Adaptive Segmentation Techniques for Efficient Representation of Time Series Datasets

Lamia Djebour

▶ **To cite this version:**

Lamia Djebour. Adaptive Segmentation Techniques for Efficient Representation of Time Series Datasets. Computation and Language [cs.CL]. Université de Montpellier, 2022. English. NNT : 2022UMONS040 . tel-03904591v2

HAL Id: tel-03904591

https://theses.hal.science/tel-03904591v2

Submitted on 5 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale I2S

Unité de recherche LIRMM, UMR 5506

Adaptive Segmentation Techniques for Efficient Representation of Time Series Datasets

Présentée par Lamia DJEBOUR
le 13/09/2022

Sous la direction de Florent MASSEGLIA
et Reza AKBARINIA

Devant le jury composé de

Omar BOUCELMA	Professeur, Univ. Aix-Marseille	Rapporteur
Thomas GUYET	CR, INRIA	Rapporteur
Anne LAURENT	Professeur, Laboratoire LIRMM	Examinatrice
Dennis SHASHA	Professeur, Univ. New-York	Examinateur
Reza AKBARINIA	CR, INRIA, Laboratoire LIRMM	Co-directeur
Florent MASSEGLIA	DR, INRIA, Laboratoire LIRMM	Directeur

UNIVERSITÉ
DE MONTPELLIER

ABSTRACT

Many applications in different domains generate time series data at an increasing rate. 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.

In recent years, there has been an explosion of interest in time series data mining. As a general rule, large time series come along with super-high dimensionality. As a consequence, it is difficult and inefficient to directly mine the raw time series without relying on dimensionality reduction. Therefore, the representation of the data is the key to efficient and effective solutions. Given this high data volumes in time series applications, or simply the need for fast response times, it is 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 thesis, we focus on the problem of segmenting time series before their transformation into symbolic representations. For this, we propose solutions to adaptively segment time series databases by adopting a variable segment size that depends on the time series distribution. These methods reduce significantly the information loss incurred by possible splittings at different steps of the representation calculation, particularly for datasets with unbalanced (non-uniform) distributions.

After reviewing the state of the art, we propose three novel approaches for efficiently segmenting time series datasets by means of variable size segments. First, we propose ASAX_EN a novel approach that performs the splitting based on the representation's entropy with an approximate algorithm using a top-down strategy.

Second, we propose ASAX_SSE approach that segments the time series by taking into account the sum of squared errors (SSE) with an approximate algorithm using a bottom-up strategy. This method provides high quality results in time series representation. Also, efficient algorithms for improving the execution time of our segmentation approach have been proposed.

Third, we propose EASAX_SSE our segmentation method that finds the time domain division that guarantees to minimize the SSE of the representation with an exact method.

Our solutions propose a lower bounding method that allows approximating the distance between the original time series based on their representations in these approaches. They have been evaluated using several real world datasets. The results illustrate that our techniques can significantly improve the time series representation quality.

Keywords

Time Series, Symbolic Representations, Segmentation, SAX, Entropy, Approximation Error, Similarity Search, Information Retrieval

RÉSUMÉ

De nombreuses applications dans différents domaines génèrent des données de séries temporelles à un rythme croissant. Le flux continu de données émises peut concerner des activités personnelles (par exemple, au moyen de compteurs intelligents ou de prises connectées pour la consommation d'électricité ou d'eau) ou professionnelles (par exemple, pour la surveillance de l'activité cardiaque ou à travers les capteurs installés sur les plantes par les agriculteurs). Il en résulte une production de données volumineuses et complexes, généralement sous la forme de séries temporelles.

Généralement, les bases de données de séries temporelles sont caractérisées par leur très grand volume. Par conséquent, il est difficile et inefficace d'exploiter directement les données de séries temporelles brutes sans avoir recours à la réduction de la dimensionnalité. Ce verrou motive l'étude de représentations alternatives, plus courtes, qui résument les séries d'origine avec une perte d'information acceptable. Les comparaisons de séries temporelles qui se basent sur ces représentations sont alors approximatives, ce qui fait de la précision un enjeu majeur.

Dans cette thèse, nous étudions le problème de la segmentation des séries temporelles avant qu'elles soient transformées en représentations symboliques. Pour cela, nous proposons des solutions de segmentation adaptative des séries temporelles en adoptant une taille de segment variable qui dépend de la distribution de ces séries. Ces méthodes réduisent de manière significative la perte d'information due aux découpages possibles dans les différentes étapes du calcul de la représentation, en particulier pour les ensembles de données dont les distributions sont non uniformes. Nous fournissons des garanties théoriques sur la borne inférieure des mesures de similitude entre séries temporelles, et nos résultats montrent que nos techniques peuvent améliorer considérablement la qualité de la représentation des séries temporelles.

Titre en français

Techniques de segmentation adaptative pour une représentation efficace des séries temporelles

Mots-clés

Séries temporelles, Représentations Symboliques, Segmentation, SAX, Entropie, Erreur d'Approximation, Recherche de Similarité, Extraction d'Informations

CONTENTS

LIST OF FIGURES

LIST OF TABLES

List of Algorithms

CHAPTER 1

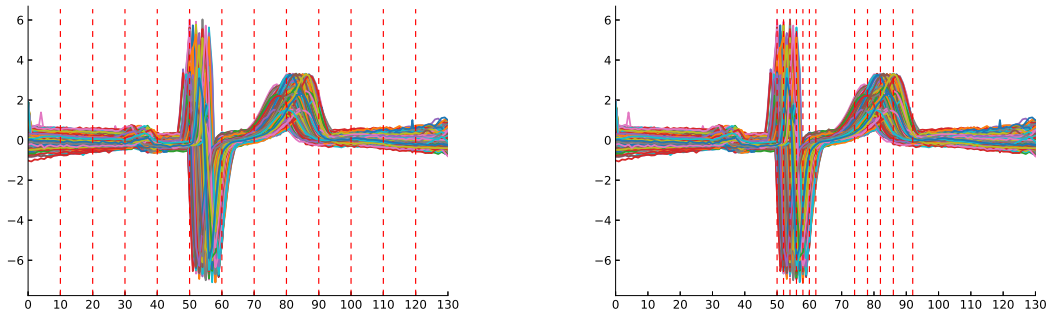
INTRODUCTION

1.1 Context

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 [14, 3, 7, 20, 10, 38, 5] that challenges knowledge discovery. 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 classification, clustering and motif discovery [31, 25, 48]. The problem of high dimensionality in time series data is the main obstacle for time series data mining, and mainly for defining a form of similarity measure based on human perception.

Because of the considered data volumes in such applications, similarity search can be slow on raw data. This is why time series approximation is often regarded as a means to allow fast computation of similarity search. SAX [23] is one of the most popular representations of time series, allowing dimensionality reduction on classic data mining tasks. SAX constructs symbolic representations by splitting the time domain into segments of equal size. This approximation model 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 "one size fits all" division into segments of fixed length is not advantageous.

To illustrate the impact of a fixed length division of the series into segments, let us consider Figure 1.1. It shows a set D of time series, taken from *ECGFiveDays* dataset of UCR Archive [11], where the time series length is 130. We can notice that there is almost no variation from time point 1 to 45 and from 95 to 130. On the other hand, the remaining part, from time point 45 to 95, shows an important variation in the data values. Figure 1.1a 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



(a) SAX (PAA) segmentation on D, with 13 segments

(b) ASAX_LSSE segmentation on D, with 13 segments

Figure 1.1: SAX segmentation Vs. ASAX_LSSE segmentation

these segments.

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 1.1b. The splitting of Figure 1.1b is the actual splitting obtained by our approach ASAX_SSE with a segment budget limited to 13. It would be rather counter-intuitive to merge segments 1-5 and 10-13 of Figure 1.1b, while it is the opposite for the same segment ranges in Figure 1.1a. This observation encourages us to provide solutions where the time intervals where data values show important differences would be split to create more segments, *e.g.*, between time point 50 and 90 in the dataset of this illustration. By proposing such a customized splitting, we aim at improving the performance of information retrieval algorithms that will rely on our data representation.

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 length of the time series. 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 improvement in the precision of kNN queries. This issue motivated us for proposing an adaptive representation aiming at minimizing information loss.

1.2 Contributions

The objective of this thesis is to develop new techniques in order to improve the quality of similarity search, and to achieve adaptive splitting as illustrated above. For this, we propose new approximation methods for time series that consider the time series shape and does the splitting based on some specific measures. These approaches allow reducing the information loss of the representation, and thus increasing the accuracy of time series representations leading to better precision during retrieval phases. Our main contributions are as following:

- **Variable size segmentation for efficient representation of non-uniform time series datasets based on entropy**. In this work, we propose ASAX_EN a novel and efficient method for time series that considers the information carried by the series and does the splitting by means of segments of variable size on the time domain by measuring the entropy of symbolic representations. This algorithm chooses between different possible splittings at each step of the representation computation using a top-down approach. In this method, we create an initial segmentation and then the top-down approach refines the segmentation considering every possible partitioning of the time serie and splitting it at the best location. The choice of division for possible splittings at different steps of the representation calculation is based on the entropy measurement. The results obtained in the experiments for this approach show the that this approximate solution allows reducing information loss and thus increasing the accuracy of time series representations, particularly for datasets with unbalanced (non-uniform) distributions.

- **Optimized techniques for time series segmentation based on the approximation error**. In this work, we study the problem of finding the variable-size segmentation that minimizes the approximation error of the time series representation. We propose ASAX_SSE, an efficient solution that allows obtaining a variable-size segmentation of time series based on *sum of squared error (SSE)* that measures the error of the representation. We adopted a bottom-up algorithm for this approach in order to achieve better results in accuracy of time series representations since it creates the finest possible approximation of the time series. After the initialisation with a large number of fine segments, they are then gradually merged depending on the lowest cost pair of adjacent segments. To improve the execution time of this segmentation approach, we propose ASAX_Dyn that finds the adaptive segmentation by means of dynamic programming. We also propose efficient parallel algorithms, called PASAX_DP and PASAX_SP, that improve the execution time of our segmentation approach using GPUs. The experimental results illustrate the good performance of ASAX_SSE, which confirms the effectiveness of our approach.

- **Time series representation based on the exact error**. In this work, we propose EASAX_SSE an exact algorithm for solving the segmentation problem based on

the SSE measurement to achieve optimal segmentation with our representation. This method finds the adequate variable-size segments such that the combined error of all segments is minimal. The optimal segmentation of time series is defined as the segmentation that results in the lowest segmentation error in relation to other possible combinations of segmentation. This approach provides excellent performance gains in terms of precision for similarity search, although its execution time is higher than the approximate techniques.

1.3 Organization of the Thesis

The rest of the thesis is organized as follows.

In Chapter 2, we review the state of the art. It is divided into four main sections: In Section 2.1, we define the time series, and give a general overview of the main techniques for mining time series. In Section 2.2, we introduce the time series representation techniques and solutions that have been proposed to deal with the problem of high dimensionality in time series data. In Section 2.3, We present the detail of SAX, one of the most popular methods that have been proposed in the literature for dimentionality reduction. Finally, we discuss the major limitation of SAX and present some related research initiatives work on enhancing the SAX representation.

In Chapter 3, we deal with the problem of time series segmentation. In Section 3.3, we propose ASAX_EN that allows obtaining a variable-size segmentation of time series with better precision in retrieval tasks based on entropy measurement and using a top-down algorithm. In Section 3.5, we assess the efficiency of our proposed approach by carrying out experiments with several datasets.

In Chapter 4, we propose our second solution to deal with the problem of segmenting time series called ASAX_SSE. In Section 4.1, we propose the basic algorithm ASAX_SSE that allows obtaining a variable-size segmentation of time series using a bottom-up strategy and based on the SSE (Sum of Squared Error) of the representation. In Section 4.2 we propose an efficient algorithm called ASAX_Dyn for improving the execution time of our segmentation approach, by means of dynamic programming. In Section 4.3, we propose efficient parallel algorithms for improving the execution time of our segmentation approach using GPUs. In Section 4.4, we validate our approach by carrying out various experiments using more than 120 datasets.

In Chapter 5, we propose an exact approach for time series segmentation using a dynamic programming algorithm. In Section 5.2, we propose our exact segmentation technique, called EASAX_DP, that finds the exact variable size segmentation which minimizes the SSE of the representation. In Section 5.3, we validate our proposal through different experiments using real world datasets.

Finally in Chapter 6, we conclude our work and give suggestions for further improvements and possible directions of research.

CHAPTER 2

STATE OF THE ART

In this chapter, we introduce the basics and the necessary background of this thesis. It is organized as follows. First, we formally define time series. Then, we introduce the problem of similarity search in time series datasets, and discuss the main existing techniques and methods that have been proposed for this problem. Afterwards, we focus on time series representation, and discuss the most efficient approaches proposed in the literature.

2.1 Time Series Data Mining

The increasing use of time series data has initiated a great deal of research and development attempts in the field of data mining. Mining is the final goal to discover hidden information or knowledge from either the original or the transformed time series data.

2.1.1 Time Series

Definition 1 *A time series T is a series of n data points indexed in time order.*

$$T = (t_1, t_2, ..., t_n), t_i \in \mathbb{R}$$

Most commonly, a time series is a sequence taken at evenly-spaced intervals. Thus it is a sequence of discrete-time data. A time series is often the result of the observation of an underlying process. Examples of time series data are weather records, economic indicators and patient health evolution metrics. An obvious example for a time series is the daily closing value of the Dow Jones Industrial Average. Figure 2.1 shows an example of a time series from an earthquake occurred few kilometers from Greve in Chianti, Italy [1].

2.1.2 Time Series Data Mining Tasks

Time series data mining has attracted an increasing interest due to its wide applications in many domains. Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Nowadays, several techniques have been developed and applied to time series data, *e.g.* , clustering , classification, indexing, etc. This section provides an overview of the main tasks that have attracted wide research interest in time series data mining.

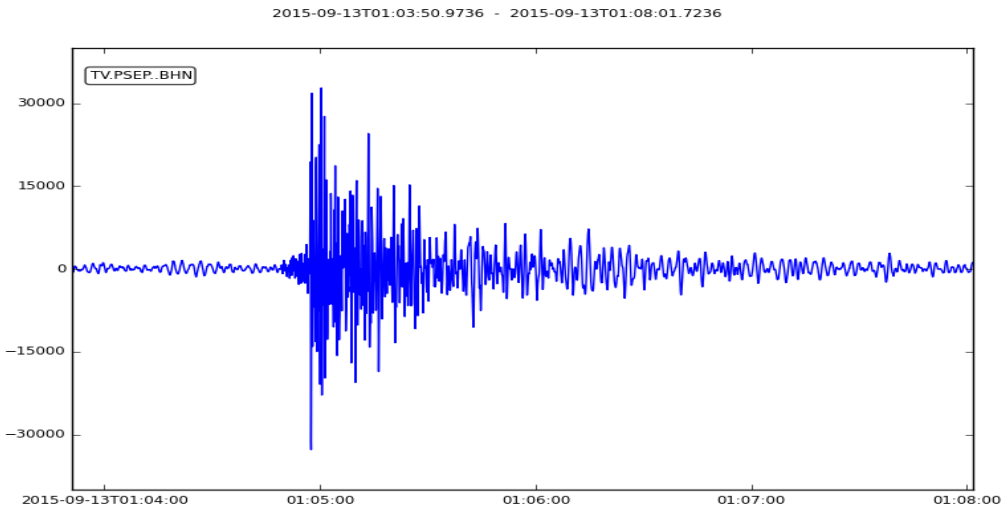


Figure 2.1: Examples of time series data relative to seismic signal from an earthquake occurred few kilometers from Greve in Chianti, Italy [1]

2.1.2.1 Similarity Search and Indexing

Time series indexing schemes are designed for efficient time series data organization and especially for fast similarity search in large databases. Given a query time series Q the goal is to retrieve similar time series from a collection using a similarity measure. Using a sequential or linear scan of a whole database of raw data to find the most similar time series to a given query is very costly. Therefore, in order to speedup the similarity search over time series, indexing techniques have been developed. To speed up sequence retrieval and to deal with the high dimensionality of this data, first, dimensionality reduction techniques are applied on the raw data and then, the results are stored in index structures adapted for the given representation.

2.1.2.2 Clustering

Clustering is the process of finding natural groups, called clusters, in a dataset under some similarity or dissimilarity measure. The objective is to find the most homogeneous clusters that are as distinct as possible from other clusters. In other words, those groups should minimise intra cluster variance while maximising inter cluster variance. In [33] the clustering is defined as the unsupervised version of classification since the instances are not previously labeled with class.

2.1.2.3 Classification

Classification of time series is a further traditional data mining task. While clustering aims at finding the naturally present groups in a dataset, classification creates a mapping from given unlabeled time series to existing classes (predefined in advance). Classification approaches first build a classification model based on a training data set containing labeled observation or time series. Then, the built models are used to

predict the label of a new, unlabeled observation or sequence of a time series. Classification as data mining task is assigned to the supervised learning algorithms in the machine learning jargon.

2.1.2.4 Segmentation

Time series segmentation can be considered either as a preprocessing step for many data mining tasks. It is also considered as a discretization problem. The segmentation (summarization) task aims at creating an accurate approximation of time series, by reducing its dimensionality while retaining its essential features.

2.1.2.5 Anomaly Detection

The detection of anomalies seeks to find abnormal subsequences in a series. Given a time series T and a model of its normal behavior, the goal is to find all subsequences of T which contain anomalies, *i.e.* which do not fit the model.

2.1.2.6 Motif discovery

Motif discovery is another typical task in the field of time series data mining, and consists in finding the subsequences, frequent patterns, or motifs that appear recurrently in a longer time series. This idea was transferred from gene analysis in bioinformatics. Motif discovery comes usually along with clustering methods, as the occurrence frequency of patterns in time series subsequences can naturally be found by clustering.

2.2 Time Series Representations and Distance Measures

Many time series datasets are usually large and high dimensional, and this is a major issue with time series data mining. Similarity search over these data is computationally too expensive, so it is important to estimate the distance between two time series very quickly. Therefore the analytical tasks are often performed not on the raw data itself, but on a more abstract representation by reducing the dimension (i.e. the number of data points). In order to reduce execution time and storage space, many high level representations or abstractions of the raw time series data have been proposed. In this section, we present these representations techniques and tools for large time series. We also introduce similarity measures that are the backbone of many data mining applications.

2.2.1 Time Series Representations

Definition 2 ***Time series representations*** *The representation of a time series $X = (x_1, ..., x_n)$ of length n is a model $\tilde{X}$ of reduced dimensionality w ($w << n$), such that $\tilde{X}$ approximates X.*

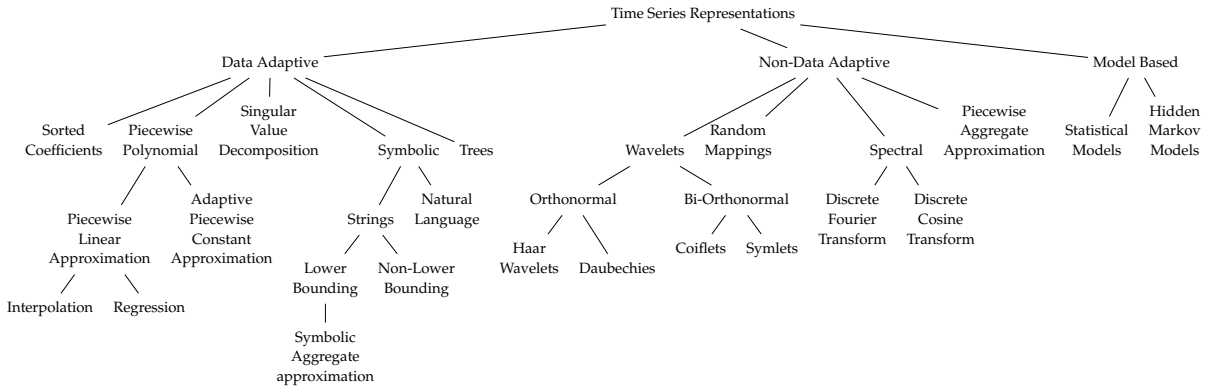


Figure 2.2: A hierarchy of various time series representations found in the literature. The leaf nodes refer to the actual representation, and the internal nodes refer to the classification of the approach

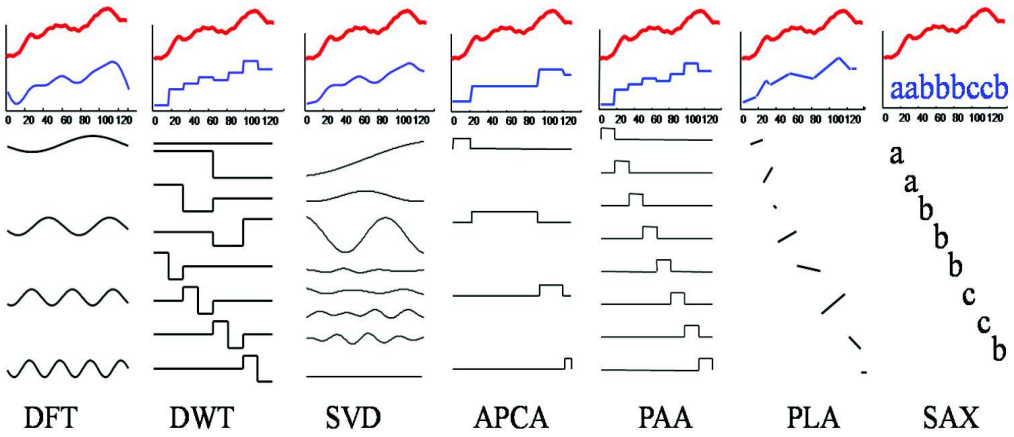


Figure 2.3: Example of techniques that can significantly reduce the dimensionality of time series [33]

In the literature, many techniques have been proposed that represent time series with reduced dimensionality, and then apply a distance function to measure the similarity between transformed time series. For example, Discrete Fourier Transformation (DFT) [3], Single Value Decomposition (SVD) [14], Discrete Wavelet Transformation (DWT) [7], Piecewise Aggregate Approximation (PAA) [20], Adaptive Piecewise Constant Approximation (APCA) [6], Chebyshev polynomials (CHEB) [4], Piecewise Linear Approximation (PLA) [37, 9] and Symbolic Aggregate approXimation (SAX) [24]. This latter takes the PAA representation as an input and discretizes it into a small alphabet of symbols as we will show later. Figure 2.3 shows examples of such techniques that can significantly reduce the time and space.

Time series representations can be classified into three main categories of dimensionality reduction techniques according to the kind of transformations applied. In this section, we will discuss these approaches and outline the most relevant methods. Figure 2.2 shows the different categories proposed by [27, 23, 12, 13] and some of their methods.

2.2.1.1 Non Data-Adaptive

Non data-adaptive representation techniques use the same transformation parameters regardless the features of the the underlying data for dimensionality reduction. So, the transformation parameters are fixed a priori. These techniques are based on the idea of spectral decomposition, that any time sequence can be represented by a finite number of trigonometric functions. Operating in the frequency domain is valid as the Euclidean distances between two time series is the same in the time and frequency domains, and hereby preserve distances. For example, Discrete Fourier Transform (DFT) [3] projects a time series into the frequency domain, by decomposing the series into a finite number of sine and cosine waves which are represented by complex coefficients, the Fourier coefficients. Only the first few waves appear to be dominant and therefore are kept for lower bounding of the actual distances, and the rest can be omitted without any great impact on the reconstruction error. Similar to DFT, wavelet coefficients give local contributions to the reconstruction of the signal, while Fourier coefficients always represent global contributions to the signal over all the time. [30] demonstrate that a large class of wavelets are applicable for time series dimension reduction. One popular wavelet is the so called "Haar" wavelet proposed to use in the time series data mining context by [39]. Haar wavelet is the simplest type of wavelet. In discrete form, Haar wavelets are related to a mathematical operation called the Haar transform. The Haar transform serves as a prototype for all other wavelet transforms. The Haar transform is a series of averaging and differencing operations on a time series [7]. The average and difference between every two adjacent data points are computed and used for representing the time series. A completely different approach, especially tailored to time series data mining, is the Piecewise Aggregate Approximation (PAA) proposed by [20, 42]. The very simple idea appears to be competitive in comparison to the more sophisticated transformations [27]. In this approach, the time series is divided into w equal sized segments. The mean value of the data falling within a segment is calculated and a vector of these values called "PAA coefficients" becomes the data-reduced representation.

2.2.1.2 Data-Adaptive

Data adaptive representation techniques are (more) sensitive to the nature of the data at hand. The transformation parameters are chosen depending on the available data and not fixed a priori as for non data-adaptive techniques. By adding a data-sensitive selection step, almost all non data-adaptive techniques can become data-adaptive approaches. As DFT and DWT, Singular Value Decomposition (SVD) [14] is another transformation-based approach, while DFT and DWT apply local transformations, SVD acts globally. SVD examines the entire data and rotates the axes to maximise variance along the first few dimensions [34]. Although SVD is an optimal transformation in the sense of minimal reconstruction error [20], it requires the computation of eigenvalues for large data matrices making it computationally very expensive [13, 6] PLA [37] is a widely used approach for the segmentation task. The set of polynomial coefficients can be obtained either by interpolation [21] or regression [16]. Many

derivatives of this technique have been introduced. In [6], the authors propose an extension of the PAA approach, called Adaptive Piecewise Constant Approximation (APCA). While PAA stores the means of consecutive fixed length segments, APCA allows the segments to be of different length, thus more adapting to the data. In this way, we try to minimize the individual reconstruction error of the reduced time series. A very frequently used and probably the most popular dimensional reduction and indexing technique is based on a symbolic representation and called SAX. It was introduced by [23]. Based on the same underlying idea as PAA, SAX (Symbolic Aggregate Approximation) calls on equal frequency histograms on sliding windows to create a sequence of short words. SAX is said to outperform all other dimensionality reduction techniques (we will present this approach in detail in Section 2.3). An extension of SAX, called indexable Symbolic Aggregate approXimation (iSAX) [38], has been proposed to make fast indexing possible by providing zero overlap at leaf nodes. It allows extensible hashing and indexing of terabyte sized time series.

2.2.1.3 Model-based

The paradigm of model based representation techniques builds on the idea that the observed time series are usually produced by an underlying model. Therefore, the goal of these techniques is to find the parameters of the corresponding underlying model. Dimensionality reduction is obtained by representing the time series by the model's parameters, used to produce the series. As a consequence, time series similarity is measured based on the model parameters [13]. Several parametric temporal models can be considered, such as statistical modeling via feature extraction [28], or more complex models such as Auto Regressive Moving Average (ARMA) [19], Markov Chains (MCs) [35] or Hidden Markov Models (HMM) [29]. The objective of those approaches is not often the explicit reduction of dimensionality, but the improvement of similarity distances for further tasks such as clustering or classification [19, 29].

2.2.2 Similarity Measures

Similarity measures indicate the level of similarity/dissimilarity between time series. Similarity measure is of fundamental importance for almost all data mining tasks (i.e. clustering, classification, pattern discovery, etc.).

Definition 3 ***Similarity measure*** *A similarity measure $D(X, Y)$ between the time series X and Y is a function that takes two times series as input and returns their distance d ($d \geq 0$).*

Similarity measures impose the major capacity constraints on time series data mining algorithms [31]. The faster the similarity measure computation algorithm, the faster is the whole time series data mining procedure.The list of approaches for dealing with time series similarity is vast. Many different systematizations of time series similarity measures exist.
In [13], four categories of similarity measures are defined in order to calculate the similarity of the time series:

- Shape-based distances : that compare the overall appearance of the time series.
- Feature-based distances : that extract the features that usually describe the time independent aspects of the series that are compared with static distance functions.
- Model-based distances : that fit a model to the data and measure the similarity by comparing the models.
- Compression-based distances : that analyze how well time series can be compressed alone and together.

Furthermore, we distinguish between elastic and lock-step similarity measures. Lock-step measures compare the i^{th} point of time series X to the i^{th} point of time series Y. In contrast to that, elastic similarity measures allow a flexible comparison and additionally compare one-to-many or one-to-none points of X to Y. In this section, we present the most popular representative examples of different families of time series similarity measures: lock-step measures (Euclidean distance) and elastic measures (Dynamic Time Warping).

2.2.2.1 Euclidean distance

The simplest way to estimate the similarity between two time series is to use any L_p norm which correspond to the group of lock-step measures such that :

$$L_p(X,Y) = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

where X and Y are time series of length n and x_i and y_i are the i^{th} element of time series x and y, respectively. p is a positive integer that denotes the norm in use.
When $p = 2$ we obtain the Euclidean distance, one of the most straightforward similarity measure used in time series similarity measures, favored by its computational simplicity and indexing capabilities.

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 [14]:

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

Figure 2.4 shows an example of the Euclidean distance between two time series X and Y.

The Euclidean distance is an effective measurement of similarity between two time series, however it presents several drawbacks, which make it inappropriate in certain applications, for example it cannot be applied to time series of different lengths and it doesn't handle outliers or noise.

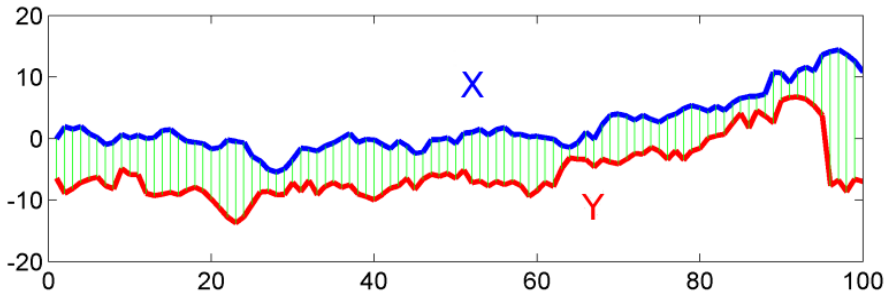


Figure 2.4: The Euclidean distance between two time series X and Y results in the sum of the point-to-point distances (green lines), along all the time series.

The Z-normalized Euclidean Distance

Normalization is a transformation process to obtain numerical and comparable input data by using a common scale. Data normalization is essential for many decision-making problems, allowing to obtain dimensionless units from heterogeneous data measurements, which can be aggregated for rating and ranking decision alternatives.

The z-normalized Euclidean distance D_{ze} is defined as the Euclidean distance between the z-normalized or normal form of two sequences, where the z-normalized form is obtained by transforming a sequence X of length n such that it has mean $\mu = 0$ and standard deviation $\sigma = 1$.

Given two time series X and Y, let μ_X and μ_Y be the mean of the values in X and Y respectively. Also, let σ_X and σ_Y be the standard deviation of the values in X and Y respectively. Then, the z-normalized Euclidean distance between X and Y is defined as:

$$D_{ze}(X,Y) = \sqrt{\sum_{i=1}^{n}(\frac{x_i - \mu_X}{\sigma_X} - \frac{y_i - \mu_Y}{\sigma_Y})^2}$$

Matrix profile [41] has been recently proposed as a promising technique to the problem of all-pairs-similarity search on time series. Given a time series X and a subsequence length m, the matrix profile returns for each subsequence included in X its distance to the most similar subsequence in the time series. The matrix profile is itself a time series very useful for data analysis, e.g., detecting the motifs (represented by low values), discords (represented by high values), etc. Efficient algorithms have been proposed for computing it, *e.g.*, STAMP [41], STOMP [46] and SCRIMP++ [45]. All these algorithms use the z-normalized Euclidean distance to measure the distance between subsequences.

2.2.2.2 Dynamic time warping

Dynamic Time Warping (DTW) [32] is the most popular elastic shape based similarity measure proposed to handle warps in the temporal dimension. DTW gives more robustness to the similarity computation. By this method, time series of different length can be compared, because it replaces the one-to-one point comparison, used in Euclidean distances with both many-to-one point and one-to-many point comparisons.

The main feature of this distance measure is that it allows to recognize similar shapes, even if they present signal transformations, such as shifting and/or scaling. Let us take two time series $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_m)$ of length n and m respectively. An alignment by DTW method exploits information contained in a $n \times m$ distance matrix, we compute the cost measure $d(x_i, y_i)$ for each sample pair of both series, we obtain the distance matrix $DM \in \mathbb{R}^{n \times m}$ where :

$$dm(j,j) = d(x_i, y_j) = (x_i - y_j)^2, 1 \leq i \leq n, 1 \leq j \leq m$$

The DTW objective is to find the warping path $P = (p_1, p_2, ..., p_L)$ by optimally aligning the time series in the temporal domain so that the accumulated distances of this alignment is minimal. The optimal path that minimizes the total cost is defined as :

$$DTW(X,Y) = min\Big(\sqrt{\sum_{l=1}^{L} p_l}\Big)$$

This path has to fulfill some constraints in order to optimize the solution found : Given $p_l = (i,j)$ and $p_{l-1} = (i',j')$ with $i, i' \leq n$ and $j, j' \leq m$:

- **Boundary conditions** : $p_1 = (1,1)$ and $p_L = (n,m)$, i.e, the alignment path starts at the bottom left and ends at the top right of the distance matrix DM which guarantees that the alignment does not consider partially one of the sequences.

- **Monotonicity** : $i - i' \geq 0$ and $j - j' \geq 0$, i.e, the alignment path does not go back in time index which guarantees that features are not repeated in the alignment.

- **Continuity** : $i - i' \leq 1$ and $j - j' \leq 1$, i.e, the alignment path does not jump in "time" index which guarantees that the alignment does not omit important features.

- **Warping window** : $|i - j| \leq r$, where $r > 0$ is the window length. This defines that a good alignment path is unlikely to wander too far from the diagonal which guarantees that the alignment does not try to skip different features and gets stuck at similar features.

This warping path can be obtained by dynamic programming using an accumulated distance matrix DM', recursively applying :

$$DM'(i,j) = d(x_i, y_j) + min\{DM'(i-1, j-1), DM'(i-1, j), DM'(i, j-1)\}$$

In [66], the authors show that DTW is significantly more accurate than the Euclidean distance for small datasets, and the difference diminishes as the datasets get larger until there is no measurable difference.

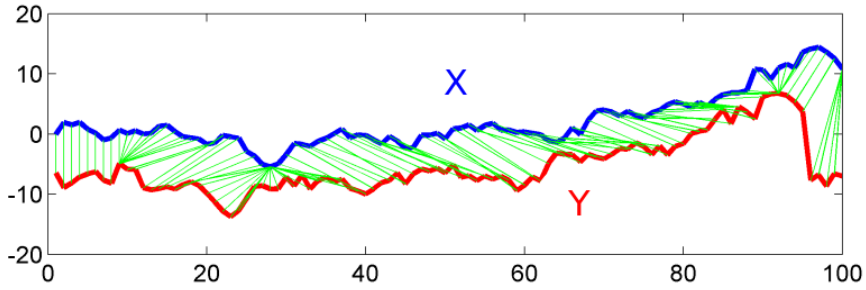


Figure 2.5: The Dynamic Time Warping distance between two time series X and Y allows many-to-one point comparisons.

2.2.2.3 The Sketch Approach

In [47], Zhu et al. propose random projection method based on random vectors. The basic idea is to multiply each time series with a set of random vectors. The result of that operation is a "sketch" for each time series consisting of the distance (or similarity) of the time series to each random vector. Then two time series can be compared by comparing sketches. The sketches are used to approximate the distance between each pair of time series. The random projection can approximate different types of distances like Euclidean Distance and Lp Distance. The sketch approach is a kind of Locality Sensitive Hashing [15], by which similar items are hashed to the same buckets with high probability. In particular, it is similar in spirit to SimHash [8], in which the vectors of data items are hashed based on their angles with random vectors.

The sketch approach, as developed by Kushilevitz et al. [22], Indyk et al. [17], and Achlioptas [2], provides a very nice guarantee: with high probability a random mapping taking b points in R^m to points in $(R^d)^{2b+1}$ (the (2b+1)-fold cross-product of R^d with itself) approximately preserves distances.

The sketch of a time series is computed as follows. Given a time series $\mathbf{t} \in R^m$, we compute its dot product with N random vectors $\mathbf{r}_i \in \{1, -1\}^m$. This results in N inner products called the *sketch* (or random projection) of t_i. Specifically, $sketch(t_i) = (\mathbf{t_i} \bullet \mathbf{r_1}, \mathbf{t_i} \bullet \mathbf{r_2}, ..., \mathbf{t_i} \bullet \mathbf{r_N})$. We compute sketches for all time series of a dataset using the same random vectors $r_1, ..., r_N$.

The theoretical underpinning for the utilization of sketches is given by the Johnson-Lindenstrauss lemma [18].

Lemma 1 *Given a collection C of m time series, for any two time series* $\overrightarrow{x}, \overrightarrow{y} \in C$, *if* $\epsilon < 1/2$ *and* $n = \dfrac{9logm}{\epsilon^2}$, *then*

$$(1-\epsilon) \leq \frac{\| \overrightarrow{s}(\overrightarrow{x}) - \overrightarrow{s}(\overrightarrow{y}) \|^2}{\| \overrightarrow{x} - \overrightarrow{y} \|} \leq (1+\epsilon)$$

holds with probability $1/2$, *where* $\overrightarrow{s}(\overrightarrow{x})$ *is the sketch of* $\overrightarrow{x}$ *of at least n dimensions.*

The Johnson-Lindenstrauss lemma implies that the distance $\|\mathbf{sketch(t_i)} - \mathbf{sketch(t_j)}\|$ is a good appproximation of $\|\mathbf{t_i} - \mathbf{t_j}\|$ provided the dimensionality of the sketches (r) is

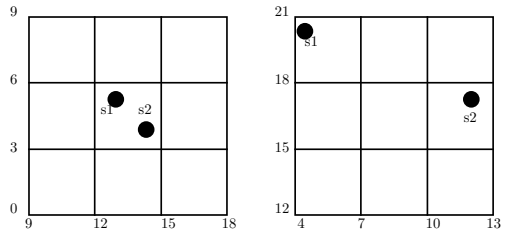


Figure 2.6: Two series (s_1 and s_2) may be similar in some dimensions (here, illustrated by $Grid_1$) and dissimilar in other dimensions ($Grid_2$). The higher the similarity between t_1 and t_2, the larger the fraction of grids in which the series are close.

large enough. Specifically, if $\|\mathbf{sketch(t_i)} - \mathbf{sketch(t_j)}\| < \|\mathbf{sketch(t_k)} - \mathbf{sketch(t_m)}\|$, then it's likely that $\|\mathbf{t_i} - \mathbf{t_j}\| < \|\mathbf{t_k} - \mathbf{t_m}\|$, because the ratio between the sketch distance and the real distance is close to one.

A sketch of a time series t is a vector of dot products: element i of the sketch is the dot product between t and the ith random vector. Thus, the full sketch contains as many dot products as there are random vectors.

The data structure consists of a set of grids. Each grid maintains the sketch values corresponding to the dot products between a specific set of random vectors and all time series. Let $|g|$ be the number of random vectors assigned to each grid, and N be the total number of random vectors, then the total number of grids is $b = N/|g|$. (We make sure that $|g|$ divides N.) The distance between two time series in different grids may differ. We consider two time series similar if they are similar in a given (large) fraction of grids.

Example 1 *Let's consider two time series t_1=(2, 2, 5, 2, 6, 5) and t_2=(2, 1, 6, 5, 5, 6). Suppose that we have generated four random vectors as follows : r_1=(1, -1, 1, -1, 1, 1), r_2=(1, 1, 1, -1, -1, 1), r_3=(-1, 1, 1, 1, -1, 1) and r_4=(1, 1, 1, -1, 1, 1). Then the sketches of t_1 and t_2, i.e. the inner products computed as described above, are respectively s_1=(14, 6, 6, 18) and s_2=(13, 5, 11, 15). In this example, we create two grids, $Grid_1$ and $Grid_2$, as depicted in figure 2.6. $Grid_1$ is built according to the sketches calculated with respect to vectors r_1 and r_2 (where t_1 has sketch values 14 and 6 and t_2 has sketch values 13 and 5). In other words, $Grid_1$ captures the values of the sketches of t_1 and t_2 on the first two dimensions (vectors). $Grid_2$ is built according to vectors r_3 and r_4 (where t_1 has sketch values 6 and 18 and t_2 has sketch values 11 and 15). Thus, $Grid_2$ captures the values of the sketches on the last two dimensions. We observe that t_1 and t_2 are close to one another in $Grid_1$. On the other hand, t_1 and t_2 are far apart in $Grid_2$.*

Partitioning Sketch Vectors

Multi-dimensional search structures don't work well for more than four dimensions in practice [36]. For this reason, as indicated in Example 1, each sketch vector is partitioned into subvectors, then grid structures are built for the subvectors as follows:

- Each sketch vector s of size N is partitioned into groups of some size $|g|$.
- The ith group of each sketch vector s is placed in the ith grid structure (of dimension $|g|$).

- If two sketch vectors s and s' are in the same cell in more than a given fraction f of the grids, then the corresponding time series are candidates for similar time series and should be checked exactly.

For example, if each sketch vector is of length $N = 40$, we might partition each one into ten groups of size $|g| = 4$. This would yield 10 grid structures, where time series items are assigned to grid cells, so that close items are grouped in the same grid cells. Suppose that the fraction f is 90%, then a time series t is considered as similar to a time series t', if they are similar (assigned to the same cell) in at least nine grids.

Grid granularity can be adjusted to control the tradeoff between efficiency and accuracy. Coarser grids have larger grid cells (i.e. more time series assigned to the same cell), which leads to a larger number of candidates to process (slower execution), but lower probability to miss a true positive (higher accuracy). The grid granularity is defined through the parameter *grid_size* that specifies the number of cells per grid dimension. At the cell assignment step, grids are divided into cells in a way that results in a uniform distribution of items across grid cells. This is supported by a sampling phase that infers the distribution and defines the cell borders along each dimension of each grid.

2.3 Symbolic Aggregate Approximation (SAX)

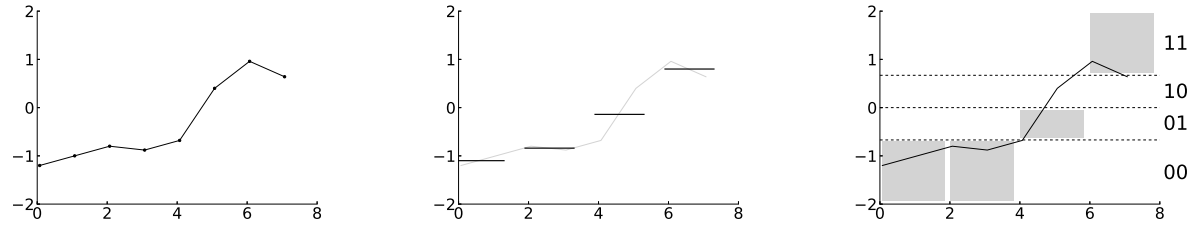
Many symbolic representations of time series have been introduced over the past decades. The challenge in this field is to create a real correlation between the distance measure defined on the symbolic representation, and that defined on original time series. SAX [23] is the most known symbolic representation technique on time series data mining that ensures both a considerable dimensionality reduction and the lower bounding property, allowing enhancing of time performances on most of data mining algorithms.
SAX allows a time series X of length n to be reduced to a string of arbitrary length w (and, usually, $w << n$). This transformation is done in two major steps: (1) transforming the raw data into a (PAA) representation, (2) transforming the PAA representation into a sequence of symbols belonging to a predefined set of alphabets, with a given cardinality.

2.3.1 Dimensionality Reduction Via PAA

A time series X of length n can be represented in a w-dimensional space that is $\overline{X} = (\overline{x_1}, \overline{x_2}, ..., \overline{x_w})$, the i_{th} element of $\overline{X}$ is the average of the i_{th} segment values and is calculated by the following equation :

$$\overline{x_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} x_j$$

In other words, the time series is divided into w equal-sized segments, for each segment, the mean value of the data value falling in it is calculated and a vector of these



(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, $[00, 00, 01, 11]$

Figure 2.7: 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 2.7a in the representation of Figure 2.7c is [first, first, second, fourth] (which becomes $[00, 00, 01, 11]$ in binary).

values becomes the data-reduced representation. Example 2 gives an illustration of PAA.

Example 2 *Figure 2.7b shows the PAA representation of X, the time series of Figure 2.7a. 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, $w << |X|$).*

2.3.2 Discretization

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. In this step, PAA coefficients are mapped into alphabetic symbols by using a lookup table that contains *breakpoints* for separating the symbols. The breakpoints β_i depend on the alphabet set size. They are derived using a Gaussian distribution such that the area from β_i to β_{i+1} is $\frac{1}{a}$ where a is the alphabet size. These breakpoints may be determined by looking them up in a statistical table. Table 2.1 shows the breakpoints for different alphabet sizes (a from 3 to 7) .

Example 3 gives an illustration of the SAX representation.

Example 3 *In Figure 2.7c, we have converted the time series X to SAX representation with size 4, and cardinality 4 using the PAA representation shown in Figure 2.7b. We denote SAX(X) =* $[00, 00, 01, 11]$.

2.3.3 Distance Measures

As discussed in Section 2.2.2.1, the Euclidean distance is one of the most popular similarity measurement methods used in time series analysis. Given two time series $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_n\}$, the Euclidean distance between X and Y is

$\beta \backslash a$	3	4	5	6	7
β_1	-0.43	-0.67	-0.84	-0.97	-1.07
β_2	0.43	0	-0.25	-0.43	-0.57
β_3		0.67	0.25	0	-0.18
β_4			0.84	0.43	0.18
β_5				0.97	0.57
β_6					1.07

Table 2.1: A lookup table that contains the breakpoints that divide a Gaussian distribution in an arbitrary number (from 3 to 7) of equiprobable regions

defined as [14]:

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

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:

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

If we further transform the data into the symbolic representation, 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:

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

This function resembles the DR_f function except for the fact that the distance between the two PAA coefficients has been replaced with the sub-function dist(). 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:

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

2.3.4 Indexing Extensions

The classic SAX representation offers the potential to be indexed. iSAX [38] is an indexable version of SAX designed for indexing large collections of time series. iSAX is based on a modification of the SAX representation to allow extensible hashing. The advantage of iSAX over SAX is that it allows the comparison of words with different cardinalities, and even iSAX words where each word has its own cardinality. The iSax tree index is built as follows. Given a cardinality a, an iSAX word length w and leaf capacity th, we produce a set of b^w children for the root node, insert the time series to their corresponding leaf, and gradually split the leaves by increasing the cardinality

by one if the number of time series in a leaf node rises above the given threshold *th*. Other extensions of SAX have been proposed for improving the similarity search performance via indexing. For example, iSAX 2.0 [5] proposes a new mechanism and also algorithms for efficient bulk loading and node splitting policy, which is not supported by iSAX index. In [5], 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.

2.3.5 Limitation of SAX

Despite the benefits of time series representation techniques, the reduction of dimensionality of a time series will highly influence the outcome of further processing. We can not expect an analysis performed on an approximation to yield results of same quality as one would obtain on the original data. A big challenge in elaborating a reduction technique is to establish a good compromise between dimensionality reduction and accuracy. This means that the features, which are extracted by the reduction technique from the data, should keep to a high extent information about the original data. The SAX representation proceeds to an approximation by minimizing the dimensionality: the original time series are divided into segments of equal size. SAX enjoys a good reduction, but its major limitation is that it only relies on the computation of the mean values to derive the symbols on fixed-size segments.

2.3.6 SAX Extensions Based on Trend Feature

Several research works raised the loss of trend issue that comes with SAX. Indeed, averaging the values of a segment makes possible to have different segments trends with the same average value. Such segments will be mapped to the same symbol. Accordingly, related research initiatives worked on enhancing the SAX representation. There have been SAX extensions designed to improve the representation of each segment by capturing the trend with different feature, while using the SAX fixed-size segmentation. ESAX [26] is an extension of SAX in which a time series segment uses two additional points, max and min, in equal sized segments besides the mean value for data approximation in order to improve the classic SAX.

SAX_TD [40] uses the starting and end points of a segment for the sake of improving the SAX distance with the trend calculation. It captures the trends of time series in numerical form by approximating the measure of trends using the difference between the average and the starting point of the segment, and the difference between the average and the ending point of the segment.

In SAX_SD [43], the standard deviation of the segment is considered as another feature to improve SAX. It includes the difference between standard deviations of different segments, and comes with a new distance that has a tighter bound compared to the original SAX.

In EN_SAX [44], the authors propose to calculate the entropy values of the segments and to use them as additional features to improve the SAX method. However, the segmentation in SAX and these extensions is not adaptive to the data since they divide the time domain based on fixed-size segments, which causes the missing of some important features in certain time series datasets.

2.4 Conclusion

In this chapter, we discussed the state of the art about time series representations. We gave an overview of time series data mining and a brief description of the main tasks in time series data mining that have attracted extensive research interest. Also, we discussed the time series representations techniques that have been widely used, particularly SAX, being the most known symbolic representation technique on time series data mining, that ensures both a considerable dimensionality reduction, and the lower bounding property, allowing enhancing of time performances on most of data mining algorithms. Despite the advantage of the SAX representation, its dimensionality reduction technique by means of equal sized segments is not efficient. In this thesis, we propose our adaptive segmentation approaches based on variable-length segmentation to increase the quality of time series approximation by taking into account various metrics allowing to measure the representation error. Our solutions are 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.

CHAPTER 3

VARIABLE SIZE SEGMENTATION FOR EFFICIENT REPRESENTATION OF NON-UNIFORM TIME SERIES DATASETS BASED ON ENTROPY

Many time series datasets are large and high dimensional. Accessing each point is computationally too expensive, therefore analysis is often performed not on the raw data itself, but on a more abstract representation obtained through dimensionality reduction techniques. This has the advantage of requiring less space and speeding up calculation procedures but comes at the price of an approximation in the uses of these representations like, *e.g.*, comparisons for retrieval. Usually, the goal of dimensionality reduction for time series representations is to lower computation time and memory usage, while guaranteeing minimum performances like precision of retrieval requests that are based on comparisons of the representations. The results of these comparisons are expected to be close to the results obtained on the original data.

In this chapter, we propose an approximation method for time series that considers the time series shape and does the splitting by means of segments of variable size on the time domain based on the representation entropy with a top-down strategy .

3.1 Motivation and Overview of the Proposal

Time series segmentation is a discretization problem and aims at accurately approximating time series. In the last decade, a significant number of approaches have been proposed to tackle the time series approximation problem. SAX [23] is one of the most popular representations of time series, allowing dimensionality reduction on the classic data mining tasks. SAX is fast because it constructs symbolic representations by splitting the time domain into segments of equal size. Therefore, its time complexity is linear with the time series length. However, it does not consider the time series shape, or distribution. Actually, we observe that, for time series having a non-uniform distribution over the time domain, this division into segments of fixed length is not advantageous since the model does not adapt to the shape of the data. Our claim is that an adaptive segmentation of time series, by distributing segments where they will increase the efficiency of uses of the representation, is necessary.

Therefore, in order to improve the quality of similarity search, and to achieve an adaptive splitting on the data, we propose a new approximation method for time se-

ries that considers the time series shape and does the splitting by means of segments of variable size on the time domain adopting a top-down strategy. Top-down approaches recursively segment the raw data until some stopping criteria are met. The top-down algorithm works by considering every possible partitioning of the time serie and splitting it at the best location. We create an initial segmentation and then the top-down approach refines the segmentation. By measuring the entropy of symbolic representations, our algorithm chooses between different possible splittings at each step of the representation computation. This approach allows reducing information loss, and thus increasing the accuracy of time series representations leading to better precision during retrieval phases, particularly from non-uniform datasets. In this work, we make the following contributions:

- We propose a new representation technique, called ASAX_EN (Adaptive SAX based on Entropy), 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 entropy measurement for detecting what time intervals should be split using a top-down strategy.

- We propose a lower bounding method that allows approximating the distance between the original time series based on their representations in ASAX_EN.

- We implement our approach and conduct empirical experiments using several real world datasets. The results suggest that ASAX_EN 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, the greater is the precision gain of ASAX_EN.

The rest of the chapter is organized as follows. In Section 3.2, we formally define the problem we address. In Section 3.3 we describe the details of ASAX_EN representation and in Section 3.4 we define the distance measure on the proposed ASAX_EN representation. In Section 3.5, we present the experimental evaluation of our approach. Finally, we conclude in Section 3.6

3.2 Problem Definition

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 kNN queries.

3.3 Adaptive SAX based on Entropy

In this section, we propose ASAX_EN, a 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. Basically, the idea is that the higher the entropy of a segment, the more important the distinguishing power of this segment between two time series. Our goal is to give more importance to segments that will allow distinguishing between to time series, therefore increasing the efficiency of k-NN queries based on such representations.

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.

3.3.1 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 4 *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 4 *Consider the database D={x,y,z} in Figure 3.1 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), and then 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.*

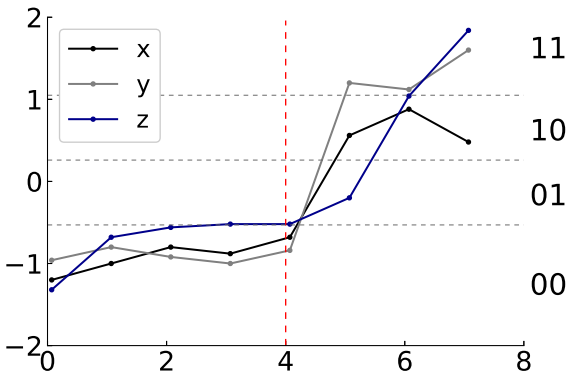


Figure 3.1: ASAX_EN segmentation with 2 segments

In the next subsection, we describe our algorithm to create variable-size segments based on entropy with a top-down strategy.

3.3.2 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.

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], [n/2, n]}; // 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 = ASAX_EN(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
```

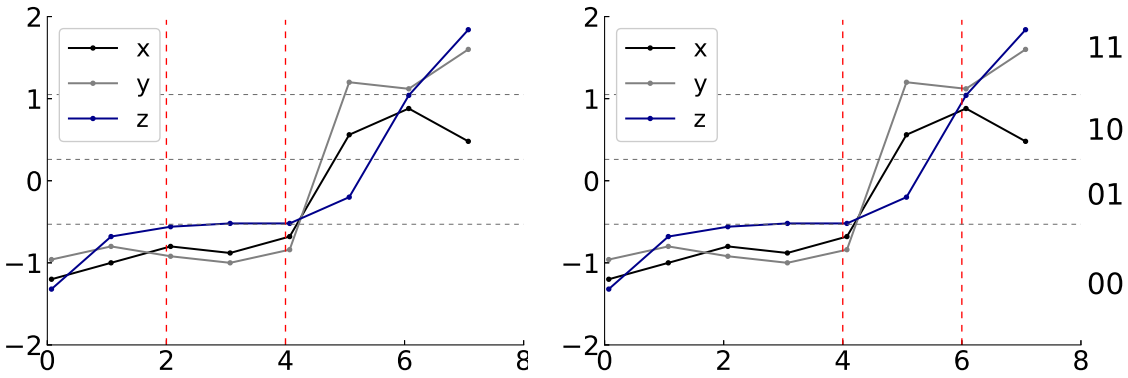
Example 5 *Let us consider the dataset D in Figure 3.1 which represents the initialization of the algorithm, i.e., the time domain is divided into two segments of the same size. The next step is to create the 3rd segment by splitting one of the two existing segments. Two different scenarios are possible.*

Scenario 1 : The first scenario is shown in Figure 3.2a 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 3.2b 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* $P(x = [00, 01, 11]) = \frac{2}{3}$. *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 at maximizing 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

Figure 3.2: The two different scenarios of ASAX_EN segmentation with 3 segments. Scenario 3.2b is the one chosen because it optimizes the entropy.

3.3.3 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 3.3, with three time series from our motivating example in Figure 1.1 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 3.3 the 10 symbol regions will be evenly distributed in the range of data values.

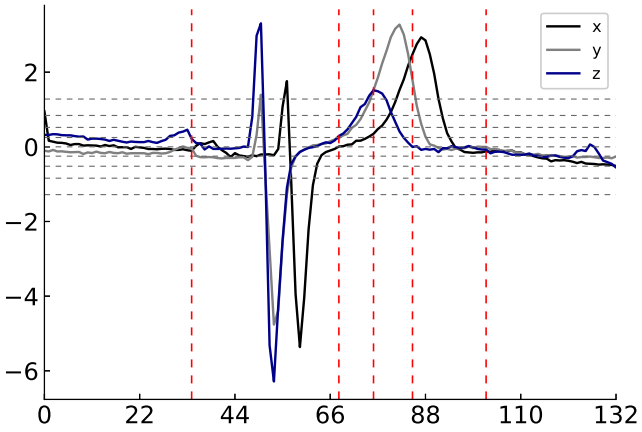


Figure 3.3: The Gaussian based distribution of symbols in SAX are not suitable for ASAX_EN since they would favor minor information gain.

3.4 Lower Bounding of the Similarity Measure

Having introduced the new representation of time series, we will now define a distance measure on it. SAX [24] defines a distance measure on the representation of time series as described in Section 2.3. 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_EN. Note that this distance measure will be used as a lower bounding approximation in all our variable size segmentation algorithms which will follow in the next chapters.

Theorem 1 *Let X and Y be two time series. Suppose that by using ASAX_EN 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_EN, $DR_v(\overline{X}, \overline{Y})$

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_EN 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_EN 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_EN 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 :

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \geq \sqrt{\sum_{j=1}^{w} ((\overline{x}_j - \overline{y}_j)^2 \times l_j)} \tag{3.1}$$

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:

$$\sqrt{\sum_{i=1}^{l_1}(x_i - y_i)^2} \geq \sqrt{\sum_{i=1}^{l_1} ((\overline{x}_i - \overline{y}_i)^2} \tag{3.2}$$

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.

3.5 Evaluation and results

In this section, we report the results of experimental studies on the proposed ASAX_EN segmentation approach that illustrate its performance in improving the accuracy of time series representations in order to get better precision during information search operations.

3.5.1 Datasets and Experimental Settings

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 3.1: Datasets basic information

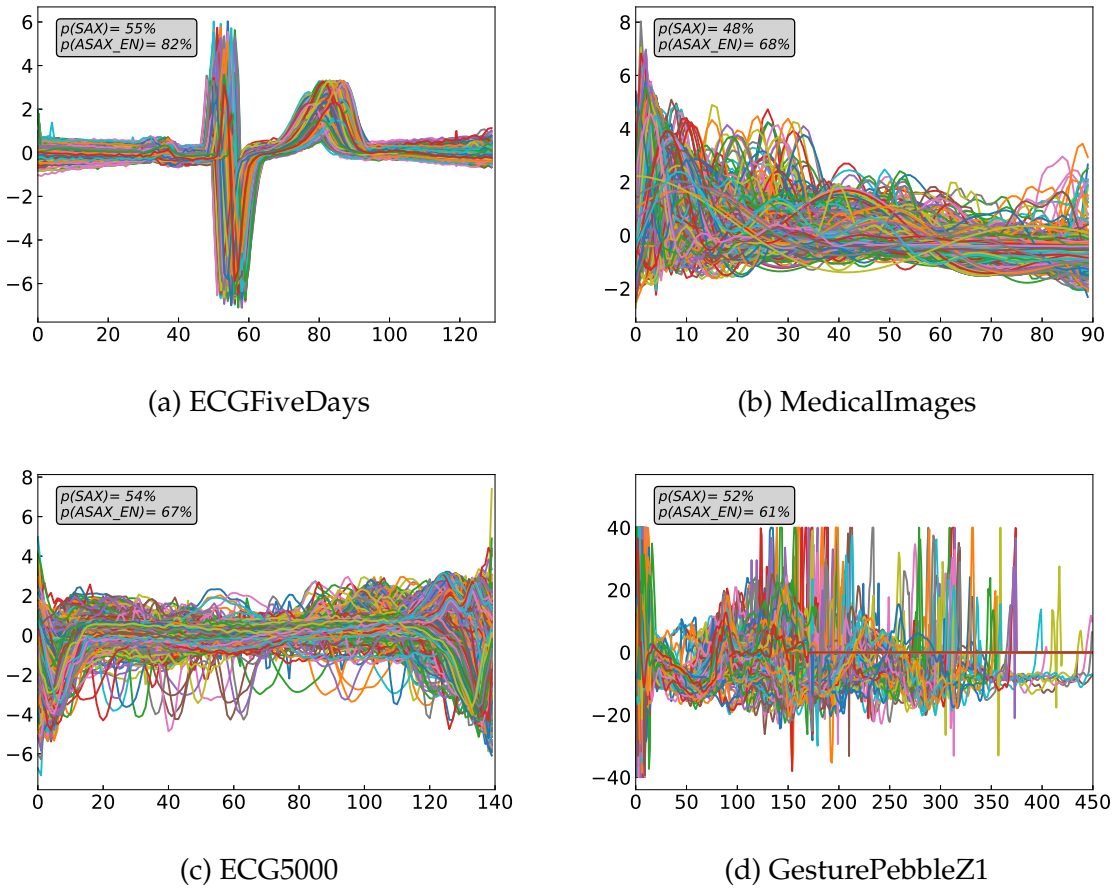
We compared the ASAX_EN representation with the existing SAX representation on datasets selected for their particular (lack of) uniformity. Notice that SAX and its extensions in the literature use a fixed-size segmentation of the time domain. But, ASAX_EN proposes a variable-size segmentation based on information theory techniques.

The approaches are implemented in Python programming language and Numba JIT compiler is used to optimize machine code at runtime. 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 [11]. Table 3.1 gives basic information about the datasets: name, type, length of the time series (number of values). Notice that almost all selected datasets have non-uniform distributions over time domain (see Figure 3.4), else SyntheticControl that has a quasi uniform distribution.

For each approach, we set the default cardinality value to 32 and the length w of the approximate representations is reduced to 10% of the original time series length.

In the experiments, we measure the ASAX_EN and SAX precision in similarity search by applying a k-Nearest Neighbor (k-NN) search, as detailed in Subsection 3.5.2. For ASAX_EN, we measure the time cost of the variable-size segmentation in Subsection 3.5.3.



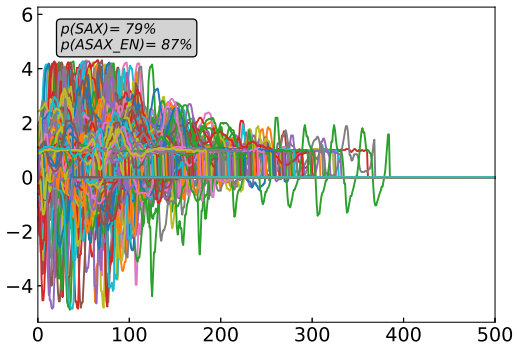
(a) ECGFiveDays

(b) MedicalImages

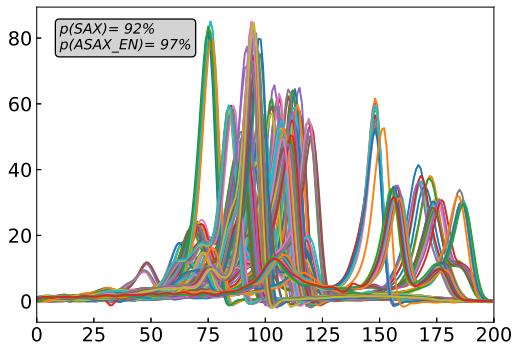
(c) ECG5000

(d) GesturePebbleZ1

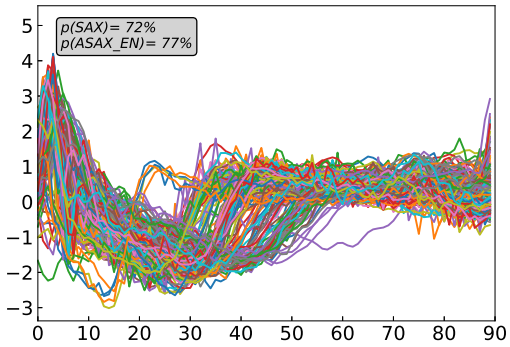
Figure 3.4: The data distribution of the tested datasets, and the precision results for each dataset. p(SAX) and p(ASAX_EN) show the precision of SAX and ASAX_EN respectively. The datasets are sorted in descending order of precision gain.



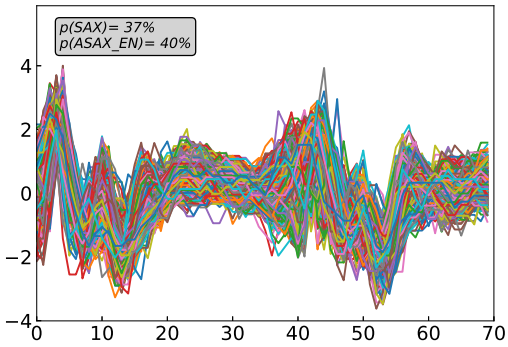
(e) AllGestureWiimoteZ



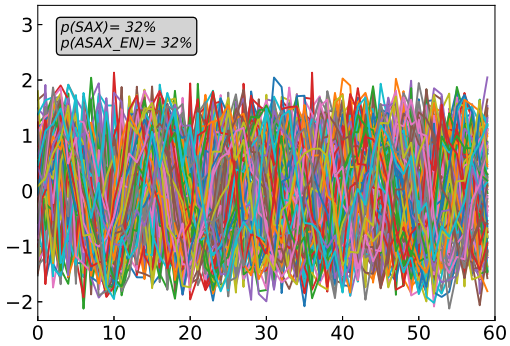
(f) Fungi



(g) ECG200



(h) SonyAIBORobotSurface1



(i) SyntheticControl

Figure 3.4: The data distribution of the tested datasets, and the precision results for each dataset. p(SAX) and p(ASAX_EN) show the precision of SAX and ASAX_EN respectively. The datasets are sorted in descending order of precision gain.

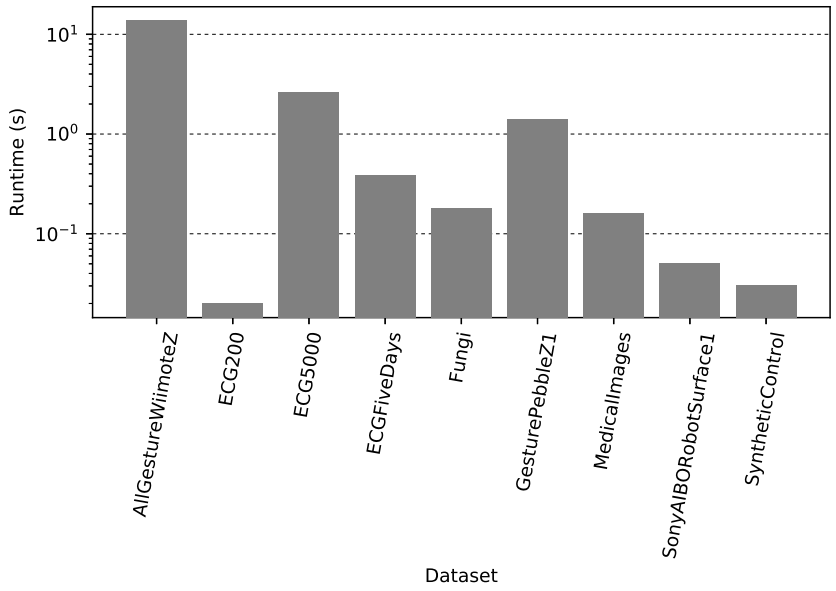


Figure 3.5: Runtime of ASAX_EN segmentation algorithm for each dataset

3.5.2 Precision of k-Nearest Neighbor Search

In this part of experiments, we compare the quality of ASAX_EN and SAX representation on the different datasets described in Table 3.1 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_EN 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 3.4 where each dataset is plotted with the precision obtained (as percentage) for both approaches and the datasets are sorted in descending order of precision gain. 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. Let us take for example the *ECGFiveDays* dataset presented in Figure 3.4a and *SyntheticControl* shown in Figure 3.4i. On the first one, we were able to achieve a precision of 82% for ASAX_EN while it is 55% for SAX, which is a significant gain in precision. This higher precision for ASAX_EN is due to the variable-size segmentation which created segments in the parts that undergo a significant variation (from time point 44 to 95) as discussed in our motivating example in

the introduction, allowing ASAX_EN to perform a better distribution of the segments according to information gain.

For *SyntheticControl* we can see that the precision of the approximate k-NN search is the same for both ASAX_EN and SAX approaches which is 32%. In this dataset, the shape of the time series is balanced over the time, and the segmentations obtained by ASAX_EN and SAX are the same, resulting in equivalent precision.

These results suggest the advantage of our approach over SAX when applied to time series with unbalanced distribution.

3.5.3 Time cost of ASAX_EN segmentation algorithm

Figure 3.5 reports the time cost of our proposed approach. It gives the segmentation time of ASAX_EN 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 longest segmentation time is approximately 13 seconds, while the shortest one is around 20 milliseconds. It depends on both the number of time series in the dataset and their length.

3.6 Conclusion

In this chapter, we proposed a new approximation technique, called ASAX_EN, that considers the time series distribution on the time domain and performs variable-size segmentation, by using the entropy of symbolic representations. Our technique allows reducing information loss and thus increasing the accuracy of time series representations. We implemented our technique and evaluated its performance using several real world datasets. The experimental results suggest that ASAX_EN can obtain significant performance gains in terms of precision for similarity search compared to SAX. The results show that the more the data distribution in the time domain is unbalanced (non-uniform), the greater is the precision gain of ASAX_EN, *e.g.*, for the *EGCFiveDays* dataset that has a non-uniform distribution in the time domain, the precision of ASAX_EN is 82% compared to 55% for SAX.

CHAPTER 4

OPTIMIZED TECHNIQUES FOR TIME SERIES SEGMENTATION BASED ON THE APPROXIMATION ERROR

In the previous chapter, we proposed a segmentation method based on the entropy measure that uses a top-down algorithm. In this chapter, we address the problem of finding the variable-size segmentation that minimizes the approximation error of the time series representation using a bottom-up approach. In contrast to top-down approaches, bottom-up approaches start with the finest possible approximation and join segments until some stopping criteria are met. We propose our variable-size segmentation approach for time series representation based on the Sum of Squared Error (SSE) of the representation. Our contributions are as follows:

- We propose a new representation technique, called ASAX_SSE, that allows obtaining a variable-size segmentation of time series based on SSE measure and by using a bottom-up approach.
- We propose an efficient algorithm, called ASAX_Dyn, for improving the execution time of our segmentation approach, by means of dynamic programming.
- 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 illustrate that ASAX_SSE can obtain significant performance gains in terms of precision for similarity search compared to SAX and our ASAX_EN presented in Chapter 3.

The rest of this chapter is organized as follows. In Section 4.1, we describe the details of our segmentation approach. In section 4.2, we present an improved version of our segmentation using dynamic programming and in Section 4.3, we present parallel versions of our algorithms. In section 4.4, we present a detailed experimental evaluation to verify the effectiveness of our algorithms. Finally, we conclude in Section 4.5.

4.1 Adaptive SAX based on Sum of Squared Error

In this section, we propose ASAX_SSE, a variable-size segmentation approach for time series representation using a bottom-up strategy. Here, to create a segmentation with

minimum information loss on time series approximation, our algorithm divides the time domain based on the sum of squared errors (SSE) of the representation with a bottom-up strategy instead of a top-down strategy . The results of the experiments have shown that this method is more efficient than the ASAX_EN approach.

In the rest of this section, we first describe the notion of Sum of Squared Errors (SSE) for the time series representation, and then, we present the algorithm that creates the variable-size segments based on SSE measurement.

4.1.1 Sum of Squared Errors (SSE)

In Statistics, 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 5 *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 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 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.

4.1.2 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. Let us illustrate LSSE using an example.

Example 6 *Consider the time series X in Figure 4.1 where n=10. Let's convert X into its PAA representation having 5 segments by dividing the time domain into segments of equal size (l=2)*

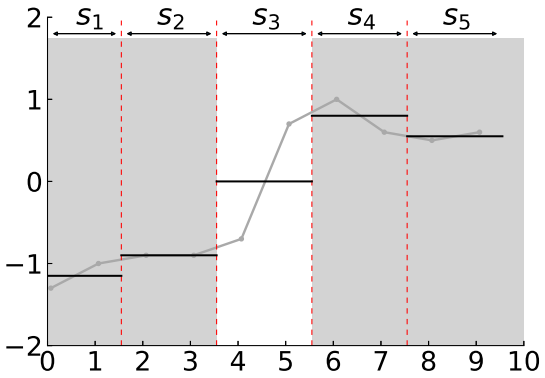


Figure 4.1: The PAA representation of time series X contains 5 segments. LSSE is computed on the selected segment S_3.

shown in the same Figure. We have $X = [-1.3, -1, -0.9, -0.9, -0.7, 0.7, 1, 0.6, 0.5, 0.6]$ *and* $\overline{X} = [-1.15, -0.9, 0, 0.8, 0.55]$. *Suppose that we want to measure the approximation error on the 3rd segment* s_3 *where* $s_3 = [4, 5]$ *(*s_3 *is delimited by time points 4 and 5), then, we compute the SSE of* s_3 *as follows:*

- *We first compute the individual error values of* s_3*, that is, the difference between each value in* s_3 *and its PAA value:* $e_4 = x_4 - \overline{x_3} = -0.7 - 0 = -0.7$, $e_5 = x_5 - \overline{x_3} = 0.7 - 0 = 0.7$.
- *Then, we calculate the squares of the errors :* $e_4^2 = (-0.7)^2 = 0.49$, $e_5^2 = (0.7)^2 = 0.49$.
- *Finally, the sum of the squared errors is calculated for all values of* s_3 *that is* $sum = 0.98$.

Notice that in the above example, for ease of presentation we have taken only one time series. For a database containing m time series, LSSE is computed by summing up the LSSE of all m individual time series. Then, the error approximation of s_3 calculated with LSSE is 0.98.

4.1.3 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.

Example 7 *Let us consider the time series X and its PAA representation* $\overline{X}$ *of Figure 4.2. We have* $X = [-1.3, -1, -0.9, -0.9, -0.7, 0.7, 1, 0.6, 0.5, 0.6]$ *and* $\overline{X} = [-1.15, -0.9, 0, 0.8, 0.55]$. *First, for each segment, we calculate the difference between each value in this segment and its PAA value. Table 4.1 details the result obtained for this example. Then, the sum of the squared errors is calculated for all values of the time series, i.e.,* $sum = 1.105$. *According to the current segmentation applied on X, the error in this approximation calculated with GSSE is* 1.11. *If we choose a different segmentation, then the corresponding GSSE value will also change. So, the approximation error is influenced by the chosen segmentation.*

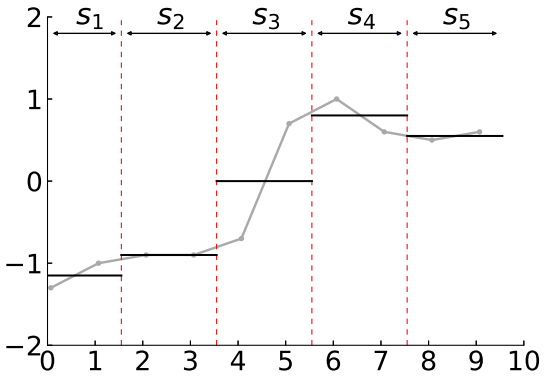


Figure 4.2: PAA representation of a time series X of length 10 with 5 segments. GSSE is computed on all segments.

Segment	Values	PAA	Error	Squared Error
s_1=[0,1]	-1.3 -1	-1.15	-0.15 0.15	0.0225 0.0225
s_2=[2,3]	-0.9 -0.9	-0.9	0 0	0 0
s_3=[4,5]	-0.7 0.7	0	-0.7 0.7	0.49 0.49
s_4=[6,7]	1 0.6	0.8	0.2 -0.2	0.04 0.04
s_5=[8,9]	0.5 0.6	0.55	-0.05 0.05	0.0025 0.0025

Table 4.1: Error calculation for each point in X.

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

4.1.4 Variable-Size Segmentation Based on SSE Measurement

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

Algorithm 2: ASAX_SSE 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
```

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 pseudo-code 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).

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 LSSE approach.

Example 8 *Let us apply our algorithm on the time series X in Figure 4.2 by taking the initial size of 2 for the starting segments. 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 SSE. Four different scenarios are possible:*

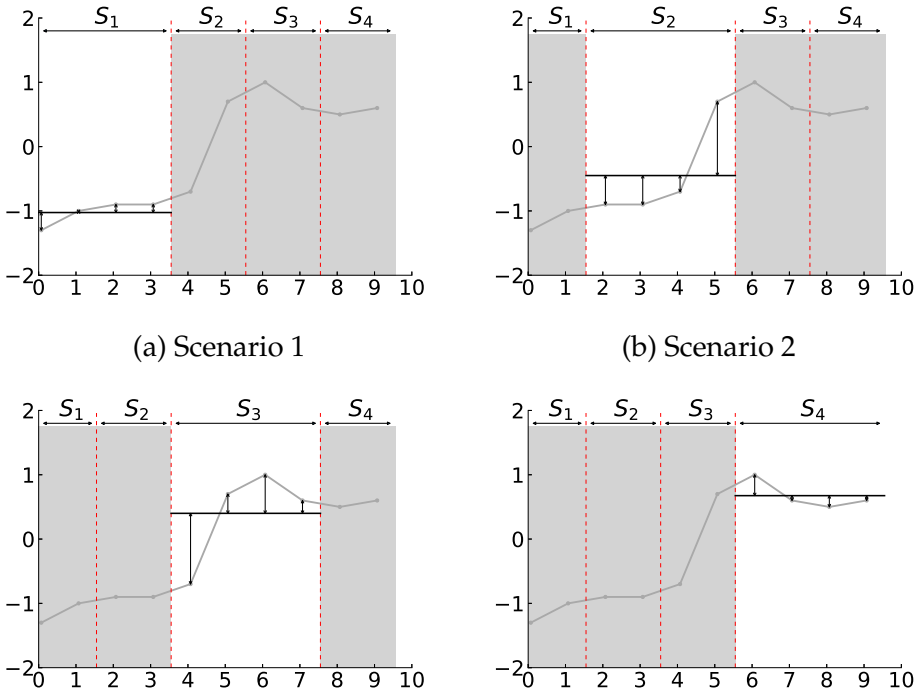
Scenario 1: The first scenario is shown in Figure 4.3a where s_1 and s_2 of the initial segmentation (shown in figure 4.2) are merged into one segment. We calculate the values's mean on the resulting segment (denoted S1 in figure 4.3a), and then compute the SSE of this approximation that is $SSE_1(X, \overline{X}) = 0.11$.

Scenario 2: This scenario is shown in Figure 4.3b 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, $SSE_2(X, \overline{X}) = 1.79$.

Scenario 3: This scenario is shown in Figure 4.3c, where we merge s_3 and s_4. For this merging (S_3), $SSE_3(X, \overline{X}) = 1.70$.

Scenario 4: The last scenario is shown in Figure 4.3d, where we merge s_4 and s_5. For this segment S_4, $SSE_4(X, \overline{X}) = 0.15$.

We have calculated the SSE for the 4 scenarios. Since we aim to minimize the SSE, we choose Scenario 1 that leads to the minimum SSE value (MSSE), that is $MSSE = 0.11$ (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.



(a) Scenario 1

(b) Scenario 2

(c) Scenario 3

(d) Scenario 4

Figure 4.3: The four different scenarios of ASAX_LSSE segmentation with 4 segments. Scenario 1 is the one chosen because it provides the minimum SSE.

4.2 ASAX_LSSE based on Dynamic Programming

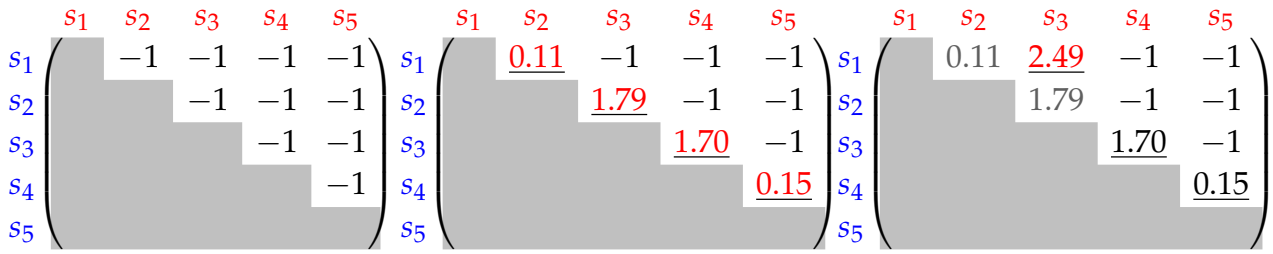
The ASAX_LSSE algorithm, which we presented in the previous section, can reduce significantly the information loss in time series representations, as illustrated by our experiments, especially the ASAX_LSSE approach. The results shows that this latter performs better than ASAX_GSSE in similarity search precision and in execution time. However, its execution time may be high, particularly over large time series datasets. In this section, we present an efficient version of ASAX_LSSE, called ASAX_Dyn, for improving the execution time of our segmentation technique using dynamic programming. In ASAX_Dyn, we use a data structure (matrix) to keep track of the result of the

Algorithm 3: ASAX_Dyn 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  matrix = matrix of k × k values initialized to -1
4  while k ≠ w do
5      segmentsToMerge = null
6      msse = ∞
7      for i=1 to k − 1 do
8          s = merge (s_i, s_{i+1})
9          r, c =Compute the position in matrix corresponding to s
10         sse = 0
11         if matrix[r,c] = -1 then
12             foreach ts in D do
13                 sse = sse + SSE(ts, s)
14             matrix[r,c]=sse
15         else
16             sse = matrix[r,c]
17         if sse < msse then
18             segmentsToMerge = i
19             msse = sse
20     s = merge (s_{segmentsToMerge}, s_{segmentsToMerge+1})
21     segments = segments − {s_{segmentsToMerge}, s_{segmentsToMerge+1}}
22     segments = segments ⋃ s
23     k = k-1
24 return segments
```



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

Figure 4.4: 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.

SSE computation for each iteration. In the matrix, if the value of a cell (i, j) is positive, then it corresponds to the SSE 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 SSE 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_Dyn algorithm in more details. Algorithm 3 presents the pseudo-code of the improved approach. As for the ASAX_SSE algorithm (described in the previous section), ASAX_Dyn 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 SSE value in the corresponding cell in the matrix is equal to -1), then the algorithm has to compute the SSE 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 SSE of PAA representation of all the time series contained in D, ASAX_Dyn adds the result of the computed SSE to *sse* and stores this value in the matrix by replacing the existing value (-1) by the calculated SSE (Line 14). In the case where the merging of s_i and s_{i+1} has already been tested (*i.e.*, if the SSE value in the matrix is not -1), the algorithm simply has to get the SSE value from the matrix which is already computed and sets *sse* to this value (Line 16). After having obtained the SSE value, if it 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 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 SSE computations. The algorithm selects the merging whose SSE 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.

Bellow, we illustrate our algorithm using an example.

Example 9 *Let us apply ASAX_Dyn on the time series X shown in Figure 4.1 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 SSE computation (Figure 4.4a). 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 SSE 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 9). Each of these 4 merging possibilities are tested, the corresponding SSE for each possible merging is computed, and the results are stored in the matrix. Figure 4.4b 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 4.3a, resulting from merging s_1 and s_2, since it provides the minimum SSE value (MSSE), that is $MSSE = 0.11$.

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 4.3a. To reduce the 4 segments to 3, three merging scenarios are possible:

Scenario 1: The first scenario is shown in Figure 4.5a where the first segment S_1 of Figure 4.3a(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 SSE that is $SSE_1(X, \overline{X}) = 2.49$. The matrix cell that corresponds to this merging is updated (Figure 4.4c).

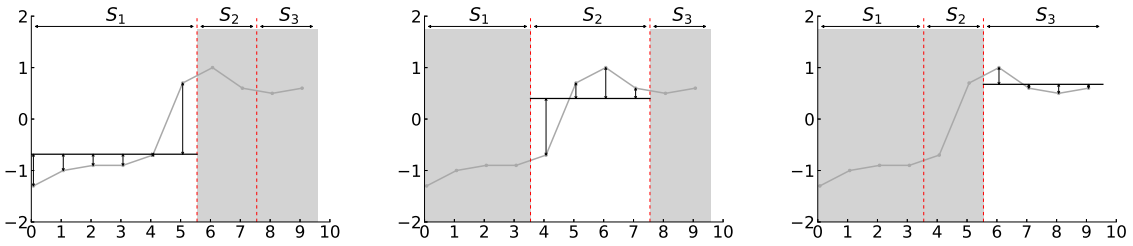
Scenario 2: This scenario is shown in Figure 4.5b in which s_3 and s_4 of the initial segmentation are merged. This merging has already been tested in the previous step ($matrix[3,4] \neq$-1), the SSE value is retrieved from our matrix from the cell $(3, 4)$. Here, $SSE_2(X, \overline{X}) = 1.70$.

Scenario 3: The last scenario is shown in Figure 4.5c, where we merge s_4 and s_5. The SSE for this merging has been already computed in step 1, that is $SSE_3(X, \overline{X}) = 0.15$.

We have now the SSE for the three scenarios. We choose the minimum SSE value, that is $MSSE = 0.15$ corresponding to the segmentation generated in Scenario 3, which is chosen for this iteration.
After this step, we have 3 segments shown in Figure 4.5c. Since, the number of segment reaches w, the algorithm ends.

4.3 PASAX : Parallel ASAX_SSE

In order to improve the accuracy of time series representations, we proposed the ASAX_SSE segmentation approach. This method is efficient when the database consists of few small time series but in case of large sets of time series, using this algorithm is highly time-consuming. ASAX_Dyn allows to improve the execution time of our segmentation approach, by means of dynamic programming. However, this method



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

Figure 4.5: The three different scenarios of ASAX_SSE segmentation with 3 segments. Scenario 4.5c is selected since it provides the minimum SSE.

may require significant system resources: memory resources especially for long time series as the size of the matrix increases with the length of the series ; and the SSE computation for all time series of the dataset is done sequentially and for this, many sequential loops have to be done at each step.

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) PASAX_DP that performs the parallelization on data; 2) PASAX_SP that makes the parallelization on segments.

4.3.1 Parallelization on Data

The main computational part of the sequential algorithm is located in the calculation of the SSE on the entire dataset that is done in a sequential loop for each segmentation, the larger is the dataset, the more this computation is long. It may therefore be advantageous to transform this computation into a GPU module. The main idea of our first parallel algorithm, called PASAX_DP (PASAX 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. Algorithm 4 presents the pseudo-code of the corresponding GPU kernel. 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.

Algorithm 4: PASAX_DP SSE computation Kernel

Input: D: time series database; *seg*: current segment(s);
Output: *sseArray*: result of SSE computation

```
1 tx = threadIdx.x // thread id in a 1D block
2 ty = blockIdx.x // block id in a 1D grid
3 bw = blockDim.x // block width, i.e number of threads per block
4 i= tx + ty * bw // compute flattened index inside the array
5 if i < D.size() then
6     sseArray[i]=SSE(D[i],seg)
```

4.3.2 Parallelization on Segments

To find the k variable size segments on a dataset of long time series with the sequential version of the algorithm many possible segmentations are tested for each step, the number of these possibilities increases as the length of time serie increases. Here, we propose PASAX_SP (PASAX 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 reached w, the host calls the GPU kernel described in Algorithm 5 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.

Algorithm 5: PASAX_SP SSE computation Kernel

Input: D: time series database; *segments*: initial segmentation; k: number of segments

Output: *sseArray*: result of SSE computation

```
1  tx = threadIdx.x // thread id in a 1D block
2  ty = blockIdx.x // block id in a 1D grid
3  bw = blockDim.x // block width, i.e number of threads per block
4  i= tx + ty * bw // compute flattened index inside the array
5  if i < k then
6      find the current segmentation seg by merging s_i and s_{i+1}
7      sse = 0
8      foreach ts in D do
9          sse = sse + SSE(ts,seg)
10     sseArray[i]=sse
```

4.4 Evaluation and results

In this section, we present the experimental evaluation of ASAX_SSE, ASAX_Dyn and PASAX. The section is organized as follows. We first present the experimental setup. Then, in Subsection 4.4.2, we compare the precision of ASAX_SSE representation with that of the existing SAX representation and the ASAX_EN proposed in chapter 3. Then, in Subsection 4.4.3, we evaluate the performance of ASAX_Dyn and PASAX by measuring the execution time of the variable-size segmentation.

4.4.1 Setup

All approaches are implemented with Python programming language. The implementation use Numba JIT compiler to optimize machine code at runtime.

The experimental evaluation was conducted on the same machine used for the experimentation of ASAX_EN presented in chapter 3. 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 precision of the approaches using several real world datasets from the UCR Time Series Classification Archive in similarity search by applying a k-Nearest Neighbor (k-NN) search, as detailed previously in Subsection 4.4.2. We evaluate the performance of the improved algorithms on the datasets taken from the same archive [11]. 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_SSE is initialized by splitting the time domain into segments of length 2 and for ASAX_EN we set the default cardinality value to 32.

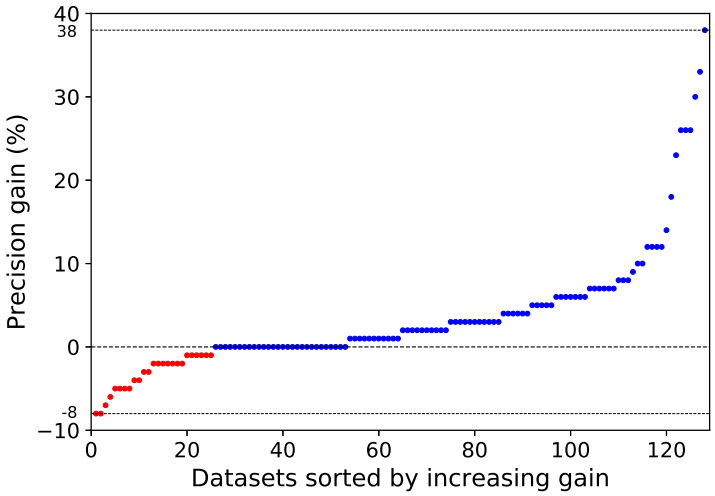
4.4.2 Precision of k-Nearest Neighbor Search

We compare the proposed ASAX_SSE and SAX in terms of precision on all 128 datasets available in the UCR Time Series Classification Archive. The precision results are reported in Figure 4.8 where the precision gain/loss (as percentage) for ASAX_SSE compared to SAX precision is measured for each dataset. Figure 4.6a shows the precision results for ASAX_GSSE (*i.e.*, ASAX_SSE using GSSE) and Figure 4.6b 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 loss in precision in represented by red dots).
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 4.8 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 (as shown in Figure 1.1). 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 kNN search.

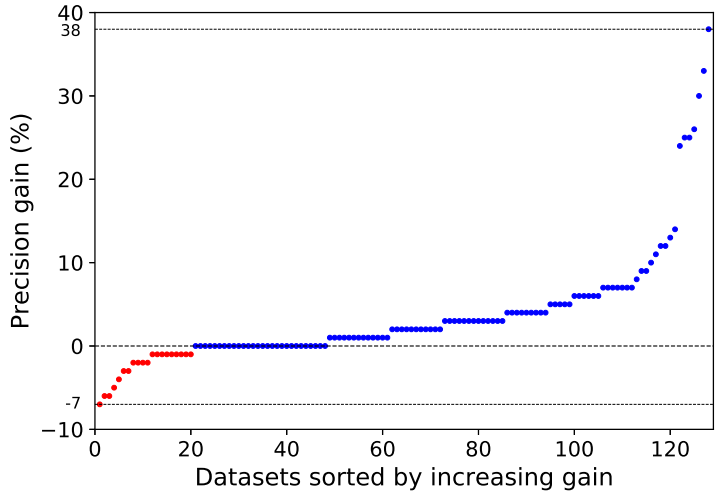
We can see that for some datasets the computed gain is zero meaning equivalent precision for ASAX_SSE and SAX due to the balanced shape of the time series over the time domain.

Regarding the few datasets where we obtain lower precision, the loss is relatively low (mostly near zero) and corresponds to a very specific, well identified, type of distribution. It is illustrated by Figure 4.7 that shows a set of time series taken from *MiddlePhalanxOutlineAgeGroup* dataset of UCR Archive. It basically contains time series that split into two distributions. The first one concerns complete time series, while the second one concerns time series that are flat on the last values (probably due to missing values that have been replaced by a constant). The right part of the dataset, with flat values, will always give higher SSE compared to the left part, forcing the algorithm to split on the left part. However, when a request Q arrives, there are two possibilities. The first one is that Q is complete and it is compared only on the left part, leading to possible errors (the observed loss in our experiments). The second one is that Q is flat on the right part and, in this case, the comparison will be accurate since the right part has low impact on retrieving its kNN.

Here, we compare the quality of ASAX_SSE, ASAX_EN and SAX representation on



(a) Precision gain for ASAX_GSSE



(b) Precision gain for ASAX_LSSE

Figure 4.6: The precision gain for ASAX_GSSE and ASAX_LSSE compared to SAX. The obtained gain is up to 38% for both methods

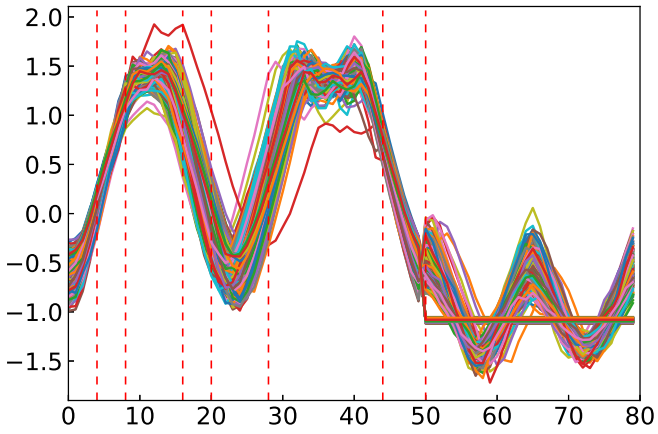


Figure 4.7: An example of dataset on which ASAX_SSE segmentation produces a precision loss

the same datasets described in Table 3.1 of the UCR Time Series Classification Archive used to evaluate ASAX_EN. For these comparison we consider only the results of ASAX_LSSE precision since The percentage of precision gain computed previously shows that the gain obtained with this method in better than the one obtained with ASAX_GSSE.

The precision results are reported in Figure 4.8. We can observe that ASAX_SSE is often much more efficient than ASAX_EN and SAX. For example, on the *ECGFiveDays* dataset presented in Figure 4.8a, we were able to achieve a precision of 93% for ASAX_SSE, 82% for ASAX_EN while it is 55% for SAX, which is a significant gain in precision.

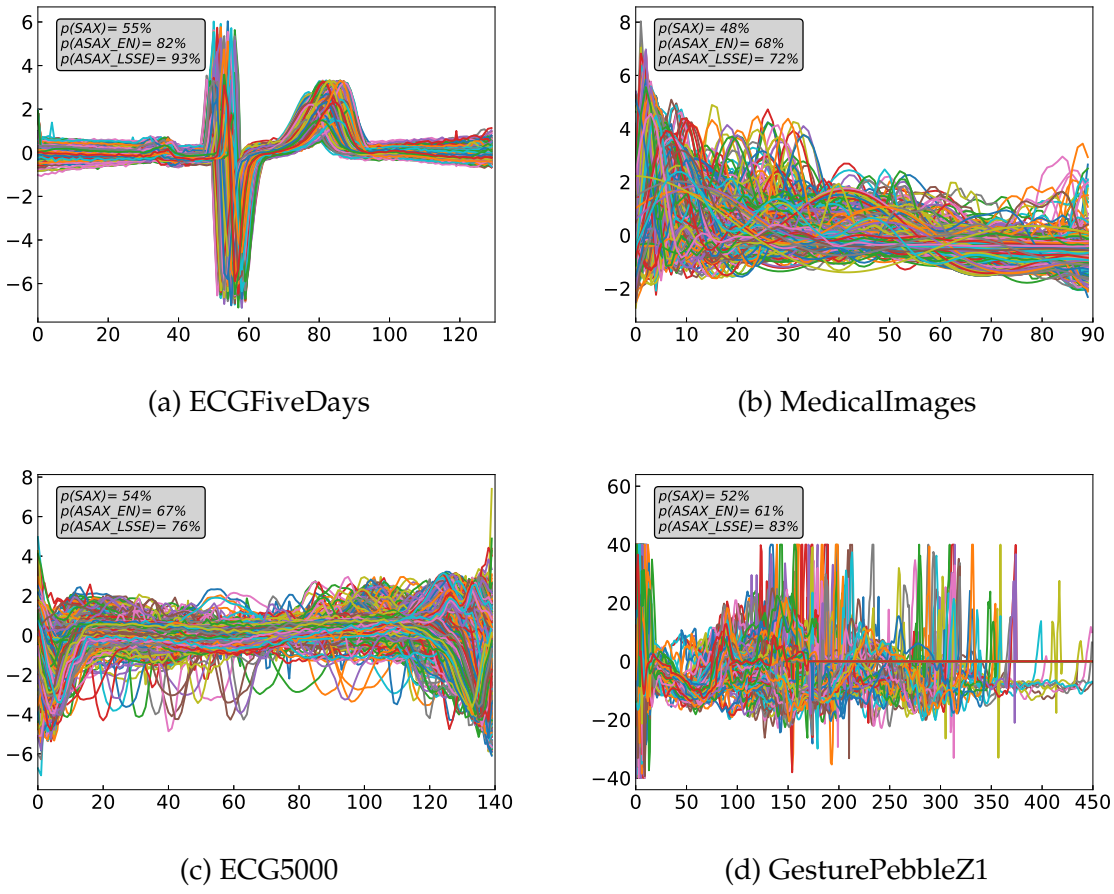
Globally, our results suggest the effectiveness of our approach and its advantage over SAX when applied to time series especially those with unbalanced distribution over the time domain.

4.4.3 Execution time of variable-size segmentation algorithms

Figure 4.9 gives the segmentation time of ASAX_EN and ASAX_LSSE on the datasets of Table 3.1. We can observe that the time cost of ASAX_LSSE is always less than the one for ASAX_EN.

Next, we measure the variable-size segmentation time cost of the improved algorithm ASAX_Dyn, and compare it to that of the basic algorithm ASAX_SSE. Figure 4.10 reports the performance gains of our improved approach ASAX_Dyn compared to the basic version of ASAX_LSSE. The variable-size segmentation time cost for the two methods is evaluated for all the datasets of the archive. We can observe that ASAX_Dyn is much more efficient and allows to have significant performance gains.

Figure 4.11 reports the computation time of variable-size segmentation for ASAX_Dyn and ASAX_LSSE over *HandOutlines* dataset, by varying the time series length. The



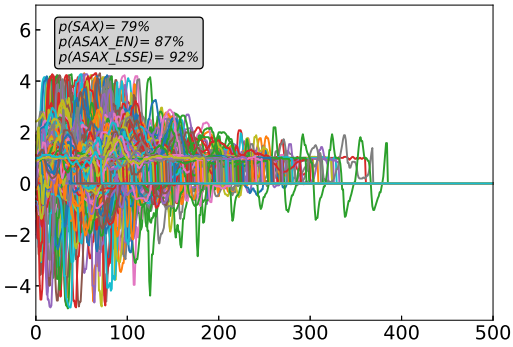
(a) ECGFiveDays

(b) MedicalImages

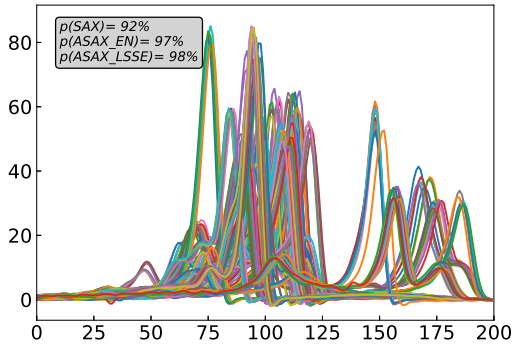
(c) ECG5000

(d) GesturePebbleZ1

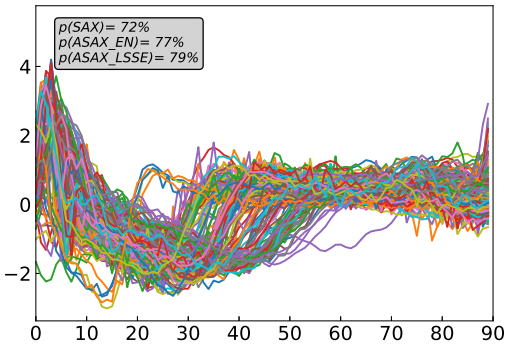
Figure 4.8: The data distribution of the tested datasets, and the precision results for each dataset. p(SAX), p(ASAX_EN) and p(ASAX_LSSE) show the precision of SAX, ASAX_EN and ASAX_LSSE respectively.



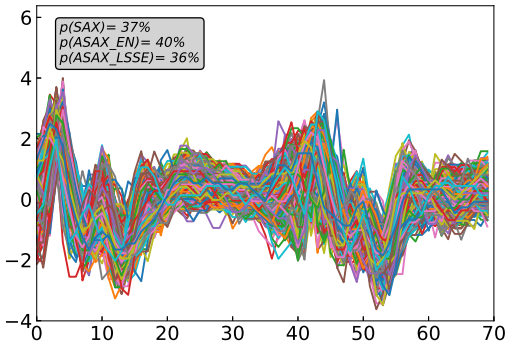
(e) AllGestureWiimoteZ



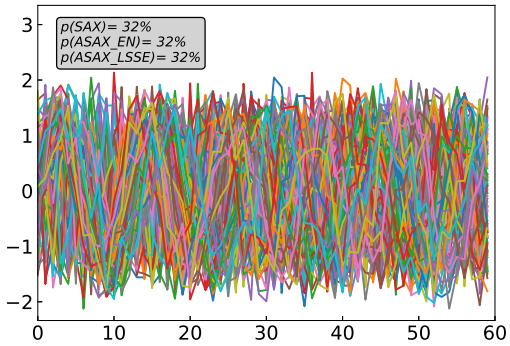
(f) Fungi



(g) ECG200



(h) SonyAIBORobotSurface1



(i) SyntheticControl

Figure 4.8: The data distribution of the tested datasets, and the precision results for each dataset. p(SAX), p(ASAX_EN) and p(ASAX_LSSE) show the precision of SAX, ASAX_EN and ASAX_LSSE respectively.

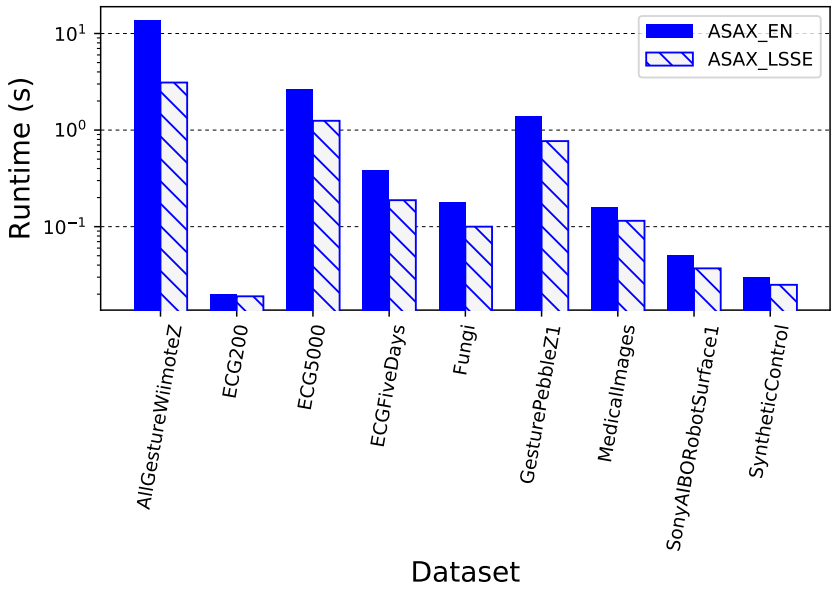


Figure 4.9: Logarithmic scale. Runtime of ASAX_LSSE and ASAX_EN segmentation algorithms for each dataset

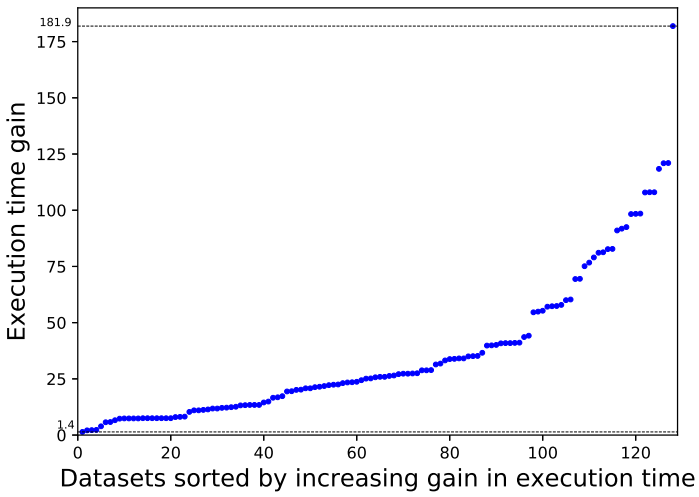


Figure 4.10: ASAX_Dyn's performance gain on ASAX_SSE in segmentation time, over all datasets of the archive

running time increases with the length of time series and, as one could expect. The basic approach ASAX_LSSE takes much more time than ASAX_Dyn. Depending on time series length, ASAX_Dyn shows performance gains that can reach $\times 182$ for the 1340 time series with length 2700 of the *HandOutlines* dataset.
Figure 4.12 illustrate ASAX_Dyn's performance gain on ASAX_LSSE in segmentation time for 5 datasets with different time series lengths. As seen, the performance gains vary significantly depending on the number of time series in relation with their length.

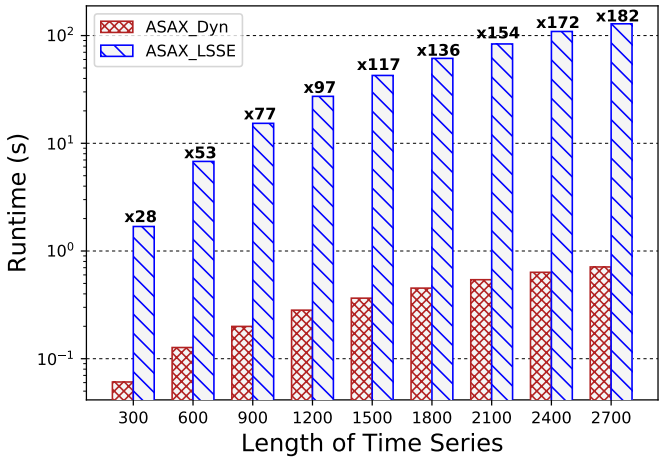


Figure 4.11: Logarithmic scale. Variable-size segmentation time for ASAX_Dyn and ASAX_LSSE as a function of time series length, over the *HandOutlines* dataset.

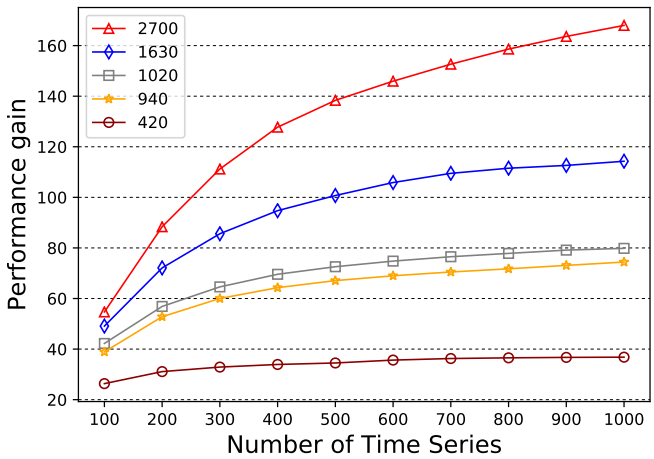


Figure 4.12: ASAX_Dyn's performance gain on ASAX_LSSE in segmentation time as a function of dataset size. The time series length is shown in the figure for each dataset.

Here, we presents the time cost of the variable-size segmentation for our proposed parallel algorithms. We measure the variable-size segmentation time costs of the parallel algorithms PASAX_DP and PASAX_SP, and compare them to that of the variable-size segmentation for the sequential algorithm PASAX. 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, PASAX_DP and PASAX_SP, only using the LSSE measurement.

Figure 4.13 and Figure 4.14 report the performance gains of our parallel approaches compared to the sequential version of ASAX_LSSE. Figure 4.13 reports the variable-

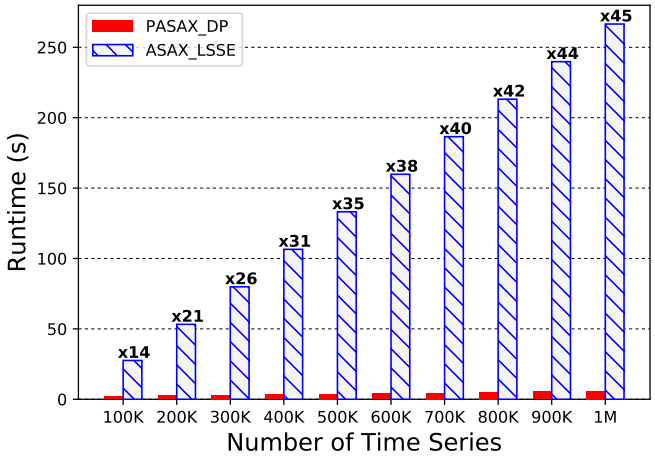


Figure 4.13: Variable-size segmentation time for PASAX_DP and ASAX_LSSE as a function of dataset size. The original time series are of length 130.

size segmentation time for the PASAX_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 PASAX_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 4.14 reports the computation time of variable-size segmentation for the PASAX_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 PASAX_SP. Depending on time series length, PASAX_SP shows performance gains that can reach $\times 24$ for 1000 time series of length 2700.

Figure 4.15 and Figure 4.16 compare the parallel segmentation computation time of our approaches. In Figure 4.15, we evaluate the two approaches with varying dataset size (number of time series) and fixed time series length. For this case, we observe that PASAX_DP is always faster than PASAX_SP. The results show that using PASAX_DP is advantageous in the case of databases of many small time series.

In Figure 4.16, we vary the time series length and we fix the dataset size for the evaluation. We notice that when time series length $n = 100$, PASAX_DP is a little faster than PASAX_SP, but when the length of time series increases, PASAX_SP becomes faster than PASAX_DP. The performance gain reaches $\times 7.5$ for time series of length

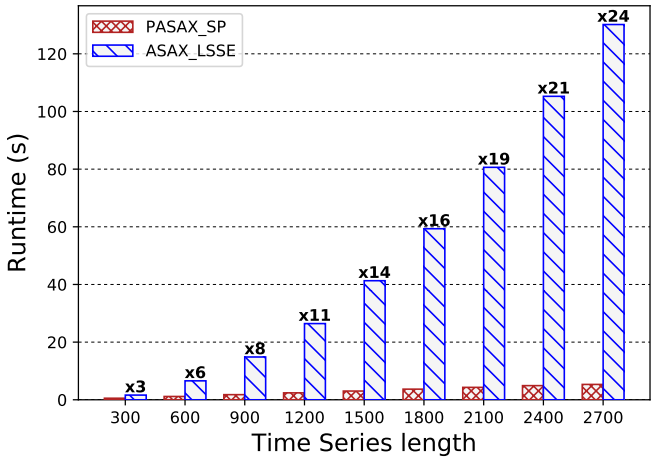


Figure 4.14: Variable-size segmentation time for PASAX_SP and ASAX_LSSE as a function of time series length. The dataset size is fixed to 1000.

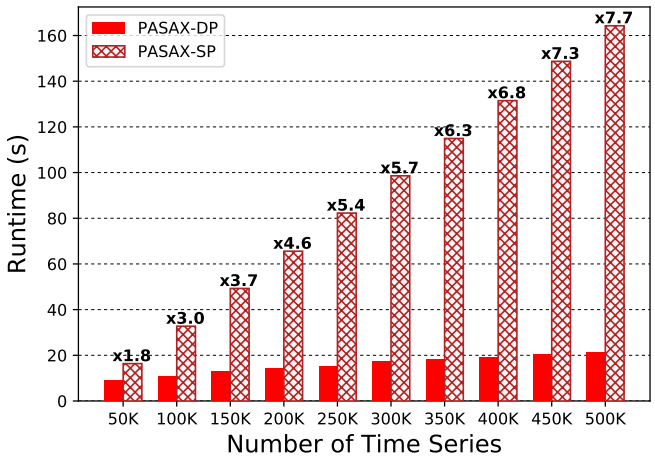


Figure 4.15: Comparison of parallel segmentation time using PASAX_DP and PASAX_SP, as a function of dataset size. The original time series are of length 300.

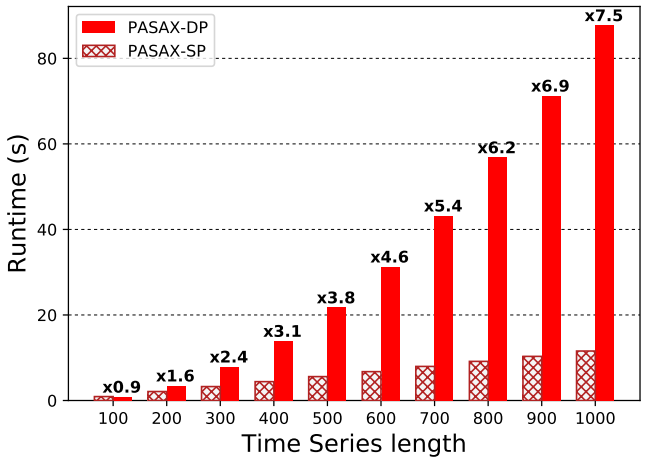


Figure 4.16: Comparison of parallel segmentation time using PASAX_DP and PASAX_SP, as a function of time series length. The dataset size is fixed to 10 000.

1000. PASAX_SP allows better performance gains when the database consists of few and long time series.

4.5 Conclusion

We proposed ASAX_SSE, an efficient solution for segmenting time series. ASAX_SSE is a simple method with high performance, it can reduce significantly the error of the representation calculation, by taking into account the sum of squared errors (SSE). We evaluated the performance of our segmentation approach through experiments using more than 120 real world datasets. The experimental results illustrate the excellent performance of ASAX_SSE compared to SAX and our algorithm ASAX_EN (*e.g.*, for the *ECGFiveDays* dataset, the precision of ASAX_SSE is 93% compared to 55% for SAX and 82% for ASAX_EN).

We proposed an efficient algorithm, called ASAX_Dyn, for improving the execution time of our segmentation approach ASAX_LSSE, by means of dynamic programming. The results also show the effectiveness of our dynamic programming algorithm, *e.g.*, up to $\times$182 faster than the basic ASAX_SSE algorithm over *HandOutlines* dataset.

We also proposed two parallel algorithms for improving the execution time of ASAX_SSE using GPUs. The results illustrate the effectiveness of our parallel algorithms, *e.g.*, up to $\times$45 faster than the sequential algorithm for 1M time series.

CHAPTER 5

TIME SERIES REPRESENTATION BASED ON THE EXACT ERROR

In this chapter, we propose an exact approach for time series segmentation using a dynamic programming algorithm. The rest of this chapter is organized as follows. In Section 5.2, we describe the details of our new segmentation approach. In section 5.3, we evaluate the performance of our solution through experiments on all datasets of the UCR archive. Finally, in Section 5.4, we conclude.

5.1 Motivation and Overview of the Proposal

Several ways exist for approaching hard optimization problems. They may either be of exact or heuristic nature. Exact approaches are guaranteed to yield proven optimal solutions when they are given enough computation time. In contrast, heuristics only aim at finding reasonably good approximate solutions usually in a more restricted time, and performance guarantees are typically not provided. In the previous works of this thesis, we proposed two methods for segmenting time series. These methods try to find the w segments that minimize the approximation error based on a top-down (ASAX_EN) and a bottom-up (ASAX_SSE) algorithms. The splitting produced by these methods gives good approximation results in a reasonable time due to the their heuristic approach. In this chapter, we propose an exact algorithm for solving the segmentation problem based on the SSE measurement in order to achieve better results and compare them with those of our approximate methods. As for the previous work, we choose the SSE measure since it gives good accuracy results for the ASAX_SSE approach. Our contributions are as follows:

- We propose an exact segmentation technique, called EASAX_Dyn, that finds the exact variable size segmentation with minimum SSE of the representation using dynamic programming.

- We implemented our approach and conducted empirical experiments using more than 120 real world datasets. The results illustrate that EASAX_Dyn can obtain better performance gains in terms of precision for similarity search compared to the heuristic approach ASAX_SSE.

5.2 EASAX_Dyn DP Algorithm description

Given a set of time series, the problem we address is to find the most efficient segmentation that minimizes the SSE of the representation. In other words, the problem consists of deciding how to do the splittings in order to guarantee the minimum error. We have many options to make the segments. A simple solution is to try all possible segmentations, calculate the cost for each segmentation and return the one that leads to the minimum SSE. For time series of length n, we can place the first split in n-1 positions. So when we place a split, we divide the problem into subproblems of smaller size. Therefore, the problem has optimal substructure property and can be easily solved using recursion. Minimum SSE value of the segmentation is equal to the minimum of all $n-1$ split placements. Since similar sub-problems are called again, this problem has overlapping sub-problems property. So this segmentation problem has the properties of a dynamic programming problem.

In this section, we propose EASAX_Dyn, our exact method that finds the variable size segmentation that guarantees the minimum SSE on the representation using dynamic programming.

Algorithm 6: EASAX_Dyn variable-size segmentation

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

1 EM = 3D matrix of size (w, n, n) values initialized to -1 // the matrix containing the minimum approximation error for each possible segment
2 SM = 3D matrix of size (w, n, n) values initialized to -1 // the matrix containing the split position done for each possible segments
3 $segments$ = { }
4 init_EM(D,n)
5 segmentation(w,1,n)
6 find_segmentation(w,1,n)
7 $segments$.sort()
8 **return** $segments$

The pseudo-code of this segmentation method can be seen in Algorithm 6. The input is a dataset D that contains time series, the time series length n and the final number of segments w. First, the algorithm starts by allocating the data structure needed : EM and SM 3D matrices, all values are initialized to -1 (Lines 1,2). Next, it calls the *init_EM* function to calculate the SSE for each possible segment that can be formed on the time series length using a single frame. This calculation will be the basis for getting the overall segmentation error when placing two or more segments, until reaching w segments in total. Then, the algorithm calls the *segmentation* and $find\_segmentation$ function that finds the optimal segmentation which minimizes the representation error (Lines 5,6). Finally, we can obtain our final segmentation after sorting the set *segments* (Line 7).

Let us now describe in details of each function used in this algorithm. We start with the first function called the *init_EM* function described in Algorithm 7. This function takes as input a database of time series denoted as D and their length denoted n. The goal of this function is to calculate the SSE of each possible segment that we can form on the n values of the time domain (using a single frame). For each segment (i,j) with i from 1 to n and j from i to n, we compute the representation error using SSE on this segment for each time series ts of the database D. If i is equal to j, meaning that the segment contains a single value, the representation error on this frame is then 0, this value is stored in the corresponding position in the EM matrix denoted $EM[1, i, j]$ (Line 4). If $i \neq j$ the sum of the SSE is computed for all time series ts in the database D on the segment s and stored in EM (Lines 9,10). Notice that when storing the obtained SSE value in $EM[1, i, j]$ the first parameter here is always set to 1 meaning that one single segment is formed between i and j timestamps positions. For other values of this parameter the *segmentation* function computes the error and stores the results gradually in the process.

Algorithm 7: init_errorMatrix

Input: D: time series database; n: the length of time series

```
1  for i = 1 to n do
2      for j = i to n do
3          if i == j then
4              EM[1,i,j]=0
5          else
6              sse=0
7              s=[i,j]
8              foreach ts in D do
9                  sse = sse + SSE(ts,s)
10             EM[1,i,j] = sse
```

Let's move to the *segmentation* function of Algorithm 8 which is the most important part of the algorithm. This function tests all possible segmentations that can be performed, in order to find the segmentation that minimizes the SSE measurement of the representation. This procedure is done recursively. The first call takes as input all n values of the series that form a single starting segment (i,j) and the goal is to find the splitting positions of the the w segments by calculating the minimum error on the segmentation. In the general case, for a given time interval (i,j) and a number of segments s (budget given to a segment), it has to find the split position that divides the segment into 2 parts that guarantee the minimum SSE: the first part (left part) with a segment budget equal to 1 and the second part (right part) the remaining s-1 segments. This function works as follows.

For the input (i,j) and s, if the number of segments that needs to be formed s is equal to 1, meaning that there is only one segment left to form between i and j, it returns

the corresponding error on the EM matrix whose values have been pre-calculated by $init_E M$ function (Line 2).

If the estimated error between i and j with a budget limit of s is different from -1,*i.e,* the error has already been calculated during the previous steps of the process, this value is simply returned (Line 4). Otherwise, we need to test all possible splittings of the interval (i,j). For each split position denoted as k (k from i to j), the sum of the error for placing one segment on the first interval (i,k) and the minimum error for placing $s-1$ segments on the second interval ($k+1$,j) is computed and stored in *error* (Line 9). This latter is obtained recursively until the budget of segments s reaches 1. If the total error computed for the two resulting parts of the segmentation according to the split k is the minimum obtained so far, the algorithm keeps track of this segmentation, the minimum error between i and j with s segments denoted as $EM[s,i,j]$ and the split position $split$ is updated (Line 11, 12). After having tested all possible divisions at this step of the segmentation, the algorithm stores the split position $split$ that provides the minimum SSE in the SM matrix (Line 13). Finally, the minimum error is returned.

At the end of this process, we can obtain the global segmentation from the information contained in the EM and SM matrices using the function which will be described bellow.

Algorithm 8: segmentation

Input: s: the segment budget ; i: start point of the segment ; j: end point of the segment

Output: $EM[s,i,j]$ the minimum error between timestamp i and j with s segments

```
1  if s == 1 then
2      return EM[1,i,j]
3  if EM[s,i,j] ≠ −1 then
4      return EM[s,i,j]
5  EM[s,i,j]=∞
6  split = null
7  for k = i to j − 1 do
8      if (k − i + 1) ≥ 1 and (j − k) ≥ s − 1 then
9          error = segmentation(1,i,k) + segmentation(s − 1,k + 1,j)
10         if error < EM[s,i,j] then
11             EM[s,i,j]=error
12             split = k
13 SM[s,i,j] = split
14 return EM[s,i,j]
15
```

Finally, we describe our function $find\_segmentation$ of Algorithm 9. For a given interval in the time domain delimited between i and j and a number of segments s

formed between i and j, this recursive function returns the position of the first split done that is stored in the SM matrix (Line 2) and this position is added in the set *segments* (Line 3). Recall that the first resulting part has a segment budget of 1, thus the first part forms a final segment that is no longer divided, and for the remaining part the function is called recursively to find the next divisions. This function is called recursively for the resulting intervals (Line 4), until all intervals segments budget becomes equal to 1.

After having obtained the splittings positions stored in *segments*, they are sorted on the time domain to obtain the correct order of the splits of the final segmentation.

Algorithm 9: find_segmentation

Input: s: the segment budget ; i: start point of the segment ; j: end point of the segment

1 **if** $s > 1$ **then**
2 $split$=$SM[s, i, j]$ // find the split position for the segment (i,j)
3 *segments*.add($split$) // add the position to the set *segments*
4 find_segmentation($s-1$,$split$+1,j) // find the segmentation for the right resulting segment

5.3 Performance Evaluation

In this section, we report experimental results that show the quality and the performance of our exact segmentation algorithm EASAX_Dyn, illustrating the precision of the segmentation. We compare our work with SAX and the previously proposed ASAX_SSE.

We implemented the approaches in Python programming language and Numba JIT compiler is used to optimize machine code at runtime.

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 the UCR Time Series Classification Archive datasets. As for the previous experiments, the default value of w the length of the approximate representations is reduced to 10% of the original time series length.

5.3.1 Precision of k-Nearest Neighbor Search

The precision results are reported in Figure 5.1 where the precision gain/loss (as percentage) for EASAX_Dyn compared to SAX is measured for each dataset. We can observe a precision gain for the large majority of datasets. We obtained a gain in precision for 84% of the datasets with ASAX_SSE. The maximum gain achieved 39% for EASAX_Dyn on the *ECGFiveDays* dataset where the precision is 94% for EASAX_Dyn and 55% for SAX.

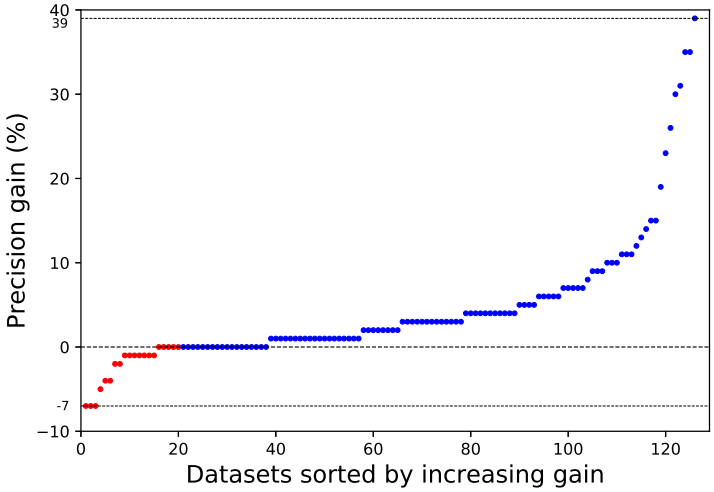


Figure 5.1: The precision gain computation result for EASAX_Dyn approach compared to SAX. The maximum gain achieved is 39 percent.

	Gain on archive datasets	max gain	average gain
ASAX_LSSE	84%	38%	4.8%
EASAX_Dyn	88%	39%	5.5%

Table 5.1: Some statistic information to compare EASAX_Dyn and ASAX_SSE

Let us now compare these results with those of our best heuristic method ASAX_LSSE. Table 5.1 shows some statistic information for both methods. We can see that the number of datasets on which we observed a better gain in precision is higher for EASAX_Dyn (88%) compared to ASAX_LSSE (84%), as well as for the max gain and the average precision gain. From these sresult, we can state that the exact method performs better than ASAX_LSSE.

5.3.2 Time cost of EASAX_Dyn segmentation algorithm

In this section, we measure the execution time of EASAX_Dyn and compare it to that of ASAX_dyn.

Figure 5.2 reports the performance gains of our approach ASAX_Dyn compared to EASAX_Dyn. The variable-size segmentation time cost for the two methods is evaluated for all the datasets of the archive. As expected, we can clearly see that that EASAX_Dyn is highly time consuming compared to ASAX_Dyn, and its execution time increases considerably as the size of the database increases.

5.4 Conclusion

In this chapter, we proposed EASAX_Dyn, an efficient solution that finds the segmentation leading to the minimum SSE of the representation. We evaluated the performance of our solution over more than 120 real world datasets. The experimental

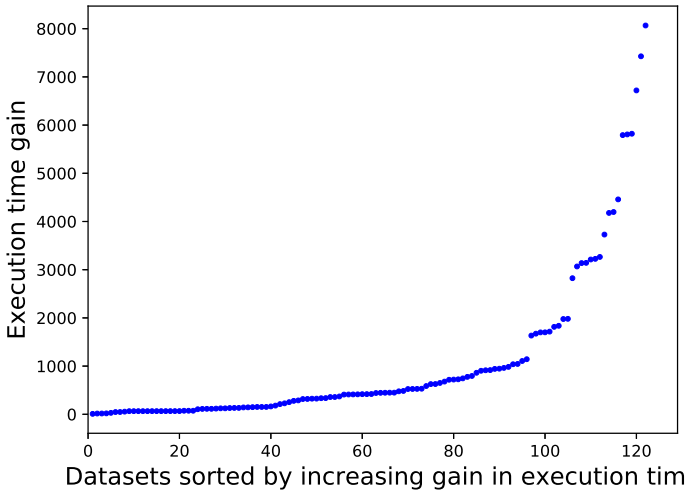


Figure 5.2: ASAX_Dyn's performance gain compared to EASAX_Dyn in segmentation time, over the datasets of the UCR archive.

results illustrate the excellent performance of EASAX_Dyn compared to SAX in terms of precision. Compared with the previous work ASAX_SSE, our exact solution provides better performance gains in terms of precision for similarity search. However, the difference is not high between the two methods. This shows that the approximate algorithm performs well on the segmentation and produces a representation whose precision is very close to that of the exact method. Furthermore, the experiments show that the segmentation time of the approximate algorithm, i.e., ASAX_Dyn, is very low compared to EASAX_Dyn, e.g., up to 8000 times lower. Therefore, our approximate approach ASAX_Dyn provides a good compromise between accuracy and the computation time for variable-size segmentation of time series.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

It is generally known that the time series, as essential high-dimensional collections of data, permeate all areas of business and scientific research. Therefore, the time series analysis is an active and attractive research area.
The key component in the data mining process of discovering the structure in time series data is the segmentation of time series. As a data mining research problem, it is focused on the division of time series into adequate internally homogeneous segments. Time series segmentation is often used as a preprocessing step in time series data mining applications, so that the quality of the output of the segmentation process is the key factor that determines the quality of the analysis and validity of the time series models.
In this thesis, we focused on the problem of time series representation. We developed efficient techniques for variable-size segmentation of the time domain, aiming at improving the accuracy of time series representations in order to get better precision during similarity search operations. In this chapter, we summarize and discuss our main contributions and then give some research directions for future work.

6.1 Contributions

This thesis includes the following main contributions related to time series approximation.

- **Variable size segmentation for efficient representation of non-uniform time series datasets based on entropy**. In this contribution, our main challenge was the approximation of time series using a variable-size segmentation based on entropy adopting a top-down strategy. In this part of the thesis, the representation is built based on the principles of SAX. We proposed ASAX_EN (Adaptive SAX based on entropy), a new approximation technique that considers the time series distribution on the time domain and performs variable-size segmentation, by using the entropy of symbolic representations. This technique allows reducing information loss and thus increasing the accuracy of time series representations. We evaluated its performance using several real world datasets. The experimental results show that ASAX_EN can obtain significant performance gains in terms of precision for similarity search compared to SAX, particularly for dataset with unbalanced (non-uniform) distribution, *e.g.*, for the *EGCFiveDays* dataset that has

a non-uniform distribution in the time domain, the precision of ASAX_EN is 82% compared to 55% for SAX.

- **Optimized techniques for time series segmentation based on the approximation error**. In this part of the thesis, we addressed the problem of finding the variable-size segmentation that minimizes the approximation error of the time series representation using a bottom-up approach. We have proposed ASAX_SSE, a simple and efficient method with high performance that can reduce significantly the error of the representation calculation, by taking into account the sum of squared errors (SSE). We evaluated the performance of our segmentation approach through experiments using more than 120 real world datasets. The experimental results illustrate the excellent performance of ASAX_SSE compared to SAX and our algorithm ASAX_EN (*e.g.*, for the *ECGFiveDays* dataset, the precision of ASAX_SSE is 93% compared to 55% for SAX and 82% for ASAX_EN). Also, efficient algorithms for improving the execution time of our segmentation approach have been proposed. We proposed an efficient algorithm, called ASAX_Dyn, for improving the execution time of our segmentation approach ASAX_LSSE, by means of dynamic programming. The results also show the effectiveness of our dynamic programming algorithm, *e.g.*, up to $\times 182$ faster than the basic ASAX_SSE algorithm over *HandOutlines* dataset.

  We also proposed two parallel algorithms for improving the execution time of ASAX_SSE using GPUs. The results illustrate the effectiveness of our parallel algorithms, *e.g.*, up to $\times 45$ faster than the sequential algorithm for 1M time series.

- **Time series representation based on the exact error**. We addressed the problem of segmenting the time series based on the exact approximation error of the representation. We proposed EASAX_Dyn, an exact segmentation technique that finds the exact variable size segmentation with minimum SSE of the representation using dynamic programming. We evaluated its performance over more than 120 real world datasets. The experimental results illustrate the excellent performance of EASAX_Dyn compared to SAX in terms of precision. Also, we compared EASAX_dyn to ASAX_dyn in terms of precision, the result have shown that ESAX_Dyn provides better performance gains. However, the difference in precision is not high between the two methods which shows that the approximate algorithm performs well on the segmentation and produces a representation whose precision is very close to that of the exact method. The results of other experiments show that the segmentation time of the approximate algorithm is very low compared to EASAX_Dyn, and therefore we can conclude that our approximate approach ASAX_Dyn provides a good compromise between accuracy and the computation time for variable-size segmentation of time series.

6.2 Directions for Future Work

The results presented in this thesis leave room to further improvement. Below, we present some research directions for future work:

- **Trend-aware symbolic representation**. Our approximation methods have illustrated their effectiveness in improving the time series representation quality compared to SAX. However, as SAX, they only consider the average value of the segment and miss important information in a segment, particularly the trend of the value change in the segment. As a result, two segments with different shapes but similar average values are transformed into the same symbol. To overcome this drawback, our segmentation methods can be improved by capturing the trend through the variations between segment points and the mean. These variations can be generated for each segment and coded as string. For using such trend-aware representation in kNN search tasks, we need a modified similarity measurement to compute the similarity between pairwise time-series not only based on the segments mean, but also by taking into account time series' trends.

- **Indexing**. Another possible future work concerns the construction of an indexing method for the proposed representation. There is an increasingly demand, by several applications in diverse domains, for developing techniques able to index and mine very large collections of time series. Examples of such applications come from astronomy, biology, web, and other domains. iSAX is the indexable version of SAX, which allows extensible hashing and indexing of very large time series databases.. A potential future work is to adapt iSAX for the variable-size segments.

- **Extension to other data mining tasks**. In this thesis, we optimized our segmentation methods for kNN search over time series. A future research direction is the extension of our segmentation methods to other data mining tasks such as classification, anomaly detection and motif discovery.

- **Multidimensional time series**. Another future work is the extension of our segmentation methods to multidimensional time series. The recent advances in sensor and GPS technology have made it possible to collect large amounts of spatiotemporal data, so there is increasing interest to perform data analysis tasks over multidimensional data. Such data types arise in many applications where the location of a given object is measured repeatedly over time. Examples include animal mobility experiments, sign language recognition, mobile phone usage, etc. The trajectory of a moving object is typically modeled as a sequence of consecutive locations in a multidimensional Euclidean space.

PUBLICATIONS

- Lamia Djebour, Reza Akbarinia, Florent Masseglia. 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, 2022.
- Lamia Djebour, Reza Akbarinia, Florent Masseglia. Parallel Techniques for Variable Size Segmentation of Time Series Datasets. In : 26th European Conference on Advances in Databases and Information Systems (ADBIS), 2022.
- Lamia Djebour, Reza Akbarinia, Florent Masseglia. ASAX : Segmentation adaptative basée sur la quantité d'information pour SAX. 37e Conférence sur la Gestion de Données - Principes, Technologies et Applications (BDA), 2021.
- Lamia Djebour, Reza Akbarinia, Florent Masseglia. Variable-Size Segmentation for Time Series Representation (under review in a journal)

BIBLIOGRAPHY

[1] La gilberta. http://lagilberta.pi.ingv.it/seismo/en/new-earthquake-in-the-chianti-area/.

[2] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.

[3] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *Proc. of the 4th Int. Conf. on FODO*, 1993.

[4] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD Conf.*, pages 599–610, 2004.

[5] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with i SAX2+. *Knowl. Inf. Syst.*, 2014.

[6] Kaushik Chakrabarti, Eamonn Keogh, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Trans. Database Syst.*, 27(2):188–228, 2002.

[7] Kin-pong Chan and Ada Wai-Chee Fu. Efficient time series matching by wavelets. In *Proc. of the ICDE*, 1999.

[8] Moses Charikar. Similarity estimation techniques from rounding algorithms. pages 380–388, 01 2002.

[9] Q. Chen, L. Chen, X. Lian, Y. Liu, and J. X. Yu. Indexable pla for efficient similarity search. In *VLDB Conf.*, pages 435–446, 2007.

[10] Richard Cole, Dennis Shasha, and Xiaojian Zhao. Fast window correlations over uncooperative time series. In *KDD Conf.*, pages 743–749, 2005.

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

[12] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1:1542–1552, 2008.

[13] Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, December 2012.

[14] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. of the SIGMOD*, 1994.

[15] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. *Proceeding VLDB '99 Proceedings of the 25th International Conference on Very Large Data Bases*, 99, 05 2000.

[16] Yun-Wu Huang and Philip Yu. Adaptive query processing for time-series data. pages 282–286, 08 1999.

[17] Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 189–197, 2000.

[18] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability*, volume 26 of *Contemporary Mathematics*, pages 189–206, 1984.

[19] K. Kalpakis, D. Gada, and V. Puttagunta. Distance measures for effective clustering of arima time-series. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 273–280, 2001.

[20] Eamonn J. Keogh, Kaushik Chakrabarti, Michael J. Pazzani, and Sharad Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowl. Inf. Syst.*, 3(3):263–286, 2001.

[21] Eamonn J. Keogh and Michael J. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *KDD*, 1998.

[22] Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 614–623, 1998.

[23] J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *SIGMOD*, 2003.

[24] J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: A novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 2007.

[25] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. Matrix profile X: VALMOD - scalable discovery of variable-length motifs in data series. In *SIGMOD*, 2018.

[26] B. Lkhagva, Yu Suzuki, and K. Kawagoe. New time series data representation esax for financial applications. In *ICDE Workshops*, 2006.

[27] Oded Maimon and Lior Rokach. *The Data Mining and Knowledge Discovery Handbook*, volume 1. 01 2005.

[28] Alex Nanopoulos, Rob Alcock, and Yannis Manolopoulos. Feature-based classification of time-series data. *International Journal of Computer Research*, 10:49–61, 01 2001.

[29] Antonello Panuccio, Manuele Bicego, and Vittorio Murino. A hidden markov model-based approach to sequential data clustering. pages 734–742, 08 2002.

[30] Ivan Popivanov and Renée J. Miller. Similarity search over time-series data using wavelets. *Proceedings 18th International Conference on Data Engineering*, pages 212–221, 2002.

[31] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, 2012.

[32] Chotirat Ratanamahatana and E. Keogh. Everything you know about dynamic time warping is wrong. 01 2004.

[33] Chotirat Ratanamahatana, Jessica Lin, Dimitrios Gunopulos, Eamonn Keogh, Michalis Vlachos, and Gautam Das. *Mining Time Series Data*, pages 1049–1077. 07 2010.

[34] K.V. Ravi Kanth, Divyakant Agrawal, Amr El Abbadi, and Ambuj Singh. Dimensionality reduction for similarity searching in dynamic databases. *Computer Vision and Image Understanding*, 75(1):59–72, 1999.

[35] Paola Sebastiani, Marco Ramoni, Paul Cohen, John Warwick, and James Davis. Discovering dynamics using bayesian clustering. volume 1642, pages 199–210, 08 1999.

[36] D. Shasha and Y. Zhu. *High Performance Discovery in Time series, Techniques and Case Studies*. Springer, 2004.

[37] Hagit Shatkay and Stan Zdonik. Approximate queries and representations for large data sequences. volume 536-545, pages 536–545, 01 1996.

[38] J. Shieh and E. Keogh. isax: Indexing and mining terabyte sized time series. In *KDD Conf.*, 2008.

[39] Zbigniew R. Struzik and Arno Siebes. The haar wavelet transform in the time series similarity paradigm. In Jan M. Żytkow and Jan Rauch, editors, *Principles of Data Mining and Knowledge Discovery*, pages 12–22. Springer Berlin Heidelberg, 1999.

[40] Youqiang Sun, Jiuyong Li, Jixue Liu, Bingyu Sun, and Christopher Chow. An improvement of symbolic aggregate approximation distance measure for time series. *Neurocomputing*, 138:189–198, 08 2014.

[41] C. C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. J. Keogh. Matrix profile I: all pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *ICDM, 2016*.

[42] Byoung-Kee Yi and Christos Faloutsos. Fast time sequence indexing for arbitrary lp norms. *Proceedings of the 26th International Conference on Very Large Data Bases, VLDB'00*, pages 385–394, 01 2000.

[43] Chaw Zan and Hayato Yamana. An improved symbolic aggregate approximation distance measure based on its statistical features. pages 72–80, 11 2016.

[44] Haowen Zhang, Yabo Dong, and Duanqing Xu. Entropy-based symbolic aggregate approximation representation method for time series. In *IEEE Joint Int. Information Technology and Artificial Intelligence Conference (ITAIC)*, pages 905–909, 2020.

[45] Yan Zhu, Chin-Chia Michael Yeh, Zachary Zimmerman, Kaveh Kamgar, and Eamonn Keogh. Matrix profile xi: Scrimp++: Time series motif discovery at interactive speeds. pages 837–846, 11 2018.

[46] Yan Zhu, Zachary Zimmerman, Nader Shakibay Senobari, Chin-Chia Michael Yeh, Gareth Funning, Abdullah Mueen, Philip Brisk, and Eamonn Keogh. Matrix profile ii: Exploiting a novel algorithm and gpus to break the one hundred million barrier for time series motifs and joins. pages 739–748, 12 2016.

[47] Yunyue Zhu. *High Performance Data Mining in Time Series: Techniques and Case Studies*. Phd thesis, New York University, 2004.

[48] Kostas Zoumpatianos and Themis Palpanas. Data series management: Fulfilling the need for big sequence analytics. In *ICDE*, 2018.