Joint symbolic aggregate approximation of time series

Xinye Chen
Faculty of Mathematics and Physics
Charles University
Prague, Czech Republic
xinye.chen@mff.cuni.cz

ABSTRACT

The increasing availability of temporal data poses a challenge to time-series and signal-processing domains due to its high numerosity and complexity. Symbolic representation outperforms raw data in a variety of engineering applications due to its storage efficiency, reduced numerosity, and noise reduction. The most recent symbolic aggregate approximation technique called ABBA demonstrates outstanding performance in preserving essential shape information of time series and enhancing the downstream applications. However, ABBA cannot handle multiple time series with consistent symbols, i.e., the same symbols from distinct time series are not identical. Also, working with appropriate ABBA digitization involves the tedious task of tuning the hyperparameters, such as the number of symbols or tolerance. Therefore, we present a joint symbolic aggregate approximation that has symbolic consistency, and show how the hyperparameter of digitization can itself be optimized alongside the compression tolerance ahead of time. Besides, we propose a novel computing paradigm that enables parallel computing of symbolic approximation. The extensive experiments demonstrate its superb performance and outstanding speed regarding symbolic approximation and reconstruction.

KEYWORDS: time series analysis, symbolic aggregate approximation, data compression, parallel computing

1 INTRODUCTION

Time series is of naturally high numerosity in the real world. Most algorithms are limited to the computational load for dealing with large-scale data. Therefore, it is very desired to compute a representation that reduces the numerosity while preserving the essential characteristics of time series, and the reasonable representation in time series often leads to a boost in algorithmic performance and dramatically alleviates the pressure of compute resources, i.e., symbolic approximation of time series has been demonstrated to speed up neural network inference [13]. However, computing symbolic representation for large-scale time series is tricky due to its high computational complexity.

The adaptive Brownian bridge-based symbolic aggregation (ABBA) method as well as its accelerant variant fABBA is one of the state-of-the-art symbolic approximation techniques regarding reconstruction error in time series domains. However, it requires transforming one single time series at a time, which shows clumsy behavior for multiple time series, especially in a large-scale manner. Besides, this method is inherently sequential, which makes it hard to fully utilize available computing resources. More importantly, the consistency of symbols is not guaranteed. The consistency here means each distinct symbol carries the same information in any sample of multiple time series. For example, the symbol "a" that appeared in

a time series should be identical to the "a" in another time series. Besides, the parameter tuning is intractable without prior knowledge, although this problem is already mitigated with fABBA by using tolerance-dominated digitization.

Our application of interests focuses on symbolizing multivariate/multiple time series in a unified manner. We propose a joint symbolic representation framework that addresses the aforementioned issues and enables parallelism. The extensive experiments demonstrate that the proposed algorithm can achieve significant speedup while retaining the competing performance of representation reconstruction, particularly for large-scale time series. The software has been integrated into PyPI registered software fABBA[1].

Our contribution is summarized as follows:

(1) This paper analyzes the clustering in the digitization between ABBA and fABBA, and proposes a sampling-based k-means to accelerate the ABBA method while retaining its original accuracy.

(2) A joint symbolic aggregate approximation method is proposed that enables a consistent symbolization for multivariate or multiple time series. Based on that, a novel parallel computing scheme for the symbolic approximation of time series, a multithreading test was performed to show its significant speedup over ABBA and fABBA.

(3) Based on the Brownian bridge modeling, the provably error-bound method is proposed to automatically determine the hyper-parameter setting for fABBA digitization, which enables less prior knowledge of hyper-parameter tuning required for users.

The remainder of this paper is structured as follows. Section 2 discusses related work of symbolic representation as well its applications. Section 3 reviews the necessary notions of ABBA framework. Section 4 expands the existing digitization analysis and presents a sampling-based algorithm that can speed up the vector quantization-based digitization, and also introduce a hyperparameter choosing method based on Brownian bridge modeling. Section 5 formally introduces our framework of joint symbolic approximation. Section 6 shows the empirical results of various competing algorithms and section 7 concludes the paper.

2 RELATED WORK

Symbolic time series representation has important applications in time series analysis, such as clustering [20, 23, 32] and time series classification [19, 30, 34], forecasting [13], event prediction [37], anomaly detection [33], and motif discovery [15, 21, 23]. In this section, we briefly review some works on symbolic time series representation as well as its applications. The symbolic representation

---

methods for time series as well as its analysis are too large a pool of literature to survey in detail, due to the limited space we only discuss a few typical ones that are mostly related to our research.

SAX [22] is the first symbolic time series representation that reduces the dimensionality of time series and allows indexing with a lower-bounding distance measure. It starts a trend that employs symbolic representation in numerous downstream time series tasks which achieves significant success, e.g., pattern search (SAXRegEx [36]), clustering (SAX Navigator [32], SPF [20]), anomaly detection (HOT SAX [17], TARZAN [18]) and time series classification (SAX-VSM [34], BOPF [19], MrSQM[30]). SAX spawns various enhanced variants, e.g., 1d-SAX [29], ESAX [25], pSAX and cSAX [4]; Their success is achieved either by acceleration or accuracy, but SAX still receives a wide popularity due to its appealing simplicity and speed.

ABBA [12] utilizes adaptive polygonal chain approximation followed by mean-based clustering to achieve symbolization of time series. The reconstruction error of the representation can be modeled as a random walk with pinned start and end points, i.e., a *Brownian bridge*. fABBA [6], the variant, uses an efficient greedy aggregation (GA) method to replace the k-means clustering, which speedups the digitization by order of magnitudes. Both ABBA and fABBA have been empirically demonstrated that have a better preservation of the shape of time series against SAX, especially the ups and downs behavior of time series. The application of ABBA has been shown effective regarding time series prediction and anomaly detection; e.g., the LSTM with ABBA shows robust performance over inference [13], the TARZAN replacing SAX with ABBA or fABBA compares favorably with SAX-based TARZAN [6, 12]. However, computing an ABBA symbolic representation for multiple time series is strenuous due to a vast number of features to be extracted, especially dealing with symbolic consistency.

3 PRELIMINARY OF ABBA

Here we briefly recap the preliminaries of ABBA method. ABBA is a symbolic time series representation based on an adaptive polygonal chain approximation, followed by the mean-based clustering algorithm. ABBA symbolization mainly contains two steps, namely *compression* and *digitization*, to aggregate time series $T = [t_1, t_2, \ldots, t_n] \in \mathbb{R}^n$ into a symbolic approximation

$$A = [a_1, a_2, \ldots, a_N], \tag{1}$$

where $N \ll n$ and $a_i \in \mathcal{L}$.

Table 1 shows the procedure of *symbolization* (the first three steps) and *inverse-symbolization* (the last three steps) [2]. ABBA method essentially comprise six steps as summarized in Table 1. The difference between T and $\widehat{T}$ is referred to as reconstruction error. Obviously, a bad symbolization often leads to a high reconstruction error. We will mainly review the phase of compression and digitization below.

3.1 Compression

The ABBA compression step aims to compute an adaptive piecewise linear continuous approximation of T, that is, to obtain time series pieces $P = [(\text{len}_1, \text{inc}_1), \ldots, (\text{len}_N, \text{inc}_N)] \in \mathbb{R}^{N \times 2}$, followed

[2]For the naming convenience, we define $\widehat{\text{inc}} = \widetilde{\text{inc}}$, same follows the [12].

Table 1: Summarized notation of ABBA procedure

time series	$T = [t_0, t_1, \ldots, t_n] \in \mathbb{R}^n$
after compression	$P = [(\text{len}_1, \text{inc}_1), \ldots, (\text{len}_N, \text{inc}_N)] \in \mathbb{R}^{2 \times N}$
after digitization	$A = [a_1, \ldots, a_N] \in \mathcal{L}^N$
inverse-digitization	$\widetilde{P} = [(\widetilde{\text{len}}_1, \widetilde{\text{inc}}_1), \ldots, (\widetilde{\text{len}}_N, \widetilde{\text{inc}}_N)] \in \mathbb{R}^{2 \times N}$
quantization	$\widehat{P} = [(\widehat{\text{len}}_1, \widehat{\text{inc}}_1), \ldots, (\widehat{\text{len}}_N, \widehat{\text{inc}}_N)] \in \mathbb{R}^{2 \times N}$
inverse-compression	$\widehat{T} = [\widehat{t_1}, \widehat{t_2}, \ldots, \widehat{t_n}] \in \mathbb{R}^n$

by a reasonable digitization that results in *symbolic sequence* $A = [a_1, a_2, \ldots, a_N] \in \mathcal{L}^N$, $N \ll n$, and each a_j is an element of a finite alphabet set $\mathcal{L}$ where $|\mathcal{L}| \ll N$. $\mathcal{L}$ can be referred to as dictionary in the procedure. The ABBA compression adaptively selects $N + 1$ indices $i_0 = 0 < i_1 < \cdots < i_N = n$ given a tolerance tol so that the time series T is well approximated by a polygonal chain going through the points (i_j, t_{i_j}) for $j = 0, 1, \ldots, N$. This results in a partition of T into N pieces $p_j = (\text{len}_j, \text{inc}_j)$ that is determined by $T_{i_{j-1}:i_j} = [t_{i_{j-1}}, t_{i_{j-1}+1}, \ldots, t_{i_j}]$, each of integer length $\text{len}_j := i_j - i_{j-1} \geq 1$ in the time direction. Visually, each piece p_j is represented by a straight line connecting the endpoint values $t_{i_{j-1}}$ and t_{i_j} This partitioning criterion is the squared Euclidean distance of the values in p_j from the straight polygonal line is upper bounded by $(\text{len}_j - 1) \cdot \text{tol}^2$. For simplicity, given an index i_{j-1} and starts with $i_0 = 0$, the procedure seeks the largest possible i_j such that $i_{j-1} < i_j \leq n$ and

$$\sum_{i=i_{j-1}}^{i_j} \left(t_{i_{j-1}} + (t_{i_j} - t_{i_{j-1}}) \cdot \frac{i - i_{j-1}}{i_j - i_{j-1}} - t_i \right)^2 \leq (i_j - i_{j-1} - 1) \cdot \text{tol}^2. \tag{2}$$

Each linear piece p_j of the resulting polygonal chain $\widetilde{T}$ is referred to as a tuple $(\text{len}_j, \text{inc}_j)$, where $\text{inc}_j = t_{i_j} - t_{i_{j-1}}$ is the increment in value, i.e., the subtraction of ending and starting value of $T_{i_{j-1}:i_j}$. The whole polygonal chain can be recovered exactly from the first value t_0 and the tuple sequence $p_1, p_2, \ldots, p_N$, i.e.,

$$(\text{len}_1, \text{inc}_1), \ldots, (\text{len}_N, \text{inc}_N) \in \mathbb{R}^2. \tag{3}$$

where the reconstruction error of this representation is with pinned start and end points, and can be naturally modeled as a Brownian bridge.

3.2 Digitization

The next step is referred to as digitization, which we further transformed the resulting polygonal chain $\widetilde{T}$ into the symbolic representation in the form of (1).

Following [12], prior to digitizing, the tuple lengths and increments are separately normalized by their standard deviations σ_{len} and σ_{inc}, respectively. After that, further scaling is employed by using a parameter scl to assign different weights to the length of each piece p_i, which denotes importance assigned to its length value in relation to its increment value. Hence, the clustering is effectively performed on the *scaled tuples*

$$p_1 = \left(\text{scl} \frac{\text{len}_1}{\sigma_{\text{len}}}, \frac{\text{inc}_1}{\sigma_{\text{inc}}} \right), \ldots, p_n = \left(\text{scl} \frac{\text{len}_n}{\sigma_{\text{len}}}, \frac{\text{inc}_n}{\sigma_{\text{inc}}} \right). \tag{4}$$

In particular, if scl = 0, then clustering will be only performed on the increment values of P, while if scl = 1, the lengths and increments are clustered with equal importance.

The steps after normalization proceed with a lossy compression technique, e.g., vector quantization (VQ), which is often achieved by mean-based clustering. The concept of vector quantization can be referenced in [9, 16]. Given an input of N vectors $P = [p_1, \ldots, p_N] \in \mathbb{R}^{\ell \times N}$, VQ seeks a codebook of k vectors, i.e., $C = [c_1, \ldots, c_k] \in \mathbb{R}^{\ell \times k}$ such that k is much smaller than N where each c_i is associated with a unique cluster S_i. A quality codebook enables the sum of squared errors SSE to be small enough to an optimal level. Suppose k clusters $S_1, S_2, \ldots, S_k \subseteq P$ are computed, VQ aims to minimize

$$\text{SSE} = \sum_{i=1}^{k} \phi(c_i, S_i) = \sum_{i=1}^{k} \sum_{p \in S_i} \text{dist}(p, c_i)^2, \quad (5)$$

where ϕ denotes energy function, c_i denotes the center of cluster S_i and $\text{dist}(p_i, p_j)$ often denotes the Euclidean norm $\|p_i - p_j\|_2$. We often choose the mean center μ_i as c_i for Euclidean space, i.e., $\mu_i = \frac{1}{|S_i|} \sum_{p \in S_i} p$, and then (5) can be written as $\text{SSE} = \sum_{i=1}^{k} |S_i| \text{Var} S_i$. Lyold's algorithm [26] (also known as k-means algorithm) is a suboptimal solution of vector quantization to minimize SSE.

The ABBA digitization can be performed by a suitable partitional clustering algorithm that finds k clusters from $P \in \mathbb{R}^{2 \times N}$ such that the sum of Euclidean distance SSE constructed by C is minimized. The obtained codebook vectors are referred to *symbolic centers* here. Each symbolic center is associated with an identical symbol and each time series snippet p_i is assigned with the closest symbolic center c^i associated with its symbol

$$c^i = \arg\min_{c \in C} (\|p - c\|). \quad (6)$$

The symbolic centers to symbols are one-to-one mapping, denoted by $I_d : C \to A$, thus the digitization $f_d : P \to A$ is given by

$$f_d(p_i) = I_d(c^i) = I_d(\arg\min_{c \in C} (\|p - c\|)). \quad (7)$$

Each symbol is associated with a unique cluster. In practice, each clustering label (membership) corresponds to a unique byte-size integer value. The symbols used in ABBA can be represented by text characters, which are not limited to English alphabet letters—often more clusters will be used. Each character in most computer systems is used by the ASCII strings with a unique byte-size integer value (a unique cluster membership). Besides, it can be any combination of symbols, or ASCII representation.

Besides, it is fun to discuss compression rates in some cases. The digitization is the key to compression rate, which is the size of codebook C (i.e., the number of distinct symbols $|\mathcal{L}|$) divided by the length of time series. We use τ_c to denote the compression rate, which is given by

$$\tau_c \in (0, 1] := 1 - \frac{|\mathcal{L}|}{n}. \quad (8)$$

3.3 Inverse symbolization

The inverse symbolization refers to the process from A to $\widehat{T}$, the intuition is to reconstruct time series from (1) such that the reconstructed time series $\widehat{T}$ is as close to T as possible. The inverse symbolization contains three steps.

The first step is referred to as *inverse-digitization*, simply written as f_d^{-1}, which uses the k representative elements c_i (in terms of, e.g., mean centers or median center of the groups S) from codebook C to replace the symbol in A orderly, and thus results in a 2-by-N array $\widetilde{P}$, i.e., an approximation of P, where each $\widetilde{p}_i \in \widetilde{P}$ is the closest symbolic center $c \in C$ to $p_i \in P$. The inverse digitization often leads to a non-integer value to the reconstructed length len, so [12] proposes a novel rounding method, which is referred to as *quantization*, to align the cumulated lengths with the closest integers. The method is as follows: start with rounding the first length into an integer value, i.e., $\widehat{\text{len}}_1 := \text{round}(\widetilde{\text{len}}_1)$ and calculate the rounding error $e := \widetilde{\text{len}}_1 - \widehat{\text{len}}_1$. The the error is added to the rounding to $\widetilde{\text{len}}_2$, i.e., $\widehat{\text{len}}_2 := \text{round}(\widetilde{\text{len}}_2 + e)$ and new error e' is calculated as $\widetilde{\text{len}}_2 + e - \widehat{\text{len}}_2$. Then e' is involved in the next rounding similarly. After all rounding is computed, we obtain

$$\widehat{P} = [(\widehat{\text{len}}_1, \widehat{\text{inc}}_1), \ldots, (\widehat{\text{len}}_N, \widehat{\text{inc}}_N)] \in \mathbb{R}^{2 \times N}, \quad (9)$$

where increments inc are unchanged, i.e., $\widehat{\text{inc}} = \widetilde{\text{inc}}$. Then, the whole polygonal chain can be recovered exactly from the initial time value t_0 and the tuple sequence (9) via the inverse-compression.

The lower reconstruction error means a higher approximation accuracy. The reconstruction error can be defined by mean squared error (MSE), which is given by

$$\text{MSE} = \frac{1}{i} \sum_{i}^{n} (t_i - \widehat{t}_n)^2. \quad (10)$$

4 CLUSTERING-BASED DIGITIZATION

In this section we discuss two commonly used clustering approaches—VQ and GA—for ABBA digitization, on which the two ABBA variants, namely ABBA and fABBA, essentially rely. The pseudocode for VQ and GA is as illustrated in Algorithm 2 and Algorithm 3. As aforementioned, the symbolic centers are represented by the centers of clusters, which is key to the inverse symbolization. The concept of starting points sp_i (the outset of each group forming, which we will elaborate later) is introduced in GA [6], but the mean centers μ_i are preferred in inverse digitization rather than starting points sp_i to seek an accurate inverse symbolization in fABBA. Let μ be the mean center of set S, denoted $\frac{1}{|S|} \sum_{p \in S} p$, we can easily obtain the relationship of the energy function based on starting point sp and mean center μ:

LEMMA 4.1. *Given arbitrary data point p (can be starting point) in group S, the mean center of S is denoted by μ, we have:*

$$\phi(p, S) = \phi(\mu, S) + |S| \text{dist}(p, \mu). \quad (11)$$

In terms of (11) μ is thus the unique value that minimizes the energy function $\phi(p, S)$ [2].

The default setting to ABBA digitization is to use k-means clustering. fABBA [6] uses GA to replace VQ, which achieves significant speed while resulting in a minor loss of approximation accuracy. Both ABBA and fABBA are dominated by a hyper-parameter for digitization, we refer to k for ABBA while α for fABBA. The k determines how many distinct symbols (i.e., clusters) were used for symbolic representation, and the α acts as a tolerance for greedy data aggregation that determines the number of distinct symbols. As discussed in [6], not all clustering (see e.g., BIRCH [38], CLIQUE [1],

spectral clustering [35], DBSCAN [14] and HDBSCAN [5]) is suitable for the partitioning, particularly the density clustering methods, which often result in insufficient symbols that required to fully reflect time series patterns since density clustering methods suffer from chaining effect, and also they are less likely to result in satisfying SSE, thus leads to high reconstruction error.

The visual difference between the two clustering methods is as shown in Figure 1. We can see that VQ (all achieved by k-means++ throughout the paper) assigns groups to form a Voronoi diagram while the GA partitions data of 10,000 points into groups that exist overlap. The partitions with overlap clusters (symbols) inherently model the natural semantic information of words in the real world, e.g., landlady and queen all refer to a woman. Therefore, we believe our joint symbolic representation has promising applications in time series with natural language processing techniques.

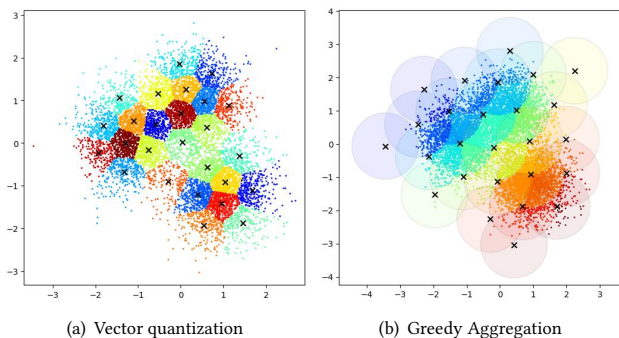|	
(a) Vector quantization	(b) Greedy Aggregation

Figure 1: 2-dimensional data partition using vector quantization achieved by k-means clustering and aggregation with 26 groups: The aggregation uses 0.025 seconds to finish the task while k-means uses 0.18 seconds. The dark points refer to starting point and centers in the two figures, respectively.

Both VQ and GA can perform clustering-based image segmentation tasks, where segmentation is completed by clustering the image's pixels (each pixel represented as a 5-dimensional vector consisting of spatial coordinates and RGB color). Figure 2 shows the result of image segmentation of two images from the COCO dataset [24] by VQ and GA using the same number of clusters, respectively. GA performs clustering in image segmentation significantly faster than VQ, and we can also observe that GA performs well-separated segmentation which is closer to human perception compared to that of VQ which approaches a Voronoi-style segmentation.

4.1 Vector quantization

The k-means problems aim to find k clusters within data in d-dimensional space, so as to minimize the (5). However, solving this problem is NP-hard even k is restricted to 2 [8, 11] or in the plane [28]. Typically, the sub-optimal k-means problem can be solved by Lloyd's algorithm [27]. In the implementation of ABBA software[3], the k-means algorithm is performed by scikit-learn library [31] which runs a few times (this is controlled by the parameter n_init[4])

[3]https://github.com/nla-group/ABBA
[4]The default to scikit-learn is 10 before the version 1.3.2.

Figure 2: Image segmentation with VQ and GA: The three images are achieved by 2,332 and 678 clusters.

of Lylod's algorithm with D^2 seeding and pick up the best result. In the setting of this paper, we found setting n_init to 1 is good enough for the ABBA performance.

As already mentioned, Lloyd's algorithm is a widely used method to solve the k-means problem, it starts with uniformly sampling k centers from data, often referred to as seeding, and then each point is allocated to a cluster with the closest center, and the mean centers of clusters are recomputed again. The procedure keeps repeating until the iteration converges.

The seeding has a great impact on the final result. The improved algorithm is combined with optimal seeding "D^2 weighting" introduced by [2], which can significantly improve Lloyd's algorithm. Lloyd's algorithm with D^2 weighting is called the "k-means++" algorithm. The k-means algorithm with "D^2 weighting" shows $O(\log k)$-competitive with the optimal clustering.

Algorithm 1 D^2 weighting

   Input: $P = [p_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, k
1: Initialize the first center c_1 uniformly arbitrarily from P
2: Select the next center $c_i \in P(i \geq 2)$ with probability $\frac{D(p')^2}{\sum_{p \in P} D(p)^2}$, $p' \in P$
3: Repeat Step 2, until a total of k centers is chosen.
4: **Return:** $c_1, c_2, \ldots, c_k$

Algorithm 2 k-means++ algorithm

Input: $P = [p_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, k
1: Use Algorithm 1 to select k initial centers $c_1, \ldots, c_k$ from P'
2: **for** $c_i \in \{c_1, \ldots, c_k\}$ **do**
3:     Compute C_i, the set of points in P where c_i is the closest center
4:     Update c_i by computing the mean center of cluster C_i, i.e., $c_i = \frac{1}{|C_i|} \sum_{p \in C_i} p$
5: **end for**
6: Repeat Steps $2 \sim 5$ until meet maximum iterations or clusters set converge
7: Assign p_i to the closest center c^i with a unique cluster label
8: **Return** Assigned points $p_1, p_2, \ldots, p_N$

ABBA digitization using this clustering method has been shown incredibly slow speed, though the reconstruction error meets the needs of most applications. It is very desired to design a faster clustering alternative while retaining the original reconstruction error to an ultimate degree. For this reason, we propose a sampling-based k-means clustering algorithm that can address the above concern. The idea is to perform k-means++ on a uniform sample of data where only r percent of original data is used. The algorithm is as described in Algorithm 3. Section 6 will demonstrate its performance empirically.

Algorithm 3 Sampling-based k-means algorithm

Input: $P = [p_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, k, r
1: Uniformly sample $\lfloor r \cdot |P| \rfloor$ points from P as P'
2: Use Algorithm 1 to select k initial centers $c_1, \ldots, c_k$ from P'
3: **for** $c_i \in \{c_1, \ldots, c_k\}$ **do**
4:     Compute C_i, the set of points in P' where c_i is the closest center
5:     Update c_i by computing the mean center of cluster C_i, i.e., $c_i = \frac{1}{|C_i|} \sum_{p \in C_i} p$
6: **end for**
7: Repeat Steps $2 \sim 5$ until iterations end or clusters set converge
8: Assign p_i to the closest center c^i with a unique cluster label
9: **Return** Assigned points $p_1, p_2, \ldots, p_N$

Figure 3 shows the simulation of k-means++ and sampling-based k-means (sampling $r = 10\%$ of data) on Gaussian blobs data with 10 clusters, proceeding by increasing data sizes. The result is as illustrated in Figure 3. We can observe that sampling-based k-means runs in a fraction of the time compared to k-means++, while giving competitive performance in terms of adjusted mutual information (AMI).

4.2 Greedy aggregation

The greedy aggregation is introduced in [6], which proceeds first by sorting the data and performing greedy aggregation of data into groups. The sorting order naturally avoids unnecessary computations in aggregation by triggering an early stopping. The codebook set is constructed by the mean centers of the groups resulting from the aggregation as a suboptimal solution to the k-means problem.
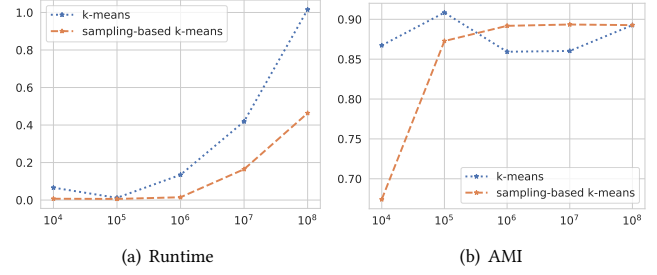

(a) Runtime   (b) AMI

Figure 3: Performance comparison of k-means++ and sampling-based k-means.

Though its accuracy is less significant than Lylod's algorithm, it achieves a significant speedup and the SSE is upper bounded by $\alpha^2(N - k)$ for N data points.

Algorithm 4 Greedy aggregation

Input: $P = [p_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, α
1: Sort data points P such that $\varrho_1 \leq \varrho_2 \leq \cdots \leq \varrho_N$.
2: Label all of sorted data points as "unassigned"
3: Select the first unassigned point x_i as initial starting point, and set $j = i + 1$
4: Check whether or not early stopping can be triggered by sorting property, if not, compute $\text{dist}(p_i, p_j)$
5: **if** $d_{ij} \leq \alpha$ and x_j is unassigned **then**
6:     Assign p_j to the same group as p_i
7: **end if**
8: Increase j by 1, repeat Steps 4~8 until $j > N$
9: Repeat Steps 3~8 until there are no unassigned points left
10: **Return** Assigned points $p_1, p_2, \ldots, p_N$

Sorting is essential to the success of aggregation in our context. Since sorting can determine the starting points selection and forming of groups, even helps to discard unnecessary distance computations. A bad sorting will result in inefficiency of aggregation and bad-performed SSE. For example, [7] proposes PCA sorting which ensures the pairwise distance between p_i and p_j is bounded by $|\varrho_i - \varrho_j| + 2\sigma_2^2$ where σ_2 is the second largest singular value of data matrix P.

4.3 Parameter elimination

As aforementioned, digitization aims to partition P described in (3) into k clusters $S_1, \ldots, S_k$ such that (5) is minimized. The tolerance-oriented digitization enables the natural relationship between compression tol and digitization α. In this section, we discuss a novel way to eliminate the need of choosing a parameter for fABBA digitization. The Lemma 4.2 shows the reconstruction error still ensure the pin of start and end of time series.

LEMMA 4.2 ([12]). *Mean-based clustering naturally leads to* $\sum_{i=1}^N \widehat{\text{inc}}_i = \sum_i^N \text{inc}_i$

Proof to Lemma 4.2 is as follows:

$$\sum_{i=1}^{N} \widehat{\text{inc}}_i = \sum_{i=1}^{N} \widetilde{\text{inc}}_i = \sum_{i=1}^{k} \sum_{j=1}^{|S_i|} \mu_i^{\text{inc}} = \sum_{i=1}^{k} \sum_{j}^{|S_i|} \frac{1}{|S_i|} \sum_{\text{inc}_l \in S_i} \text{inc}_l$$

$$= \sum_{i}^{k} \frac{1}{|S_i|} \sum_{j}^{|S_i|} \sum_{\text{inc}_l \in S_i} \text{inc}_l = \sum_{i}^{k} \frac{1}{|S_i|} |S_i| \sum_{\text{inc}_l \in S_i} \text{inc}_l$$

$$= \sum_{i}^{k} \sum_{\text{inc}_l \in S_i} \text{inc}_l = \sum_{i}^{N} \text{inc}_i$$

Also, we know that $t_{i_N} = t_0 + \sum_{i}^{N} \text{inc}_i$, hence, the reconstruction $\widetilde{T}$ starts and ends at the same values as T so is $\widehat{T}$.

We assume variance of length and increment of pieces, denoted by Var_{len} and Var_{inc}, are:

$$\text{Var}_{\text{len}} = \max_{i=1,\dots,k} \frac{1}{|S_i|} \sum_{\text{len} \in S_i} |\text{len} - \mu_i^{\text{len}}|^2,$$

$$\text{Var}_{\text{inc}} = \max_{i=1,\dots,k} \frac{1}{|S_i|} \sum_{\text{inc} \in S_i} |\text{inc} - \mu_i^{\text{inc}}|^2. \tag{12}$$

Here we suppose the aggregation is performed on the length and increment values (1-dimensional data) of pieces simultaneously, which is referred to as *hierarchical aggregation*, and we denote the digitization tolerance for length and increment α_{len} and α_{inc}, respectively. Obviously, we have

$$\max(\text{Var}_{\text{len}}, \text{Var}_{\text{inc}}) \leq \max(\alpha_{\text{len}}, \alpha_{\text{inc}})^2 \tag{13}$$

We assume $\alpha_{\text{len}} = \alpha_{\text{inc}} = \alpha$, this yields

$$\max(\text{Var}_{\text{len}}, \text{Var}_{\text{inc}}) \leq \alpha^2 \tag{14}$$

In the following, we will demonstrate that the Brownian bridge property as illustrated in [12] still holds in hierarchical aggregation for time series reconstruction. Though the length of each piece may not be consistent because of rounding error, we assume the length of the reconstructed time series is equal to the original length, i.e., $\sum_{i=0}^{N} \widehat{\text{len}}_i = \sum_{i=0}^{N} \text{len}_i$ (In practice, the assumption is true in most cases, but in some special cases, this does not hold true because of rounding). To simplify the modeling and facilitate the analysis, we consider only aggregating increment and assume each cluster of increment has the same mean length, i,e, $\mu_i^{\text{len}} = n/N$. The local deviation of the increment and length value of $\widehat{T}$ on piece P_ℓ from the true increment and length of T, which are given by

$$\dot{e}_{\ell,\text{len}} := \widehat{\text{len}}_\ell - \text{len}_\ell,$$

$$\dot{e}_{\ell,\text{inc}} := \widehat{\text{inc}}_\ell - \text{inc}_\ell, \tag{15}$$

respectively.

The global incremental errors, i.e., the accumulated incremental errors, according to (15), $\ddot{e}_{i_j,\text{inc}}$ are given by:

$$\ddot{e}_{i_j,\text{inc}} := \widehat{t}_{i_j} - t_{i_j} = \sum_{\ell=1}^{j} \dot{e}_{\ell,\text{inc}}, \quad j = 0,\dots,N \tag{16}$$

Also, we must consider the error arisen from the rounding error of length. Similarly, the global length errors, i.e., the accumulated

length errors, according to (12) and (13), can be calculated as:

$$\ddot{e}_{i_j,\text{len}} := \sum_{\ell=1}^{j} \dot{e}_{\ell,\text{len}} \leq \sum_{\ell=1}^{j} \left(\max_{i=1,\dots,k} \frac{1}{|S_i|} \sum_{\text{len} \in S_i} |\text{len} - \mu_i^{\text{len}}| \right)$$

$$\leq \sum_{\ell=1}^{j} \sqrt{ \max_{i=1,\dots,k} \frac{1}{|S_i|} \sum_{\text{len} \in S_i} |\text{len} - \mu_i^{\text{len}}|^2 }$$

$$\leq j \cdot \alpha, \quad j = 0,\dots,N \tag{17}$$

The global error of the reconstructed time series, denoted by e_{i_j}, is caused by errors from reconstructed length and increment. Up to this point, the global error of the reconstructed time series is still difficult to determine since the estimated error caused by the length displacement is hard to get, so we consider an approximation:

$$e_{i_j} \approx \ddot{e}_{i_j,\text{len}} \cdot \ddot{e}_{i_j,\text{inc}} \tag{18}$$

According to (14) and Lemma 4.2, the $\dot{e}_{\ell,\text{inc}}$ is bounded by α, and $\mathbb{E}(\ddot{e}_{i_j,\text{inc}}) = 0$ since they are consistent with the deviations from their respective cluster center. Also, $\ddot{e}_{i_0,\text{inc}} = \ddot{e}_{i_n,\text{inc}} = 0$ as proved earlier. Therefore, referred to [12], we can model a random process of incremental errors e_{i_j}, and its associated variance:

$$\text{Var}(\ddot{e}_{i_j,\text{inc}}) = \alpha^2 \cdot \frac{j(N-j)}{N}, \quad j = 0,\dots,N$$

Following [12], e_{i_j} is considered to stay η standard deviations away from its zeros mean. That is, we consider a realization

$$\ddot{e}_{i_j,\text{inc}} = \eta \cdot \alpha \cdot \sqrt{\frac{j(N-j)}{N}}, \quad j = 0,\dots,N. \tag{19}$$

Then, with (17) and (19), we have,

$$e_{i_j} \leq \eta \cdot \alpha^2 \cdot \sqrt{\frac{j^3(N-j)}{N}}, \quad j = 0,\dots,N.$$

The process of e_{i_j} is modeled as a Brownian bridge following [12]. Considering the interpolated quadratic function on the right-hand side is concave, based on the linear stitching procedure used in the reconstruction and by piecewise linear interpolation of the incremental errors from the course time grid $i_0, i_1, \dots, i_N$ to the fine time grid $i = 0, 1, \dots, n$, it is natural to deduce that

$$e_i \leq \sqrt{\frac{N}{n}} e_{i_j} = \frac{\eta}{n} \cdot \alpha^2 \cdot \sqrt{N \cdot i^3 \cdot (n-i)}, \quad i = 0,\dots,N. \tag{20}$$

Therefore, the squared Euclidean norm of this fine-grid "worst-case" realization is upper bounded by

$$\sum_{i=0}^{n} e_i^2 \leq \frac{N \cdot \eta^2 \cdot \alpha^4}{n^2} \cdot \sum_{i=0}^{n} i^3(n-i)$$

$$= \frac{N \cdot \eta^2 \cdot \alpha^4}{n^2} \cdot \left(\frac{1}{30}n - \frac{1}{12}n^3 + \frac{1}{20}N^5 \right)$$

$$= \frac{N \cdot \eta^2 \cdot \alpha^4}{n^2} \cdot \left(\frac{3n^5 + 2n - 5n^3}{60} \right)$$

$$= \frac{N(3n^4 + 2 - 5n^2) \cdot \eta^2 \cdot \alpha^4}{60n}.$$

It has previously been established that $\text{euclid}(T, \widetilde{T})^2 \leq (n-N) \cdot \text{tol}^2$ by [12], and based on this (4.3) is the worst-case realization of the Brownian bridge and thereby we have a probabilistic bound on

the error incurred from digitization. By making $\text{euclid}(T, \widetilde{T})^2 = \sum_{i=0}^{n} e_i^2$, we can smartly choose

$$\alpha = \sqrt[4]{\frac{60n \cdot (n-N) \cdot \text{tol}^2}{N \cdot \eta^2 \cdot (3n^4 + 2 - 5n^2)}}. \tag{21}$$

For simplicity we can set this hyperparameter controlling the tolerance of length to be the same as that of increment, i.e., $\alpha_{\text{len}} = \alpha_{\text{inc}} = \alpha$. Therefore, the parameter of digitization is automatically determined by the compression tolerance, resulting in a non-parametric and error-bounded digitization procedure.

The procedure detailed above is referred to as *auto digitization*. On top of that, the method introduced in the Section 5 of [12] can also be used to approximate an error-bounded fABBA digitization and eliminate the need for tuning α, however, this is not practical as it results in a linear relationship between tol and α—the difference is as shown in Figure 4 which shows the method in [12] depicts a straight line (marked as green color). As a consequence, we can see our method (marked as orange color) as an improvement for choosing α to some degree.
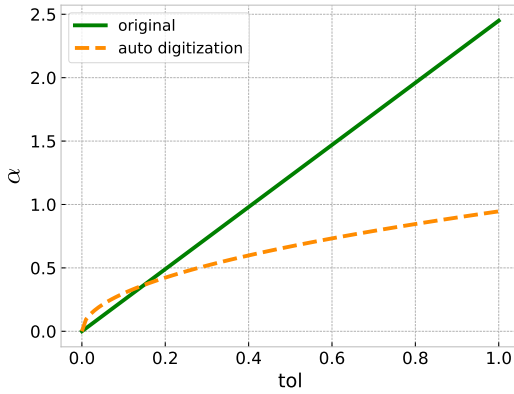

Figure 4: Value of α as tol increases.

5 JOINT SYMBOLIC APPROXIMATION

After discussing the two ABBA methods, we introduce a joint symbolic aggregate approximation on how to perform fast ABBA symbolization on multiple time series while retaining the symbolic consistency. This joint approximation framework is also applicable to large-scale univariate time series and multivariate time series.

The ideal case of symbolization of multiple time series is that the symbolization should have consistent symbols used in each time series and as less distinct symbols used as possible. One intuitive idea is to fit one (or a given number of) time series and use the previous symbolic information to transform the rest of the data. However it does not consider the variety of characteristics in every single time series, this might result in serious information loss in some time series. Henceforth, we require an approach, i.e., joint symbolic approximation, that can symbolize the multiple time series simultaneously.

The essential idea of joint symbolic approximation is partitional compression. Let $\mathcal{T}$ be a dataset of m time series (If $m = 1$, simply

partition the time series into multiple series). In contrast to the original compression, it proceeds by first computing the compression for each series. Then all outputs will be concatenated as an input to digitization which results in a single symbolic sequence. But for multiple time series, an additional step is required, i.e., divide the final symbolic sequence such that each partition corresponds to the symbolic representation of the original time series. The algorithm is as described in Algorithm 5. Since no dependencies occur between compression tasks, this allows for efficient parallel computing. The joint symbolic approximation as well as the parallel computing paradigm is as depicted in Figure 5 and the integral algorithm description is as illustrated in Algorithm 6. For the inverse symbolization, each time series can be reconstructed exactly from its first value and reconstructed pieces from inverse digitization.

The joint symbolic aggregate approximation essentially performs the same steps as the original ABBA method, the major difference is that the compression in ABBA is replaced with partitional compression. Since that, we refer to the method of joint symbolic aggregate approximation as JABBA for simplicity. The framework of joint symbolic aggregate approximation spawns two variants: (1) JABBA (VQ): performs partitional compression and digitization with k-means clustering; (2) JABBA (GA): performs partitional compression and digitization with greedy aggregation. Their performance will be evaluated in Section 6.

Algorithm 5 Partitional compression

1: **Input:** Time series $\mathcal{T}$, tol, m (optional)
2: **if** $\mathcal{T}$ is multivariate **then**
3:     $\mathcal{T}' = \mathcal{T}$
4:     $m \leftarrow$ Compute the number of dimensions of $\mathcal{T}$
5: **else**
6:     $\mathcal{T}' = \{T_i, \ldots, T_m\} \leftarrow$ Partition time series $\mathcal{T}$ into m segments evenly
7: **end if**
8: **for** $i = 1 : m$ **do**   ▷ can do in parallel
9:     $P_i \leftarrow$ Compress time series $T_i \in \mathcal{T}$ according to (2)
10: **end for**
11: $P \leftarrow$ Concatenate $\{P_i\}$
12: **Return:** P, m

As aforementioned, the approach can be applied to datasets storing multiple time series such as UCR time series archive [10]. With the availability of consistent symbols information, techniques of text mining and natural language processing are becoming promising in time series analysis.

Algorithm 6 Joint symbolic aggregate approximation

1: **Input:** $\mathcal{T}$, m, tol
2: $P, m \leftarrow$ Perform Algorithm 5 on $\mathcal{T}$ ▷ Compute partitional compression
3: Compute ABBA digitization on P
4: Partition the symbolic sequence to m subsequences and assign them to the corresponding dimensions, respectively.
5: **Return:** Symbolic representation

---

6 EMPIRICAL RESULTS

In this section, we focus on the experiments regarding runtime and reconstruction errors of symbolic representation. We conduct
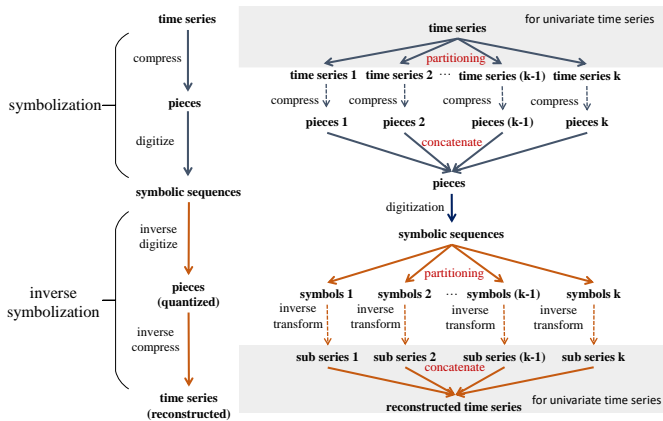
Figure 5: Fork and join model: since there is no dependency among compression tasks, the parallelism is easy to be executed with fork and join model.

extensive experiments on the UEA Archive [3], which is a well-established dataset, and synthetic Gaussian noises for the multi-threading test. We select the competing algorithms that provide publicly available software[5], which is for simplicity and efficiency.

6.1 Multivariate time series test

The UEA Archive contains 30 multivariate time series datasets with a variety of dimensions and lengths. The datasets are very huge, therefore it is inefficient for the original ABBA and its variant fABBA to perform computations one at a time, so we select some datasets in the UEA Archive for the test, which are summarized in Table 2. Though impossible for ABBA and fABBA to symbolize time series in each dimension of multivariate time series with unified symbols, we still use them for benchmarking but without considering the symbolic consistency. In order to unify their compressed time series pieces P as specified in (3), we use partitional compression for all four competing methods and reassign the subset of output corresponding to each multivariate time series dimension to ABBA and fABBA. We start with performing the partitional compression as described in Algorithm 5 with `tol` of 0.01, then for the two JABBA variants we perform their digitization all at once while for ABBA and fABBA we just perform their digitization on time series pieces for each dimension of the multivariate time series one at a time, and then record the runtime for their digitization, respectively.

It's known that the more symbols are used the more accurate the reconstruction of the representation is. In order to use roughly the same number of symbols for each method as much as possible, we first perform JABBA (GA) digitization using (21) to confirm an α value, and a total number of symbols used for the multivariate time series, denoted by k_m, then we feed the number of symbols k_m for JABBA (VQ) digitization (see Algorithm 3, we set r to 0.5, same in the following). Then we feed the same α value to fABBA

digitization and use k_m/d for ABBA where d is the dimension of the multivariate time series. As a consequence, the number of symbols used will be unified for ABBA, JABBA (VQ), and JABBA (GA), but not guaranteed for fABBA since its digitization is tolerance-oriented.

Table 3 showcases the average value of MSE, dynamic time warping (DTW), runtime for digitization, and the number of symbols used for each dataset. Information of compression tolerance `tol` used for each dataset is also given in Table 3. Accordingly, JABBA (GA) shows significant speedup over fABBA by order of magnitude though both use the same GA-based digitization. The speedup of JABBA (VQ) over ABBA is also remarkable while the reconstruction error of JABBA (VQ) is lower than ABBA in five out of the eight datasets though using sampling-based k-means is employed. Additionally, an example of reconstruction from symbolic representation for multivariate time series is presented in Figure 6, which only shows 4 out of 61 dimensions.

Table 2: Selected multivariate time series datasets in UEA Archive.

Dataset	Size	Dimension	Length
AtrialFibrillation	30	2	640
BasicMotions	80	6	100
CharacterTrajectories	2,858	3	182
Epilepsy	275	3	206
Heartbeat	409	61	405
NATOPS	360	24	51
StandWalkJump	27	4	2,500
UWaveGestureLibrary	440	3	315

Figure 6: Reconstruction of symbolic representation for Heartbeat.

---

[5]Available at https://github.com/nla-group/ABBA and https://github.com/nla-group/fABBA.

Table 3: Result in selected UEA multivariate time series datasets (all values are preserved to 2 significant digits, and the best results for MSE, DTW, and runtime are marked as boldface font).

Dataset	Metric	ABBA	fABBA	JABBA (VQ)	JABBA (GA)
AtrialFibrillation (tol = 0.01)	MSE	**6.7**	69	9.7	63
	DTW	**880**	52,000	1,600	18,000
	Runtime	160	15	23	**2.6**
	Symbols	21	550	21	21
BasicMotions (tol = 0.01)	MSE	22	17	**14**	33
	DTW	710	920	**690**	1,800
	Runtime	100	14	13	**2.7**
	Symbols	17	460	18	18
CharacterTrajectories (tol = 0.01)	MSE	**3.5**	**3.5**	6.9	17
	DTW	540	**350**	650	1,600
	Runtime	24	3.3	4.8	**1.7**
	Symbols	3.1	47	3.5	3.5
Epilepsy (tol = 0.01)	MSE	20	50	**15**	86
	DTW	1,700	20,000	**1,400**	13,000
	Runtime	83	12	31	**2.2**
	Symbols	14	480	14	14
Heartbeat (tol = 0.0001)	MSE	5.2	**0.0017**	1.8	0.017
	DTW	1,400	**0.69**	430	6.8
	Runtime	17,000	1,500	3,500	**110**
	Symbols	2,000	23,000	2,000	2,000
NATOPS (tol = 0.01)	MSE	38	18	**9.1**	28
	DTW	1,700	270	**150**	430
	Runtime	110	28	100	**2.5**
	Symbols	24	450	23	23
StandWalkJump (tol = 0.005)	MSE	**2.2**	8.7	3.7	5.7
	DTW	**550**	5,100	730	1,200
	Runtime	900	60	55	**11**
	Symbols	190	1,900	190	190
UWaveGestureLibrary (tol = 0.01)	MSE	3.1	**2**	3	8
	DTW	350	**180**	340	1,100
	Runtime	37	3.8	5.7	**1.9**
	Symbols	5.4	52	5.4	5.4

6.2 Multithreading simulation

In this experiment, we will compare ABBA, fABBA, JABBA (GA), and JABBA (VQ) on synthetic Gaussian noise series in terms of runtime, and reconstruction accuracy with various number of time series partitions. The reconstruction accuracy is measured by MSE here.

We used Gaussian noises as the time series for benchmarking. The data generated for the test are of length 100,000 with zero mean and unit standard deviation. We first ran fABBA with tol = 0.01 and α = 0.05 to compute the number of symbols it used. This simulation used 358 symbols accordingly. Second, we ran ABBA by feeding the same number of symbols fABBA used to k, i.e., 358 symbols. After that, we run the JABBA (VQ) and JABBA (GA) with varying partitions by the same tol and specifying a consistent hyperparameter setting for digitization, i.e., k = 358 and α = 0.05, respectively. The number of threads scheduled for JABBA is set the same as the partition number. The result shows the compression rate τ_c computed for ABBA, fABBA, JABBA (GA), and JABBA (VQ) are , respectively.

The experimental result is as exhibited in Figure 7 and Figure 8. We mainly compare the methods with the same digitization technique. We can see that there is an obvious negative correlation between reconstruction error and the number of partitions, this can be explained by the increasing partition points that will be used for reconstruction. Figure 7 also shows that JABBA (VQ) which uses sampling-based k-means achieves similar performance against ABBA regarding MSE while performing speedup by orders of magnitude, a similar result applies to JABBA (GA) and fABBA.
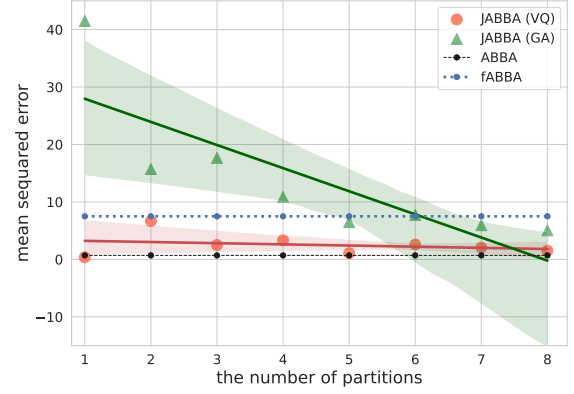

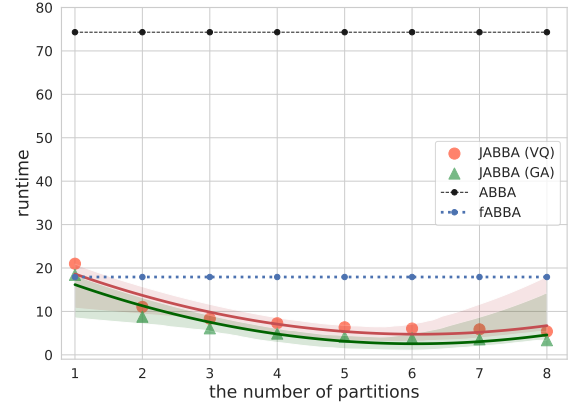**Figure 7: MSE of JABBA with varying number of partitions (the black line marks the result of fABBA).**

Figure 8: Runtime of JABBA with varying number of partitions (the black line marks the result of fABBA).

The parallel speedup m processors is given by

$$\Phi(m) = \frac{v(1)}{v(m)}$$

where $v(m)$ is referred to as the runtime of the m processors. Without loss of generality, we only evaluate the speedup of Parallelism for JABBA (GA) as shown in Figure 9. We can see the speedup $\Phi(m)$ scale almost linearly with the number of threads m. Since our algorithm is partially parallel in compression, which is hindered by the sequential part of the algorithm, that is, the digitization. This phenomenon can be naturally explained by Amdahl's law which gives the theoretical speedup at a fixed workload where there are limits on the benefits one can derive from parallelizing a computation.

7 SUMMARY AND FUTURE WORK

The existing ABBA methods are incapable of handling the consistency of symbols for multiple time series and are inherently sequential, and it is not clear how to leverage the extra computational power such as multithreading processing. In this paper, we
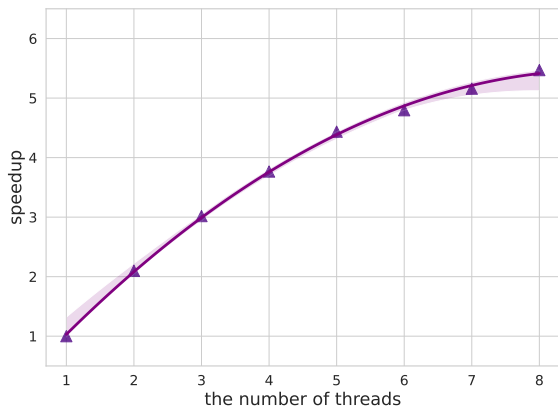
Figure 9: Speedup.

introduce a joint symbolic approximation method that improves the speed of ABBA symbolization and achieves symbolic consistency in each representation. The framework of joint symbolic approximation enables parallel computing for further speedup. Attributed to the symbolic consistency, a manipulation of natural language processing and text mining techniques is available in time series. The convergence analysis of our proposed sampling k-means method will be left as future work.

REFERENCES

[1] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. 1998. Automatic subspace clustering of high dimensional data for data mining applications. *Proceedings of the ACM SIGMOD International Conference on Management of Data* 27 (1998), 94–105.

[2] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms.* Society for Industrial and Applied Mathematics, 1027–1035.

[3] Anthony J. Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn J. Keogh. 2018. The UEA multivariate time series classification archive. *CoRR* (2018).

[4] Konstantinos Bountrogiannis, George Tzagkarakis, and Panagiotis Tsakalides. 2023. Distribution Agnostic Symbolic Representations for Time Series Dimensionality Reduction and Online Anomaly Detection. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2023), 5752–5766.

[5] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. 2013. Density-Based Clustering Based on Hierarchical Density Estimates. In *Advances in Knowledge Discovery and Data Mining.* Springer, 160–172.

[6] Xinye Chen and Stefan Güttel. 2022. An Efficient Aggregation Method for the Symbolic Representation of Temporal Data. *ACM Transactions on Knowledge Discovery from Data* (2022).

[7] Xinye Chen and Stefan Güttel. 2022. Fast and explainable clustering based on sorting. (2022), 25. arXiv:2202.01456

[8] Sanjoy Dasgupta and Yoav Freund. 2008. Random Projection Trees and Low Dimensional Manifolds. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing (STOC '08).* ACM, 537–546.

[9] Sanjoy Dasgupta and Yoav Freund. 2009. Random Projection Trees for Vector Quantization. *IEEE Transactions on Information Theory* 55, 7 (2009), 3229–3242.

[10] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica* 6, 6 (2019), 1293–1305.

[11] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. 2004. Clustering large graphs via the singular value decomposition. *Machine Learning* 56, 1–3 (2004), 9–33.

[12] Steven Elsworth and Stefan Güttel. 2020. ABBA: adaptive Brownian bridge-based symbolic aggregation of time series. *Data Mining and Knowledge Discovery* 34 (2020), 1175–1200.

[13] Steven Elsworth and Stefan Güttel. 2020. Time series forecasting using LSTM networks: A symbolic approach. (2020), 12. arXiv:2003.05672

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96).* AAAI Press, 226–231.

[15] Yifeng Gao and Jessica Lin. 2019. Discovering Subdimensional Motifs of Different Lengths in Large-Scale Multivariate Time Series. In *IEEE International Conference on Data Mining.* 220–229.

[16] Robert Gray. 1984. Vector quantization. *IEEE ASSP Magazine* 1, 2 (1984), 4–29.

[17] E. Keogh, J. Lin, and A. Fu. 2005. HOT SAX: efficiently finding the most unusual time series subsequence. In *IEEE International Conference on Data Mining (ICDM'05).* 1–8.

[18] Eamonn Keogh, Stefano Lonardi, and Bill 'Yuan-chi' Chiu. 2002. Finding Surprising Patterns in a Time Series Database in Linear Time and Space. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '02).* ACM, 550–556.

[19] Xiaosheng Li and Jessica Lin. 2017. Linear Time Complexity Time Series Classification with Bag-of-Pattern-Features. In *IEEE International Conference on Data Mining.* 277–286.

[20] Xiaosheng Li, Jessica Lin, and Liang Zhao. 2021. Time Series Clustering in Linear Time Complexity. *Data Mining and Knowledge Discovery* 35, 6 (2021), 2369–2388.

[21] Yuan Li and Jessica Lin. 2010. Approximate Variable-Length Time Series Motif Discovery Using Grammar Inference. In *Proceedings of the 10th International Workshop on Multimedia Data Mining (MDMKDD '10).* ACM, 9.

[22] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery.* ACM, 2–11.

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

[24] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: Common Objects in Context. *European Conference on Computer Vision* (2014), 740–755.

[25] B. Lkhagva, Yu Suzuki, and K. Kawagoe. 2006. New Time Series Data Representation ESAX for Financial Applications. In *22nd International Conference on Data Engineering Workshops (ICDEW'06).* x115–x115. https://doi.org/10.1109/ICDEW.2006.99

[26] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.

[27] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *Transactions on Information Theory* 28 (1982), 129–137.

[28] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. 2012. The planar k-means problem is NP-hard. *Theoretical Computer Science* 442 (2012), 13–21. Special Issue on the Workshop on Algorithms and Computation (WALCOM 2009).

[29] Simon Malinowski, Thomas Guyet, René Quiniou, and Romain Tavenard. 2013. 1d-SAX: A Novel Symbolic Representation for Time Series. In *Advances in Intelligent Data Analysis XII.*

[30] Thach Le Nguyen and Georgiana Ifrim. 2023. Fast Time Series Classification with Random Symbolic Subsequences. In *Advanced Analytics and Learning on Temporal Data: 7th ECML PKDD Workshop, AALTD 2022, Grenoble, France, September 19–23, 2022, Revised Selected Papers.* Springer, 50—-65.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[32] N. Ruta, N. Sawada, K. McKeough, M. Behrisch, and J. Beyer. 2019. SAX Navigator: Time Series Exploration through Hierarchical Clustering. In *2019 IEEE Visualization Conference.* IEEE, 236–240.

[33] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. 2015. Time series anomaly discovery with grammar-based compression.. In *18th International Conference on Extending Database Technology.* OpenProceedings.org, 481–492.

[34] Pavel Senin and Sergey Malinchik. 2013. SAX-VSM: Interpretable Time Series Classification Using SAX and Vector Space Model. In *IEEE International Conference on Data Mining.* 1175–1180.

[35] Stella X. Yu and Jianbo Shi. 2003. Multiclass spectral clustering. In *Proceedings of the Ninth IEEE International Conference on Computer Vision*, Vol. 2. IEEE, 313.

[36] Yuncong Yu, Tim Becker, Le Minh Trinh, and Michael Behrisch. 2023. SAXRegEx: Multivariate time series pattern search with symbolic representation, regular expression, and query expansion. *Computers & Graphics* 112 (2023), 13–21.

[37] Shengdong Zhang, Soheil Bahrampour, Naveen Ramakrishnan, Lukas Schott, and Mohak Shah. 2017. Deep learning on symbolic representations for large-scale heterogeneous time-series event prediction. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing.* 5970–5974.

[38] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. 1996. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data.* ACM, 103–114.