decline gradually when DCR (Num_{seg}) increases. Compared to the other two MPLR methods, MPLR-TP cannot provide a corresponding data representation at some DCRs, due to its limited number of TPs. In *HSI* dataset, the *ets* and *mes* of MPLR-TP are smaller than those of the other two methods; in other words, MPLR-TP provides a more accurate PLR than do MPLR-PIP and MPLR-IDP on DCR = 10%, which proves that the TPs based MPLR method (MPLR-TP) has corresponding advantages in financial data multi-resolution representation. As the DCR of *HSI* continues to increase, the fitting precision of MPLR-TP is gradually surpassed by those of MPLR-PIP and MPLR-IDP. According to the introduction of TP importance evaluation in Sect. 2, when the DCR increases from 20% to 30%, more trivial TPs than before (DCR=10%) are utilized for MPLR; thus, their ability to reduce the corresponding fitting error in the specific data representation is limited. The above analysis can be confirmed from the three rows of *HSI* (DCR = 20%,DCR = 30%,DCR = 40%) in Table 8, where we find that the *mes* of MPLR-TP is basically the same, so its reduction of the *ets* is also relatively slow.

In contrast, according to the definition of IDP in Sect. 3, the point with *mep* in the current segment with *mes* should be selected as an IDP; in other words, each IDP selection reflects the current overall fitting precision to some extent. Therefore, even more relatively unimportant IDPs are utilized for MPLR along with the DCR increases, the corresponding *ets* and *mes* are reduced more efficiently than those of the other two methods, which can also be confirmed from the *mes* results in Table 8. In this table, we can find the *ets* of MPLR-IDP may bigger than that of MPLR-PIP in some cases due to the difference in the selection order of *mep* point introduced in Sect. 3; however, MPLR-IDP can ensure the *mes* results under all DCR conditions are smaller than those of other two methods on different datasets. In summary, MPLR-IDP provides a more accurate representation than MPLR-PIP and MPLR-TP.

According to the results of the comparison experiments presented above, compared to MPLR-PIP and MPLR-TP, although the runtime for ARI index construction is relatively long. MPLR-IDP can provide more efficient and accurate MPLR results than MPLR-PIP and MPLR-TP.

6 Extensional application

In this section, the effectiveness of MPLR-IDP as a preprocessing tool for time series classification is highlighted and further analyzed.

Time series classification (TSC) has been attracting great interest over the past decade. While dozens of techniques have been introduced, recent empirical evidence has strongly suggested that shapelet-based TSC algorithms outperform many previous TSC algorithms in terms of accuracy, efficiency and interpretability (He et al. 2012). According to the concept of shapelets in Ye and Keogh (2009), shapelets are not only subsequences extracted from single time series, but also have distinctly representative class membership characteristics. With the help of shapelets, TSC can utilize the similarity between two shapelets, rather than the similarity between two entire time series, to complete time series classification. Consequently, the overall performance of these shapelet-based TSC methods can be greatly enhanced; moreover, the appropriate shapelets can provide sufficient information to make the results of classification more explainable. Therefore, an evolutionary algorithm by utilizing shapelets for TSC has been proposed by Bagnall et al. (2016) called shapelet transformation (ST), which not only optimizes the process of shapelet evaluation, but also allows various classification strategies (SVM, random forest, etc.) to be adopted to classify time series objects after the shapelet selection process has been completed (Hills et al. 2014). In other words, the shapelet selection process and the classification operation are relatively independent between each other. After the corresponding transformation is completed, a larger number of off-the-shelf classification algorithms can be applied in ST. Accordingly, various scholars (Hills et al. 2014; Xing et al. 2011; Mueen et al. 2011) have done much work on improving efficiency of ST.

According to the above analysis, the process of ST can be summarized into the following three steps: quality measurement, shapelets generation and selection, and data transformation, as shown in Fig. 8.

However, the previous research was mainly concerned with how to reduce the time needed to evaluate every shapelet candidate, while there are still too many shapelet candidates waiting for the same evaluation processing. As shown in Lines et al. (2012), Hills et al. (2014), nearly all the subse-

Table 8 Fitting errors of MPLR by DCR

Dataset	DCR	MPLR-PIP			MPLR-TP			MPLR-IDP		
		num_{seg}	ets	mes	num_{seg}	ets	mes	num_{seg}	ets	mes
Worms (900)	DCR=10% (90)	89	76.57	5.48	89	163.08	36.04	89	72.31	1.89
	DCR = 20% (180)	179	32.41	1.07				179	29.82	0.53
Phoneme (1024)	DCR = 10%(102)	101	418.93	16.29	101	484.35	25.81	101	433.86	10.06
	DCR = 20% (204)	203	177.68	6.88	203	222.9	7.57	203	170.7	2.11
	DCR = 30% (307)	306	102.0	2.3	306	175.67	7.57	306	98.01	0.92
	DCR = 40% (409)	408	59.29	0.9				408	60.35	0.47
	DCR = 50% (512)	511	38.08	0.87				511	37.6	0.24
Haptics (1092)	DCR = 10%(109)	108	6.69	0.24	108	50.14	5.1	108	6.3	0.2
	DCR = 20% (218)	217	3.91	0.1	217	28.73	5.1	217	0.01	0.001
	DCR = 30% (327)	326	3.14	0.07	326	28.01	5.1	326	0.001	0.001
InlineSkate (1882)	DCR = 10% (188)	187	26.34	0.79	188	37.67	4.03	188	25.79	0.74
	DCR = 20% (376)	375	17.88	0.35	375	22.49	0.34	375	17.56	0.14
	DCR = 30% (564)	563	12.31	0.21	563	15.15	0.17	563	12.11	0.07
	DCR = 40% (752)	751	8.5	0.09	751	10.36	0.17	751	8.52	0.04
	DCR = 50% (941)	940	5.59	0.05	940	7.52	0.17	940	5.85	0.02
	DCR = 60% (1129)	1128	3.68	0.04				1128	3.36	0.01
HSI (4481)	DCR = 10% (448)	447	811158.06	20240.52	447	737285.37	10165.59	447	748569.21	4086.25
	DCR = 20% (896)	895	439618.53	5928.01	895	449869.09	3231.36	895	424825.24	1203.33
	DCR = 30% (1344)	1343	276280.63	2634.76	1343	343412.34	3231.36	1343	279727.75	580.63
	DCR = 40% (1792)	1791	184807.77	1282.05	1791	279313.34	3231.36	1791	181892.66	322.85
	DCR = 50% (2240)	2239	120593.53	1020.47				2239	116193.48	198.97

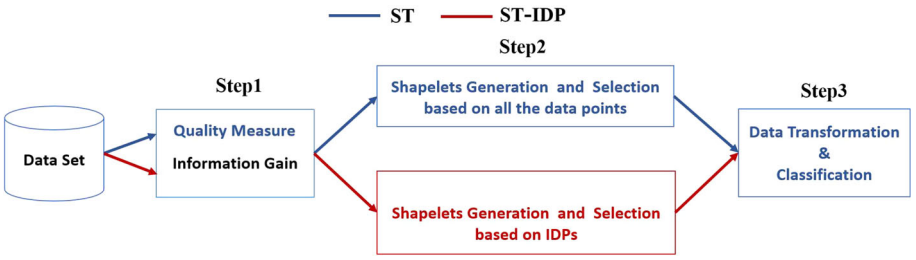


Fig. 8 The main steps of ST and ST-IDP

quences of a time series are selected as shapelet candidates. Therefore, the overall efficiency of ST is still poor, which definitely hampers the classification efficiency and obstructs the scalability of ST.

Motivated by the above analysis, considering the definition of IDP (in Sect. 3), the selection order of a certain IDP reflects the importance of that IDP on the main trend reconstruction for the entire time series. Therefore, we utilize a certain number of IDPs (stored in ARI) to reduce the number of shapelet candidates as much as possible. As shown in Fig. 8, our proposed method called ST-IDP replaces only Step 2 of the original ST (the rest steps of ST remain unchanged). The subsequent empirical results demonstrate that ST-IDP not only speeds up the shapelet selection process but also improves the overall efficiency of ST.

6.1 MPLR-IDP for shapelet selection

According to the above analysis, suppose that a training dataset TDS contains m time series, whose length is n. Without loss of generality, TDS can be expressed as $TDS = \{TS_n^1, TS_n^2, \ldots, TS_n^i, \ldots, TS_n^m\}$ $(1 \leq i \leq m)$ according to Definition 1.

The profound implication of the shapelet definition in Lines et al. (2012) implied that shapelet candidates should be refined as subsequences which maximally represent a specific class. In other words, some subsequences which have significances in TS_n^j should be further extracted to represent the class (Zhang et al. 2016). Accordingly, the overall ARIs on TDS can be denoted as $\text{ARI}_m = \{\text{ARI}_n^1, \text{ARI}_n^2, \ldots, \text{ARI}_n^i, \ldots, \text{ARI}_n^m\}$ $(1 \leq i \leq m)$, where ARI_n^i denotes the correlative ARI-List on TS_n^i. With the help of ARI_n^i, the appropriate number of IDPs in TS_n^i can be selected to help generate the corresponding shapelet candidates more efficiently.

To illustrate the shapelet generation and selection process by ST-IDP more clearly, the real time series sensor dataset named MoteStrain from the UEA and UCR Time Series Classification Repository (Bagnall et al. 2017) is selected as an example. The first time series, whose length is 84, in the training dataset of "MoteStrain" (TS_{84}^1) and the part of ARI-List (ARI_{84}^1) on TS_{84}^1 are shown in Fig. 9a and b, respectively. In addition, for simplicity, only two attributes $(rank, index)$ are given for the first eight $ListNode$s listed in Fig. 9b. With the help of ARI_{84}^1, vt_{37} (a IDP of TS_{84}^1, whose $index$ is 37) is identified immediately based on LN_6 in ARI_{84}^1. Similarly, other 7 IDPs are obtained efficiently and shown in Fig. 9c.

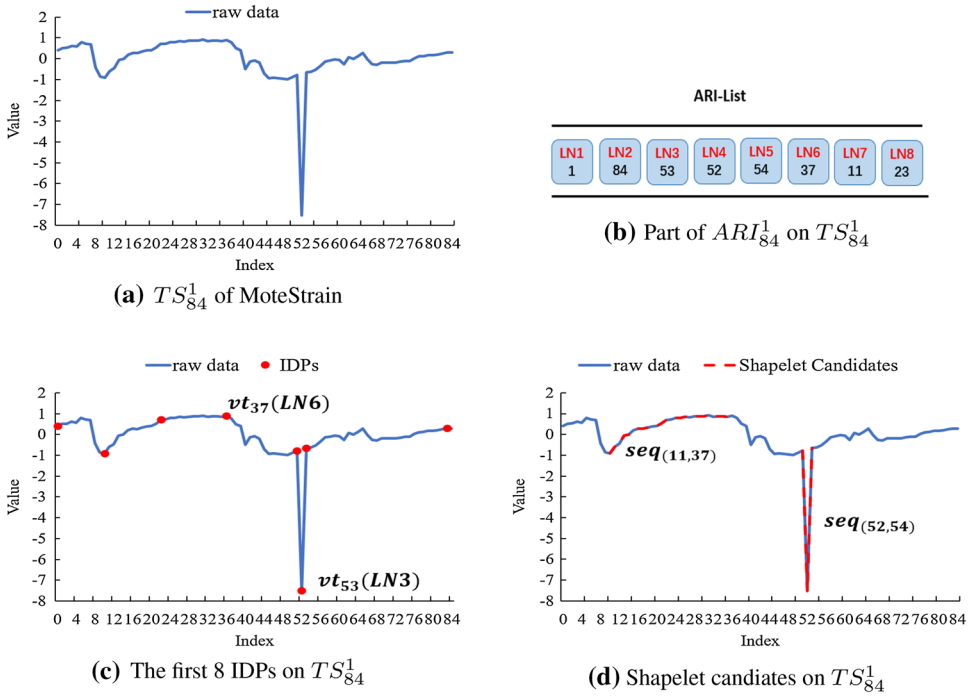
Subsequently, a series of subsequences are generated based on the number of IDPs, according to the following two rules.

1. The beginning and ending points of the subsequence must be IDPs.
2. The above two points cannot be adjacent IDPs.

In the above example, for TS_{84}^1, the first eight IDPs stored in ARI_{84}^1 can be utilized to generate 21 subsequences, two of which ($seq_{(11,37)}$,$seq_{(52,54)}$) are shown in Fig. 9d. Obviously, these subsequences do reflect the corresponding temporal features of TS_{84}^1 and should be selected as shapelet candidates. Furthermore, for TS_{84}^1 of length 84, the number of shapelet candidates generated by ST-IDP (21) is much smaller than the number of shapelet candidates produced by ST ($3468 = C_{84}^2$). These 21 shapelet candidates form the set of candidates for TS_{84}^1, named as sub_1.

We adopt a similar strategy to process the rest of the time series in TDS. After all the shapelet candidates in each sub_i have been generated, the new data collection $SubTS$, expressed as $SubTS = \{sub_1, sub_2, \ldots, sub_i, \ldots, sub_n\}$ $(1 \leq i \leq m)$, has been completely formed. Subsequently, $information\ gain$ in Ye and Keogh (2011) is utilized to select K shapelets in $SubTS$ to form the final shapelets set, named FSS for short. FSS can be expressed as $FSS = \{S_1, S_2, \ldots, S_j, \ldots, S_K\}$ $(1 \leq j \leq K)$. The algorithm for shapelets selection is shown in Algorithm 6 and can be subdivided into three steps as follows.

1. Get all the IDPs in each TS in TDS.
2. Generate a series of sequences from each T by IDPs to form the entire $SubTS$
3. Select K shapelets from $SubTS$ using the information gain strategy from Bagnall et al. (2016) to form FSS.



(a) TS^1_{84} of MoteStrain

(b) Part of ARI^1_{84} on TS^1_{84}

(c) The first 8 IDPs on TS^1_{84}

(d) Shapelet candiates on TS^1_{84}

Fig. 9 IDP selection and shapelets generation

Algorithm 6 Shapelets Simplification and Selection

Input: All the time series in $TDS = \{TS_1, TS_2, \ldots, TS_j, \ldots, TS_n\}$, the corresponding ARIs constructed on TDS
Output: FSS for storing the final shapelets.
1: **for all** TS in TDS **do**
2: $IDPList$ = ARI$_{TS}$.Search(TS)
3: $IDPSet$.add($IDPList$)
4: **end for**
5: **for** i = 1 to $IDPSet.length$ **do**
6: $IDPList = IDPSet$.get(i)
7: $subList$ = subTS($IDPList$,TDS.get(i))
8: $SubTS$.add($subList$)
9: **end for**
10: FSS = infoJudge($cSet$,K)
11: **return** FSS

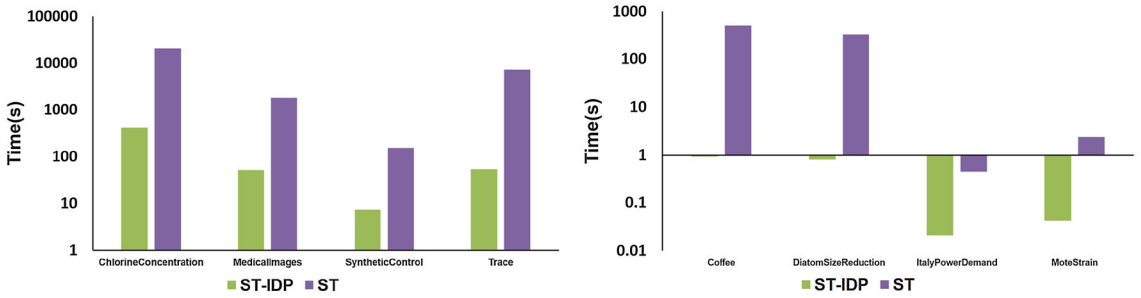
Table 9 Two types of datasets

Type	Dataset	
	Name	Size
DS_b with Size ≥ 100	ChlorineConcentration	467
	MedicalImages	381
	SyntheticControl	300
	Trace	100
DS_s with Size < 100	Coffee	28
	DiatomSizeReduction	16
	ItalyPowerDemand	67
	MoteStrain	20

Finally, after completing the above operations in Algorithm 6, we can use the FSS for further data transformation and TSC. According to the analysis in Fig. 8, subsequent data transformation and classification operations of ST-IDP are completely consistent with ST; therefore, the corresponding operation steps will not be described here (more details can be found in Bagnall et al. (2017)).

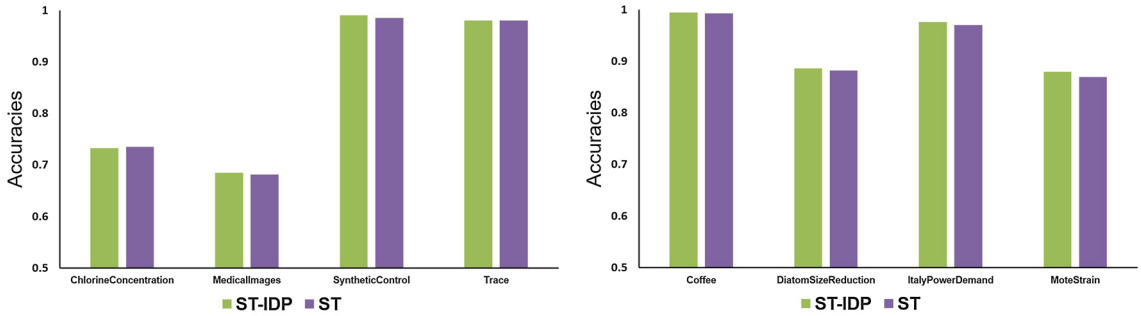
6.2 Performance analysis

According to the above analysis, to speed up the efficiency of shapelet selection, the original shapelet selection process is replaced by our acceleration strategy based on IDPs . To distinguish the above two methods, we name our proposed method ST-IDP, which has the identical post-processing operations of ST. We conduct a series of comparative experiments on eight highly cited datasets in different fields from the UEA and UCR Time Series Classification Repository (Bagnall et al. 2017). Based on the published research results from Bagnall et al. (2016), the number of shapelets (k) is set to half the number of time series sequences m in TDS, i.e., $k = m/2$. Moreover, due to the diverse needs of different users, the number of IDPs selected for subsequences generation are also disparate. Therefore, an automatic approach to obtain the corresponding number of IDPs in accordance with the specific needs of users is necessary. With the help of ARI-List and the definition of DCR in Sect. 5.2, the specific number of IDPs can be obtained efficiently. Assume that a user wants to get five IDPs for subsequences generation, the first five $ListNode$s in ARI-List can be obtained conveniently and converted to their corresponding IDPs. Accordingly, in this paper, the



(a) Running Time Comparison on DS_b **(b)** Running Time Comparison on DS_s

Fig. 10 Running time comparison on different datasets



(a) Accuracies Comparison on DS_b **(b)** Accuracy Comparison on DS_s

Fig. 11 Accuracy comparison on different datasets

number of selected IDPs in each TS_n^i is set to DCR $*$ n, where DCR $=$ 5%, i.e., the number of selected IDPs is $\lceil 0.05 * n \rceil$.

Furthermore, considering that different training dataset sizes may have some impact on shapelet selection based on IDPs, the above eight datasets are divided into two types based on the training dataset sizes. Accordingly, the "**Size**" in Table 9 lists the sizes of the training dataset of the eight selected datasets. The corresponding running time comparison results are shown in Fig. 10, and the classification accuracy comparison results are shown in Fig. 11.

As shown in Fig. 10, the efficiency of ST-IDP is substantially higher than ST by more than one order of magnitude on average. Moreover, the efficiency of ST-IDP in Fig. 10b is much higher than that of ST-IDP in Fig. 10a, which means that the efficiency of shapelet selection and transformation based on IDPs in relatively small datasets is higher than those in large datasets. The reason is that in general, the number of IDPs in DS_s is also smaller accordingly; the corresponding processing efficiency on DS_s is relatively higher than on DS_b.

According to the results in Fig. 11, the average classification accuracy of the above two methods is basically in the same level, which means that our proposed shapelet selection strategy does not reduce the accuracy of time series classification, but greatly improves the corresponding classification efficiency.

7 Conclusion

In this paper, we propose a novel multi-resolution piecewise linear representation algorithm MPLR-IDP which preserves the main time series characteristics and guarantees more accurate piecewise linear representation. More importantly, MPLR-IDP can provide two MPLR strategies to meet the diverse needs from different users. Thus, MPLR-IDP can not only efficiently produce adaptive MPLR in accordance with $TSRS$, but also support more flexible local representation adjustments. Furthermore, a certain number of IDPs can be selected by the adaptive representation index (ARI) and utilized for efficient shapelet selection. The corresponding experimental results demonstrate the following conclusions. For one thing, our proposed shapelet selection approach greatly improves the TSC efficiency of the original ST. For another, our proposed shapelet selection strategy does not affect the TSC accuracy. In the future, we plan to use MPLR-IDP as a useful tool for time series prediction and time series anomaly detection.

Acknowledgements The authors would like to thank the anonymous reviewers and the editors for their insightful comments and suggestions,

which are greatly helpful for improving the quality of this paper. This work is supported by the National Natural Science Foundation of China, No.: 61772310, No.: 61702300, No.: 61702302, No.: 61802231; the Science and Technology Development Funds of Shandong Province, No.: 2014GGX101028; the Project of Qingdao Postdoctoral Applied Research.

Compliance with ethical standards

Conflict of interest All authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

Informed consent Informed consent was obtained from all individual participants included in the study.

References

Agrawal R, Faloutsos C, Swami A (1993) Efficient similarity search in sequence databases. In: International conference on foundations of data organization and algorithms. Springer, pp 69–84

Bagnall A, Bostrom A, Large J, Lines J (2016) The great time series classification bake off: an experimental evaluation of recently proposed algorithms. Extended version CoRR. arXiv:1602.01711

Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Discov 31(3):606–660

Chan KP, Fu AWC (1999) Efficient time series matching by wavelets. In: 15th international conference on data engineering, 1999. Proceedings. IEEE, pp 126–133

Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, Batista G (2015) The ucr time series classification archive

Doerr B, Fischer P, Hilbert A, Witt C (2016) Detecting structural breaks in time series via genetic algorithms. Soft Comput 21(16):4707–4720

Fu TC (2011) A review on time series data mining. Eng Appl Artif Intell 24(1):164–181

Fu TC, Chung Fl, Luk R, Ng CM (2008) Representing financial time series based on data point importance. Eng Appl Artif Intell 21(2):277–300

He Q, Dong Z, Zhuang F, Shang T, Shi Z (2012) Fast time series classification based on infrequent shapelets. In: 2012 11th international conference on machine learning and applications (ICMLA), vol 1. IEEE, pp 215–219

Hills J, Lines J, Baranauskas E, Mapp J, Bagnall A (2014) Classification of time series by shapelet transformation. Data Min Knowl Discov 28(4):851–881

Keogh EJ, Smyth P (1997) A probabilistic approach to fast pattern matching in time series databases. KDD 1997:24–30

Keogh EJ, Pazzani MJ (1998) An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. KDD 98:239–243

Keogh E, Chu S, Hart D, Pazzani M (2001) An online algorithm for segmenting time series. In: Proceedings IEEE international conference on data mining, 2001. ICDM 2001. IEEE, pp 289–296

Korn F, Jagadish HV, Faloutsos C (1997) Efficiently supporting ad hoc queries in large datasets of time sequences. ACM Sigmod Record 26:289–300

Lines J, Davis LM, Hills J, Bagnall A (2012) A shapelet transform for time series classification. In: Proceedings of the 18th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 289–297

Lomet D, Hong M, Nehme R, Zhang R (2008) Transaction time indexing with version compression. Proc VLDB Endow 1(1):870–881

Mueen A, Keogh E, Young N (2011) Logical-shapelets: an expressive primitive for time series classification. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 1154–1162

Park S, Lee D, Chu WW (1999) Fast retrieval of similar subsequences in long sequence databases. In: 1999 workshop on knowledge and data engineering exchange 1999. (KDEX'99) proceedings. IEEE, pp 60–67

Perng CS, Wang H, Zhang SR, Parker DS (2000) Landmarks: a new model for similarity-based pattern querying in time series databases. In: 16th international conference on data engineering 2000. Proceedings. IEEE, pp 33–42

Pratt KB, Fink E (2002) Search for patterns in compressed time series. Int J Image Graph 2(01):89–106

Qu Y, Wang C, Wang XS (1998) Supporting fast search in time series for movement patterns in multiple scales. In: Proceedings of the seventh international conference on information and knowledge management. ACM, pp 251–258

Shatkay H, Zdonik SB (1996a) Approximate queries and representations for large data sequences. In: Twelfth international conference on data engineering, pp 536–545

Shatkay H, Zdonik SB (1996b) Approximate queries and representations for large data sequences. In: Proceedings of the twelfth international conference on data engineering, 1996. IEEE, pp 536–545

Si YW, Yin J (2013) Obst-based segmentation approach to financial time series. Eng Appl Artif Intell 26:2581–2596

Wan Y, Si YW (2017) A hidden semi-Markov model for chart pattern matching in financial time series. Soft Comput 22(19):6525–6544

Xing Z, Pei J, Yu PS, Wang K (2011) Extracting interpretable features for early classification on time series. In: Proceedings of the 2011 SIAM international conference on data mining. SIAM, pp 247–258

Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Discov 22(1):149–182

Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp 947–956

Yin S, Kaynak O (2015) Big data for modern industry: challenges and trends [point of view]. Proc IEEE 103(2):143–146

Zhang Z, Zhang H, Wen Y, Yuan X (2016) Accelerating time series shapelets discovery with key points. In: Asia-Pacific web conference. Springer, pp 330–342

Zhou DZ, Li MQ (2008) Time series segmentation based on series importance point. Comput Eng 23:14–16