Variable-Size Segmentation for Time Series Representation

Lamia Djebour, Reza Akbarinia, Florent Masseglia

Variable-Size Segmentation for Time Series Representation

Lamia Djebour, Reza Akbarinia, and Florent Masseglia

Inria, University of Montpellier, CNRS, LIRMM, Montpellier, France
firstname.lastname@inria.fr

Abstract. Given the high data volumes in time series applications, or simply the need for fast response times, it is usually necessary to rely on alternative, shorter representations of time series, usually with information loss. This incurs approximate comparisons of time series where precision is a major issue. We propose a new representation approach called ASAX, coming with two techniques ASAX_EN and ASAX_SAE, for segmenting time series before their transformation into symbolic representations. Our solution can reduce significantly the error incurred by possible splittings at different steps of the representation calculation, by taking into account the entropy of the representations (ASAX_EN) or the sum of absolute errors (ASAX_SAE), particularly for datasets with unbalanced (non-uniform) distributions. This is particularly useful for time series similarity search, which is the core of many data analytics tasks. We provide theoretical guarantees on the lower bound of similarity measures, and our experiments illustrate that our approach can improve significantly the time series representation quality.

Keywords: Time Series, SAX, Representation, Segmentation, Similarity Search, Information Retrieval

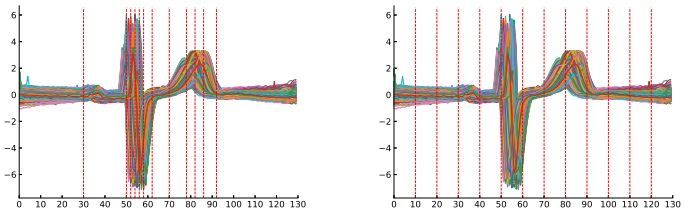
1 Introduction

Many applications in different domains generate time series data at an increasing rate. That continuous flow of emitted data may concern personal activities (*e.g.*, through smart-meters or smart-plugs for electricity or water consumption) or professional activities (*e.g.*, for monitoring heart activity or through the sensors installed on plants by farmers). This results in the production of large and complex data, usually in the form of time series that challenges knowledge discovery [11,22,27,20,21,10,24,19,15,17]. Data mining techniques on such massive sets of time series have drawn a lot of interest since their application may lead to improvements in a large number of these activities, relying on fast and accurate similarity search in time series for performing tasks like, *e.g.*, Classification, Clustering, and Motifs Discovery [20,16,30].

As a consequence of the high data volumes in such applications, similarity search can be slow on raw data. One of the issues that hinder the analysis of such

data is the dimensionality. This is why time series approximation is often used as a means to allow fast similarity search. SAX (Symbolic Aggregate ApproXimation) [13] is one of the most popular time series representations, allowing dimensionality reduction on the classic data mining tasks. SAX constructs symbolic representations by splitting the time domain into segments of equal size where the mean values of segments represent the time series intervals. This approximation technique is effective for time series having a uniform and balanced distribution over the time domain. However, we observe that, in the case of time series having high variation over given time intervals, this division into segments of fixed length is not efficient.

To illustrate the impact of a fixed length division of the series into segments, let us consider Figure 1. It shows a set D of time series, taken from *ECGFiveDays* dataset of UCR Archive [6]. In this dataset, the time series length is 136 and the data distribution in the time domain is unbalanced. We can notice that there is almost no variation from time point 1 to 45 and from 95 to the end. On the other hand, the remaining part, from time point 45 to 95, shows an important variation in the data values. Figure 1b shows the SAX division on D, with a fixed-size segmentation on the time series. In this example, the segment size is 10, leading to 13 segments in total. If we take any time series X from D and convert it into its SAX representation, the first 4 segments are always represented by the same symbol, all the values of these 4 segments being close to each other. Actually, there is no need to consider these 4 distinct segments. And the same applies to the last 3 segments. Meanwhile, for segments 5-10, all the values of each segment are represented by a single symbol while the data values present great variations, causing a significant loss of information on these segments.



(a) ASAX_SAE segmentation on D, with 13 segments.

(b) SAX segmentation on D, with 13 segments

Fig. 1: ASAX_SAE segmentation vs. SAX segmentation

As one can observe, it is not necessary to split the parts that are constant or where the variation is low since they don't carry any relevant information and would therefore better form a single segment. It is more efficient to divide into several small segments the parts where variation is important in order to preserve potentially relevant information as shown in Figure 1a. The splitting

of Figure 1a is the actual splitting obtained by our variable-size segmentation approach using our ASAX_SAE technique (based on the representations's sum of absolute errors) with a segment budget limited to 13. It would be rather counter-intuitive to merge segments 1-5 and 10-13, while it is the opposite for Figure 1b. But, the time intervals where data values show important differences should be split to create more segments, *e.g.*, between time point 50 and 90. By proposing such a customized splitting, we aim at improving the performance of information retrieval algorithms that will rely on our data representation. As illustrated by our experiments, the precision gain of ASAX_SAE compared to SAX for kNN search is 38% over the dataset of Figure 1 (*i.e.*, *ECGFiveDays* dataset).

Our main contribution is to provide an adaptive interval distribution, rather than an equal distribution in time. However, the number of possible segmentations of k segments with n can be very high. Furthermore, when searching for the best variable-size segmentation, a large number of computation may be repeated. Therefore, it is important to efficiently carry out the computations involved. We propose an new time series representation approach, called *ASAX*, that by smart selection of variable-size segments from the time domain allows to significantly reduce the information loss in the time series representation. To perform the variable-size segmentation in ASAX, we propose two different techniques *ASAX_EN* and *ASAX_SAE*. Briefly, we make the following contributions in this paper:

- We propose two new representation techniques, called ASAX_EN (based on the entropy) and ASAX_SAE (based on SAE the Sum of Absolute Errors), that allow obtaining a variable-size segmentation of time series with better precision in retrieval tasks thanks to the lower information loss.
- We propose a lower bounding method that allows approximating the distance between the original time series based on their representations in ASAX.
- We propose an efficient algorithm, called ASAX_DP, for improving the execution time of our segmentation approach, by means of dynamic programming.
- We implemented our solution and conducted empirical experiments using several real world datasets. The results illustrate that it can obtain significant performance gains in terms of precision for similarity search compared to SAX, particularly for datasets with non-uniform distributions. They suggest that the more the data distribution in the time domain is unbalanced (non-uniform), the greater is the precision gain of our approach. For example, for the *ECGFiveDays* dataset that has a non-uniform distribution in the time domain, the precision of ASAX_SAE is 93% and 82% for ASAX_EN compared to 55% for SAX. Furthermore, the results illustrate the effectiveness of our dynamic programming algorithm ASAX_DP, *e.g.*, up to $\times 40$ faster than the basic algorithm over some datasets.

This article is an extension of [7], with at least 30% of added value. Particularly, in Section 5, we propose ASAX_SAE, that is a variable-size segmentation algorithm based on SAE measurement. The execution time of ASAX_SAE is lower than that of previously proposed ASAX_EN, and its precision is often

better. Furthermore, in Section 6, we propose a dynamic programming solution, called ASAX_DP, which significantly improves the execution time of ASAX_SAE. In addition, in Section 8, we report new experimental results done for evaluating the performance of ASAX_SAE and ASAX_DP.

The rest of the paper is organized as follows. In Section 2, we define the problem we address. We discuss the related work in Section 3. In Section 4, we propose ASAX_EN for efficient segmentation of time seires, and in Section 5, we propose ASAX_SAE. In Section, 6 we propose ASAX_DP, the dynamic programming version of ASAX_SAE. In Section 7 we present a formula for approximating the Euclidean distance of time series based on their representation given by ASAX_EN or ASAX_SAE. In Section 8, we present the experimental evaluation of our approaches, and we conclude in Section 9.

2 Problem Definition and Background

In this section, we first present the background about SAX representation, and then define the problem we address.

D	Time series database
X, Y , Q	Time series
$n = \|X\|$	The length of time series X
l	The segment size
w	The number of PAA segments
a	The cardinality (the alphabet size)
$\overline{X}$	The PAA representation of time series X
$\hat{X}$	The SAX representation of time series X
k	the k nearest neighbors

Table 1: Some frequently used symbols

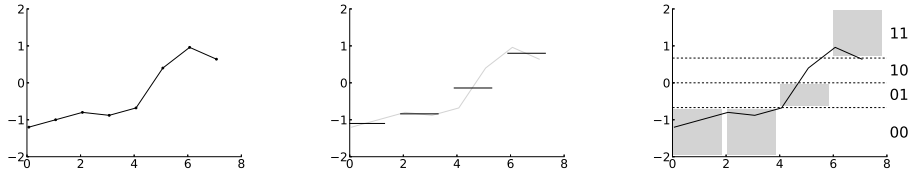
A time series X is a sequence of values $X = \{x_1, ..., x_n\}$. We assume that every time series has a value at every timestamp $t = 1, 2, ..., n$. Thus, all time series in a database have the same length. The length of X is denoted by $|X|$.

SAX allows a time series T of length n to be reduced to a string of arbitrary length w. Table 1 lists the notations used in this paper.

2.1 SAX Representation

Given two time series $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$, the Euclidean distance between X and Y is defined as [9]:

$$ED(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$



(a) A time series X of length 8

(b) A PAA representation of X, with 4 segments

(c) A SAX representation of X, with 4 segments and cardinality 4, $[00, 00, 01, 11]$

Fig. 2: A time series X is discretized by obtaining a PAA representation and then using predetermined break-points to map the PAA coefficients into SAX symbols. Here, the symbols are given in binary notation, where 00 is the first symbol, 01 is the second symbol, etc. The time series of Figure 2a in the representation of Figure 2c is [first, first, second, fourth] (which becomes $[00, 00, 01, 11]$ in binary).

The Euclidean distance is one of the most popular similarity measurement methods used in time series analysis.

The Symbolic Aggregate ApproXimation (SAX) is based on the Piecewise Aggregate Approximation (PAA) [13] which allows for dimensionality reduction while providing the important lower bounding property as we will show later. The idea of PAA is to have a fixed segment size, and minimize dimensionality by using the mean values on each segment. Example 1 gives an illustration of PAA.

Example 1. Figure 2b shows the PAA representation of X, the time series of Figure 2a. The representation is composed of $w = |X|/l$ values, where l is the segment size. For each segment, the set of values is replaced with their mean represented in the figure with solid horizontal lines. The length of the final representation w is the number of segments (and, usually, $w << |X|$).

By transforming the original time series X and Y into PAA representations $\overline{X} = \{\overline{x}_1, ..., \overline{x}_w\}$ and $\overline{Y} = \{\overline{y}_1, ..., \overline{y}_w\}$, the lower bounding approximation of the Euclidean distance for these two representations can be obtained by:

$$DR_f(\overline{X}, \overline{Y}) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (\overline{x}_i - \overline{y}_i)^2}$$

The SAX representation takes as input the reduced time series obtained using PAA. It discretizes this representation into a predefined set of symbols, with a given cardinality, where a symbol is a binary number. Example 2 gives an illustration of the SAX representation.

Example 2. In Figure 2c, we have converted the time series X to SAX representation with size 4, and cardinality 4 using the PAA representation shown in Figure 2b. The discretization is achieved by considering a series of breakpoints running parallel to the x-axis represented in the figure with dashed horizontal lines. for each segment depending on the PAA coefficient an alphabet symbol is mapped to that segment based on the breakpoints interval it falls in. We denote $\text{SAX}(X) = [00, 00, 01, 11]$.

The lower bounding approximation of the Euclidean distance for SAX representation $\hat{X} = \{\hat{x}_1, ..., \hat{x}_w\}$ and $\hat{Y} = \{\hat{y}_1, ..., \hat{y}_w\}$ of two time series X and Y is defined as:

$$MINDIST_f(\hat{X}, \hat{Y}) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (dist(\hat{x}_i, \hat{y}_i))^2}$$

where the function $dist(\hat{x}_i, \hat{y}_i)$ is the distance between two SAX symbols $\hat{x}_i$ and $\hat{x}_i$. The lower bounding condition is formulated as:

$$MINDIST_f(\hat{X}, \hat{Y}) \leq ED(X, Y)$$

2.2 Similarity Queries

The problem of similarity queries is one of the main problems in time series analysis and mining. In information retrieval, finding the k nearest neighbors (k-NN) of a query is a fundamental problem. Let us define *exact* and *approximate* k nearest neighbors.

Definition 1. (Exact k nearest neighbors) *Given a query time series Q and a set of time series D, let $R = ExactkNN(Q, D)$ be the set of k nearest neighbors of Q from D. Let $ED(X, Y)$ be the Euclidean distance between two time series X and Y, then the set R is defined as follows:*

$$(R \subseteq D) \wedge (|R| = k) \wedge (\forall a \in R, \forall b \in (D - R), ED(a, Q) \leq ED(b, Q))$$

Definition 2. (Approximate k nearest neighbors) *Given a set of time series D, a query time series Q, and $\epsilon > 0$. We say that $R = AppkNN(Q, D)$ is the approximate k nearest neighbors of Q from D, if $ED(a, Q) \leq (1+\epsilon)ED(b, Q)$, where a is the k^{th} nearest neighbor from R and b is the true k^{th} nearest neighbor.*

2.3 Time Series Approximation

The SAX representation proceeds to an approximation by minimizing the dimensionality: the original time series are divided into segments of equal size.

This representation does not depend on the time series values, but on their length. It allows SAX to perform the segmentation in $O(n)$ where n is the time series length. However, for a given reduction in dimensionality, the modeling error may not be minimal since the model does not adapt to the information carried

by the series. Our claim is that, by taking into account the information carried by time series for choosing the segments, we may obtain significant increase in the precision of kNN queries. This issue motivated us for proposing an adaptive representation aiming at minimizing the information loss.

2.4 Problem Statement

Our goal is to propose a variable-size segmentation of the time domain that minimizes the loss of information in the time series representation.

The problem we address is stated as follows. Given a database of time series D and a number w, divide the time domain into w segments of variable size such that the representation of the times series based on that segmentation lowers the error of similarity queries.

3 Related Work

Several techniques have been yet proposed to reduce the dimensionality of time series. Examples of such techniques that can significantly decrease the time and space required for similarity search are: singular value decomposition (SVD) [9], the discrete Fourier transformation (DFT) [1], discrete wavelets transformation (DWT) [4], piecewise aggregate approximation (PAA) [12], random sketches [5], Adaptive Piecewise Constant Approximation (APCA) [3], and symbolic aggregate approXimation (SAX) [14].

SAX [14] is one of the most popular techniques for time series representation. It uses a symbolic representation that segments all time series into equi-length segments and symbolizes the mean value of each segment. Some extensions of SAX have been proposed for improving the similarity search performance via indexing [23,2]. For example, iSAX [23] is an indexable version of SAX designed for indexing large collections of time series. iSAX 2.0 [2] proposes a new mechanism and also algorithms for efficient bulk loading and node splitting policy, which is not supported by iSAX index. In [2], two extensions of iSAX 2.0, namely iSAX 2.0 Clustered and iSAX2+, have been proposed. These extensions focus on the efficient handling of the raw time series data during the bulk loading process, by using a technique that uses main memory buffers to group and route similar time series together down the tree, performing the insertion in a lazy manner.

There have been SAX extensions designed to improve the representation of each segment, while using the SAX fixed-size segmentation, e.g., [18,25,29]. For example, SAX_TD improves the representation of each segment by taking into account the trend of the time series. It uses the values at the starting and ending points of the segments to measure the trend. TFSA [28] and SAX_CP [26] are other trend-based SAX representation methods. TFSA proposes a representation method for long time series based on the trend, and SAX_CP considers abrupt change points while generating the symbols in order to capture time series' trends.

ABBA [8] is a symbolic representation of time series based on an adaptive polygonal chain approximation of the time series into a sequence of tuples, followed by a mean-based clustering to obtain the symbolic representation. However, the authors of ABBA have not proposed an approximate function for comparing the time series in their representation form, and this prevents ABBA from being used for kNN search over the representations.

To increase the quality of time series approximation, our ASAX approach is based on variable-length segmentation. ASAX is complementary to the existing SAX extensions, *e.g.*, indexing based techniques or those that use the trend for representing the segments. This makes our variable-size segmentation an advantageous alternative for segmenting the time domain in indexing solutions like iSAX.

4 Adaptive SAX based on Entropy

In this section, we propose ASAX_EN, our first variable-size segmentation technique for the time series representation. To create a segmentation with minimum information loss, ASAX_EN divides the time domain based on the representation entropy.

In the rest of this section, we first describe the notion of entropy for the time series representation. Then, we describe our algorithm for creating the variable-size segments based on this measurement.

4.1 Entropy

Entropy is a mathematical function which intuitively corresponds to the amount of information contained or delivered by a source of information. This source of information can be of various types. The more the source emits different information the higher is the entropy. If the source always sends the same information, the entropy is minimal. Formally, entropy is defined as follows.

Definition 3. *Given a set X of elements, and each element $x \in X$ having a probability P_x of occurrence, the entropy H of the set X is defined as:* $H(X) = -\sum_{x \in X} P_x \times \log P_x$

In our context, we calculate the entropy on a set containing the different symbolic representations obtained from the transformation of the original time series of a dataset according to a given segmentation. The entropy computed on this set allows to measure the quantity of information contained in the time series representations. Let us illustrate this using an example.

Example 3. Consider the database D={x,y,z} in Figure 3 where x, y and z are time series with l=8. Let us create a representation having two segments (e.g., 0-4, and 4-8) obtained by dividing the time domain into two segments of the same size (the split is represented with the red dashed line). Then we compute the entropy of the representation of the set D. To generate the representation of the

time series x, y and z, they are discretized by obtaining their PAA representation and then using predetermined break-points to map the PAA coefficients into the corresponding symbols like the SAX representation proceeds. We have converted the 3 time series into symbolic representations with size 2, and cardinality 4. Thus, the symbolic representations of x, y and z are $\hat{x} = [00, 10]$, $\hat{y} = [00, 10]$ and $\hat{z} = [00, 10]$, respectively. We notice that the 3 time series have the same symbolic representation, thus, the set X consists of only this unique symbolic representation with an occurrence equal to 3., *i.e.*, $X = \{[00, 10]\}$. The entropy *H(X)* of X is computed as follows:

$H(X) = -(P(x = [00, 10]) \times \log_2 P(x = [00, 10]))$

where the probability for the word x is $P(x = [00, 10]) = \frac{3}{3} = 1$. Therefore, we have $H(X) = -(1 \log 1) = 0$ meaning that in the representation X there is no information allowing to distinguish the three original time series from each other. This is explained by the fact that they have the same representation with a fixed-size segmentation.

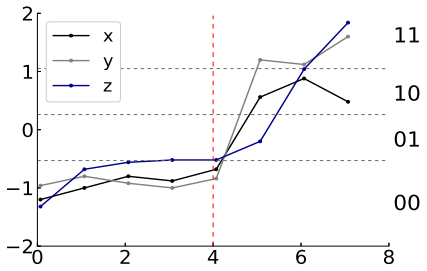


Fig. 3: ASAX_EN segmentation with 2 segments

In the next subsection, we describe our algorithm to create variable-size segments based on entropy.

4.2 Variable-Size Segmentation Based on Entropy Measurement

Given a database of time series D, and a number w, our goal is to find the k variable size segments that minimize the loss of information in time series representations.

Intuitively, our algorithm works as follows. First it splits the time domain into two segments of equal size. Then, it performs $w - 2$ iterations, and in each iteration it finds the segment s whose split makes the minimum loss in entropy, and it splits that segment. By doing this, in each iteration a new segment is added to the set of segments. This continues until having w segments.

Let us now describe ASAX_EN in more details. The pseudo-code is shown in Algorithm 1. It first splits the time domain into two equal parts and creates

two segments that are included to the set *segments* (Line 1). Then, it sets the current number of segments, denoted as k, to 2 (Line 2).

Afterwards, in a loop, until the number of segments is less than w the algorithm proceeds as follows. For each segment i (from 1 to k), i is divided into two equal parts, if its size is greater than $minSize$, which is the minimum possible size of a segment, and it's default value is 1. Then, a temporary set of segments $tempSegments$ is created including the two new segments and all previously created segments except i (i.e., expect the one that has been divided). Then, for each time series ts in the database D, the algorithm generates the symbolic representation of ts (denoted as $word$) using the segments included in $tempSegments$ with the given cardinality a (Line 12), and inserts it to a hash table (Line 13). Note that for all time series, ASAX_EN uses the same cardinality to map the PAA coefficients into the corresponding symbols. After having inserted all the representations of the time series contained in D to the hash table, the entropy of the representations is calculated (Line 14). If the entropy is higher than the maximum entropy obtained until now, the algorithm sets i as the segment to be split, and keeps the entropy of the representation. This procedure continues by splitting one of the segments at each time, and computing the entropy. The algorithm selects the one whose entropy is the highest, and updates the set of the segments by removing the selected segment, and inserting its splits to the set *segments* (Lines 18-20). Then, the variable k, which shows the number of current segments, is incremented by one. The algorithm ends if the number of segments is equal to the required number, *i.e.*, w.

Example 4. Let us consider the dataset D in Figure 3 which represents the initialization of the algorithm, i.e., the time domain is divided into two segments of the same size. The next step is to create the 3rd segment by splitting one of the two existing segments. Two different scenarios are possible.

Scenario 1 : The first scenario is shown in Figure 4a where the left segment is divided into two equal parts. We generate the symbolic representation of the time series x, y, and z by using the 3 segments. Let's assume the cardinality is 4. Then, $\hat{x} = [00, 00, 10]$, $\hat{y} = [00, 00, 10]$ and $\hat{z} = [00, 00, 10]$ are the symbolic representation of x, y and z, respectively. Thus, the set X_1 consists of only one representation *[00,00,10]* with an occurrence of 3, *i.e.*, $X_1 = [00, 00, 10]$. The entropy is then calculated as: $H(X_1) = -(P(x = [00, 00, 10]) \log P(x = [00, 00, 10]))$ where $P(x = [00, 00, 10]) = \frac{3}{3} = 1$ and we have $H(X_1) = -(1 \log 1) = 0$.

Scenario 2 : This scenario is shown in Figure 4b in which the right segment is split. As for Scenario 1 we generate the symbolic representation of time series x, y and z using the 3 segments, and cardinality of 4. $\hat{x} = [00, 01, 10]$, $\hat{y} = [00, 01, 11]$ and $\hat{z} = [00, 01, 11]$ are the symbolic representation of x, y and z, respectively. In this scenario the representation set X_2 consists of *[00,01,10]* with an occurrence of 1 and *[00,01,11]* with an occurrence of 2, *i.e.*, $X = [00, 01, 10], [00, 01, 10]$. The entropy is calculated as:
$H(X_2) = -(P(x = [00, 01, 10]) \log P(x = [00, 01, 10]) +$
$P(x = [00, 01, 11]) \log P(x = [00, 01, 11]))$ where $P(x = [00, 01, 10]) = \frac{1}{3}$ and

Algorithm 1: ASAX_EN variable-size segmentation

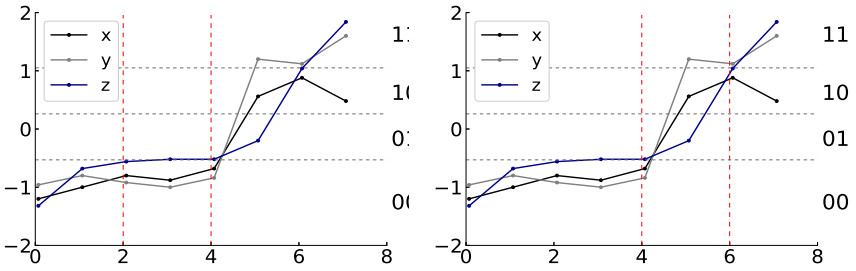
Input: D: time series database; n: the length of time series; $minSize$: the minimum possible size of a segment; a: cardinality of symbols; w: the required number of segments

Output: w variable-size segments

```
1  segments = {[0, n/2 − 1], [n/2, n − 1]}; // split time domain into two equal size segments
2  k = 2
3  while k ≠ w do
4      segmentToSplit = 1
5      entropy = 0
6      for i=1 to k do
7          tempSegments = segments
8          if length(tempSegments[i]) > minSize then
9              split segment i into two equal parts, and replace the segment i by its corresponding parts in tempSegments
10             hashtable = new HashTable
11             foreach ts ∈ D do
12                 word = Symbolic-Representation(ts, tempSegments, a)
13                 hashTable.put(word)
14             e = entropy(hashTable)
15             if e > entropy then
16                 segmentToSplit = i
17                 entropy = e
18     split segmentToSplit into two equal size segments s1 and s2
19     segments = segments - {segmentToSplit}
20     segments = segments ⋃ {s1, s2}
21     k = k+1
22 return segments
```

$P(x = [00, 01, 11]) = \frac{2}{3}$. Then, $H(X_2) = -(\frac{1}{3}\log\frac{1}{3} + \frac{2}{3}\log\frac{2}{3}) = 0.918$.

After having calculated the entropy for the two scenarios, we see that $H(X_1) < H(X_2)$. We aim to maximize the entropy, therefore we choose the segmentation generated in Scenario 2 for this iteration of our algorithm. We continue the next iterations, until the number of segment reaches w.



(a) Scenario 1 of ASAX_EN segmentation with 3 segments

(b) Scenario 2 of ASAX_EN segmentation with 3 segments

Fig. 4: The two different scenarios of ASAX_EN segmentation with 3 segments. Scenario 4b is the one chosen because it optimizes the entropy.

Let us now analyze the complexity of ASAX_EN algorithm. Let $|D|$ be the number of time series in the database, n the time series length, and w the desired number of segments. In the *while* loop, the algorithm performs $w-2+1$ iterations, and in each iteration it tries the division of each segment and computes the entropy of the segmentation. Thus, in total the number of entropy computations is $O(w^2)$. For each entropy computation, the database should be scanned, and the representation of each time series created. This is done in $O(|D| \times n)$. Therefore, in total the time complexity of the algorithm is $O(|D| \times n \times w^2)$.

4.3 Uniform Distribution of Symbols

SAX breakpoints divide the value domain into regions of different size where small regions are concentrated on the middle of the value domain and regions at extreme values are larger. This is illustrated by Figure 5, with three time series from our motivating example in Figure 1 with 6 segments. The breakpoints of SAX with 10 symbols are represented by horizontal lines, and, logically, they appear close to the center of the distribution. If we keep such distribution of symbols, then we would have two issues. First, the extreme values of the series like those above 2 or below -4 would be assigned the same symbol (their PAA value on the segment would fall in the same symbol). Second, the adaptive segmentation would consider that the slight variations around zero are more

important than the ones at extreme values, ending in irrelevant splits that favor minor information gain. For this reason, we propose to calculate the breakpoints differently. In ASAX_EN, the discretization is done based on breakpoints that produce uniform distributions of symbols. These breakpoints divide the value domain into regions of equal size. In the case of Figure 5 the 10 symbol regions will be evenly distributed in the range of data values.

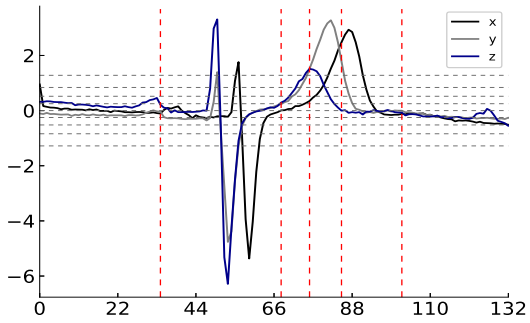


Fig. 5: The Gaussian based distribution of symbols in SAX are not suitable for ASAX_EN since they would favor minor information gain.

5 Adaptive SAX based on SAE

In this section, we propose ASAX_SAE, our second variable-size segmentation approach for time series representation. Here, to create a segmentation with minimum information loss on time series approximation, ASAX_SAE divides the time domain by taking into account the *sum of absolute errors (SAE)* of the representation with a bottom-up strategy instead of a top-down strategy as used in the previous approach. Our experimental results have shown that this method is faster than ASAX_EN, and its precision is often better.

In the rest of this section, we first describe the notion of sum of absolute errors (SAE) for the time series representation, and then, we describe the algorithm that creates the variable-size segments based on SAE measurement.

5.1 Sum of Absolute Errors (SAE)

SAE (Sum of Absolute Errors) calculates the sum of the absolute difference between the actual and the estimated values. Formally, SAE is defined as follows.

Definition 4. *Given a vector X of n elements and a vector $\tilde{X}$ being the estimated values generated from X, SAE of the estimation is given by:*

$$SAE(X, \tilde{X}) = \sum_{i=1}^{n} |x_i - \tilde{x}_i|$$

In our context, we calculate the SAE on the PAA representation obtained from the transformation of the original time series of a dataset according to a given segmentation. The SAE computed on this representation allows to measure the approximation error on the time series by the PAA representation compared to the original time series. The lower the SAE, the closer is the PAA representation to the original data.

By transforming a time series $X = \{x_1, ..., x_n\}$ into a PAA representation $\overline{X} = \{\overline{x}_1, ..., \overline{x}_w\}$, X is reduced to the PAA representation composed of w segments. For each segment, the set of values is replaced with their mean. We can compute the SAE for each segment, that is in this case, the sum of the absolute differences between each value (original value) and its segment's mean (estimated value). In the following, we explain how to compute the SAE of a PAA representation for a given segmentation.

Let $\overline{X}$ be the PAA representation of X with w segments. The SAE of $\overline{X}$ for a particular segment is the sum of the absolute errors for the time series values in this segment. Formally, SAE of $\overline{X}$ for a segment s_i is computed as:

$$SAE(s_i, \overline{x}_i) = \sum_{j=LB(s_i)}^{UB(s_i)} |x_j - \overline{x}_i|$$

where s_i is the selected segment, $LB(s_i)$ and $UB(s_i)$ are the start and end time points of s_i respectively.

In the next subsection, we describe our algorithm that creates variable-size segments thereby providing an accurate representation of time series based on the SAE measurement.

5.2 Variable-Size Segmentation Based on SAE Measurement

Given a database of time series D, and a number w, our goal is to find the w variable size segments that minimize the loss of information in time series representations by minimizing the approximation error of these representations.

Intuitively, our algorithm works as follows. Based on a starting segment size value $size$, it firstly splits the time domain into segments of length $size$. The default value of $size$ is 2. Let k be the initial number of segments. The algorithm performs $k - w$ iterations, and in each iteration it finds the two adjacent segments s_i and s_{i+1} whose merging gives the minimum SAE (MSAE) on the representations, and merges them. By doing this, in each iteration the two selected segments are merged to form a single segment which replaces them in the set of segments, reducing the number of segments by one. This continues until having w segments.

Algorithm 2: ASAX_SAE variable-size segmentation

Input: D: time series database; n: the length of time series; $size$: the starting size of segments; w: the required number of segments

Output: w variable-size segments

```
1  k = ⌈n / size⌉
2  segments = {⋃_{i=0}^{k-1} [size × i, size × (i + 1) − 1]} // split time domain into k
     segments of size size
3  while k ≠ w do
4      segmentsToMerge = null
5      msae = ∞
6      for i=1 to k − 1 do
7          s = merge (s_i, s_{i+1})
8          sae = 0
9          foreach ts in D do
10             sae = sae + SAE(ts, s)
11         if sae < msae then
12             segmentsToMerge = i
13             msae = sae
14     s = merge (s_{segmentsToMerge}, s_{segmentsToMerge+1})
15     segments = segments − {s_{segmentsToMerge}, s_{segmentsToMerge+1}}
16     segments = segments ⋃ s
17     k = k-1
18 return segments
```

Let us now describe our algorithm in more details. The pseudocode is shown in Algorithm 2. It first sets the current number of segments, denoted as k, to $\frac{n}{size}$ where n is the length of time series (Line 1). Then, it splits the time domain into k segments of length $size$ that are included to the set *segments* (the set containing the current segmentation) (Line 2).

Afterwards, in a loop, until the number of segments is more than w the algorithm proceeds as follows. For each segment s_i (i from 1 to $k-1$), s_i is merged with segment s_{i+1} to form a single segment denoted as s (Line 7). Then, for each time series ts in the database D, the algorithm generates its PAA representation on segment s and calculates the corresponding SAE and adds the result of the computed SAE to *sae* (Line 10).

After having calculated the sum of the SAE for the PAA representation of all the time series contained in D, if the SAE is less than the MSAE (minimum SAE) obtained so far, the algorithm sets i as the segment to be merged with the next one, and keeps the SAE of the representation (Lines 12, 13). This procedure continues by trying the merging of every two adjacent segments of *segments* at each time, and computing the SAE. The algorithm selects the merging whose SAE is the lowest, and updates the set of the segments by removing the selected segments, and inserting its merging to *segments* (Lines 14-16). Then, the number of current segments (*i.e.*, k) , is decremented by one (Line 17). The algorithm ends when k gets equal to the required number, *i.e.*, w.

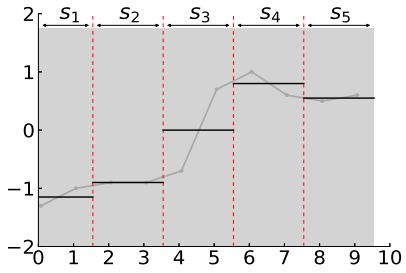
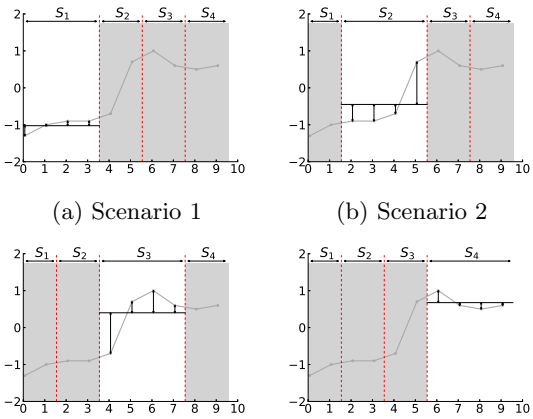


Fig. 6: The PAA representation of time series X contains 5 segments.

Let us illustrate the principle of our algorithm using an example. For simplicity, we consider a dataset containing only a single time series.

Example 5. Let us apply our algorithm on the time series X in Figure 6 by taking the initial size of 2 for the starting segments. Notice that the PAA encoding is represented with black horizontal lines. The algorithm starts by dividing the time domain into 5 segments of size 2. The next step is to reduce the number of segments from 5 to 4. For this purpose, the algorithm tests the merging of every two adjacent segments of the 5 existing segments, in order to find the one that has the minimum SAE. Four different scenarios are possible:



(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

Fig. 7: The four different scenarios of ASAX_SAE segmentation with 4 segments. Scenario 1 is the one chosen because it provides the minimum SAE.

Scenario 1: The first scenario is shown in Figure 7a where s_1 and s_2 of the initial segmentation (shown in figure 6) are merged into one segment. We calculate the values's mean on the resulting segment (denoted $S1$ in figure 7a), and then compute the SAE of this approximation that is $SAE_1(X, \overline{X}) = 0.55$.

Scenario 2: This scenario is shown in Figure 7b in which s_2 and s_3 of the initial segmentation are merged. As for Scenario 1, we compute the mean of X on the current segment S_2. Here, $SAE_2(X, \overline{X}) = 2.3$.

Scenario 3: This scenario is shown in Figure 7c, where we merge s_3 and s_4. For this merging (S_3), $SAE_3(X, \overline{X}) = 2.2$.

Scenario 4: The last scenario is shown in Figure 7d, where we merge s_4 and s_5. For this segment S_4, $SAE_4(X, \overline{X}) = 0.65$.

We have calculated the SAE for the 4 scenarios. Since we aim to minimize the SAE, we choose Scenario 1 that leads to the minimum SAE value (MSAE), that is $MSAE = 0.55$ (obtained by merging s_1 and s_2). After merging the segments s_1 and s_2, the algorithm continues the next iterations, until the number of segment reaches w.

Let us now analyze the complexity of ASAX_SAE algorithm. Let $|D|$ be the number of time series in the database, n the time series length, and w the desired number of segments. The initial number of segments is $\frac{n}{size}$, where $size$ is the initial size of the segments. In the *while* loop, the algorithm performs $\frac{n}{size} - w + 1$ iterations, and in each iteration it tries the merging of each segment with its next segment, and computes the SAE of the segmentation. The maximum value for $\frac{n}{size}$ is n, *i.e.*, with $size = 1$. Thus, the maximum number of SAE computations

Algorithm 3: ASAX_DP variable-size segmentation

Input: D: time series database; n: the length of time series; $size$: the starting size of segments; w: the required number of segments
Output: w variable-size segments

$k = \lceil \frac{n}{size} \rceil$
$segments = \{\bigcup_{i=0}^{k-1}[size \times i, size \times (i+1) - 1]\}$ // split time domain into k segments of size $size$
$matrix$ = matrix of $k \times k$ values initialized to -1
while $k \neq w$ **do**
    $segmentsToMerge = null$
    $msae = \infty$
    for i=1 to $k-1$ **do**
        s = merge (s_i, s_{i+1})
        r, c =Compute the position in $matrix$ corresponding to s
        $sae = 0$
        if matrix[r,c] = -1 **then**
            foreach ts in D **do**
                $sae = sae + SAE(ts)$
            $matrix[r,c] = sae$
        else
            $sae = matrix[r,c]$
        if $sae < msae$ **then**
            $segmentsToMerge = i$
            $msae = sae$
    s = merge $(s_{segmentsToMerge}, s_{segmentsToMerge+1})$
    $segments = segments - \{s_{segmentsToMerge}, s_{segmentsToMerge+1}\}$
    $segments = segments \bigcup s$
    k = k-1
return $segments$

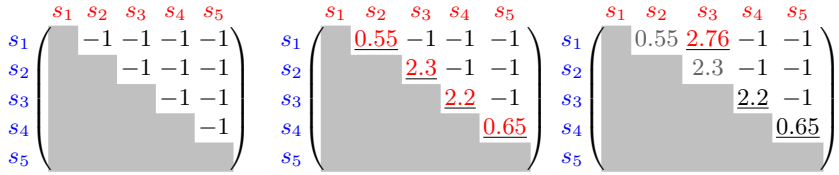
is $O(n-w)^2$. For each SAE computation, the database should be scanned, and the error of each time series representation computed. This is done in $O(|D| \times n)$. Therefore, in total the time complexity of the algorithm is $O((n-w)^2 \times |D| \times n)$.

6 ASAX_SAE based on Dynamic Programming

The ASAX_SAE algorithm, which we presented in the previous section, can reduce significantly the information loss in time series representations and is much more efficient than ASAX_EN in terms of accuracy and calculation time, as illustrated by our experiments. However, its execution time may be high, particularly over large time series datasets as shown by our experiments (e.g., see Figure 12 in Section 8). In this section, we present an efficient version of ASAX_SAE, called ASAX_DP, for improving the execution time of our segmentation technique using dynamic programming. In ASAX_DP, we use a data structure (matrix) to keep track of the result of the SAE computation for each iteration. In the matrix, if the value of a cell (i, j) is positive, then it corresponds to the SAE of merging all adjacent segments from segment s_i to segment s_j. In each iteration, after testing the merging of two adjacent segments, the computed SAE of the merging is kept in the matrix, in order to be used in the case where this merging needs to be evaluated again in the next steps.

Let us describe ASAX_DP algorithm in more details. Algorithm 3 presents the pseudocode of the improved approach. As for the ASAX_SAE algorithm (described in the previous section), ASAX_DP splits the time domain into k segments of length *size* to create the set *segments* (Lines 1, 2). A matrix of size $k \times k$ denoted as *matrix* is allocated and all its values are initialized to -1 (Line 3). Then, in a loop, until the number of segments is more than w the algorithm proceeds as follows. For each segment s_i (i from 1 to $k-1$), the algorithm tests the merging of s_i with segment s_{i+1}. For this, the algorithm computes the position in the matrix corresponding to the merging of these two segments by finding the row and column number (r, c) (Line 9). If it is the first time that these two segments merging is tested (*i.e.*, if the SAE value in the corresponding cell in the matrix is equal to -1), then the algorithm has to compute the SAE for each time series ts in the database D on the segment s made from merging s_i and s_{i+1} (Line 13). By summing up the SAE of PAA representation of all the time series contained in D, ASAX_DP adds the result of the computed SAE to *sae* and stores this value in the matrix by replacing the existing value (-1) by the calculated SAE (Line 14). In the case where the merging of s_i and s_{i+1} has already been tested (*i.e.*, if the SAE value in the matrix is not -1), the algorithm simply has to get the SAE value from the matrix which is already computed and sets *sae* to this value (Line 16). After having obtained the SAE value, if it is less than the MSAE (minimum SAE) obtained so far, the algorithm sets i as the segment to be merged with the next one, and keeps the SAE of the representation (Lines 18, 19). This procedure continues by testing the merging of every two adjacent segments of *segments* at each time by making use of the matrix to avoid redundant SAE computations. The algorithm selects the merg-

ing whose SAE is the lowest, and updates the set of the segments (Lines 20-22). The procedure continues until k reaches the required number of segments w.



(a) Matrix initialization (b) After step 1 (c) After step 2

Fig. 8: State of the matrix at different steps of the algorithm. The updated values are in red and the possible scenarios are underlined in each step.

Bellow, we illustrate our algorithm using an example.

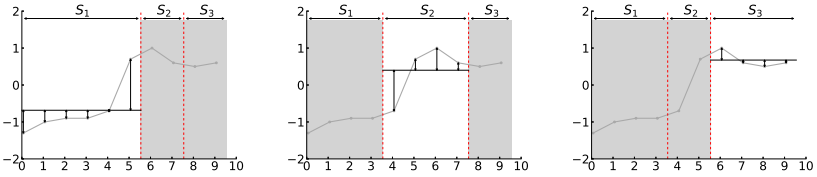
Example 6. Let us apply ASAX_DP on the time series X shown in Figure 6 by taking the initial size of 2 for the segments. Suppose the number of desired segments is $w = 3$. The algorithm starts by dividing the time domain into 5 segments of size 2 and initializes the matrix that will keep track of the SAE computation (Figure 8a). This matrix's row (and column) size is 5 which is the initial number of segments, each value corresponds to a possible merging of the initial segments (two segments or more). The value of cell (i, j) in the matrix corresponds to the SAE of merging of all adjacent segments from s_i to s_j. The final number of segments is w=3, thus the algorithm consists of 2 steps:
Step 1 : Reduce the number of segments from 5 to 4. The algorithm tests the merging of every two adjacent segments of the 5 existing segments, 4 different scenarios are possible (presented previously in Example 6). Each of these 4 merging possibilities are tested, the corresponding SAE for each possible merging is computed, and the results are stored in the matrix. Figure 8b shows the content of the matrix after the update (the updated values are in red). The possible scenarios are underlined. For this step, we choose the segmentation generated in Scenario 1 shown in Figure 7a, resulting from merging s_1 and s_2, since it provides the minimum SAE value (MSAE), that is $MSAE = 0.55$.
Step 2 : Reduce the number of segments from 4 to 3. In the previous step, the initial segments s_1 and s_2 have been merged into a single segment. Now, we have 4 segments as shown in Figure 7a. To reduce the 4 segments to 3, three merging scenarios are possible:

Scenario 1: The first scenario is shown in Figure 9a where the first segment S_1 of Figure 7a(the segment resulting from merging s_1 and s_2) and S_2 are merged, i.e s_1,s_2 and s_3 of the initial segmentation are merged into one segment. This merging is tested for the first time until now ($matrix[1, 3]$=-1), then, we calculate the values's mean of X on the resulting segment, and then compute the SAE that is $SAE_1(X, \overline{X}) = 2.76$. The matrix cell that corresponds to this merging is updated (Figure 8c).

Scenario 2: This scenario is shown in Figure 9b in which s_3 and s_4 of the initial segmentation are merged. This merging has already been tested in the previous step ($matrix[3,4]$ $\neq$-1), the SAE value is retrieved from our matrix from the cell $(3,4)$. Here, $SAE_2(X,\overline{X}) = 2.2$.

Scenario 3: The last scenario is shown in Figure 9c, where we merge s_4 and s_5. The SAE for this merging has been already computed in step 1, that is $SAE_3(X,\overline{X}) = 0.65$.

We have now the SAE for the three scenarios. We choose the minimum SAE value, that is $MSAE = 0.65$ corresponding to the segmentation generated in Scenario 3, which is chosen for this iteration.
After this step, we have 3 segments shown in Figure 9c. Since, the number of segment reaches w, the algorithm ends.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 9: The three different scenarios of ASAX_SAE segmentation with 3 segments. Scenario 9c is selected since it provides the minimum SAE.

7 Lower Bounding of the Similarity Measure

Having introduced the new representation of time series, we will now define a distance measure on it. SAX [14] defines a distance measure on the representation of time series as described in Section 2.1. Given the representation of two time series, the $MINDIST_f$ function allows obtaining a lower bounding approximation of the Euclidean distance between the original time series. By the following theorem, we propose a lower bounding approximation formula for the case of variable size segmentation in ASAX (whether it is done by ASAX_EN or ASAX_SAE).

Theorem 1. *Let X and Y be two time series. Suppose that by using ASAX we create a variable size segmentation with w segments, such that the size of the i^{th} segment is l_i.*
Let $\overline{X}$ and $\overline{Y}$ be the PAA representation of variable size of X and Y in ASAX, $DR_v(\overline{X},\overline{Y})$ gives a lower bounding approximation of the Euclidean distance between X and Y:

$$DR_v(\overline{X},\overline{Y}) = \sqrt{\sum_{i=1}^{w}((\overline{x}_i - \overline{y}_i)^2 \times l_i)}$$

Let $\hat{X}$ and $\hat{Y}$ be the representations of X and Y in ASAX obtained by converting $\overline{X}$ and $\overline{Y}$ into symbolic representation. Then, $MINDIST_v(\hat{X}, \hat{Y})$ gives a lower bounding approximation of the Euclidean distance between X and Y:

$$MINDIST_v(\hat{X}, \hat{Y}) = \sqrt{\sum_{i=1}^{w} (dist(\hat{x}_i, \hat{y}_i)^2 \times l_i)}$$

Proof : To generate the ASAX representation of a time series, we need to first generate its PAA representation using the variable size segmentation (by taking the mean of the time series in each segment), and then we convert the PAA representation to ASAX by creating a symbol for each segment.

Our proof is done in two steps. In the first step, we show that the distance of X and Y in the PAA representation, denoted as $DR_v(\overline{X}, \overline{Y})$, is lower than or equal to their Euclidean distance. In the second step, we show that $MINDIST_v(\hat{X}, \hat{Y}) \leq DR_v(\overline{X}, \overline{Y})$.

Step 1 : In the first step, we show that the DR_v distance lower bounds the Euclidean distance, that is :

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \geq \sqrt{\sum_{j=1}^{w}((\overline{x}_j - \overline{y}_j)^2 \times l_j)} \tag{1}$$

To prove the above inequality, it is sufficient to prove that the PAA distance of two time series *in each segment* is lower than or equal to their Euclidean distance in the segment. Without loss of generality, let us take the first segment S_1, and suppose that its size is l_1. Thus, we need to prove the following inequality:

$$\sqrt{\sum_{i=1}^{l_1}(x_i - y_i)^2} \geq \sqrt{\sum_{i=1}^{l_1}((\overline{x}_i - \overline{y}_i)^2} \tag{2}$$

Let $\overline{X}$ and $\overline{Y}$ be the means of time series X and Y, respectively. We can rewrite the above inequality as:

$$\sqrt{\textstyle\sum_{i=1}^{l_1}(x_i - y_i)^2} \geq \sqrt{(\overline{X} - \overline{Y})^2 \times l_1}$$
$$\text{Or : } \sqrt{\textstyle\sum_{i=1}^{l_1}(x_i - y_i)^2} \geq \sqrt{l_1}\sqrt{(\overline{X} - \overline{Y})^2}$$

By squaring both sides, we have:

$$\sum_{i=1}^{l_1}(x_i - y_i)^2 \geq l_1(\overline{X} - \overline{Y})^2$$

For each point x_i in X, $x_i = \overline{X} - \Delta x_i$. The same applies to each point y in Y. Then, we substitute the rearrangement:

$$\sum_{i=1}^{l_1}((\overline{X} - \Delta x_i) - (\overline{Y} - \Delta y_i))^2 \geq l_1(\overline{X} - \overline{Y})^2$$

After rearranging terms in the left-hand side, we have:

$$\sum_{i=1}^{l_1}((\overline{X} - \overline{Y}) - (\Delta x_i - \Delta y_i))^2 \geq l_1(\overline{X} - \overline{Y})^2$$

Then, we expand the inequality using the binomial theorem:

$$\sum_{i=1}^{l_1}((\overline{X} - \overline{Y})^2 - 2(\overline{X} - \overline{Y})(\Delta x_i - \Delta y_i) + (\Delta x_i - \Delta y_i)^2) \geq l_1(\overline{X} - \overline{Y})^2$$

By using distributive law and summation properties, we have:

$$l_1(\overline{X} - \overline{Y})^2 - 2(\overline{X} - \overline{Y})\sum_{i=1}^{l_1}(\Delta x_i - \Delta y_i) + \sum_{i=1}^{l_1}(\Delta x_i - \Delta y_i)^2 \geq l_1(\overline{X} - \overline{Y})^2$$

We know that $x_i = \overline{X} - \Delta x_i$, which means that $\Delta x_i = \overline{X} - x_i$, and the same applies for Δy_i.

$$\begin{aligned}
\sum_{i=1}^{l_1}(\Delta x_i - \Delta y_i) &= \sum_{i=1}^{l_1}((\overline{X} - x_i) - (\overline{Y} - y_i)) \\
&= (\sum_{i=1}^{l_1}\overline{X} - \sum_{i=1}^{l_1}x_i) - (\sum_{i=1}^{l_1}\overline{Y} - \sum_{i=1}^{l_1}y_i) \\
&= (l_1\overline{X} - \sum_{i=1}^{l_1}x_i) - (l_1\overline{Y} - \sum_{i=1}^{l_1}y_i) \\
&= (\sum_{i=1}^{l_1}x_i - \sum_{i=1}^{l_1}x_i) - (\sum_{i=1}^{l_1}y_i - \sum_{i=1}^{l_1}y_i) \\
&= 0 - 0 = 0
\end{aligned}$$

We substitute 0 into $\sum_{i=1}^{l_1}(\Delta x_i - \Delta y_i)$ in the inequality:

$$l_1(\overline{X} - \overline{Y})^2 - 0 + \sum_{i=1}^{l_1}(\Delta x_i - \Delta y_i)^2 \geq l_1(\overline{X} - \overline{Y})^2$$

Then by subtracting $n(\overline{X} - \overline{Y})^2$ from both sides, we have:

$$\sum_{i=1}^{l_1}(\Delta x_i - \Delta y_i)^2 \geq 0$$

This always holds true, so it completes the proof.

Step 2 : Following the same method as in *Step 1*, we will show here that $MINDIST_v$ lower bounds the DR_v distance, that is :

$$\sqrt{\sum_{j=1}^{w}((\overline{x}_j - \overline{y}_j)^2 \times l_j)} \geq \sqrt{\sum_{i=j}^{w}(dist(\hat{x}_j, \hat{y}_j)^2 \times l_j)}$$

To prove the above inequality, it is sufficient to prove that $MINDIST_v$ in each segment lower bounds the DR_v distance in the segment. Without loss of generality, let us take the first segment S_1, and assume that its size is l_1. Thus, it is sufficient to prove the following inequality:

$$\sqrt{\sum_{i=1}^{1}((\overline{x}_i - \overline{y}_i)^2) \times l_1} \geq \sqrt{\sum_{i=1}^{1}(dist(\hat{x}_1, \hat{y}_1)^2 \times l_1)}$$

The above inequality can be written as: $l_1(\overline{X} - \overline{Y})^2 \geq l_1(dist(\hat{X}, \hat{Y}))^2$. There are two possible scenarios for the symbols representing X and Y.

Case 1 : the symbols representing X and Y are either the same, or consecutive from the alphabet a, i.e. $|(\hat{X} - \hat{Y})| \leq 1$. In this case the *MINDIST* value is 0. Therefore, the inequality becomes :

$$l_1(\overline{X} - \overline{Y})^2 \geq 0$$

which always holds true.

Case 2 : the symbols representing X and Y are at least two alphabets apart, i.e. $|(\hat{X} - \hat{Y})| > 1$. Let us assume that X is at a higher region than Y, i.e. $\hat{X} > \hat{Y}$, otherwise, in the case where $\hat{X} < \hat{Y}$, it can be demonstrated in the same way.

$$dist(\hat{X}, \hat{Y}) = \beta_{\hat{X}-1} - \beta_{\hat{Y}}$$

By substituting into the inequality, we have:

$$l_1(\overline{X} - \overline{Y})^2 \geq l_1(\beta_{\hat{X}-1} - \beta_{\hat{Y}})^2$$

By removing l_1 from both sides, we have:

$$|\overline{X} - \overline{Y}| \geq |\beta_{\hat{X}-1} - \beta_{\hat{Y}}|$$

Since $\hat{X} > \hat{Y}$ and $|(\hat{X} - \hat{Y})| > 1$, we can drop the absolute value notation and rearrange the terms:

$$\overline{X} - \beta_{\hat{X}-1} \geq \overline{Y} - \beta_{\hat{Y}}$$

We know that : $\beta_{\hat{X}-1} \leq \overline{X} < \beta_{\hat{X}}$ and $\beta_{\hat{Y}-1} \leq \overline{Y} < \beta_{\hat{Y}}$ which implies that $\overline{X} - \beta_{\hat{X}-1} \geq 0$ and $\overline{Y} - \beta_{\hat{Y}} < 0$.
Then, the inequality always holds true, and this completes the proof.

8 Experiments

In this section, we report the results of experimental studies on the proposed ASAX segmentation approaches, *i.e.* ASAX_EN, ASAX_SAE and ASAX_DP.

8.1 Datasets and Experimental Settings

We compared the ASAX_EN and ASAX_SAE representations with the existing SAX representation on datasets selected for their particular (lack of) uniformity. Notice that SAX and its extensions in the literature use a fixed-size segmentation of the time domain.

The approaches are implemented in Python programming language and Numba JIT compiler is used to optimize machine code at runtime [1]. The experimental evaluation was conducted on a machine using Ubuntu 18.04.5 LTS operating system with 20 Gigabytes of main memory, and an Intel Xeon(R) 3,10 GHz processor with 4 cores.

We carried out our experiments on several real world datasets from the UCR Time Series Classification Archive [6]. Table 2 gives basic information about the datasets: name, type, length of the time series (number of values). Almost all selected datasets have non-uniform distributions over time domain (see Figure 10), else SyntheticControl that has a quasi uniform distribution.

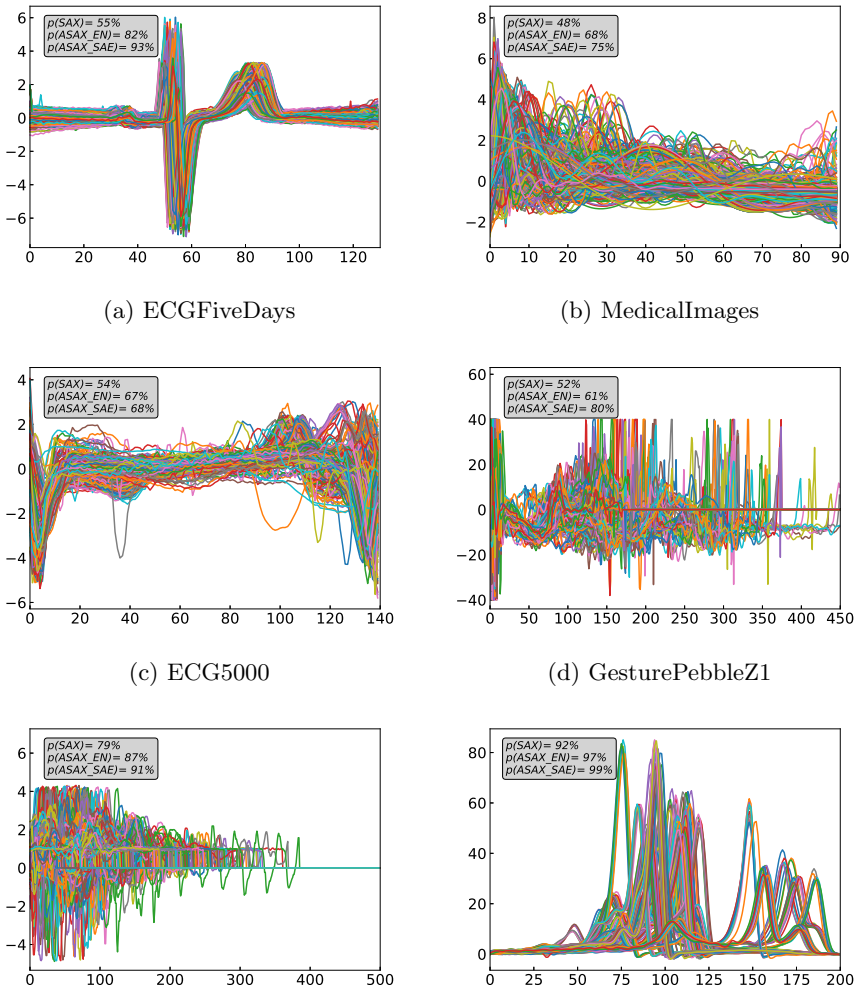
For each approach, the length w of the approximate representations is reduced to 10% of the original time series length. For the variable-size segmentation algorithms, ASAX_SAE is initialized by splitting the time domain into segments of length 2 and for ASAX_EN we set the default cardinality value to 32.

Name	Type	time series Length
AllGestureWiimoteZ	Sensor	500
ECG200	ECG	90
ECG5000	ECG	140
ECGFiveDays	ECG	130
Fungi	HRM	200
GesturePebbleZ1	Sensor	450
MedicalImages	Image	90
SonyAIBORobotSurface1	Sensor	70
SyntheticControl	Simulated	60

Table 2: Datasets basic information

In the experiments, we measure the precision of the compared approaches in similarity search, by applying a k-Nearest Neighbor (k-NN) search, as detailed in Subsection 8.2. For ASAX variable-size segmentation algorithms, we measure the time cost of the variable-size segmentation in Subsection 8.3.

[1] Our code is available at : `https://github.com/lamiad/ASAX`

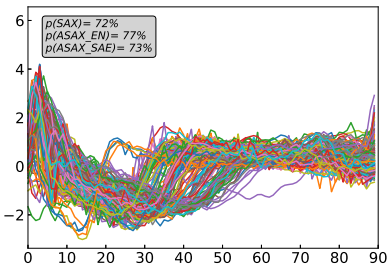


(a) ECGFiveDays (b) MedicalImages

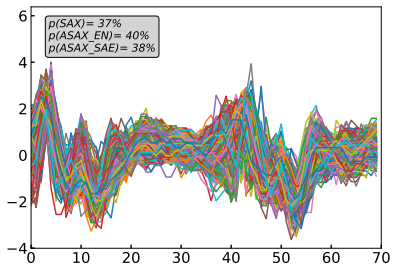
(c) ECG5000 (d) GesturePebbleZ1

(e) AllGestureWiimoteZ (f) Fungi

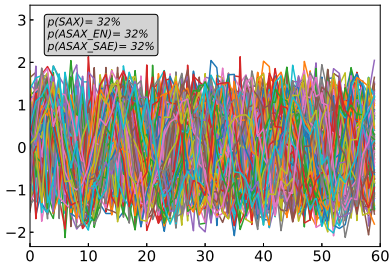
Fig. 10: The data distribution of the tested datasets, and the precision results for each dataset. p(SAX), p(ASAX_EN) and p(ASAX_SAE) show the precision of SAX, ASAX_EN and ASAX_SAE respectively.



(g) ECG200



(h) SonyAIBORobotSurface1



(i) SyntheticControl

Fig. 10: The data distribution of the tested datasets, and the precision results for each dataset. p(SAX), p(ASAX_EN) and p(ASAX_SAE) show the precision of SAX, ASAX_EN and ASAX_SAE respectively.

8.2 Precision of k-Nearest Neighbor Search

In this part of experiments, we compare the quality of ASAX and SAX representations on different datasets described in Table 2 by measuring the precision of the approximate k-NN search for both of the two approaches. The precision reported for each dataset represents the average precision for a set of arbitrary random queries taken from this dataset. The search precision for each query Q from a dataset D is calculated as follows :

$$p = \frac{|AppkNN(Q,D) \cap ExactkNN(Q,D)|}{k}$$

where *AppkNN(Q,D)* and *ExactkNN(Q,D)* are the sets of approximate k nearest neighbors and exact k nearest neighbors of Q from D, respectively. *AppkNN(Q,D)* is obtained using DR_f distance measure for SAX and DR_v for the ASAX representation and the set *ExactkNN(Q,D)* contains the k-NN of Q using the euclidean distance *ED*. *AppkNN(Q,D)* and *ExactkNN(Q,D)* use a linear search that consists in computing the distance from the query point Q to every other point in D, keeping track of the "best so far" result.
The precision results are reported in Figure 10 where each dataset is plotted with the precision obtained (as percentage) for all approaches. The plots show the shape of the different time series of each dataset and we can notice that the distribution of time series over the time domain varies from one dataset to another. There are some datasets for which the distribution is quite balanced, those which undergo some variations and others whose variation increases a lot. We have noticed that the more the distribution of the data is unbalanced the more the gain is important. Let us take for example the *ECGFiveDays* dataset presented in Figure 10a and *SyntheticControl* shown in Figure 10i. On the first one, we were able to achieve a precision of 93% for ASAX_SAE, 82% for ASAX_EN while it is 55% for SAX, which is a significant gain in precision. This higher precision for our approaches is due to the variable-size segmentation which creates segments in the parts that undergo a significant variation (from time point 45 to 95 in *ECGFiveDays*), allowing ASAX_SAE and ASAX_EN to perform a better distribution of the segments according to information gain by creating several segments in the parts that undergo a significant variation that produces more accurate times series representations leading to a better result for the approximate kNN search.

For *SyntheticControl* we can see that the precision of the approximate k-NN search is the same for the three approaches which is 32%. In this dataset, the shape of the time series is balanced over the time, and the segmentations obtained by ASAX_SAE, ASAX_EN and SAX are the same, resulting in equivalent precision.
From the results, we can observe that ASAX_SAE approach performs better than ASAX_EN for most datasets, especially for the datasets with high variation.
Globally, the results suggest the effectiveness of our approaches and their advantage over traditional SAX when applied to time series, especially those with unbalanced distribution over the time domain.

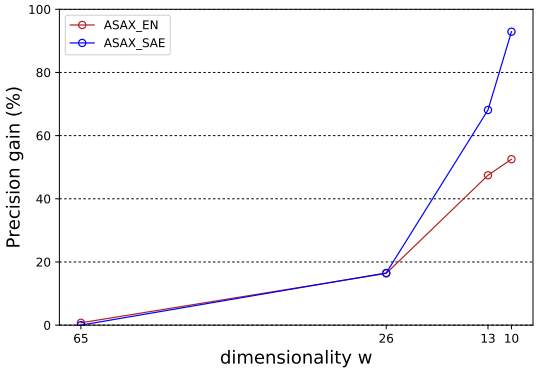


Fig. 11: Precision gain of ASAX_EN and ASAX_SAE compared to SAX on the *ECGFiveDays* dataset, for different values of segments w =[10,13,26,65].

Using *ECGFiveDays* dataset, Figure 11 shows the precision gain of ASAX_EN and ASAX_SAE compared to SAX, for different number of segments $w = [10,13,26,65]$. The initial length of the time series in the dataset is around 130. The precision gain is computed by dividing the difference between the precision of ASAX_EN (or ASAX_SAE) and SAX, over the precision of SAX.

The results illustrate that ASAX_SAE performs better than ASAX_EN for all values of w. They also show that for small values of w the precision gain is higher. The reason is that in these cases the loss of information with SAX is much higher than our approaches. This suggests the interest of using our approaches for high dimentionality reduction.

8.3 Execution time of variable-size segmentation algorithms

This subsection presents the time cost of the variable-size segmentation for our proposed algorithms. Figure 12 reports the segmentation time of ASAX_EN and ASAX_SAE on the datasets of our experiments. It does not concern SAX since SAX divides the time domain into segments of fixed size which does not require any computation beforehand. The segmentation time depends on both the number of time series in the dataset and their length. We can observe that the time cost of ASAX_SAE is always less than the one for ASAX_EN.

Let us now compare the variable-size segmentation time cost of the improved algorithm ASAX_DP with that of the basic algorithm ASAX_SAE. Figure 13 reports the performance gains of ASAX_DP compared to ASAX_SAE, in logarithmic scale. The variable-size segmentation time for the two methods is evaluated for the same datasets used previously. We can observe that ASAX_DP is much more efficient and allows to have significant performance gains.

Figure 14 reports the computation time of variable-size segmentation for ASAX_DP and ASAX_SAE over *AllGestureWiimoteZ* dataset, by varying the time series length. The runtime increases with the length of time series and, as

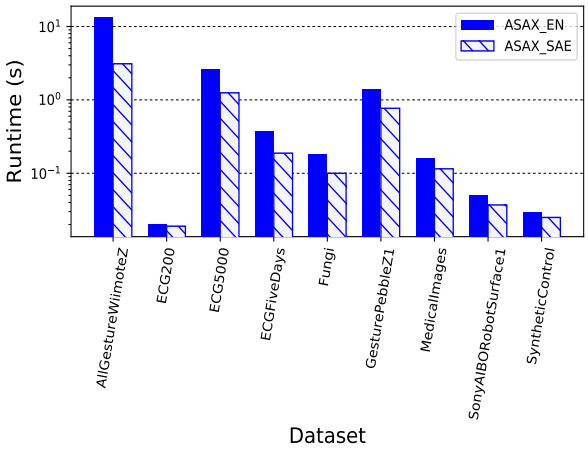


Fig. 12: Logarithmic scale. Runtime of ASAX_SAE and ASAX_EN segmentation algorithms for each dataset

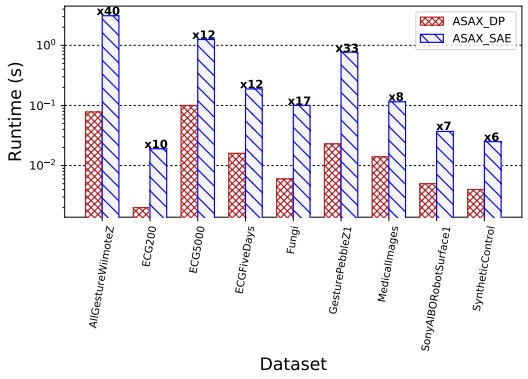


Fig. 13: Logarithmic scale. ASAX_DP's performance gain on ASAX_SAE in segmentation time for each dataset

one could expect. The basic approach ASAX_SAE takes much more time than ASAX_DP. Depending on time series length, ASAX_DP shows performance gains that can reach ×40 for the 1000 time series with length 500 of the *AllGestureWiimoteZ* dataset.
Figure 15 illustrate ASAX_DP's performance gain on ASAX_SAE in segmentation time for the same dataset, by varying the number of time series. As seen, the performance gains vary significantly depending on the number of time series.

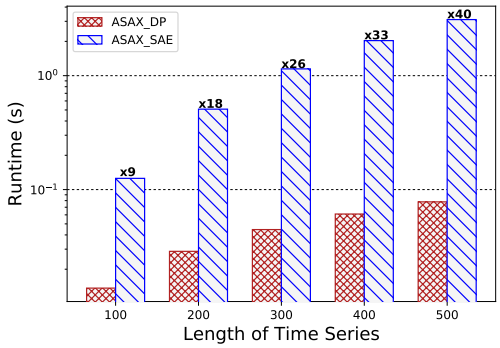


Fig. 14: Logarithmic scale. Variable-size segmentation time for ASAX_DP and ASAX_SAE as a function of time series length, over the *AllGestureWiimoteZ* dataset.

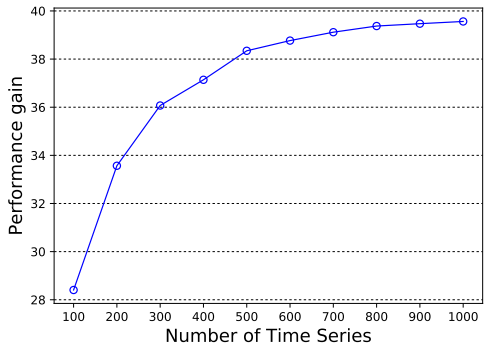


Fig. 15: ASAX_DP's performance gain on ASAX_SAE in segmentation time as a function of dataset size, over the *AllGestureWiimoteZ* dataset.

9 Conclusion

We addressed the problem of approximating time series, and proposed ASAX_EN and ASAX_SAE, new techniques for segmenting time series before their transformation into symbolic representations. Our solutions can reduce significantly the error incurred by possible splittings at different steps of the representation calculation, by taking into account the sum of absolute errors (ASAX_SAE) or the entropy (ASAX_EN). We also proposed an efficient algorithm, called ASAX_DP, for improving the execution time of our segmentation approach, by means of dynamic programming. We evaluated the performance of our segmentation approach through experiments using several real world datasets. The results suggest that the more the data distribution in the time domain is unbalanced (non-uniform), the greater is the precision gain of ASAX_EN and ASAX_SAE. For example, for the *ECGFiveDays* dataset that has a non-uniform distribution in the time domain, the precision of ASAX_SAE is 93% and 82% for ASAX_EN compared to 55% for SAX. Furthermore, the results illustrate the effectiveness of our dynamic programming algorithm ASAX_DP, *e.g.*, up to $\times 40$ faster than the basic ASAX_SAE algorithm over *AllGestureWiimoteZ* dataset.

References

1. Agrawal, R., Faloutsos, C., Swami, A.N.: Efficient similarity search in sequence databases. In: Lomet, D.B. (ed.) International Conference on Foundations of Data Organization and Algorithms (FODO). Lecture Notes in Computer Science, vol. 730, pp. 69–84. Springer (1993)
2. Camerra, A., Shieh, J., Palpanas, T., Rakthanmanon, T., Keogh, E.J.: Beyond one billion time series: indexing and mining very large time series collections with i SAX2+. Knowl. Inf. Syst. (2014)
3. Chakrabarti, K., Keogh, E., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. ACM Trans. Database Syst. **27**(2), 188–228 (2002)
4. Chan, K., Fu, A.W.: Efficient time series matching by wavelets. In: International Conference on Data Engineering (ICDE) (1999)
5. Cole, R., Shasha, D., Zhao, X.: Fast window correlations over uncooperative time series. In: International Conference on Knowledge Discovery and Data Mining (KDD). pp. 743–749 (2005)
6. Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G., Hexagon-ML: The ucr time series classification archive (October 2018), `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`
7. Djebour, L., Akbarinia, R., Masseglia, F.: Variable size segmentation for efficient representation and querying of non-uniform time series datasets. In: 37th ACM/SIGAPP Symposium on Applied Computing (SAC). pp. 395–402. ACM (2022)
8. Elsworth, S., Güttel, S.: ABBA: adaptive brownian bridge-based symbolic aggregation of time series. Data Min. Knowl. Discov. **34**(4), 1175–1200 (2020)

9. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: International Conference on Management of Data (SIGMOD) (1994)
10. Huijse, P., Estévez, P.A., Protopapas, P., Principe, J.C., Zegers, P.: Computational intelligence challenges and applications on large-scale astronomical time series databases. IEEE Comp. Int. Mag. **9**(3), 27–39 (2014)
11. Kashino, K., Smith, G., Murase, H.: Time-series active search for quick retrieval of audio and video. In: ICASSP (1999)
12. Keogh, E.J., Chakrabarti, K., Pazzani, M.J., Mehrotra, S.: Dimensionality reduction for fast similarity search in large time series databases. Knowl. Inf. Syst. **3**(3), 263–286 (2001)
13. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: International Conference on Management of Data (SIGMOD) (2003)
14. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing sax: A novel symbolic representation of time series. Data Min. Knowl. Discov. (2007)
15. Linardi, M., Palpanas, T.: ULISSE: ultra compact index for variable-length similarity search in data series. In: International Conference on Data Engineering (ICDE) (2018)
16. Linardi, M., Zhu, Y., Palpanas, T., Keogh, E.J.: Matrix profile X: VALMOD - scalable discovery of variable-length motifs in data series. In: International Conference on Management of Data (SIGMOD) (2018)
17. Linardi, M., Zhu, Y., Palpanas, T., Keogh, E.J.: VALMOD: A suite for easy and exact detection of variable length motifs in data series. In: International Conference on Management of Data (SIGMOD) (2018)
18. Lkhagva, B., Suzuki, Y., Kawagoe, K.: New time series data representation esax for financial applications. In: Workshops of International Conference on Data Engineering (ICDE) (2006)
19. Palpanas, T.: Data series management: The road to big sequence analytics. SIGMOD Record **44**(2), 47–52 (2015)
20. Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under dynamic time warping. In: International Conference on Knowledge Discovery and Data Mining (KDD) (2012)
21. Raza, U., Camerra, A., Murphy, A.L., Palpanas, T., Picco, G.P.: Practical data prediction for real-world wireless sensor networks. IEEE Trans. Knowl. Data Eng. (accepted for publication, 2015). https://doi.org/10.1109/TKDE.2015.2411594
22. Shasha, D.: Tuning time series queries in finance: Case studies and recommendations. IEEE Data Eng. Bull. **22**(2), 40–46 (1999)
23. Shieh, J., Keogh, E.: isax: Indexing and mining terabyte sized time series. In: International Conference on Knowledge Discovery and Data Mining (KDD) (2008)
24. Soldi, S., Beckmann, V., Baumgartner, W.H., Ponti, G., Shrader, C.R., Lubinski, P., Krimm, H.A., Mattana, F., Tueller, J.: Long-term variability of AGN at hard X-rays. Astronomy and Astrophysics - A&A **563**(A57), 16 (Mar 2014). https://doi.org/10.1051/0004-6361/201322653, `https://hal.archives-ouvertes.fr/hal-01171251`
25. Sun, Y., Li, J., Liu, J., Sun, B., Chow, C.: An improvement of symbolic aggregate approximation distance measure for time series. Neurocomputing **138**, 189–198 (08 2014)
26. Yahyaoui, H., Al-Daihani, R.: A novel trend based sax reduction technique for time series. Expert Systems with Applications **130** (04 2019)

27. Ye, L., Keogh, E.J.: Time series shapelets: a new primitive for data mining. In: International Conference on Knowledge Discovery and Data Mining (KDD) (2009)
28. Yin, H., Yang, S.q., Zhu, X.q., Ma, S.d., Zhang, L.m.: Symbolic representation based on trend features for knowledge discovery in long time series. Frontiers of Information Technology Electronic Engineering **16**, 744–758 (09 2015)
29. Zhang, H., Dong, Y., Xu, D.: Entropy-based symbolic aggregate approximation representation method for time series. In: IEEE Joint Int. Information Technology and Artificial Intelligence Conference (ITAIC). pp. 905–909 (2020)
30. Zoumpatianos, K., Palpanas, T.: Data series management: Fulfilling the need for big sequence analytics. In: International Conference on Data Engineering (ICDE) (2018)