Parallel Techniques for Variable Size Segmentation of Time Series Datasets

Lamia Djebour, Reza Akbarinia, Florent Masseglia

Parallel Techniques for Variable Size Segmentation of Time Series Datasets

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 these series, usually with loss. This incurs approximate comparisons of time series where precision is a major issue.In this paper, we propose a new parallel approach for segmenting time series before their transformation into symbolic representations. It can reduce significantly the error incurred by possible splittings at different steps of the representation calculation, by taking into account the sum of squared errors (SSE). 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 technique can improve significantly the time series representation quality.

Keywords: Time Series · Representations · Information Retrieval

1 Introduction

Time series have attracted an increasing interest due to their wide applications in many domains. The 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 [6,1,4,7,5,11,2] that challenges knowledge discovery.

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 high dimensionality. This is why time series approximation is often used as a means to allow fast similarity search. SAX [8] 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 (PAA approach). 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. 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 is involved in case of large sets of time series. Therefore, we propose efficient parallel techniques using GPUs for improving the execution time of our segmentation algorithm. In this paper, we make the following contributions:

- We propose a new representation technique, called ASAX_SSE, that allows obtaining a variable-size segmentation of time series with better precision in retrieval tasks thanks to its lower information loss. Our representation is based on SSE measurement for detecting what time intervals should be split.
- We propose a lower bounding method that allows approximating the distance between the original time series based on their representations in ASAX_SSE.
- We propose efficient parallel algorithms for improving the execution time of our segmentation approach using GPUs.
- We implemented our approach and conducted empirical experiments using more than 120 real world datasets. The results suggest that ASAX_SSE can obtain significant performance gains in terms of precision for similarity search compared to SAX. They illustrate that the more the data distribution in the time domain is unbalanced (non-uniform), the greater is the precision gain of ASAX_SSE. For example, for the *ECGFiveDays* dataset that has a non-uniform distribution in the time domain, the precision of ASAX_SSE is 93% compared to 55% for SAX.

The rest of the paper is organized as follows. In Section 2, we define the problem we address. In Section 3, we describe the details of ASAX_SSE representation, and in Section 4 we present parallel versions of ASAX_SSE. In Section 5, we present the experimental evaluation of our approach. Finally, we discuss the related work in Section 6 and give our conclusion in Section 7.

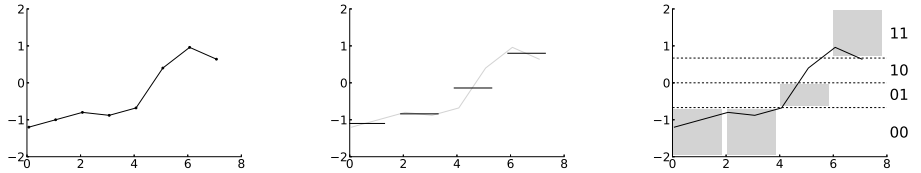
2 Problem Definition and Background

We first present the background about SAX representation, and then define the problem we address. 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 $t=1,2,...,n$. 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.

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 [6]: $ED(X,Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. The Euclidean distance is one of the most popular similarity measurement methods used in time series analysis.

The SAX representation is based on the PAA representation [8] 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.



(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. 1: 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 1a in the representation of Figure 1c is [first, first, second, fourth] (which becomes [00,00,01,11] in binary).

Example 1. Figure 1b shows the PAA representation of X, the time series of Figure 1a. 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. 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 1c, we have converted the time series X to SAX representation with size 4, and cardinality 4 using the PAA representation shown in Figure 1b. We denote 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 Problem Statement

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 goal is to propose a variable-size segmentation of the time domain that minimizes the loss of information in the time series representation. Formally, 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 Adaptive SAX based on the representation's Sum of Squared Errors (ASAX_SSE)

We propose ASAX_SSE, a variable-size segmentation technique for time series representation. To create a segmentation with minimum information loss on time series approximation, ASAX_SSE divides the time domain based on the Sum of squared errors (SSE) value of the representation.

In the rest of this section, we first describe the notion of Sum of Squared Errors (SSE) for the time series representation. Then, we describe our algorithm for creating the variable-size segments based on this measurement. Finally, we present our method for measuring the lower bound distance between time series in the proposed representation. This lower bounding is useful for efficient evaluation of kNN queries.

3.1 Sum of Squared Errors (SSE)

In Statistics, Sum of Squared Errors (SSE) is defined as the sum of the squares of the errors. In other words, SSE is the sum of the squared differences between the actual and the estimated values. Formally, SSE is defined as follows.

Definition 3. *Given a vector X of n elements and a vector $\tilde{X}$ being the estimated values generated from X, SSE of the estimation is given by: $SSE(X,\tilde{X}) = \sum_{i=1}^{n}(x_i - \tilde{x}_i)^2$*

In our context, we calculate the SSE on the PAA representation obtained from the transformation of the original time series of a dataset according to a given segmentation. The SSE 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 SSE, the closer is the PAA representation to 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 SSE for each segment, that is in this case, the sum of the squared differences between each value (actual value) and its segment's mean (estimated value). In the next subsections, we show how to compute the SSE of a PAA representation considering only one segment (called LSSE) or all segments (called GSSE). As shown by experiments, using these two different SSE measurements may lead to different results in terms of precision and execution time.

3.2 SSE of PAA Representation Considering One Segment (LSSE)

Let $\overline{X}$ be the PAA representation of X with w segments. The LSSE (local SSE) of $\overline{X}$ for a particular segment is the sum of the squared errors for the time series values in this segment. Formally, LSSE of $\overline{X}$ for a segment s_i is computed as: $LSSE(s_i, \overline{x}_i) = \sum_{j=LB(s_i)}^{UB(s_i)} (x_j - \overline{x}_i)^2$ 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.

3.3 SSE of PAA Representation Considering All Segments (GSSE)

The global SSE (GSSE), is computed by taking into account all segments of the PAA representation $\overline{X}$: $GSSE(X, \overline{X}) = \sum_{i=1}^{w} \sum_{j=LB(s_i)}^{UB(s_i)} (x_j - \overline{x}_i)^2$ where $LB(s_i)$ and $UB(s_i)$ are the start and end time points of the segment s_i respectively.

3.4 Variable-Size Segmentation Based on SSE 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 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 k segments of length $size$. The default value of $size$ is 2. 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 SSE (MSSE) 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.

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

Algorithm 1: 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     msse = ∞
 6     for i=1 to k−1 do
 7         s = merge (s_i,s_{i+1})
 8         tempSegments = segments−{s_i,s_{i+1}}
 9         tempSegments = tempSegments⋃s
10         //merge segment i and segment i+1 in tempSegments
11         sse = 0
12         foreach ts in D do
13             sse = sse + SSE(ts)
14         if sse<msse then
15             segmentsToMerge = i
16             msse = sse
17     s = merge (s_{segmentsToMerge},s_{segmentsToMerge+1})
18     segments = segments−{s_{segmentsToMerge},s_{segmentsToMerge+1}}
19     segments = segments⋃s
20     k = k-1
21 return segments
```

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, a temporary set of segments *tempSegments* is created including the new segment and all previously created segments except s_i and s_{i+1} *i.e.*, except the two that have been merged (Lines 8, 9). Then, for each time series ts in the database D, the algorithm generates its PAA representation and calculates the corresponding SSE (Line 13) calling either GSSE function in the case that the entire PAA representation is considered for the error calculation, or LSSE function if the error is computed on segment s. Then, it adds the result of the computed SSE to *sse* (Line 13). After having calculated the sum of the SSE for the PAA representation of all the time series contained in D, if the SSE is less than the MSSE (minimum SSE) obtained so far, the algorithm sets i as the segment to be merged with the next one, and keeps the SSE of the representation (Lines 15, 16). This procedure continues by trying the merging of every two adjacent segments of *segments* at each time, and computing the SSE. The algorithm selects the merging whose SSE is the lowest, and updates the set of the segments by removing the selected segments, and inserting its merging to *segments* (Lines 17-19). Then, k, which stands for the number of current segments, is decremented by one (Line 20). The algorithm ends when k gets equal to the required number, *i.e.*, w.

Let us illustrate the principle of our algorithm using an example. For simplicity, we consider a dataset containing only a single time series and we calculate the approximation error on the entire time series representation using GSSE approach.

Example 3. Let us apply our algorithm on the time series X in Figure 2 by taking the initial size of 2 for the segments. The algorithm starts by dividing the time domain

into 4 segments of size 2. The next step is to reduce the number of segments from 4 to 3. To this purpose, the algorithm tests the merging of every two adjacent segments of the 4 existing segments, in order to find the one that has the minimum SSE. Three different scenarios are possible:

Scenario 1: The first scenario is shown in Figure 3a where s_1 and s_2 of the initial segmentation (shown in figure 2) are merged into one segment. We generate the PAA representation of X using the 3 segments, and then compute the SSE of this representation that is $SSE_1(X,\overline{X}) \approx 1.167$.

Scenario 2: This scenario is shown in Figure 3b in which s_2 and s_3 of the initial segmentation are merged. As for Scenario 1, we generate the PAA representation of X using the current segmentation. Here, $SSE_2(X,\overline{X}) \approx 1.915$.

Scenario 3: The last scenario is shown in Figure 3b, where we merge s_3 and s_4. For this segmentation, $SSE_3(X,\overline{X}) \approx 1.745$.

We have calculated the SSE for the three scenarios. Since we aim to minimize the SSE, we have to choose the minimum SSE value (MSSE), that is $MSSE = 1.167$ corresponding to the segmentation generated in Scenario 1. The latter is chosen for this iteration of our algorithm and we continue the next iterations, until the number of segment reaches w.

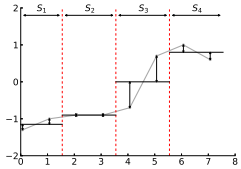
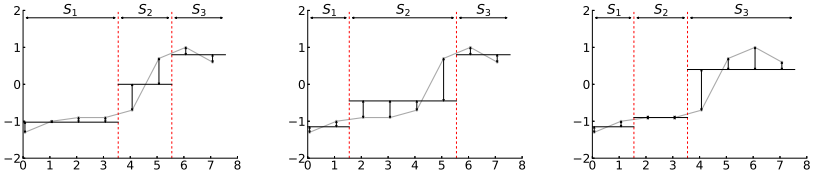


Fig. 2: PAA representation of a time series X of length 8 with 4 segments.



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

Fig. 3: The three different scenarios of ASAX_SSE segmentation with 3 segments. Scenario 1 is the one chosen because it provides the MSSE.

3.5 Lower Bounding of the Similarity Measure

SAX [9] defines a distance measure on the PAA representation of time series as described in Section 2.1. Given the representation of two time series, the DR_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_SSE.

Theorem 1. *Let X and Y be two time series. Suppose that by using ASAX_SSE 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 representations of X and Y in ASAX_SSE. Then, $DR_v(\overline{X},\overline{Y})$ gives a lower bounding approximation of the Euclidean distance between X and Y: $DR_v(\overline{X},\overline{Y}) = \sqrt{\sum_{j=1}^{w}((\overline{x}_j - \overline{y}_j)^2 \times l_j)}$*

Proof: The proof has been removed due to lack of space.

4 Parallel Versions of ASAX_SSE

We propose efficient parallel techniques using GPUs for improving the execution time of our segmentation algorithm. In our approach, the CPU controls the main loop of the segmentation computation process and does light operations, while the time-consuming tasks are parallelized on GPU, particularly the SSE computation on a dataset for a given segmentation. We propose two parallel versions of the algorithm using CUDA framework to provide a fast computation of the variable-size segmentation over long time series and/or large number of time series: 1) ASAX_DP that performs the parallelization on data; 2) ASAX_SP that makes the parallelization on segments.

4.1 Parallelization on Data

The main idea of our first parallel algorithm, called ASAX_DP (ASAX Data Parallel), is to divide the dataset into blocks (partitions), and to assign the SSE computation for the time series of each block to a core of the GPU.
Let us describe the proposed algorithm. Initially, the host (CPU) sends the whole dataset D to the GPU (this data transfer between the CPU and the GPU is done only once). Then, the host creates the initial segmentation *segments* by splitting the time domain into the k starting segments. Afterwards, in a loop, until the number of segments is more than w, it generates a candidate segmentation by merging 2 segments of the last validated segmentation. For each candidate segmentation, the GPU is used for computing SSE on D. For this, the host calls the GPU kernel that computes SSE in parallel operating on different time series of the different dataset blocks. In the kernel, each thread calculates the SSE on the time series of its block and stores the result in a shared array, called *sseArray*, that is sent back to the CPU. The host calculates the sum of the received results to get the SSE on D, and updates the MSSE (minimum SSE) if the SSE obtained in this iteration is less than the MSSE obtained until now. After testing all possible segmentations, it chooses the one that has the minimum SSE, updates the set of segments *segments* and decrements the current number of segments k by one. This process continues until k reaches the required number of segments w.

4.2 Parallelization on Segments

Here, we propose ASAX_SP (ASAX Segment in Parallel), a parallel algorithm in which the computations related to each possible merging of segments is done by a different GPU core. As shown by our experiments, this algorithm can be more efficient than the one presented previously in the cases where the time series are long (e.g., more than 1000 values per time series).

The initialization of this algorithm is the same as the algorithm presented in the previous subsection. The host starts by sending the dataset D to the GPU, and dividing the time domain into k starting segments to form the set *segments*.

Then, until the number of segments has not reached w, the host calls the GPU kernel to compute SSE on D of each possible segmentation in parallel. The number of launched threads is equal to the number of possible segmentations obtained when reducing the number of segments from k to $k-1$. In the kernel, each thread calculates its segmentation by merging two segments s_i and s_{i+1} where i is the thread position. The thread computes the SSE of the segmentation on the dataset D and stores the result in a shared array, called *sseArray*, according to its position. The result array is sent back to the CPU. Each element of the array represents the SSE for a candidate segmentation. The host selects the one having the lowest SSE value, and then updates *segments* and k. This process continues until k reaches w.

5 Experiments

In this section, we present the experimental evaluation of ASAX_SSE. We first describe the experimental setup. Then, in Subsection 5.2, we compare the precision of ASAX_SSE representation with that of the existing SAX representation. Finally, in Subsection 5.3, we evaluate the performance of the parallel versions of ASAX_SSE by measuring the execution time of the variable-size segmentation using GPUs.

5.1 Setup

All approaches are implemented with Python programming language. ASAX_SSE and SAX implementations use Numba JIT compiler to optimize machine code at runtime. The GPU-based part of the parallel algorithms is written in Numba [1].

The ASAX_SSE and SAX experiments were 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. The parallel experimental evaluation was conducted on an NVIDIA GeForce RTX 2080 Ti GPU card, equipped with 4 352 CUDA cores and 11 GB of memory installed in the same machine. We compare the proposed ASAX_SSE and SAX in terms of precision on all the real-world datasets available in the UCR Time Series Classification Archive [2]. We evaluate the performance of the parallel algorithms on two datasets taken from the same archive, the size of the datasets is increased to reach 1M by repeating the contained time series

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

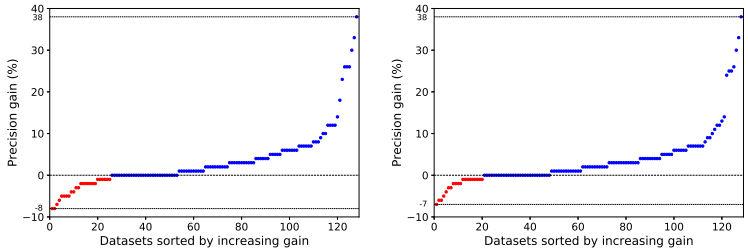
[2] `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`

multiple times. For each approach, the length w of the approximate representations is reduced to 10% of the original time series length and the variable-size segmentation algorithms are initialized by splitting the time domain into segments of length 2.

5.2 Precision of k-Nearest Neighbor Search

We compare the quality of ASAX_SSE and SAX representation on all 128 datasets of the UCR Time Series Classification Archive. For each dataset, we measure the precision of the approximate k-NN search as 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 : $p = \frac{|AppkNN(Q,D) \cap ExactkNN(Q,D)|}{k}$ 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 the DR_f distance measure for SAX and DR_v for the ASAX_SSE representation, while *ExactkNN(Q,D)* contains the exact k-NN results 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 4 where the precision gain/loss (as percentage) for ASAX_SSE compared to SAX precision is measured for each dataset. The integer part of the obtained precision is taken into consideration to compare the two methods. Figure 4a shows the precision results for ASAX_GSSE (*i.e.*, ASAX_SSE using GSSE) and Figure 4b those for ASAX_LSSE (*i.e.*, ASAX_SSE using LSSE). The results are illustrated using a scatter chart where the horizontal axis represents the dataset number and the vertical axis shows the precision gain/loss obtained. We observe a gain in precision for the large majority of datasets. We obtained a gain in precision for 80% of the datasets with ASAX_GSSE and 84% with ASAX_LSSE.
The distribution of time series over the time domain varies from one dataset to another. There are some for which the distribution is quite balanced, those which undergo some variations and others whose variation increases a lot. Figure 4 does not allow explaining the precision gain or loss since we need to have the visualisation of the time series for each datasets, for this, an analysis is done regarding the precision results obtained and the shape of data. We have noticed that the more the distribution of the data is unbalanced the more the gain is important. The maximum gain achieved is a significant 38% for both ASAX_GSSE and ASAX_LSSE methods, obtained for the *ECGFiveDays* dataset. This high gain is due to the unbalanced data distribution over the time domain on this dataset. We were able to achieve a precision of 93% for ASAX_SSE while it is 55% for SAX, because ASAX_SSE performed 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 k-NN search. We can see that for some datasets the computed gain is zero meaning equivalent precision for ASAX_SSE and SAX due to the balanced shape of the time series over the time domain. Regarding the few datasets where we obtain lower precision, the loss is relatively low (mostly near zero).



(a) Precision gain for ASAX_GSSE (b) Precision gain for ASAX_LSSE

Fig. 4: The precision gain for ASAX_GSSE and ASAX_LSSE compared to SAX. The obtained gain is up to 38% for both methods

Globally, our results suggest the effectiveness of our approach and its advantage over the state of the art when applied to time series especially those with unbalanced distribution over the time domain.

5.3 Scalability

This subsection presents the time cost of the variable-size segmentation for our proposed algorithms. We measure the variable-size segmentation time costs of the parallel algorithms ASAX_DP and ASAX_SP, and compare them to that of the variable-size segmentation for the sequential algorithm ASAX_SSE. The percentage of precision gain computed in the experiments described in the previous subsection shows that the gain obtained with the ASAX_GSSE approach is less than the one obtained with ASAX_LSSE. Furthermore, the evaluation of the time cost for ASAX_GSSE approach (sequential and parallel methods) showed that this approach is more time consuming than ASAX_LSSE. For these reasons, we present the results of our parallel algorithms, ASAX_DP and ASAX_SP, only using the LSSE measurement.

Figure 5 and Figure 6 report the performance gains of our parallel approaches compared to the sequential version of ASAX_LSSE. Figure 5 reports the variable-size segmentation time for the ASAX_DP and ASAX_LSSE with varying dataset size. The computation time increases with the number of time series for both algorithms. But, it is much lower in the case of ASAX_DP than that of the sequential ASAX_LSSE. The performance gains vary significantly depending on the number of time series. As seen, the gain reaches $\times 45$ for 1M of time series.

Figure 6 reports the computation time of variable-size segmentation for the ASAX_SP and ASAX_LSSE. Here we vary the time series length. The running time increases with the length of time series and, as one could expect, the sequential ASAX_LSSE takes much more time than ASAX_SP. Depending on time series length, ASAX_SP shows performance gains that can reach $\times 24$ for 1000 time series of length 2700.

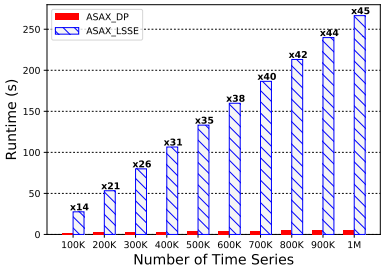


Fig. 5: Variable-size segmentation time for ASAX_DP and ASAX_LSSE as a function of dataset size. The original time series are of length 130.

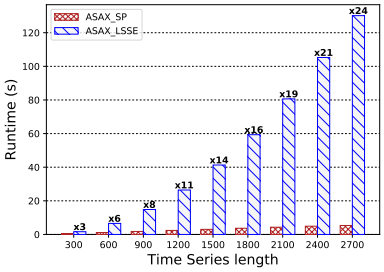


Fig. 6: Variable-size segmentation time for ASAX_SP and ASAX_LSSE as a function of time series length. The dataset size is fixed to 1000.

Figure 7 and Figure 8 compare the parallel segmentation computation time of our approaches. In Figure 7, we evaluate the two approaches with varying dataset size (number of time series) and fixed time series length. For this case, we observe that ASAX_DP is always faster than ASAX_SP. The results show that using ASAX_DP is advantageous in the case of databases of many small time series. In Figure 8, we vary the time series length and we fix the dataset size for the evaluation. We notice that when time series length $n=100$, ASAX_DP is a little faster than ASAX_SP, but when the length of time series increases, ASAX_SP becomes faster than ASAX_DP. The performance gain reaches $\times 7.5$ for time series of length 1000. ASAX_SP allows better performance gains when the database consists of few and long time series.

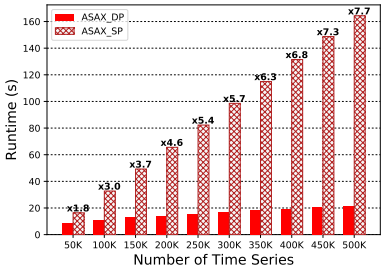


Fig. 7: Comparison of parallel segmentation time using ASAX_DP and ASAX_SP, as a function of dataset size. The original time series are of length 300.

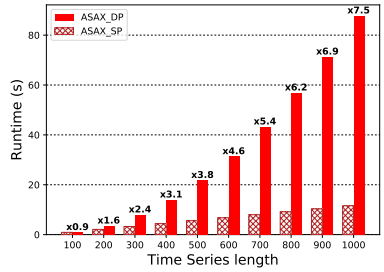


Fig. 8: Comparison of parallel segmentation time using ASAX_DP and ASAX_SP, as a function of time series length. The dataset size is fixed to 10 000.

6 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) [6], the discrete Fourier transformation (DFT) [1], discrete wavelets transformation (DWT) [4], piecewise aggregate approximation (PAA) [7], random sketches [5], Adaptive Piecewise Constant Approximation (APCA) [3], and symbolic aggregate approXimation (SAX) [9].

SAX [9] 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 [11,2]. For example, iSAX [11] 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.

There have been SAX extensions designed to improve the representation of each segment, while using the SAX fixed-size segmentation, e.g., [10,12,15]. 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 [14] and SAX_CP [13] 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.

To increase the quality of time series approximation, we propose an adaptive approach ASAX_SSE based on variable-length segmentation of time series by taking into account the sum of absolute error. Our approach is complementary to the existing SAX extensions, *e.g.*, in indexing based techniques or those that use the trend for representing the segments. For example, our variable-size segmentation can be used in iSAX, SAX_TD and SAX_CP for segmenting the time series.

7 Conclusion

We addressed the problem of approximating time series, and proposed ASAX_SSE, a new technique for segmenting time series before their transformation into symbolic representations. ASAX_SSE can reduce significantly the error incurred by possible splittings at different steps of the representation calculation, by taking into account the sum of squared errors (SSE). We also proposed two parallel algorithms for improving the execution time of ASAX_SSE using GPUs. We evaluated the performance of our segmentation approach through experimentation using more than 120 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_SSE. For example, for the *ECGFiveDays* dataset that has a non-uniform distribution in the time domain, the precision of ASAX_SSE is 93% compared to 55% for SAX. Furthermore, the results illustrate the effectiveness of our parallel algorithms, *e.g.*, up to $\times 45$ faster than the sequential algorithm for 1M time series.

References

1. Agrawal, R., Faloutsos, C., Swami, A.N.: Efficient similarity search in sequence databases. In: Proc. of the 4th Int. Conf. on FODO (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: Proc. of the ICDE (1999)
5. Cole, R., Shasha, D., Zhao, X.: Fast window correlations over uncooperative time series. In: KDD Conf. pp. 743–749 (2005)
6. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: Proc. of the SIGMOD (1994)
7. 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)
8. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: SIGMOD (2003)
9. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing sax: A novel symbolic representation of time series. Data Min. Knowl. Discov. (2007)
10. Lkhagva, B., Suzuki, Y., Kawagoe, K.: New time series data representation esax for financial applications. In: ICDE Workshops (2006)
11. Shieh, J., Keogh, E.: isax: Indexing and mining terabyte sized time series. In: KDD Conf. pp. 623–631 (2008)
12. 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)
13. Yahyaoui, H., Al-Daihani, R.: A novel trend based sax reduction technique for time series. Expert Systems with Applications **130** (04 2019)
14. 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)
15. 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)