An Indexable Time Series Dimensionality Reduction Method for Maximum Deviation Reduction and Similarity Search

Ruidong Xue
Aston University
Birmingham, United Kingdom
xuer@aston.ac.uk

Weiren Yu*
University of Warwick
Coventry, United Kingdom
weiren.yu@warwick.ac.uk

Hongxia Wang
Aston University
Birmingham, United Kingdom
wangh25@aston.ac.uk

ABSTRACT

Similarity search over time series is essential in many applications. However, it may cause "the curse of dimensionality" due to the high dimensionality of time series. Various dimensionality reduction methods have been developed. Some of them sacrifice maximum deviation to get faster dimensionality reduction. The Adaptive Piecewise Linear Approximation (*APLA*) method uses guaranteed error bounds for the best maximum deviation, but it takes a long time for dimensionality reduction. We propose an adaptive-length dimensionality reduction method, called Self Adaptive Piecewise Linear Approximation (*SAPLA*). It consists of 1) initialization; 2) split & merge iteration; and 3) segment endpoint movement iteration. Increment Area, Reconstruction Area, and several equations are applied to prune redundant computations. Experiments show that our method outperforms *APLA* by n times with a minor maximum deviation loss, where n is the length of the time series. We also propose a lower bound distance measure between time series to guarantee lower bounding lemma and tightness for adaptive-length dimensionality reduction methods. Moreover, we propose a Distance-Based Covering with Convex Hull (DBCH) structure to improve *APCA MBR* for adaptive-length dimensionality reduction methods. When mapping time series into a DBCH-tree, we split nodes and pick branches using the lower bounding distance. Our experimental evaluations on 117 datasets from the UCR2018 Archive demonstrate the efficiency and effectiveness of the proposed approaches.

1 INTRODUCTION

With the development of data collection and storage techniques, large volumes of time series in scientific domains and business processes require data mining methods to find meaningful information. Similarity search over time series is a widely studied and essential task for high-level data mining tasks such as classification, prediction, clustering, anomaly detection, motif discovery, and semantic segmentation. Time series can be regarded as a high-dimensional data type. For example, k-Nearest Neighbor (k-NN) is popularly used for classification. However, similarity search is often expensive in terms of time and space. Thus, dimensionality reduction methods and pruning methods have been proposed to reduce the computation cost. Due to the ubiquitous nature of time series data, reducing dimensionality while maintaining important characteristics is a big challenge. A Euclidean distance [10] is often used for measuring similarity between time series. Due to high dimensionality, similarity queries over the original time series may cause a "dimensionality curse" [3] with increasing time series length. Representation coefficients from dimensionality reduction methods will degrade the distance computation as some information is lost after dimensionality reduction. We need to reduce the maximum deviation for the tighter distances between the original and reconstructed time series by representation coefficients. Max deviation is used to measure the reduction performance of dimensionality reduction methods (e.g., [2, 5, 9, 17, 19]). The lower the maximum deviation, the more pruning opportunities there are in k-NN. With memory becoming cheaper, it is affordable to configure a computer with a large main memory. An efficient main memory index should minimise the distance computation. A general framework called the Generic Multimedia Indexing Method (*GEMINI*) [10] converts time series into a lower dimensional representation, and it uses a lower bound of the Euclidean distance to guarantee no-false-dismissals while filtering through the index. In a branch-and-bound [6] search (e.g. k-NN), the R-tree [11] is commonly used in multi-dimensional indexing.

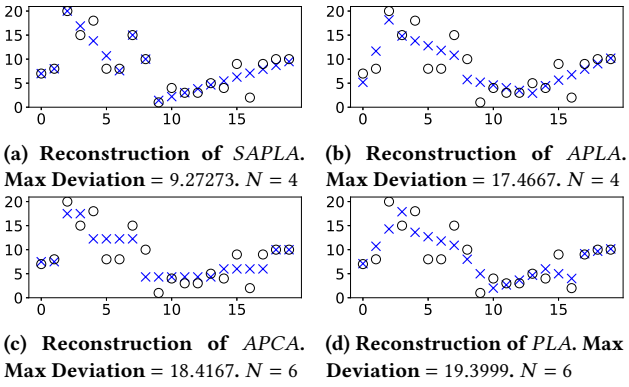
Many dimensionality reduction techniques have been proposed. The equal-length segmentation methods include Piecewise Linear Approximation (*PLA*) [5], Piecewise Aggregate Approximation (*PAA*) [12, 23], Chebyshev Polynomials (*CHEBY*) [2], and Piecewise Aggregate Approximation Lagrangian Multipliers (*PAALM*) [21]. The adaptive-length segmentation methods include Adaptive Piecewise Linear Approximation (*APLA*) [17], and Adaptive Piecewise Constant Approximation (*APCA*) [4, 13] etc. The Symbolic Aggregate Approximation (*SAX*) [15] is the symbolic representation for time series that allows for dimensionality reduction and distance measures that lower-bound Euclidean distance measures on the original time series.

1.1 Motivation

Adaptive-length dimensionality reduction aims at finding the approximate segments of the original time series but is rather time-consuming, especially for regularly changed time series, such as *EOG* datasets. Among them, *APLA* [17] combines the virtues of *APCA* [13] and *PLA* [5] to reduce maximum deviation. Because *APLA* has guaranteed error bounds in the reduction process (by scanning each point to get maximum deviation), *APLA* has $O(Nn^2)$ time complexity, where n is the original time series length, and N is the number of segments after the reduction process. Our experiments show that the *APLA* is much slower than other dimensionality reduction methods. *APLA* minimises only one segment's maximum deviation, rather than the sum of all maximum deviations. For example, two adjacent segments are denoted as $\hat{c}_i$ and $\hat{c}_{i+1}$. The left segment $\hat{c}_i$ already has the minimum max deviation. The sum of the maximum deviations would be reduced if the right endpoint of $\hat{c}_i$ was moved.

We propose an adaptive-length dimensionality reduction method called Self Adaptive Piecewise-Linear Approximation (*SAPLA*). Fig. 1 pictorially compares the sum of the maximum deviations of *SAPLA*, *APLA*, *APCA*, and *PLA*. The length of the original time series is 20. For a fair comparison, these methods have the

*The corresponding author is Weiren Yu.



(a) Reconstruction of *SAPLA*. Max Deviation = 9.27273. $N = 4$

(b) Reconstruction of *APLA*. Max Deviation = 17.4667. $N = 4$

(c) Reconstruction of *APCA*. Max Deviation = 18.4167. $N = 6$

(d) Reconstruction of *PLA*. Max Deviation = 19.3999. $N = 6$

Figure 1: A visual comparison of time series dimensionality reduction methods. ○ is original time series. × is reconstructed time series from representation coefficients. For a fair comparison, they have the same representation coefficients number (denoted as M) but not the same segment number (denoted as N).

same number of representation coefficients $M = 12$ but not the same number of segments (denoted as N). *SAPLA* and *APLA* use fewer segments ($N = 4$) to get better the sum of the maximum deviations than *APCA* and *PLA*.

APCA proposes two lower bounding distance measures that make adaptive-length dimensionality reduction methods indexable. One keeps a lower bounding lemma, called $Dist_{LB}$, and another has a tight Euclidean distance approximation but no lower bound, called $Dist_{AE}$. We propose $Dist_{PAR}$, which has a guaranteed lower bounding lemma and tightness. *APCA* also proposes a *MBR* for adaptive-length representation coefficients.

R-tree [11] splits node by finding a minimum area waste and picks a branch with a minimum area increase. However, homogeneous time series datasets are usually from the same data sources and we find that *APCA MBR* [13] of homogeneous time series could cause overlap problems. We used the proposed $Dist_{PAR}$ in node splitting and branch picking algorithms and create an updated R-tree for efficient queries of time series data.

1.2 Contributions

Our contributions in this paper are summiarised below:

- We propose an adaptive-length dimensionality reduction method (*SAPLA*) in Section 4. *SAPLA* prunes redundant computation for dimensionality reduction time reduction ($O(n(N + \log n))$), which is much faster than *APLA* [17] ($O(Nn^2)$) about n times. Furthermore, our experiment shows that *SAPLA* sacrifices little max deviation.
- We propose an adaptive-length lower bounding distance measure $Dist_{PAR}$ in Section 5.1. It has a guaranteed lower bounding lemma for tightness.
- We implement the distance based node-splitting and branch picking algorithms in DBCH structure. This improvement solves the overlap problem in time series *APCA MBR* for adaptive-length representation coefficients.
- We evaluate the max deviation, pruning power, accuracy, dimensionality reduction time and k-NN time of *SAPLA* together with $Dist_{PAR}$ and the DBCH-tree , as compared to other classic dimensionality reduction methods, lower bonding distance measures and R-tree. We evaluate all time series datasets (117) in [8] (128) with equal-length time series.

2 RELATED WORK

This section provides a quick review of seven related dimensionality reduction methods, *PLA*, *APLA*, *APCA*, *PAA*, *CHEBY*, *PAALM*, and *SAX*. Let M denote the representation coefficient number. Table 1 shows the summaries of the seven dimensionality reduction methods and our proposed *SAPLA*. *SAPLA* uses the same representation coefficients as *APLA*, but the time complexity is almost $1/n$ of *APLA*. *SAPLA* only needs the $M/3$ segment number of *PAA*, *PAALM*, *CHEBY*, *SAX*, and $M/2$ segment number of *APCA* and *PLA*.

PLA [5] uses $\check{c}_t = a \times t + b, t \in [0, l)$ of the equal-length segment to reconstruct time series $C = c_0, \ldots, c_{n-1}$. Let l denote the segment length. Let c_t denote the point value of position t in time series C. a is a slope, and b is a y-intercept in a linear function [14, 18]. Their computation is shown in Eq. (1). *PLA* has $O(n)$ time complexity. However, an equal-length segment could not help improve the tightness of an individual segment.

$$a = \frac{12\sum_{t=0}^{l-1}(t - (n-1)/2)c_t}{l(l-1)(l+1)} \quad b = \frac{2\sum_{t=0}^{l-1}(2l - 1 - 3t)c_t}{l(l+1)} \tag{1}$$

APLA [17] has time complexity $O(Nn^2)$. It builds a max deviation matrix $\varpi[n, N]$, and $\varpi[m, t]$ is a max deviation of the best t-segment representation to points 0, ..., m. Once the best (t-1)-segment representation is known for each prefix of the points, the best t-segment representation for points 0, ..., m can be computed through $\varpi[m, t] = \min_{\alpha=t}^{m-1}(\varpi[\alpha, t-1] + \varepsilon_t)$. *APLA* has guaranteed error bounds in the dimensionality reduction process. However, *APLA* has $O(Nn^2)$ time complexity.

APCA [13] uses the average value of the adaptive-length to approximate the Haar wavelet transformation. *APCA* has a time complexity of $O(n \log n)$, whereas *APCA* focuses on improving the tightness of individual segments with adaptive length and constant value.

PAA [12] uses the average value of equal-length segment to approximate C. *PAA* has $O(n)$ time complexity, and *PAA* is a simple technique. However, its segment number is three times that of *SAPLA*, as Table 1 shows.

PAALM [21] applies *PAA* and Lagrangian Multipliers on C. *PAALM* has $O(n)$ time complexity. *PAALM* represents continuous data as a series of patterns, not focusing on max deviation reduction. Thus we will evaluate it in k-NN search for showing the importance of max deviation.

CHEBY [2] uses the Chebyshev polynomial coefficient che_i to approximate C. The authors declared that Chebyshev coefficients should be smaller than 25. Our evaluation in Section 6 shows that *CHEBY* falls into the "dimensionality curse" when $N > 25$. *CHEBY* has $O(Nn)$ time complexity.

SAX [20] first transforms C into *PAA* and then symbolizes *PAA* into a discrete string. *SAX* has $O(n)$ time complexity. *SAX* is a symbolic version of *PAA*. Thus, the reconstruction of *SAX* has lower reconstruction accuracy than *PAA* (symbol → number). We do not compare *SAX*'s max deviation in this paper.

One high compression ratio method [9] does not consider final dimensionality reduction with user-defined segment numbers N. It needs a user-defined max deviation and a maximum polynomial degree. We do not compare this approach because it has an undetermined segment number after reduction and our paper does not include a user-defined max deviation.

Table 1: Dimensionality Reduction Methods Comparison

Name	Time	Coeffici.	Seg. Num	Seg. Size	Dim.
$*SAPLA$	$O(n(N+\log n))$	a_i, b_i, r_i	$N = M/3$	Adaptive	x-axis
$APLA$	$O(Nn^2)$	a_i, b_i, r_i	$N = M/3$	Adaptive	x-axis
$APCA$	$O(n \log n)$	v_i, r_i	$N = M/2$	Adaptive	x-axis
PLA	$O(n)$	a_i, b_i	$N = M/2$	Equal	x-axis
PAA	$O(n)$	v_i	$N = M$	Equal	x-axis
$PAALM$	$O(n)$	v_i	$N = M$	Equal	x-axis
$CHEBY$	$O(Nn)$	che_i	$N = M$	Equal	x-axis
SAX	$O(n)$	alphabet	$N = M$	Equal	y-axis

Table 2: Summary of Notations

No.	Meanings	No.	Meanings
n	Time series length	N	Number of Segments
C	Original time series	$\hat{C}$	Representation of C
a_i, b_i	Slope, Y-intercept	r_i	Right endpoint of $\hat{c}_i$
$\varepsilon(C, \check{C})$	$\sum_{t=0}^{n-1} \lvert c_t - \check{c}_t \rvert$	ϵ_i	Max deviation of $\hat{c}_i$
β_i	Segment upper bound	M	Baseline coefficient number
β	Sum upper bound	$\check{C}$	Reconstructed time series from $\hat{C}$
l_i	Segment length	ω	Map $\langle key, value \rangle$

There are many lower bounding measures for equal-length methods, such as $Dist_{PLA}$ [5], $Dist_{SAX}$ [16] and $Dist_{CHEBY}$ [2]. However, lower bounding measure for adaptive-length methods is difficult. *APCA* [4] proposes $Dist_{AE}$ for a tight approximation but not always lower bound the Euclidean distance. *APCA* also proposes $Dist_{LB}$ for a less tight approximation but can guarantee lower bound the Euclidean distance.

When original time series are mapped into a multidimensional index structure, the efficiency of indexing depends on the precision of the representation in the reduced dimensionality space. DCRC-tree [22] provides a structure to replace *MBR* for covering the original time series with no dimensionality reduction.

3 PRELIMINARIES

We introduce the definitions of representation ($\hat{C}$), reconstructed time series ($\check{C}$), max deviation (ϵ) and segment upper bound (β). Table 2 lists notations used throughout this paper.

Definition 3.1. ***Time Series*** ***(C).*** *Time series is a sequence of values c_i, defined as $C = \{c_0, c_1, \ldots, c_{n-1}\}$, where n is time series length. Like grey circles ○ in Fig. 1.*

Definition 3.2. ***Representation*** ***($\hat{C}$).*** *$\hat{C}$ is a N-segment sequence as a representation of C. $\hat{C} = \{\hat{c_0}, \hat{c_1}, \ldots, \hat{c}_{N-1}\} (N \leq n)$. For an adaptive-length linear curve, $\hat{c}_i := \langle a_i, b_i, r_i \rangle$ represents the i^{th} segment.*

Definition 3.3. ***Reconstructed Time Series*** ***($\check{C}$).*** *$\check{C}$ is reconstructed from representation coefficients, defined as $\check{C} = \{\check{c}_0, \check{c}_1, \ldots, \check{c}_{n-1}\} = \{\check{C}_0, \check{C}_1, \ldots, \check{C}_{N-1}\}$. One reconstructed segment $\check{C}_i = \{\check{c}_{r_{i-1}+1}, \ldots, \check{c}_{r_i}\}$. Like blue crosses × in Fig. 1.*

Definition 3.4. ***Max Deviation*** ***(ϵ).*** *Max deviation [2] is the maximum absolute difference between original time series C and reconstructed time series $\check{C}$ from representation $\hat{C}$. For the segment representation $\hat{c}_i$, its segment max deviation is $\epsilon_i := \max_{t=r_{i-1}+1}^{r_i} \lvert c_t - \check{c}_t \rvert$.*

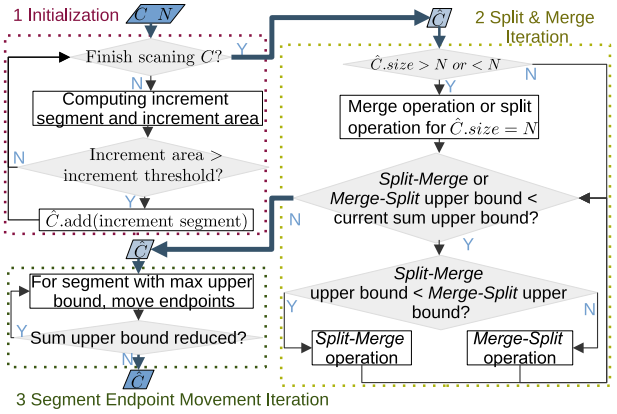


Figure 2: Framework of *SAPLA*: 1) Initializing C into $\hat{C}$. 2) Split & merge iteration reduces β of $\hat{C}$. 3) Segment endpoint movement iteration reduces β of $\hat{C}$.

Definition 3.5. ***Segment Upper Bound*** ***(β_i).*** *β_i is proposed to bound segment max deviation at different stages. β is the sum upper bound that $\beta = \sum_{i=0}^{N-1} \beta_i$. There are four stages to compute β_i in this paper. 1) Computing β_i when SAPLA initializes C into $\hat{C}$. 2) Computing β_i from a merge operation. 3) Computing β_i from a split operation. 4) Computing β_i from segment endpoints movements.*

4 SELF ADAPTIVE PIECEWISE LINEAR APPROXIMATION (*SAPLA*)

SAPLA focuses on finding segment endpoints to reduce the sum upper bound of segment max deviation. *SAPLA* consists of initialization, split & merge iteration, and segment endpoint movement iteration.

Fig. 2 shows the framework of *SAPLA*. Users specify a segment number of N during the initialization stage, and *SAPLA* converts time series C to $\hat{C}$. In the second stage, split & merge iteration reduces the sum upper bound by splitting a segment with the maximum upper bound into two segments and merging two adjacent segments with the minimum reconstruction area. Finally, the segment endpoint movement iteration reduces the sum upper bound. As a result, we will get *SAPLA* representation $\hat{C} = \{\langle a_0, b_0, r_0 \rangle, \ldots, \langle a_{N-1}, b_{N-1}, r_{N-1} \rangle\}$.

4.1 Proposed Equations and Area Computation

4.1.1 Increment Area. *SAPLA* represents the i^{th} segment by $\hat{c}_i = \langle a_i, b_i, r_i \rangle$. Let r_i' denote the next position of the right endpoint of $\hat{c}_i$ thus, $r_i' = r_i + 1$. The original point value is $c_{r_i'}$. Suppose a new segment is formed from the original time series as $C_i' = \{C_i, c_{r_i'}\}$. We called its *SAPLA* representation as Increment Segment representing as $\hat{c}_i' = \langle a_i', b_i', r_i' \rangle$. The increment segment length is $l_i' = l_i + 1$. The computation of a_i', b_i' by Eq. (1) is $O(l_i')$ time complexity. We extend Eq. (1) to Eq. (2), whose time complexity is $O(1)$.

$$
\begin{aligned}
a_i' &= \frac{(l_i - 2)(l_i - 1)a_i + 6(c_{r_i'} - b_i)}{(l_i + 1)(l_i + 2)} \\
b_i' &= \frac{2(l_i - 1)(a_i l_i - c_{r_i'}) + (l_i + 5) l_i b_i}{(l_i + 1)(l_i + 2)}
\end{aligned}
\tag{2}
$$

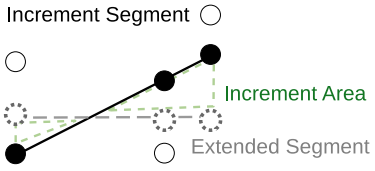


Figure 3: An example of Increment Area $\varepsilon(\check{C}'_i, \check{C}^e_i)$ is simplified as two green triangles △ . The black circle ○ is a point c_t in the original time series C. The grey dashed dot ◌ is the reconstructed point $\check{c}_t$ in $\check{C}^e_i$. The black dot ● is the reconstructed point $\check{c}'_t$ in $\check{C}'_i$. $t \in [0, n-1]$.

Let Increment Segment $\check{C}'_i$ denote the reconstructed segment time series from $\hat{c}'_i$. $\check{c}'_{r'_i} = a'_i * l_i + b'_i$ is the last point in $\check{C}'_i$, and $\check{c}_{r'_i} = a_i * l_i + b_i$ is the extended point from $\check{C}_i$. We could get $\check{C}^e_i = \{\check{C}_i, \check{c}_{r'_i}\}$, the extended segment of $\check{C}_i$, called Extended Segment. Fig. 3 presents an example of Extended Segment $\check{C}^e_i$ and Increment Segment $\check{C}'_i$. We find that $\check{C}^e_i$ and $\check{C}'_i$ always intersect. Thus, we can get Lemma 4.1.

Lemma 4.1. *Increment segment $\check{C}'_i$ and Extended segment $\check{C}^e_i$ have one intersection point. The proof is shown Section A.1.*

Because of Lemma 4.1, we can define the area between Increment Segment $\check{C}'_i$ and Extended Segment $\check{C}^e_i$ in Definition 4.1, called Increment Area. Let ε denote the summation of the absolute difference between two time series $\varepsilon(C, \check{C}) := \sum_{t=0}^{n-1} |c_t - \check{c}_t|$.

Definition 4.1. ***Increment Area*** *($\varepsilon(\check{C}'_i, \check{C}^e_i)$). $\varepsilon(\check{C}'_i, \check{C}^e_i)$ is an area between the Increment segment $\check{C}'_i$ and Extended segment $\check{C}^e_i$. $\varepsilon(\check{C}'_i, \check{C}^e_i)$ can be simplified as an area of 2 triangles. Fig. 3 presents an example of $\varepsilon(\check{C}'_i, \check{C}^e_i)$, shown as the two green triangles.*

4.1.2 (β_i) Segment Upper Bound in Initialization. Algorithm 4.1 introduces get_max() function for upper bound computation. Let [] denote the order of points in one segment, such as $\check{c}_{r_{i-1}+1} = \check{C}_i[1]$, $\check{c}_{r_i} = \check{C}_i[l_i]$. Let *max_d* denote the temp max value during the increment process. We will get upper bound like $\beta_i = \max($ get_max($[1, l_i, l'_i], C'_i, \check{C}'_i, \check{C}^e_i), max\_d) * l_i$. Theorem 4.1 proves we do not need to consider four point differences because they are always smaller than $|C'_i[l'_i] - \check{C}^e_i[l'_i]|$. Theorem 4.2 provides the conditions that make $\beta_i \geq \epsilon_i$.

Algorithm 4.1: get_max() denoted as max_d_i

```
input  : v := vector of id to compute;
         C_i, Q_i, T_i := segment time series;
output : Maximum absolute difference;
Function get_max(v, C_i, Q_i, T_i):
    m ← 0
    foreach k in v do
        m ← max(m, |C_i[k] − Q_i[k]|, |C_i[k] − T_i[k]|,
          |Q_i[k] − T_i[k]|)
    return m
```

Theorem 4.1. *Because Increment Segment $\check{C}'_i$ and Extended Segment $\check{C}^e_i$ have one intersection point (Lemma 4.1). Let $d_1 = |\check{C}'_i[1] - \check{C}^e_i[1]| = |b'_i - b_i|$, $d_2 = |\check{C}'_i[l_i] - \check{C}^e_i[l_i]| = |\check{c}'_{r_i} - \check{c}_{r_i}|$, $d_3 = |\check{C}'_i[l'_i] - C'_i[l'_i]| = |c_{r'_i} - \check{c}'_{r'_i}|$, $d_4 = |\check{C}'_i[l'_i] - \check{C}^e_i[l'_i]| = |\check{c}'_{r'_i} - \check{c}_{r'_i}|$.*

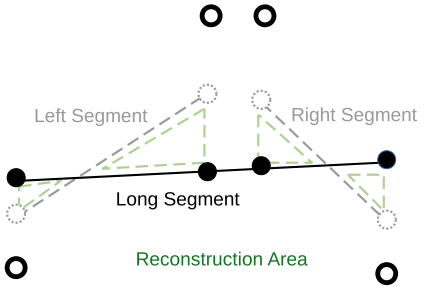


Figure 4: An example of Reconstruction Area $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ is simplified as four green triangles △. The black circle ○ is a point c_t in C. The grey dashed dot ◌ is the reconstructed point $\check{c}_t$ in $\check{C}_i + \check{C}_{i+1}$. The black dot ● is the reconstructed point $\check{c}'_t$ in $\check{C}'_{i+1}$.

$d_5 = |\check{C}^e_i[l'_i] - C'_i[l'_i]|$ So, we could get $d_4 \geq d_1$, $d_4 \geq d_2$ and $d_5 = d_3 + d_4$. Proof is shown in Section A.2.

Theorem 4.2 ($\beta_i \geq \epsilon_i$). *As Figure 17 shows when $d_5 = c_{r'_i} - \check{c}_{r'_i} \geq 0$, three factors support $\beta_i \geq \epsilon_i$: 1) $d_m = \check{c}'_{r'_i} - c_m = \check{c}'_{r'_i} - (c_{r'_i} + \check{c}''_{r'_i})/2 \leq 0$; 2) Section 4.1.2 proves $l''_i = 3 \Rightarrow get\_max([1, 2, 3], C'_i, \check{C}'_i, \check{C}^e_i) = \epsilon_i$; 3) In Eq. (21), $max\_d''_i \geq d_5$, $l''_i \in [3, n]$. Detail proof is shown in Section A.3. When $d_5 < 0$, the situation is similar.*

4.1.3 Reconstruction Area ($\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$). There are two adjacent segment representations, $\hat{c}_i$ and $\hat{c}_{i+1}$. Let C'_{i+1} denote the part of original time series covering C_i and C_{i+1} represented as $C'_{i+1} = \{C_i, C_{i+1}\}$. We could get $r'_{i+1} = r_{i+1}$, $l'_{i+1} = l_i + l_{i+1} = r'_{i+1} - r_{i-1}$. We propose Eqs.(3) and (4) that make the computation of a'_{i+1} and b'_{i+1} with $O(1)$ time complexity. We use $\check{C}'_{i+1}$ instead of $\check{C}_i, \check{C}_{i+1}$ in $\hat{C}$, called a merge operation. $\check{C}'_{i+1}$ and $\check{C}_i + \check{C}_{i+1}$ will form a reconstruction area ($\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ in Definition 4.2).

Definition 4.2. ***Reconstruction Area*** *($\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$). $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ is an area between $\check{C}_i + \check{C}_{i+1}$ and $\check{C}'_{i+1}$. $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ can be simplified as an area of several triangles or parallelograms. Fig. 4 provides an example of $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ as the four green triangles in Fig. 4.*

$$a'_{i+1} = \frac{a_i l_i (l_i - 1)(l_i + 1 - 3l_{i+1}) - 6 l_i l_{i+1} b_i}{l'_{i+1}(l'_{i+1} - 1)(l'_{i+1} + 1)} + \frac{a_{i+1} l_{i+1} (l_{i+1} - 1)(l_{i+1} + 1 + 3l_i) + 6 l_i l_{i+1} b_{i+1}}{l'_{i+1}(l'_{i+1} - 1)(l'_{i+1} + 1)} \tag{3}$$

$$b'_{i+1} = \frac{b_i l_i (l_i + 1) + 2 a_i l_{i+1} l_i (l_i - 1) + 4 l_i l_{i+1} b_i}{l'_{i+1}(l'_{i+1} + 1)} + \frac{b_{i+1} l_{i+1} (l_{i+1} + 1) - a_{i+1} l_i l_{i+1} (l_{i+1} - 1) - 2 l_i l_{i+1} b_{i+1}}{l'_{i+1}(l'_{i+1} + 1)} \tag{4}$$

4.1.4 (β_i) Segment Upper Bound in Merge Operation. A long segment $\check{C}'_{i+1}$ that is merged from $\check{C}_i$, $\check{C}_{i+1}$ by Eq. (3), (4). Let us denote $\check{C}_i + \check{C}_{i+1}$ as one reconstructed time series. $\check{C}'_{i+1}$ and $\check{C}_i + \check{C}_{i+1}$ have the same segment length. The upper bound in merging operations is defined as $\beta'_{i+1} =$ get_max($[1, l_i, l_i + 1, l'_{i+1}]$, $C'_{i+1}, \check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1}$) * $(l'_{i+1} - 1)$. In other words, we choose the max absolute point differences from endpoints in $\{C_i, C_{i+1}, \check{C}_i, \check{C}_{i+1}, \check{C}'_{i+1}\}$. There is a visual illustration of $\check{C}_i, \check{C}_{i+1}, \check{C}'_{i+1}$ in Fig. 4.

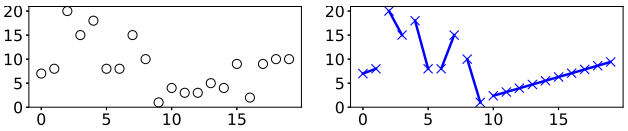


Figure 5: An example of the initialization result. Figure 5a is an original time series $\{7, 8, 20, 15, 18, 8, 8, 15, 10, 1, 4, 3, 3, 5, 4, 9, 2, 9, 10, 10\}$ **in Fig. 1a. Fig. 5b is the reconstructed time series from** *SAPLA* **representation coefficients after initialization.**

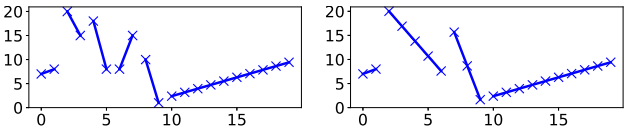


Figure 6: An example of the Split & Merge Iteration. Fig. 6a is the reconstructed time series after initialization. Fig. 6b is the reconstructed time series after Split & Merge Iteration on Fig. 6a. We can get user defined segment number $N = 4$ **by Split & Merge Iteration. The max deviation is** 10.6061**.**

4.2 Initialization

Fig. 5 shows an example of Initialization. The user defined representation coefficients number is $M = 12$. So the final segment number of *SAPLA* is $N = M/3 = 4$. The initialized *SAPLA* representation has 6 segments, $\{\langle 1, 7, 1\rangle, \langle -5, 20, 3\rangle, \langle -10, 18, 5\rangle, \langle 7, 8, 7\rangle, \langle -9, 10, 9\rangle, \langle 0.781818, 2.38182, 19\rangle\}$. Split & Merge Iteration in Section 4.3 will help get user defined four segment number. Initialization algorithm transfers original time series C into initialized representation $\hat{C}$. *SAPLA* scans original time series once to find the top N largest Increment Areas (refer to Definition 4.1) as $\hat{C}$ segment endpoints. Let $\varepsilon(\check{C}'_j, \check{C}^e_j)$ denote an increment threshold. Let $\max(\varepsilon(\check{C}'_j, \check{C}^e_j))_{N-1}$ denote the $(N-1)^{th}$ largest Increment Area. In general cases, we could get at least N segments after initialization. When scanning a new point $c_{r'_i}$, we will get increment area by Definition 4.1. We compare this increment area with the increment threshold. If the current increment area is bigger than the increment threshold, we will get a new threshold and a new segment. After initialization, *SAPLA* uses split & merge iteration in Section 4.3 to get exact N segments representation.

Algorithm 4.2 shows the process of initialization aiming to find segment endpoints by increment area $\varepsilon(\check{C}'_i, \check{C}^e_i)$. Computation of $\varepsilon(\check{C}'_i, \check{C}^e_i)$ needs a'_i and b'_i. Because of Eq. (2), Algorithm 4.2 scans original time series C once to get a'_i, b'_i $(i \in [0, n))$. If $\varepsilon(\check{C}'_i, \check{C}^e_i) > \max(\varepsilon(\check{C}'_j, \check{C}^e_j))_{N-1}$ $(i > j)$, the Increment Segment representation $\hat{c}'_i$ will be added to $\hat{C}$. $\langle(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1}), \hat{c}_{i+1}\rangle$ is stored in map ω^m from min to max. $\langle\beta_i, \hat{c}_i\rangle$ is stored in map ω^s from max to min.

4.3 Split & Merge Iteration

Fig. 6 shows an example of Split & Merge Iteration. $\hat{c}_i$ and $\hat{c}_{i+1}$ with $\min_{i=0}^{\hat{C}.size-2} \varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ are regarded as candidate merge segments. After initialization, original time series C is represented by initialized representation $\hat{C}$. Merge and split operations are applied to initialized representation $\hat{C}$ for sum upper bound β reduction. Segment upper bound β_i is proposed to bound segment maximum deviation (ϵ_i in Definition 3.4) with $O(1)$ time complexity. The maximum segment upper bound $\max_{i=0}^{\hat{C}.size-1} \beta_i$ is regarded as a split threshold, and sum upper bound β is regarded as an iteration threshold. Because a merge operation involves adjacent segments $\hat{c}_i$ and $\hat{c}_{i+1}$, the minimum reconstruction area $\min_{i=0}^{\hat{C}.size-2} \varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ is proposed as a merge threshold.

Several merge operations cannot guarantee a small max deviation ϵ. Therefore, we further apply split operations. Splitting could be regarded as a reverse operation of the merging operation in Section 4.1.3. In other words, we use two short segments $\check{C}_i + \check{C}_{i+1}$ to replace one long segment denoted as $\check{C}'_{i+1}$. We could get $l'_{i+1} = l_i + l_{i+1}$ and $r'_{i+1} = r_{i+1}$. *SAPLA* proposes the maximum segment upper bound $\max_{i=0}^{\hat{C}.size-1} \beta_i$ as the segment split

Algorithm 4.2: Initialization

input : $C : \{c_0, c_1, \ldots, c_{n-1}\}$;
N; // User defined segment number.
output: Initialized $\hat{C}$, $\hat{C}.size \in [1, \frac{n}{2}]$;
ω^m: Map storing $\hat{c}_{i+1}$ by $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ from min to max;
$\omega^m.top := \langle \min_{i=0}^{\hat{C}.size-2} \varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1}), \hat{c}_{i+1}\rangle$;
ω^s: Map storing $\hat{c}_i$ by β_i from max to min;
$\omega^s.top := \langle \max_{j=0}^{\hat{C}.size-1} \beta_j, \hat{c}_j\rangle$;
η: priority_queue. store $\varepsilon(\check{C}'_i, \check{C}^e_i)$;// $\eta.size \in [0, N)$;
$\eta.top :=$ min $\varepsilon(\check{C}'_i, \check{C}^e_i)$ in η;
$\hat{c}_i := \langle a_i, b_i, r_i\rangle \leftarrow \langle c_1 - c_0, c_0, 1\rangle$;
$l_i \leftarrow 2$;
$i \leftarrow 0$;
while $r_i < n$ **do**
  Compute $\hat{c}'_i$ and $\varepsilon(\check{C}'_i, \check{C}^e_i)$ from $\hat{c}_i$ by Eq. (2) and Definition 4.1;
  Compute β_i by Section 4.1.2;
  if $\eta.size < N - 1$ **then**
    $\eta.push(\varepsilon(\check{C}'_i, \check{C}^e_i))$;
    $\hat{C}$.insert$(\hat{c}_i)$;
    ω^s.add$\langle\beta_i, \hat{c}_i\rangle$;
    if $i > 0$ **then**
      Compute $\varepsilon(\check{C}'_i, \check{C}_{i-1} + \check{C}_i)$ by Eq. (3) (4) and Definition 4.2;
      Compute β'_i by Section 4.1.4;
      ω^m.add$\langle\varepsilon(\check{C}'_i, \check{C}_{i-1} + \check{C}_i), \hat{c}_i\rangle$;
    i++, r_i+ = 2, $l_i \leftarrow 2$;
  else if $\varepsilon(\check{C}'_i, \check{C}^e_i) > \eta.top$ **then**
    Update η by $\varepsilon(\check{C}'_i, \check{C}^e_i)$;
    $\hat{C}$.insert$(\hat{c}_i)$;
    ω^s.add$\langle\beta_i, \hat{c}_i\rangle$;
    ω^m.add$\langle\varepsilon(\check{C}'_i, \check{C}_{i-1} + \check{C}_i), \hat{c}_i\rangle$;
    i++, r_i+ = 2, $l_i \leftarrow 2$;
  else $\hat{c}_i \leftarrow \hat{c}'_i$;
  $r'_i = r_i + 1$;
  $l'_i = l_i + 1$;

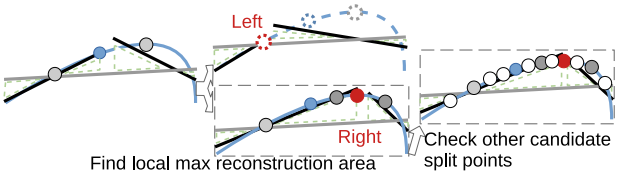


Figure 7: An example of finding a split point in $\check{C}'_{i+1}$. The blue dot ● is the middle point of C_i. The grey dot ● is the candidate split point in step 1), it is the middle point between endpoint and ● . 1) *SAPLA* gets split point ● with local max $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ and maximum magnitude (1). 2) *SAPLA* will check other candidate points ○ until has bigger $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ or all magnitudes of candidate split points are equal to ● magnitude.

threshold. *SAPLA* applies the peak finding technique [7] to find the split point in segment $\hat{c}_i$ with $\max_{i=0}^{\hat{C}.size-1} \beta_i$.

In this split & merge iteration, we update segment upper bound β_i in merge operations presented in Section 4.1.4 and in split operations presented in Section 4.3.1.

4.3.1 (β_i) Segment Upper Bound in Split Operation. The upper bound in the split operation could be regarded as the reverse operation of upper bound computation in merging operation (Section 4.1.4). We could get the left segment upper bound β_i = get_max([1,l_i], C_i, $\check{C}'_{i+1}$, $\check{C}_i$) * (l_i − 1) and the right segment upper bound β_{i+1} = get_max([1,l_{i+1}], C_{i+1}, $\check{C}'_{i+1}$, $\check{C}_{i+1}$) * (l_{i+1} − 1). Note that the order in segment for $\check{C}'_{i+1}$ should be transformed as $[1 - l_i, \ldots, l'_{i+1} - l_i]$. Thus, C_{i+1}, $\check{C}_{i+1}$ and $\check{C}'_{i+1}$ will have the same order value.

In merging operation, upper bound β'_{i+1} is defined in Section 4.1.4, and Fig. 4 shows an example of merging operation when left segment $\check{C}_i$ and right segment $\check{C}_{i+1}$ are merged into a long segment $\check{C}'_{i+1}$. In split operation, the upper bound β_i is defined in Section 4.3.1, and Fig. 4 also could be an example of a split operation where $\check{C}'_{i+1}$ is split into $\check{C}_i$ and $\check{C}_{i+1}$. Theorem 4.3 provides the conditions that make $\beta_i \geq \epsilon_i$.

Theorem 4.3. *In merging operation, suppose $\epsilon'_{i+1} = |c_t - \check{c}_t|$. We compute the sum absolute point difference except max deviation, denoted as $s = \sum_{j=r_{i-1}+1}^{r'_{i+1}} |c_j - \check{c}_j|$, $j \neq t$. We could get $\epsilon'_{i+1} \leq s$ because $\sum_{j=r_{i-1}+1}^{r'_{i+1}} (c_j - \check{c}_j) = 0$. The average point difference is $\frac{s}{l'_{i+1}-1}$. So, if $max\_d'_{i+1} \geq \frac{s}{l'_{i+1}-1}$, we will get $\beta_{i+1} \geq \epsilon'_{i+1}$. Detail proof is shown in Section A.4. In spitting operation, $\beta_i \geq \epsilon_i$ has same situation.*

4.3.2 Finding Split Point in segment $\check{C}'_{i+1}$. Fig. 7 provides an example of finding a split point in long segment $\check{C}'_{i+1}$. *SAPLA* regards the segment $\check{C}'_{i+1}$ with maximum segment upper bound $\max_{i=-1}^{\hat{C}.size-2} \beta'_{i+1}$ as candidate split segment. Split point is regarded as right endpoint r_i in left segment representation $\hat{c}_i$ after splitting long segment $\check{C}'_{i+1}$. *SAPLA* finds r_i with nearly maximum reconstruction area $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$ by peak finding technique [7]. We use Eq. (5) and (6) to compute representation coefficients a_i and b_i in $\hat{c}_i$ by a'_{i+1}, b'_{i+1}, a_{i+1}, and b_{i+1}. Eq. (7) and (8) help the computation of a_{i+1} and b_{i+1} in $\hat{c}_{i+1}$ in the same way.

$$a_i = a'_{i+1} \frac{l'_{i+1}(l'_{i+1}-1)(l'_{i+1}+1+3l_{i+1})}{l_i({l_i}^2-1)} - a_{i+1} \frac{l_{i+1}(l_{i+1}-1)(3l_i+4l_{i+1}+1)}{l_i({l_i}^2-1)} + \frac{6l_{i+1}l'_{i+1}(b'_{i+1}-b_{i+1})}{l_i({l_i}^2-1)} \quad (5)$$

$$b_i = b'_{i+1} \frac{l'_{i+1}(l'_{i+1}+1-4l_{i+1})}{l_i(l_i+1)} + b_{i+1}l_{i+1} \frac{2l'_{i+1}+l_{i+1}-1}{l_i(l_i+1)} + a_{i+1} * \frac{(l'_{i+1}+l_{i+1})l_{i+1}(l_{i+1}-1)}{l_i(l_i+1)} - a'_{i+1} \frac{2l_{i+1}l'_{i+1}(l'_{i+1}-1)}{l_i(l_i+1)} \quad (6)$$

$$a_{i+1} = a'_{i+1} \frac{l'_{i+1}(l'_{i+1}-1)(l'_{i+1}+1-3l_i)}{l_{i+1}({l_{i+1}}^2-1)} + a_i \frac{l_i(l_i-1)(2l'_{i+1}+l_{i+1}-1)}{l_{i+1}({l_{i+1}}^2-1)} + \frac{6l_il'_{i+1}(b_i-b'_{i+1})}{l_{i+1}({l_{i+1}}^2-1)} \quad (7)$$

$$b_{i+1} = a'_{i+1} \frac{l_il'_{i+1}(l'_{i+1}-1)}{l_{i+1}(l_{i+1}+1)} + b'_{i+1} \frac{l'_{i+1}(l'_{i+1}+1+2l_i)}{l_{i+1}(l_{i+1}+1)} - \frac{a_il_i(l_i-1)(l'_{i+1}+l_{i+1})}{l_{i+1}(l_{i+1}+1)} - \frac{b_il_i(3l'_{i+1}+l_{i+1}+1)}{l_{i+1}(l_{i+1}+1)} \quad (8)$$

Algorithm 4.3 presents the split & merge iteration process. We have got the minimum reconstruction area $\omega^m.top$ and the maximum reconstruction area $\omega^s.top$ in Algorithm 4.2. When the segment number of $\hat{C}$ is more than user defined segment number N, *SAPLA* applies a merge iteration for $\omega^m.top$ to reduce the segment number. Eq. (3), (4) help to reduce the computation time of representation coefficients. When the segment number of $\hat{C}$ is fewer than user defined segment number N, *SAPLA* applies split iteration for $\omega^s.top$ to increase the segment number of $\hat{C}$. Eq. (5), (6), (7), (8) help reduce the computation time of representation coefficients.

When them segment number of $\hat{C}$ is equal to user defined number N, *SAPLA* computes the segment upper bound β_j, β_{j+1} from $\omega^s.top$ after split operation and β_{i+1} from $\min\{\omega^m.top, \varepsilon(\check{C}'_j, \check{C}_{j-1} + \check{C}_j), \varepsilon(\check{C}'_{j+2}, \check{C}_{j+1} + \check{C}_{j+2})\}$ after merge operation. Thus, we will get a temp sum upper bound β^{sm} from the above split-merge computation. And *SAPLA* computes segment upper bound β_{y+1} from $\omega^m.top$ after merge operation and β_t, β_{t+1} from $\omega^s.top$ after split operation. Thus, we will get a temp sum upper bound β^{ms} from the above merge-split computation. Finally, if the current sum upper bound $\beta \leq \min\{\beta^{ms}, \beta^{sm}\}$, iteration will be transferred to the segment endpoint movement iteration (Section 4.3). If the current sum upper bound $\beta > \min\{\beta^{ms}, \beta^{sm}\}$, $\{\beta, \hat{C}, \omega^s, \omega^m\}$ will be updated by the above computation results and Algorithm 4.3 will continue the iteration.

4.4 Segment Endpoint Movement Iteration

Fig. 8 shows an example of Segment Endpoint Movement Iteration. In the segment endpoint movement iteration, *SAPLA* moves left and right endpoints of the segment $\check{C}_i$ with maximum segment upper bound $\max_{i=0}^{N-1} \beta_i$ for sum upper bound β reduction. Fig. 9 provides an example of one segment $\check{C}_i$ endpoints movement. There are four cases: 1) $\check{C}_i$ increases the right endpoint. 2) $\check{C}_i$ decreases the right endpoint. 3) $\check{C}_i$ increases the left endpoint. 4) $\check{C}_i$ decreases the left endpoint. *SAPLA* computes each movement's updated sum upper bound β of and finds the movement has the minimal sum upper bound β.

Algorithm 4.3: Split & Merge Iteration

input : $C : \{c_0, c_1, \ldots, c_{n-1}\}$;
$\hat{C}$, ω^m, and ω^s are from Algorithm 4.2;
output: $\hat{C} = \{\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{N-1}\}$ with reduced β;

while $\hat{C}.size > N$ **do**
  $\hat{c}'_{i+1} := \omega^m.top$;
  Update $\hat{C}$ by $\hat{c}'_{i+1}$;
  Update ω^s by β_{i+1};
  Update ω^m by $\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1})$; //Eq. (3) (4), Section 4.1.4; Definition 4.2.
while $\hat{C}.size < N$ **do**
  $\hat{c}'_{j+1} := \omega^s.top$;
  Update $\hat{C}$ by $\hat{c}_j, \hat{c}_{j+1}$;
  Update ω^s by β_j, β_{j+1} ;
  Update ω^m by $\varepsilon(\check{C}'_{j+1}, \check{C}_j + \check{C}_{j+1})$;//Section 4.3.2; Section 4.3.1.
$\beta^{sm} \leftarrow \beta^{ms} \leftarrow 0$;// all segments are labeled as unsplitted and unmerged
while $\beta \geq \min\{\beta^{sm}, \beta^{ms}\}$ **do**
  if $\omega^s.top$ *has been split* **then** get next segment.
  if $\omega^m.top$ *has been merged* **then** get next segment.
  Compute $\beta_j, \beta_{j+1}, \hat{c}_j, \hat{c}_{j+1}$ from $\omega^s.top$;
  Compute $\beta_{i+1}, \hat{c}'_{i+1}$ from $\min\{\omega^m.top, \varepsilon(\check{C}'_j, \check{C}_{j-1} + \check{C}_j), \varepsilon(\check{C}'_{j+2}, \check{C}_{j+1} + \check{C}_{j+2})\}$; // $O(1)$.
  Compute $\beta_{y+1}, \hat{c}'_{y+1}$ from $\omega^m.top$;//$O(1)$
  Compute $\beta_t, \beta_{t+1}, \hat{c}_t, \hat{c}_{t+1}$ from $\omega^s.top$;
  Compute β^{sm} by $\{\beta, \beta_j, \beta_{j+1}, \beta_{i+1}\}$;
  Compute β^{ms} by $\{\beta, \beta_t, \beta_{t+1}, \beta_{y+1}\}$;
  if $\beta^{sm} < \beta$ *or* $\beta^{ms} < \beta$ **then**
    if $\beta^{sm} < \beta^{ms}$ **then**
      Update $\hat{C}$ by $\hat{c}_j, \hat{c}_{j+1}, \hat{c}'_{i+1}$;
    else Update $\hat{C}$ by $\hat{c}_t, \hat{c}_{t+1}, \hat{c}'_{y+1}$;
    //label split or merged
    Update ω^s, ω^m;
    $\beta \leftarrow \min\{\beta^{sm}, \beta^{ms}\}$;

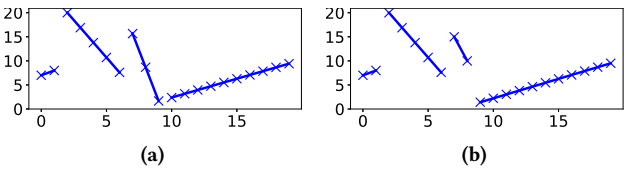


Figure 8: An example of the Segment Endpoint Movement Iteration. Fig. 8a is the reconstructed time series after Split & Merge Iteration. Fig. 8b is the reconstructed time series after Segment Endpoint Movement Iteration on Fig. 8a. The max deviation is 9.27273.

4.4.1 β_i Segment Upper Bound in Endpoint Movement. One case is that segment $\check{C}_i$ increases the right endpoint, we could get segment upper bound β_i as presented in Section 4.1.2. Because segment upper bound β_i computation for the other 3 cases are similar, we will not discuss them here. Eq. (2), (9), (10), (11) help to reduce the computation time of representation coefficients.

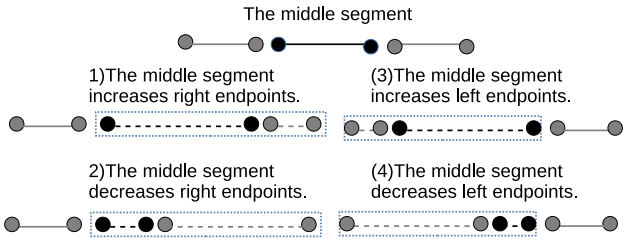


Figure 9: Example of segment endpoint movement for $\check{C}_i$. $\check{C}_i$ tries to move endpoints for β reduction. There are four cases. The grey dot ● is the endpoint in $\check{C}_{i-1}$, $\check{C}_{i+1}$. The black dot ● is the endpoint in $\check{C}_i$.

Segment $\check{C}_i$ with maximum segment upper bound $\max_{i=0}^{N-1} \beta_i$ increases and decreases its left and right endpoints during iteration. We could get four updated sum upper bound values (β), called β^a, β^b, β^c, β^d. Algorithm 4.5 shows how to compute two updated segment representations ($\hat{c}'_i$,$\hat{c}'_{i+1}$), two updated segment upper bounds (β_i, β_{i+1}), and an updated sum upper bound β.

1) β^a is from two segment upper bounds $\{\beta_i, \beta_{i+1}\}$ when one segment $\check{C}_i$ increases right endpoints and its right segment $\check{C}_{i+1}$ decreases left endpoints. 2) β^b is from $\{\beta_i, \beta_{i+1}\}$ when $\check{C}_i$ decreases right endpoints and $\check{C}_{i+1}$ increases left endpoints. 3) β^c is from $\{\beta_{i-1}, \beta_i\}$ when $\check{C}_i$ increases left endpoints and $\check{C}_{i-1}$ decreases right endpoints. 4) β^d is from $\{\beta_{i-1}, \beta_i\}$ when $\check{C}_i$ decreases left endpoints and $\check{C}_{i-1}$ increases right endpoints. Each endpoint increase or decrease movement will continue until the sum upper bound β cannot be reduced. *SAPLA* pops out the computed segment $\hat{c}_i$ and repeats the above process for sum upper bound β reduction.

Algorithm 4.4: Segment Endpoint Movement Iteration

input : $C : \{c_0, c_1, \ldots, c_{n-1}\}$;
$\hat{C}$ and β are from Algorithm 4.3;
output: $\hat{C} = \{\hat{c}_0, \hat{c}_1, \ldots, \hat{c}_{N-1}\}$ with reduced β;

$\eta :=$ priority_queue;
$\eta.top := \hat{c}_i$ with $\max_{i=0}^{N-1} \beta_i$;
$\beta^a \leftarrow \beta^b \leftarrow \beta^c \leftarrow \beta^d \leftarrow \beta$;
while $\beta \geq \min\{\beta^a, \beta^b, \beta^c, \beta^d\}$ *and* $\eta! = \emptyset$ **do**
  $\beta^a \leftarrow$ increase_right(β, $\hat{c}_i$,$\hat{c}_{i+1}$); //Algorithm 4.5
  $\beta^b \leftarrow$ decrease_right(β, $\hat{c}_i$,$\hat{c}_{i+1}$); //Algorithm 4.5
  $\beta^c \leftarrow$ decrease_right(β, $\hat{c}_{i-1}$,$\hat{c}_i$);
  $\beta^d \leftarrow$ increase_right(β, $\hat{c}_{i-1}$,$\hat{c}_i$);
  if $\beta > \min\{\beta^a, \beta^b, \beta^c, \beta^d\}$ **then**
    if $\min(\beta^a,\beta^b) < \min(\beta^c,\beta^d)$ **then**
      $\hat{c}_i \leftarrow \hat{c}'_i$;
      $\hat{c}_{i+1} \leftarrow \hat{c}'_{i+1}$;
    else
      $\hat{c}_{i-1} \leftarrow \hat{c}'_{i-1}$;
      $\hat{c}_i \leftarrow \hat{c}'_i$;
    $\beta \leftarrow \min\{\beta^a, \beta^b, \beta^c, \beta^d\}$;
    η.pop;

Algorithm 4.5: Coefficients for Endpoint Movement

$\beta' \leftarrow \beta$;
Function increase_right(β, $\hat{c}_i$, $\hat{c}_{i+1}$):
  while $\beta' \leq \beta$ *and* $l'_{i+1} \geq 2$ **do**
    $\beta \leftarrow \beta'$; $r'_i \leftarrow r_i + 1$;
    Computes $\hat{c}'_i$ by Eq. (2); //$O(1)$
    Computes $\hat{c}'_{i+1}$ by Eq. (11); //$O(1)$
    Computes β_i, β_{i+1} by Section 4.4.1;//$O(1)$;
    Updates β' by β_i, β_{i+1} ;
  return β;
decrease_right() is similar, it uses Eq. (9) (10)

$$a'_i = \frac{(l_i+4)a_i}{l_i-2} + \frac{6(b_i - c_{r_i})}{(l_i-1)(l_i-2)} \qquad b'_i = \frac{(l_i-3)b_i}{l_i-1} - 2a_i + \frac{2c_{r_i}}{l_i-1} \tag{9}$$

$$a'_i = \frac{a_i(l_i-1)(l_i+4) + 6(b_i - c_{r_{i-1}})}{(l_i+1)(l_i+2)} \qquad b'_i = \frac{2(2l_i+1)c_{r_{i-1}} + l_i(l_i-1)(b_i-a_i)}{(l_i+1)(l_i+2)} \tag{10}$$

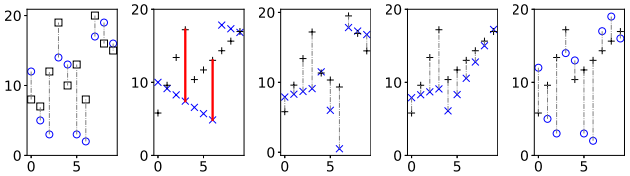
$$a'_i = a_i + \frac{6(c_{r_{i-1}+1} - b_i)}{(l_i-1)(l_i-2)} \qquad b'_i = a_i + \frac{(l_i+3)b_i - 4c_{r_{i-1}+1}}{l_i-1} \tag{11}$$

4.5 Time Complexity Analysis

The worst time complexity of *SAPLA* is $O(n(N + \log n))$. The iteration threshold β decides iteration time in *SAPLA*. For the initialization, the worst time complexity is $O(n \log N)$. When the first segment $\hat{c} = \langle c_0, c_1 - c_0, 1\rangle$ is constructed ($l = 2$), *SAPLA* will apply $\hat{C}$.insert($\hat{c}$) and update the increment threshold $\max(\varepsilon(\check{C}', \check{C}^e))_{N-1}$. The increment threshold $\max(\varepsilon(\check{C}', \check{C}^e))_{N-1}$ will be updated $\frac{n}{2}$ times and cost $O(\log N)$ during each updating.

For the split & merge iteration, when $\hat{C}.size > N$, the worst case is $\hat{C}.size = \frac{n}{2}$ with $\frac{n}{2} - N$ merge operation times. So the worst time complexity is $O(\sum_{i=0}^{\frac{n}{2}-N} 2\log(\frac{n}{2} - i)) = O(\sum_{i=N}^{\frac{n}{2}} \log i) \rightarrow O(n \log n)$. When $\hat{C}.size < N$, the worst time complexity is $O(Nn)$. In split iteration, the worst time complexity of finding split points is $O(n - 2\hat{C}.size)$. After one split operation, we need to sort segment upper bound β_i from big to small and reconstruction area (Definition 4.2) from small to big. Thus, the time complexity for sorting operation is $O(2\log \hat{C}.size)$. The worst case in split operations is when $\hat{C}.size = 1$, we need $N - 1$ loop times to get N segments. So, the time complexity is $O(n(N-1) - N(N+1) + 2\sum_{i=1}^{N} \log i) = O(Nn)$.

When $\hat{C}.size = N$, we need to sort $\max_{j=-1}^{N-1} \beta'_{j+1}$ for split operation and $\min_{i=0}^{N-1} (\varepsilon(\check{C}'_{i+1}, \check{C}_i + \check{C}_{i+1}))$ for merge operation, the time complexity is $O(2\log N)$. So, the time complexity in each while loop is $O(n - 2N + 2\log N)$. We do not apply both split operation and merge operation on the same segment in each while loop. A segment is only split one time in this iteration. We follow the same strategy for merge operations. Thus, the while loop times are $\in [1, N]$, and the worst time complexity is $O(N(n-2N)+2N\log(N)) = O(Nn)$. The whole time complexity is $O(n(N + \log n))$.



(a) Euclidean distance = 17 **(b)** Reduce to 2 dimensions. **(c)** $Dist_{PAR}$ = 14. **(d)** $Dist_{LB}$ = 11. **(e)** $Dist_{AE}$ = 20.

Figure 10: A visual comparison of lower bounding distance measures for adaptive-length representations. ◯ and □ are original time series points. × and + are reconstructed time series points from representation coefficients. Fig.10a is the Euclidean distance between two original time series. Fig.10b is two reconstructed time series from the *SAPLA* dimensionality reduction method ($N = 2$). The red line is the position to partition for Fig.10c.

For segment endpoint movements, the worst case is that each $\check{C}_i$ has the longest $l_i = n - 2N$ movements. Thus time complexity is $O(N(n-2N)) = O(Nn)$.

5 INDEXING USING A LOWER BOUNDING DISTANCE MEASURE

We propose a lower bounding measure for adaptive-length segment dimensionality reduction methods (*SAPLA*, *APLA* [17], *APCA* [13]), denoted as $Dist_{PAR}$. When we reduce two original time series into the lower dimensional *SAPLA* spaces, $Dist_{PAR}$ between them is a lower bound of the Euclidean distance between these two original time series. Lower bound lemma can guarantee no-false-dismissals in k-NN search. Tightness of lower bound distance measure could help to improve k-NN performance. We will prove the lower bounding lemma and the tightness of $Dist_{PAR}$ for adaptive-length methods. Because of the tightness of $Dist_{PAR}$, we propose an indexing structure, Distance Based Covering with Convex Hull (DBCH), that uses two representations with the maximum $Dist_{PAR}$ as the convex hull. In Section 5.3, we could use distance based node splitting and branching picking algorithms to build a DBCH-tree.

5.1 Lower Bound Distance Measure for Adaptive-Length Segment Dimensionality Reduction Method

$Dist_{PAR}$ can guarantee to be the lower bound and a tight approximation of the Euclidean distance. For two *SAPLA* segment representations $\hat{q}_i$ and $\hat{c}_i$, suppose they have the same right endpoint, and segment length denoted as l_i. Let $\check{q}_j$, $\check{c}_j$ denote the reconstructed point in $\hat{q}_i$, $\hat{c}_i$ by a linear function $a * j + b$. Their Euclidean distance square is shown in Eq. (12).

Fig. 11 shows an example of $Dist_{PAR}$. The Euclidean distance of two original time series is 17. Two original time series of length 10 are reduced to 2 dimensions by *SAPLA*. Fig. 10b are two reconstructed time series from *SAPLA* representation coefficients. $Dist_{PAR}$ partitioned these representations by Definition 5.1. $Dist_{PAR} = 14$ is a very tight approximation of the Euclidean distance. $Dist_{AE} = 20$ does not lower bound the Euclidean distance. $Dist_{LB} = 11$ is a less tight approximation of the Euclidean distance. We could find that $Dist_{LB} < Dist_{PAR} < Dist$, which

means $Dist_{PAR}$ is lower bound Euclidean distance and tighter than $Dist_{LB}$. $Dist_{AE}$ is bigger than Euclidean distance, which breaks the lower bounding lemma.

$$Dist_S(\hat{q}_i, \hat{c}_i) = \sum_{j=0}^{l_i-1} (\check{q}_j - \check{c}_j)^2 = \frac{l_i(l_i-1)(2l_i-1)}{6}(\hat{q}_{a_i} - \hat{c}_{a_i})^2 + l_i(l_i-1)(\hat{q}_{a_i} - \hat{c}_{a_i})(\hat{q}_{b_i} - \hat{c}_{b_i}) + l_i(\hat{q}_{b_i} - \hat{c}_{b_i})^2 \quad (12)$$

Definition 5.1. *($Dist_{PAR}$) There are two SAPLA representations $\hat{Q}$ and $\hat{C}$. Let $\hat{Q}_R$ denote all r_i in $\hat{Q}$. Let $\hat{C}_R$ denote all r_i in $\hat{C}$. We define $R = \hat{Q}_R \cup \hat{C}_R$. We will get $\hat{Q}_R \subseteq R$ and $\hat{C}_R \subseteq R$. The partition process is applied to these two represents so that they have the same segment endpoints. This is similar to the split operations in Section 4.3. We get the new shorter segments whose a, b can be computed by Eq. (5) (6) (7) (8). After the partition, the new representations $\hat{Q}^P$ and $\hat{C}^P$ have the same segment endpoints $\hat{Q}^P_R = \hat{C}^P_R$. Thus, $Dist_{PAR}(\hat{Q}^P, \hat{C}^P)$ is defined in Eq. (13).*

$$Dist_{PAR}(\hat{Q}^P, \hat{C}^P) = \sqrt{\sum_{i=0}^{R.size-1} Dist_S(\hat{q}^p_i, \hat{c}^p_i)} \quad (13)$$

The proof of $Dist_{PAR}$ being a lower bound on Euclidean distance is shown in A.5. The proof of $Dist_{PAR}$ is a tight approximation of the Euclidean distance is shown in A.6. We already know $Dist_{AE}$ has $O(n)$ time complexity, and $Dist_{LB}$ also needs $O(n)$ time complexity for "projecting" new endpoints [4]. Because of Eq. (5) (6) (7) (8), the worst time complexity of $Dist_{PAR}$ is smaller than $O(n)$.

5.2 Distance Based Covering with Convex Hull (DBCH structure)

APCA [13] proposes a *MBR* for adaptive-length representation coefficients that the distance between query time series Q and *MBR* is lower bounds the $Dist_{euc}(Q, C)$ for any C in *MBR*. However, we find that *MBR* would cause serious overlap problems as Fig. 11a shows how homogeneous time series overlap each other. The *MBR* based on homogeneous time series also overlaps each other. Our experiments in Section 6 show that building R-tree based on *MBR* will degrade the space efficiency of R-tree and k-NN pruning power. We have proved the proposed lower bounding measure for adaptive-length segment dimensionality reduction method $Dist_{PAR}$ is tighter than $Dist_{LB}$ and satisfies the lower bound lemma. Therefore, we propose a convex hull structure instead of *APCA MBR*. We use two representation coefficients that have the maximum $Dist_{PAR}$ in one node as the bound of the node. Figures 11b-11d show an example of construction of DBCH structure. Fig. 11b shows three reconstructed time series from three *SAPLA* representations which have different segment endpoints. The partition process is applied so that they have the same segment endpoints. We compute the $Dist_{PAR}$ between them as shown in Fig. 11c. We choose two *SAPLA* representations with maximum $Dist_{PAR}$ as the convex hull in Fig. 11d. We regard these two representations as the lower and upper bound of DBCH structure.

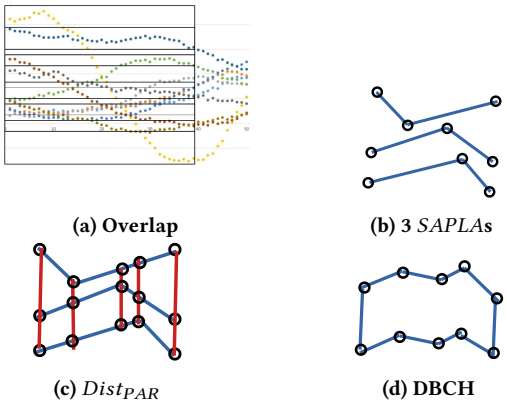


Figure 11: (a) shows how homogeneous time series *MBR*s overlap each other. (b) shows three *SAPLA*s. (c) computes $Dist_{PAR}$ between them. (d) DBCH structure chooses the two *SAPLA*s with maximum $Dist_{PAR}$ as the convex hull.

5.3 Distance Based Node Splitting and Branch Picking.

Node splitting in R-tree attempts to find a small-area split. The branch picking algorithm also picks a branch with a minimum area increase. We propose improved node splitting and branch picking algorithms for time series. We split node and pick branch using lower bounding distance instead of *MBR* area. We can avoid serious overlap problems in node splitting and branch picking process.

The lower bounding distance measure has been applied to improve the node splitting in the R-tree, not the waste area. We will not show the detailed algorithms here because they are similar to the algorithms in R-tree [11]. When one node needs to split, we first choose the pair with the maximum lower bounding distance and denote them as the seed. $seed_1$ and $seed_2$. Then, we compute the lower bounding distance between the rest of the entries and these two seeds. If they are close to $seed_1$, we put them in $node_1$. Otherwise, we put them in $node_2()$. The branch picking algorithm does similar work by choosing the branch with the minimum distance increase. We compute the distance between all entries in the leaf node. We only compute the pair's distance that constructs the convex hull in each subnode for the internal node.

Let u and l denote the upper and lower bound of the DBCH structure in k-NN search. We call the $Dist_{PAR}(u, l)$ as volume. Let q denote the query representation. If the distances from q to u and l are smaller than volume, we set the $Dist(q, DBCH) = 0$. If the distances from q to u or l are bigger than volume, we choose the smaller distance as $Dist(q, DBCH)$.

6 EXPERIMENTAL EVALUATION

Implementation: We have implemented *SAPLA*, *APLA*, *APCA*, *PLA*, *PAA*, *CHEBY*, *PAALM*, *SAX*, R-tree, and DBCH-tree by C++ [1]. The summary of dimensionality reduction methods is shown in Table 1. Note that we focus on adaptive-length segment dimensionality reduction methods. Our proposed DBCH-tree is proposed to solve the overlap problem of *APCA MBR*. *PAA*, *PAALM*, *SAX*, *SAPLA*, and *APLA* use *MBR* in R-tree because they are special transformation of *APCA*. *PLA* use its own *MBR* computation method because *PLA* proposes a robust distance

measure between query time series and *PLA MBR*. *CHEBY* coefficient computation is different from *APCA*. We could find that *PLA* and *CHEBY* have similar performance in R-tree and DBCH-tree. However, DBCH-tree shows a big improvement than R-tree with *APCA MBR*. It is important for index building of adaptive-length segment dimensionality reduction methods.

Hardware: The processor is Intel(R) Core(TM) i5-7600 CPU @ 3.50 GHz. RAM is 8 GB. We use VS2019 in Windows 10 system.

Datasets: There are 128 different datasets in UCR2018 [8]. We have evaluated all the datasets with equal length time series (117 datasets). We set the time series length as 1024, and the time series number is 100. We randomly test five query time series on each dataset and summarise the experiment results. Due to the space limitation, the detailed comparisons of each parameter in each dataset are shown in our technical report [1].

Parameters: We evaluate max deviation (ϵ), pruning power (ρ), accuracy, dimensionality reduction time and k-NN (CPU) time. We do not measure wall clock time because our index structure is memory based. The database may be memory based in the future. Thus, we measure the performance of methods by CPU time in addition to the number of disk access (pruning power). We also evaluate ingest data time, space efficiency of R-tree and our proposed DBCH-tree. The parameter is $M = \{12, 18, 24\}$, $K = \{4, 8, 16, 32, 64\}$. The maximum number of entries in one node is 5, and the minimum number of entries are 2.

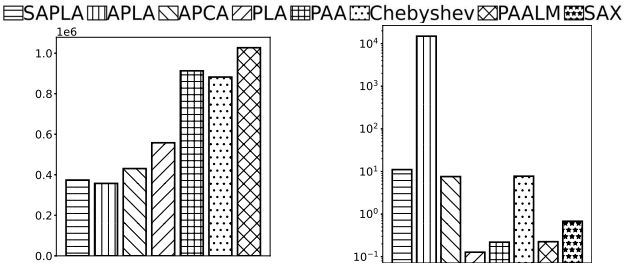
We evaluate the indexing performance by testing pruning power (ρ) and accuracy. ρ could be got by Eq.(14), which can avoid implementation bias. Accuracy is given by Eq.(15), which can evaluate false positive k-NN results.

$$\rho = \frac{the\ number\ of\ time\ series\ which\ have\ to\ be\ measured}{all\ time\ series\ number} \tag{14}$$

$$Accuracy = size\ of\ true\ nearest\ neighbor \div K \tag{15}$$

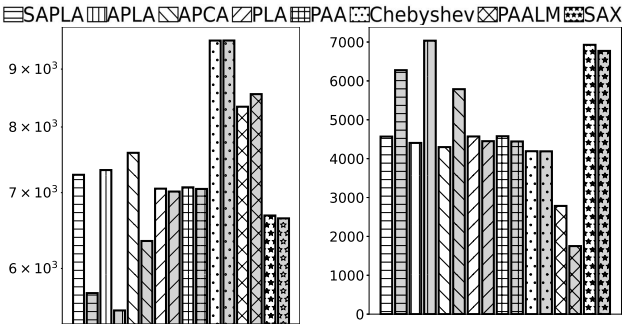
Max Deviation (ϵ) and Dimensionality Reduction Time. Max deviation in Fig. 12a is used to evaluate the tightness between the original time series and representation coefficients. *SAX* is not compared here because *SAX* is a symbolic version of *PAA*. The reconstructed time series of *SAX* has lower reconstruction accuracy than the *PAA* (symbol → number). We directly compare *PAA* instead of *SAX*. Dimensionality reduction time in Fig. 12a is consistent with the time complexity in Table 1. Adaptive-length methods *SAPLA*, *APLA* and *APCA*, have better max deviation than equal-length methods with fewer segment numbers N when their representation coefficients number M is same. Adaptive-length methods need more dimensionality reduction time but help to improve pruning power and accuracy. *APLA* has the best max deviation but the worst dimensionality reduction time. This is because *APLA* has the best pruning power and accuracy in Fig. 13. Our proposed *SAPLA* uses $1/n$ dimensionality reduction time of *APLA* to get similar pruning power in Fig. 13a, accuracy in Fig. 13b.

Pruning Power (ρ) and Accuracy. Fig. 13a shows the adaptive-length methods *SAPLA*, *APLA* and *APCA* get a huge improvement in pruning power and accuracy. This proves that our proposed DBCH-tree overcomes the overlap problem in *APCA MBR*. *PLA* and *CHEBY* use their own *MBR* that their pruning power and accuracy are similar in R-tree and DBCH-tree. *PAA* and *SAX* are equal-length segment methods that their pruning power and accuracy are similar in different index structures. Because *PAALM* has the worst max deviation, its accuracy becomes worse



(a) Max Deviation **(b) Dimensionality Reduction Time (s)**

Figure 12: Summary comparison on 117 datasets.



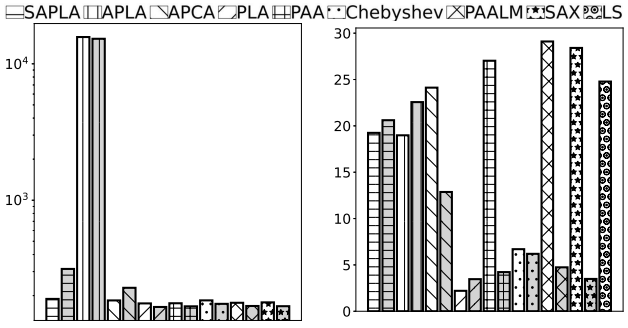
(a) Pruning power **(b) Accuracy**

Figure 13: Summary comparison on 117 datasets. The □ is original R-tree. The ■ is DBCH-tree.

in DBCH-tree. This shows DBCH-tree depends on accurate distance measures.

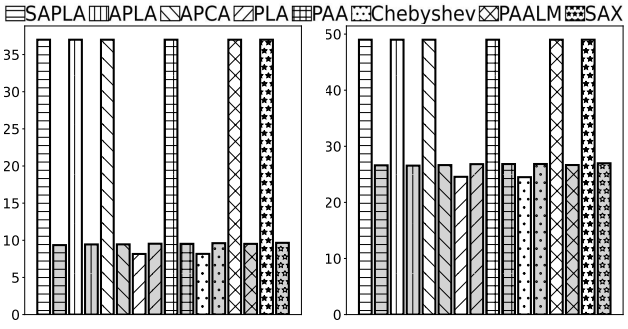
Data Ingest Time and k-NN CPU Time. Fig. 14a shows that *APLA* needs more data ingest time than other methods. This shows the importance of the *SAPLA*, which is much faster than *APLA*. Adaptive-length methods could use fewer segments to get better pruning power and accuracy in Fig. 13. Fig. 14b shows the k-NN CPU time. The last bar is the linear scan method. We record the Euclidean distance computation time of the linear scan. We could find *SAPLA* and *APLA* need a little more k-NN time in DBCH-tree. Because *SAPLA* and *APLA* have the best pruning power and accuracy by our proposed $Dist_{PAR}$. $Dist_{PAR}$ in *SAPLA* and *APLA* shows tightness. Equal-length methods have similar pruning power and accuracy, but Fig. 14b shows they use less k-NN in DBCH-tree.

Comparison on Index Size and Space Efficiency Analysis The number of time series in each tree is 100. We set the minimum fill as 2, and the maximum fill is 5. The height of the tree could be between 3 and 7. Fig. 16b shows the average tree height of one R-tree is 5, and one DBCH-tree is 4. The maximum of the total node number is 102. Fig. 15a shows the average internal node number in one tree, and Fig. 15b shows the leaf node number. We could find that the leaf node in DBCH-tree contains 4 entries on average, and the R-tree is 2. The number of internal nodes in the R-tree is about four times that of DBCH-tree. Our experiment in Fig. 15, 16 show the superiority of the DBCH-tree. Because *PLA* and *CHEBY* do not apply *APCA MBR*, *PLA* and *CHEBY* have a minor difference between R-tree and DBCH-tree. We can conclude that DBCH-tree makes adaptive-length representations have similar space efficiency with equal-length representations.



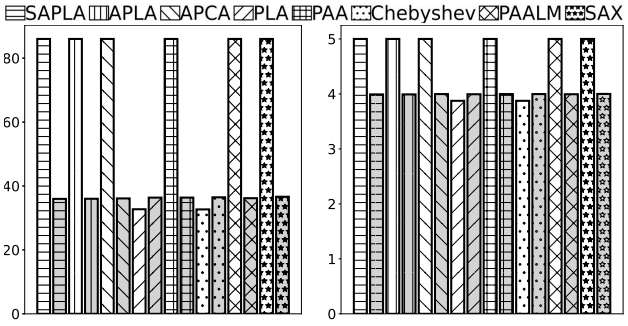
(a) Ingest Data Time (s) (b) k-NN CPU Time (s)

Figure 14: The □ is original R-tree. The ■ is DBCH-tree. The last bar in Fig. 14b is Linear Scan.



(a) Average Internal Node Number (b) Average Leaf Node Number

Figure 15: The □ is original R-tree. The ■ is DBCH-tree. The average number of internal node and leaf node in one tree.



(a) Average Total Node Number (b) Average Tree Height

Figure 16: The □ is original R-tree. The ■ is DBCH-tree. The average total node number and height in one tree.

7 CONCLUSION

Similarity search over the original time series may fall into the "dimensionality curse". We propose an adaptive-length segment dimensionality reduction method, *SAPLA*, to improve *APLA*. *SAPLA* uses linear adaptive-length segments to represent original time series for small max deviation with fewer segments. As our experiments show, *SAPLA* is faster than *APLA* about n times with minor max deviation loss. The limitation of *SAPLA* is that it applies conditional upper bounds, not unconditional upper bounds for max deviation reduction.

APCA proposes two lower bounding distance measures on adaptive-length dimensionality reduction methods. $Dist_{LB}$ keeps lower bounding lemma, $Dist_{AE}$ has tight Euclidean distance approximation but non-lower bounding. Thus, we propose $Dist_{PAR}$ with guaranteed lower bounding lemma and tightness.

APCA MBR of homogeneous time series could cause overlap problems in R-tree. We split the node and pick branch by the proposed lower bounding distance instead of waste area to implement a DBCH-tree. DBCH-tree helps to improve the space efficiency and pruning power of adaptive-length segment representation coefficients. DBCH-tree depends on the tightness of distance measurements. DBCH-tree needs more time to build an index than R-tree for adaptive-length segment dimensionality reduction methods. DBCH-tree could guarantee the lower bounding lemma between the query time series and the leaf node. However, the distance between query time series and an internal node cannot guarantee the lower bounding lemma.

A PROOFS OF THEOREMS & LEMMAS

A.1 Proof of Lemma 4.1

Proof. We could get the difference value between the last reconstructed point in Increment Segment, and the last extended point in Extended Segment is shown in Eq.16.

$$d_4 = \check{c}'_{r'_i} - \check{c}_{r'_i} = a'_i l_i + b'_i - (a_i l_i + b_i) \Rightarrow \frac{2(2l_i+1)(c_{r'_i} - \check{c}_{r'_i})}{(l_i+1)(l_i+2)} \tag{16}$$

We also get the difference value between the first reconstructed point in Increment Segment, and the first extended point in Extended Segment is shown in Eq.17.

$$d_1 = b'_i - b_i \Rightarrow \frac{(l_i-1)(\check{c}_{r'_i} - c_{r'_i})}{(l_i+1)(l_i+2)} \tag{17}$$

We could get $d_4 * d_1 \leq 0$. Thus, the Increment Segment $\check{C}'_i$ and the Extended Segment $\check{C}^e_i$ have one intersection point unless they are same. □

A.2 Proof of Theorem 4.1

Proof. We define the segment length is longer than one in this paper ($l_i > 1$). As Fig. 3 shows, Eq. (18) shows $d_4 \geq d_1$. Eq. (19) shows $d_4 \geq d_2$. Eq. (20) shows $d_5 = d_3 + d_4$.

$$d_4 \geq d_1 \Rightarrow |a'_i l_i + b'_i - (a_i l_i + b_i)| \geq |b'_i - b_i| \Rightarrow \frac{2(2l_i+1)|c_{r'_i} - \check{c}_{r'_i}|}{(l_i+1)(l_i+2)} \geq \frac{(l_i-1)|c_{r'_i} - \check{c}_{r'_i}|}{(l_i+1)(l_i+2)} \tag{18}$$

$$d_4 \geq d_2 \Rightarrow 2(2l_i+1) > 4(l_i-1) \tag{19}$$

$$d_3 + d_4 \Rightarrow \frac{l_i(l_i-1) + 2(2l_i+1)}{(l_i+1)(l_i+2)} |c_{r'_i} - \check{c}_{r'_i}| = d_5 \tag{20}$$

□

A.3 Proof of Theorem 4.2: In general case, $\beta_i \geq \epsilon_i$ in initialization operation

Proof. As Fig. 17 shows, $\check{C}''_i$ is an initialized segment. When $\epsilon_i = c_{r'_i} - \check{c}''_{r'_i}$, we will prove $\beta_i = max\_d''_i * (l''_i - 1) \geq \epsilon_i$ in

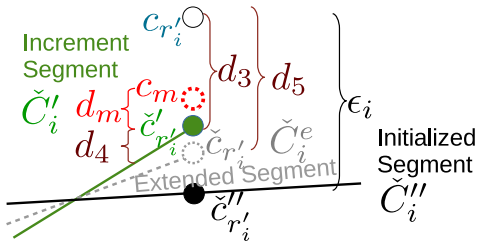


Figure 17: Example of proof $\beta_i \geq \epsilon_i$ in general cases, suppose $\epsilon_i = c_{r'_i} - \check{c}''_{r'_i}$. The black circle ○ is original point c_t in C,$t \in [0, n-1]$. The grey dashed dot ◌ is reconstructed point $\check{c}_t$ in $\check{C}^e_i$. The green dot ● is reconstructed point $\check{c}'_t$ in $\check{C}'_i$. The black dot ● is reconstructed point $\check{c}''_t$ in $\check{C}''_i$. $c_m = \frac{c_{r'_i} + \check{c}''_{r'_i}}{2}$ is middle point between original point $c_{r'_i}$ and reconstructed point $\check{c}''_{r'_i}$.

general cases. Since $l_i \geq 2$, $l''_i - 3 \geq 0$. When $d_5 = c_{r'_i} - \check{c}_{r'_i} \geq 0$, $d_3 = c_{r'_i} - \check{c}'_{r'_i} \geq 0$ in Theorem 4.1, we can prove

$$\begin{aligned} &\beta_i \geq (l''_i - 1)d_3 \geq \epsilon_i \Rightarrow (l''_i - 2)d_3 + \check{c}''_{r'_i} - \check{c}'_{r'_i} \geq 0 \\ &\Rightarrow \frac{(l''_i - 3)d_3}{2} \geq \check{c}'_{r'_i} - \frac{c_{r'_i} + \check{c}''_{r'_i}}{2} \Rightarrow \frac{(l''_i - 3)d_5}{2} \geq d_m \end{aligned} \tag{21}$$

One special case $l''_i = 4$, $\frac{(l''_i - 3)d_5}{2} \geq d_m \Rightarrow d_5 \geq \frac{\epsilon_i}{2}$. If $max\_d''_i = d_5$ and $d_5 < \frac{\epsilon_i}{2}$, we will get $\beta_i < \epsilon_i$. Another special case is $d_m > 0$, so $\frac{(l''_i - 3)d_5}{2}$ may be smaller than d_m, especially when $max\_d''_i = d_5$. During our experiment, we have not found these two special cases that cause $\beta_i < \epsilon_i$. For $d_5 < 0$, the proof is similar to Eq. (21). We will not discuss this in details. □

A.4 Proof of Theorem 4.3: In general case, $\beta_i \geq \epsilon_i$ in merge operation

Lemma A.1. *Let C_i denote the i^{th} segment of original time series C. Let $\check{C}_i$ denote the i^{th} segment of reconstructed time series $\check{C}$ from SAPLA representation $\hat{C}$. Let $s_p = \{\sum_{j=r_{i-1}+1}^{r_i}(c_j - \check{c}_j), if\ c_j - \check{c}_j > 0\}$ and $s_n = \{\sum_{j=r_{i-1}+1}^{r_i}(c_j - \check{c}_j), if\ c_j - \check{c}_j < 0\}$. $s_p + s_n = 0$, as Eq.(22) shows.*

$$\sum_{j=r_{i-1}+1}^{r_i}(c_j - \check{c}_j) = \frac{l_i(l_i - 1)}{2}a_i + l_i b_i - \sum_{j=r_{i-1}+1}^{r_i} c_j = 0 \tag{22}$$

Proof. $\beta_{i+1} = max\_d'_{i+1} * (l'_{i+1} - 1)$. We suppose $\epsilon'_{i+1} = c_{r_i} - \check{c}'_{r_i}$ in Fig. 4. According to Lemma A.1, ϵ'_{i+1} also has maximum value. That is when only c_{r_i} is located above $\check{C}_i$, other points are all below $\check{C}_i$ and average value is $\frac{c_{r_i} - \check{c}'_{r_i}}{l'_{i+1} - 1}$. So, the worst case is $max\_d'_{i+1} < \frac{c_{r_i} - \check{c}'_{r_i}}{l'_{i+1} - 1}$. During our experiment, we have not found this extreme case. For proof β_i in split operation is similar with β_{i+1}. We will not discuss in detail. □

A.5 Lower Bounding Lemma for $Dist_{PAR}$

Let Q and C denote two original time series. Let $\hat{Q}$ and $\hat{C}$ denote the *SAPLA*. Let $\hat{Q}^p$ and $\hat{C}^p$ denote the partitioned *SAPLA* of $\hat{Q}$ and $\hat{C}$. In order to guarantee no false dismissal, $Dist_{PAR}(\hat{Q}, \hat{C})$ between two partitioned *SAPLA* representations, $\hat{Q}^p$ and $\hat{C}^p$, should be smaller than the Euclidean distance $Dist(Q, C)$ between two time series Q and C. Let N' denote the partitioned dimension.

$Dist(Q, C)$ and $Dist_{PAR}(\hat{Q}, \hat{C})$ are the summations of the distance of all segments. $\hat{Q}^p$ and $\hat{C}^p$ have the same right endpoints. Thus, it is sufficient to prove one segment that the lower bound distance from the partitioned *SAPLA* is smaller than or equal to the Euclidean distance of the same segments. For the first segment $\hat{q}_0 = \langle \hat{q}_a, \hat{q}_b, \hat{q}_r \rangle$,$\hat{c}_0 = \langle \hat{c}_a, \hat{c}_b, \hat{c}_r \rangle$, their right endpoints are equal ($\hat{q}_r = \hat{c}_r$). So, the proof of $Dist(Q_0, C_0) \geq Dist_{PAR}(\hat{Q}_0, \hat{C}_0)$ is the same as $Dist(Q_0, C_0) \geq Dist_{PLA}(\hat{Q}_0, \hat{C}_0)$ (proved in [5]). Because the entire distance is the summation of all segments, we can guarantee that our proposed lower bound distance measure is smaller than Euclidean distance $Dist_{PAR}(\hat{Q}, \hat{C}) \leq Dist(Q, C)$.

A.6 Proof of $Dist_{PAR}$ Tightness

Let $\hat{Q}^p$ and $\hat{C}^p$ denote the partitioned *SAPLA* representations of $\hat{Q}$ and $\hat{C}$. In order to prove the tightness of $Dist_{PAR}$, we need to guarantee $Dist_{LB} \leq Dist_{PAR}$. We have proved the $Dist_{PAR}(\hat{Q}, \hat{C}) \leq Dist(Q, C)$ in Section A.5.

$\hat{Q}^p$ and $\hat{C}^p$ have the same right endpoints r_i. Let $\hat{C}^p_R$ denote all the partitioned right endpoints in $\hat{C}^p$. Let $\hat{Q}_R$ denote all r_i in $\hat{Q}$. Let $\hat{C}_R$ denote all r_i in $\hat{C}$. $Dist_{LB}$ [13] converts Q into a *SAPLA* representation $\hat{Q}^{LB}$ with the exact r_i as $\hat{C}$. It is obvious that $\hat{Q}^{LB}_R = \hat{C}_R \subseteq \hat{C}^p_R$. Thus, we can conclude that any segment $\hat{c}_i$ in $\hat{C}$ could be computed by one segment or merged (Eq. (3) (4)) from several segments in $\hat{C}^p$. Meanwhile, let $l_i = r_i - r_{i-1}$ denote the segment length. $\exists m \in [0, k]$ satisfies $\sum_{j=m}^{k} \hat{C}^p_{l_j} = \hat{C}_{l_i}$. It is sufficient to prove one segment that $Dist_{LB}(\hat{q}_i, \hat{c}_i)$ is smaller than or equal to the $Dist_{PAR}$ of one segment or several segments. If the segment $\hat{c}_i$ has the same right endpoint with $\hat{c}^p_k$ and their segment length are equal, their distance will be equal. If more than 2 segments in $\hat{C}^p$ that their summation of segments length are equal to segment length in $\hat{c}_i$, they can be merged into 2 segments $\hat{c}^p_{k-1}$ and $\hat{c}^p_k$. The above conclusions can also work on $\hat{Q}^p$ and $\hat{Q}^{LB}$.

We will prove the tightness of $Dist_{PAR}$ on one segment. Let $\hat{q}$ and $\hat{c}$ denote the first segment in $Dist_{LB}$ distance approximation. Suppose there are two segments $\hat{c}_0$, $\hat{c}_1$ in $Dist_{PAR}$ distance approximation that their summation of segment length is equal to the length of $\hat{c}$, denoted as $l = l_0 + l_1$. $r = r_1$. We could regard the $\hat{c}$ as the *SAPLA* representation of reconstructed time series from $\hat{c}_0$ plus $\hat{c}_1$. Their reconstructed time series are denoted as $\check{C}_0 + \check{C}_1$. If $\check{C}_0 + \check{C}_1$ and $\check{Q}_0 + \check{Q}_1$ are regarded as original time series, $\hat{q}$ and $\hat{c}$ are regarded as their *SAPLA* representation, we will get $Dist(\check{C}_0 + \check{C}_1, \check{Q}_0 + \check{Q}_1) \geq Dist_{PAR}(\hat{q}, \hat{c})$ (proved in Section A.5). According to Section A.5, $Dist(\check{C}_0, \check{Q}_0) \geq Dist_{PAR}(\hat{q}_0, \hat{c}_0)$ and $Dist(\check{C}_1, \check{Q}_1) \geq Dist_{PAR}(\hat{q}_1, \hat{c}_1)$. Let Q and C denote the original time series in first segment of $Dist_{LB}$ distance approximation. We will have $(Dist(C, Q))^2 = (Dist(C_0, Q_0))^2 + (Dist(C_1, Q_1))^2 \geq (Dist_{PAR}(\hat{q}_0, \hat{c}_0))^2 + (Dist_{PAR}(\hat{q}_1, \hat{c}_1))^2 \geq (Dist_{LB}(\hat{q}, \hat{c}))^2$. We complete the proof of tightness on the first segment. Because the entire distance is the summation of those segments, we can conclude $Dist_{LB}(\hat{Q}, \hat{C}) \leq Dist_{PAR}(\hat{Q}, \hat{C}) \leq Dist(Q, C)$.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (NSFC61972203), and the Natural Science Foundation of Jiangsu Province (BK20190442).

REFERENCES

[1] 2021. *Appendix, Source Code, Full Report, Datasets.* Retrieved May 26, 2021 from https://sites.google.com/view/edbt2022/home

[2] Yuhan Cai and Raymond Ng. 2004. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 599–610.

[3] Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn Keogh. 2014. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and information systems* 39, 1 (2014), 123–151.

[4] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. 2002. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)* 27, 2 (2002), 188–228.

[5] Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. 2007. Indexable PLA for efficient similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 435–446.

[6] Jens Clausen. 1999. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen* (1999), 1–30.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2009. *Introduction to algorithms*. MIT press.

[8] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[9] Frank Eichinger, Pavel Efros, Stamatis Karnouskos, and Klemens Böhm. 2015. A time-series compression technique and its application to the smart grid. *The VLDB Journal* 24, 2 (2015), 193–218.

[10] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. 1994. Fast subsequence matching in time-series databases. *ACM Sigmod Record* 23, 2 (1994), 419–429.

[11] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[12] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems* 3, 3 (2001), 263–286.

[13] Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM Sigmod Record*, Vol. 30. ACM, 151–162.

[14] Eamonn J Keogh and Michael J Pazzani. 1998. An Enhanced Representation of Time Series Which Allows Fast and Accurate Classification, Clustering and Relevance Feedback.. In *Kdd*, Vol. 98. 239–243.

[15] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. 2–11.

[16] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery* 15, 2 (2007), 107–144.

[17] Vebjorn Ljosa and Ambuj K Singh. 2007. APLA: Indexing arbitrary probability distributions. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 946–955.

[18] Yuu Morinaka, Masatoshi Yoshikawa, Toshiyuki Amagasa, and Shunsuke Uemura. 2001. The L-index: An indexing structure for efficient subsequence matching in time sequence databases. In *Proc. 5th PacificAisa Conf. on Knowledge Discovery and Data Mining*. 51–60.

[19] Rakesh Agrawal Giuseppe Psaila and Edward L Wimmers Mohamed &It. 1995. Querying shapes of histories. *Very Large Data Bases. Zurich, Switzerland: IEEE* (1995).

[20] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 262–270.

[21] Roonak Rezvani, Payam Barnaghi, and Shirin Enshaeifar. 2019. A New Pattern Representation Method for Time-series Data. *IEEE Transactions on Knowledge and Data Engineering* (2019).

[22] Tao Sun, Hongbo Liu, Seán McLoone, Shaoxiong Ji, and Xindong Wu. 2020. Time series indexing by dynamic covering with cross-range constraints. *The VLDB Journal* 29 (2020), 1365–1384.

[23] Byoung-Kee Yi and Christos Faloutsos. 2000. Fast time sequence indexing for arbitrary Lp norms. (2000).