ABBA: *Adaptive Brownian bridge-based symbolic aggregation of time series*

Elsworth, Steven and Güttel, Stefan

2019

MIMS EPrint: **2019.11**

Manchester Institute for Mathematical Sciences

School of Mathematics

The University of Manchester

ABBA: Adaptive Brownian bridge-based symbolic aggregation of time series

Steven Elsworth · **Stefan Güttel**

Abstract A new symbolic representation of time series, called ABBA, is introduced. It is based on an adaptive polygonal chain approximation of the time series into a sequence of tuples, followed by a mean-based clustering to obtain the symbolic representation. We show that the reconstruction error of this representation can be modelled as a random walk with pinned start and end points, a so-called Brownian bridge. This insight allows us to make ABBA essentially parameter-free, except for the approximation tolerance which must be chosen. Extensive comparisons with the SAX and 1d-SAX representations are included in the form of performance profiles, showing that ABBA is able to better preserve the essential shape information of time series compared to other approaches. Advantages and applications of ABBA are discussed, including its in-built differencing property and use for anomaly detection, and Python implementations provided.

Keywords time series · symbolic aggregation · dimension reduction · Brownian bridge

1 Introduction

Symbolic representations of time series are an active area of research, being useful for many data mining tasks including dimension reduction, motif and rule discovery, prediction, and clustering of time series. Symbolic time series representations allow for the use of algorithms from text processing and bioinformatics, which often take advantage of the discrete nature of the data. Our focus in this work is to develop a symbolic representation which is dimension reducing whilst preserving the essential *shape* of the time series. Our definition of shape is different from the one commonly implied in the context of time series: we focus on representing the peaks and troughs of the time series in their correct order of appearance, but we are happy to slightly stretch the time series in *both* the time and value directions. In other words, our focus is not necessarily on approximating the time series values at the correct time points, but on representing the local up-and-down behavior of the time series and identifying repeated motifs. This is obviously not appropriate in all applications, but we believe it is close to how humans summarize the overall behavior of a time series, and in that our representation might be useful for trend prediction, anomaly detection, and motif discovery.

To illustrate, let us consider the time series shown in Figure 1. This series is sampled at equidistant time points with values $t_0, t_1, \ldots, t_N \in \mathbb{R}$, where $N = 230$. There are various ways of describing this time series, for example:

(a) It is exactly representable as a high-dimensional vector $T = [t_0, t_1, \ldots, t_N] \in \mathbb{R}^{N+1}$.
(b) It starts at a value of about -3, then climbs up to a value of about 0 within 25 time steps, then it stays at about 0 for 100 time steps, after which it goes up to a value of about 3 within 25 time steps, and so on.
(c) It starts at a value of about -3, then goes up rapidly by about 3 units, followed by a longer period with almost no change in value, after which it *again* goes up rapidly by about 3 units, and so on.

Note how in (a) and (b) the emphasis is on the actual values of the time series, whereas in (c) we mainly refer to trends in the time series in relation to previously observed trends. High-level information might be difficult to

Steven Elsworth
Department of Mathematics, The University of Manchester, M13 9PL, Manchester, UK
E-mail: steven.elsworth@manchester.ac.uk

Stefan Güttel
Department of Mathematics, The University of Manchester, M13 9PL, Manchester, UK
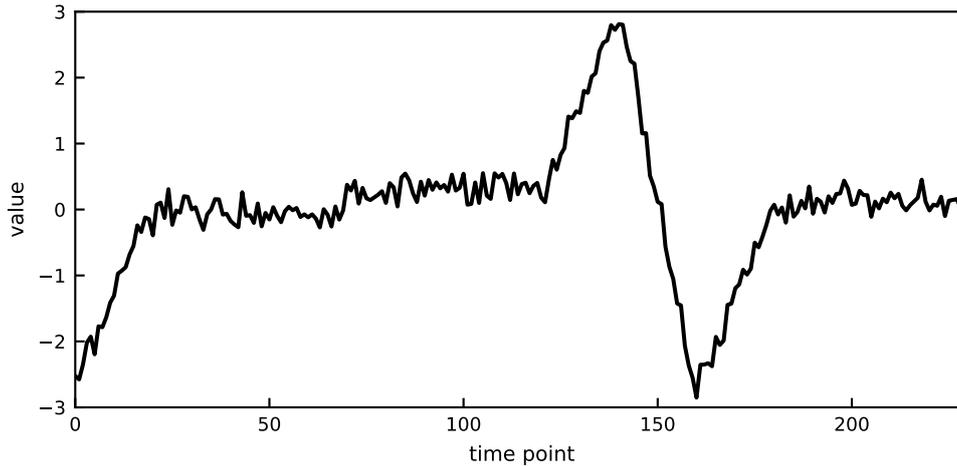E-mail: stefan.guettel@manchester.ac.uk

Fig. 1: Illustrative example time series T used throughout the paper.

extract from (a) directly, while (b) could be seen as putting too much emphasis on the time series values instead of the overall shape. The symbolic representation developed in this paper, called *adaptive Brownian bridge-based aggregation* (ABBA), adaptively reduces T to a shorter sequence of symbols with an emphasis on the shape information. The resulting description will be conceptually similar to (c) from the examples above.

To formalize the discussion and introduce notation, we consider the problem of aggregating a time series $T = [t_0, t_1, \ldots, t_N] \in \mathbb{R}^{N+1}$ into a *symbolic representation* $S = [s_1, s_2, \ldots, s_n] \in \mathbb{A}^n$, where each s_j is an element of an alphabet $\mathbb{A} = \{a_1, a_2, \ldots, a_k\}$ of k symbols. The sequence S should be of considerably lower dimension than the original time series T, that is $n \ll N$, and it should only use a small number of meaningful symbols, that is $k \ll n$. The representation should also allow for the approximate reconstruction of the original time series with a controllable error, with the shape of the reconstruction suitably close to that of the original. Both n, the length of the symbolic representation, and k, the number of symbols, should be chosen automatically without parameter tuning required.

This paper is organized as follows. In Section 2 we give an overview of existing symbolic representations and other algorithms which are conceptually similar to ABBA. To evaluate the approximation accuracy of ABBA, we must compare the shape of the original time series and the reconstruction from its symbolic representation. Section 3 reviews existing distance measures for this purpose and discusses how well they perform in measuring shape. Sections 4–7 contain the key contributions of this paper:

- Section 4 introduces ABBA, our novel dimension-reducing symbolic time series representation which aims to preserve the shape of the original time series. We explain in detail how ABBA's compression and reconstruction procedures work.
- In Section 5 we show that the error of the ABBA reconstruction behaves like a random walk with pinned start and end values. This observation appears to be novel in itself and allows us to balance the error of the piecewise linear approximation with that of the digitization procedure, thereby allowing the method to choose the number of symbols k automatically.
- Section 6 contains performance comparisons of ABBA with other popular symbolic representations using various distance measures, with a particular emphasis on the compression versus accuracy relation. Aside from verifying that ABBA can represent time series to higher accuracy than SAX and 1d-SAX using a comparable number of symbols k and string length n, we also find that SAX outperforms 1d-SAX when the same number of symbols k is used for both.
- In Section 7 we discuss some practical applications of ABBA including the handling of linear trends, anomaly detection, and VizTree visualization.

Finally, we conclude in Section 8 with an outlook on future work.

2 Background and related work

Despite the large number of dimension-reducing time series representations in the literature, very few are *symbolic*. Most techniques are *numeric* in the sense that they reduce a time series to a lower-dimensional vector with its

components taken from a continuous range; see [9, 17, 32] for reviews. Here we provide an overview of existing symbolic representations relevant to ABBA.

The construction of symbolic time series representations typically consists of two parts. First, the time series is segmented, with the length of each segment being either specified by the user or found adaptively via a bottom-up, top-down, or sliding window approach [22]. The segmentation procedure intrinsically controls the degree of dimension reduction. The second part, the discretization process, assigns a symbol to each segment.

Symbolic Aggregate approXimation (SAX), a very popular symbolic representation, consists of a piecewise approximation of the time series followed by a symbolic conversion using Gaussian breakpoints [32]. SAX starts by partitioning T into segments of constant length `len`, and then represents each segment by the mean of its values (i.e., a piecewise constant approximation). The means are converted into symbols using breakpoints that partition a Gaussian bell curve into k equally-sized areas. In addition to its simplicity, an attractive feature of SAX is the existence of distance measures that serve as lower bounds for the Euclidean distance between the original time series. On the other hand, both the segment length `len` and the number of symbols k must be specified in advance. SAX is designed such that each symbol appears with equal probability, which works best when the time series values are approximately normally distributed.

The literature on applications of SAX is extensive and many variants have been proposed. Most variants modify the symbolic representation to incorporate the slope of the time series on each segment. This is often justified by applications in finance, where the extreme values of time series provide valuable information which is lost with the piecewise constant approximation used in SAX. The modifications often come at the cost of losing the lower bounds on distance measures. We now provide a brief overview of some of these variants.

Trend-based and Valued-based Approximation (TVA) uses SAX to symbolically represent the time series values, enhanced with U, D, or S symbols to represent an upwards, downwards, or straight trend, respectively [16]. The TVA representation alternates between value symbols and slope symbols, making the symbolic representation twice as long as a SAX representation with the same number of segments. A similar approach is *Trend-based SAX* (TSAX) which uses two trend symbols per segment [43].

Extended SAX (ESAX) represents each segment by the minimum, maximum, and mean value of the time series ordered according to their appearance in the segment, defining the mean to appear in the center of the segment [33]. This results in a symbolic representation three times longer than the corresponding SAX representation with the same number of segments. *ENhanced SAX* (EN-SAX) forms a vector for each segment consisting of the minimum, maximum and mean value. The vectors are then clustered and a symbol is allocated to each cluster [5]. *Time-Weighted Average for SAX* (TWA_SAX) uses the time weighted average for each segment instead of the mean [7]. This can encapsulate important patterns which are missed by the mean.

Trend-based Symbolic approximation (TSX) represents each segment by four symbols [28]. The first symbol corresponds to the SAX representation. The following three symbols correspond to the slopes between the first, last, most peak and most dip points, which are defined in terms of vertical distance from the trend line (the straight line connecting the end point values of a segment). The slopes are converted to symbols using a lookup table. This results in a symbolic representation four times longer than the SAX representation with the same number of segments.

The *1d-SAX algorithm* uses linear regression to fit a straight line to each segment [35]. Each segment is then represented by the gradient and the average value of the line. Two sets of Gaussian breakpoints are used to provide symbols for both the averages and the slopes. It is unclear how many breakpoints should be allocated for the averages, and how many should be allocated for the slopes. The total number of symbols is the product of the respective number of breakpoints.

Using the same number of segments, the above SAX variants result in an increase in the length of the symbolic representation by some factor. It is unclear whether any of these approaches performs better than SAX when the SAX segment length `len` is decreased by the same factor (keeping the overall length of the symbolic representation constant). As with the original SAX approach, all of these variants require the user to specify the segment length `len` and the number of symbols k in advance.

In many time series applications, the assumption that the values of the normalized time series follow a normal distribution is a strong one. To overcome this, the *adaptive SAX algorithm* (aSAX) uses k-means clustering to find the breakpoints for the symbolic conversion [40]. However, as piecewise constant approximations are used, the aSAX approach fails to represent the extreme points of the time series.

SAX's digitization procedure based on Gaussian breakpoints allows its extension to a multi-resolution symbolic representation known as indexable SAX (iSAX) [41]. This clever indexing procedure allows mining of datasets containing millions of time series. At the heart of the algorithm is a SAX representation where each window uses Gaussian breakpoints with 2^c regions, where c can change from segment to segment.

The *sensorPCA algorithm* overcomes the fixed window length problem by using a sliding window to start a new segment when the standard deviation of the approximation exceeds some prespecified tolerance [18]. However, [18] does not provide a method to convert the mean values and window lengths to a symbolic representation.

Symbolic Aggregate approXimation Optimized by data (SAXO) is a data-driven approach based on a regularized Bayesian coclustering method called minimum optimized description length [10, 11]. The discretization of the time series is optimized using Bayesian statistics. The number of symbols and the underlying distribution change for each time interval. The computational complexity of SAXO is far greater than that of SAX.

The authors in [37] take a completely different approach based on the persistence of a time series. A persistent time series is one where the value at a certain point is closely related to the previous value; see also [27]. The authors provide "persist", a symbolic representation based on the Kullback–Leibler divergence between the marginal and the self-transition probability distributions of the discretization symbols.

Symbolic Polynomial (SP) [19] is a symbolic representation designed to detect local patterns. It is constructed by an overlapping sliding window of length w and stepsize 1. For each window, one computes the coefficients of a regression polynomial of degree d. The coefficients of each order are collected and allocated a symbol using an equi-area discretization. This symbolic representation provides no dimensional reduction as each window is represented by d symbols.

The authors in [6] introduce a symbolic representation of multivariate time series called SMTS. They construct a data table consisting of time index, time values, and first differences of the time series. A tree learner is trained on the data and each of the leaf nodes is allocated a symbol. Their approach allows multiple tree learners, which in the univariate case results in a symbolic representation much larger than the original.

Piecewise linear approximations of time series have been used for many years. The lengths of the linear pieces (segments) can be prespecified or chosen adaptively. Each segment is approximated using either linear interpolation or linear regression [22]. The authors of [34] describe how the linear segments can be stitched so that each piece is represented by two parameters rather than three. An example of a piecewise linear approximation algorithm is the Ramer–Douglas–Peucker algorithm, an iterative endpoint fitting procedure which uses adaptive linear interpolation with a prespecified tolerance. These methods provide an effective shape-preserving and dimension-reducing representation but not a symbolic representation.

3 Distance measures

The accuracy of a symbolic time series representation S can be assessed by the distance between the original time series T and its reconstruction $\widehat{T}$ from S. We note that the original time series should first be normalized to have zero mean and unit variance. This ensures that distance measures are comparable across different time series; see [23] for a discussion of the importance of normalization.

A detailed overview of time series distance measures and their applications can be found in [2]. Distance measures for time series typically fall into two main categories: *lock-step alignment* and *elastic alignment* [1]. Lock-step alignment refers to the element-wise comparison of time series, i.e., the i-th element of one time series is compared to the i-th element of another. Such measures can only compare time series of equal length. The most popular lock-step distance is the Euclidean distance. The Euclidean distance is a poor measure of shape similarity in two particular cases: if the time series have the same shape but are stretched in value (see Figure 2a), or if the time series have the same shape but are warped in time (see Figure 2b). The first issue can be mitigated by differencing the time series before measuring the distance. The second issue is intrinsic to lock-step alignment distance measures.

Elastic alignment distance measures construct a nonlinear mapping between time series elements, effectively allowing for one value in a time series to be compared to multiple consecutive values in another. The most popular elastic alignment method is Dynamic Time Warping (DTW), originally proposed in [8]. The DTW distance measure corresponds to the Euclidean distance between two DTW-aligned time series. This distance measure can be used to compare time series of different lengths but it has a quadratic computational complexity in both time and space; for further details see [25]. Many methods have been proposed to either approximate the DTW distance at a reduced cost or calculate bounds to avoid computing the DTW alignment altogether. The authors of [26] notice that DTW may pair a rising trend in one time series with a falling trend in another, and they overcome this problem by a variant known as Derivative Dynamic Time Warping (DDTW). The elastic alignment allows DTW to overcome the issues when two time series have the same shape but are warped in time (see Figure 2b), but DTW is still a poor measure of shape similarity if the time series have the same shape but are vertically stretched (see Figure 2a). Again, this can be mitigated by differencing the time series before measuring their DTW distance.

It is because of these advantages and drawbacks of the Euclidean and DTW distance measures and their differenced counterparts that we will test the performance of ABBA with all these distance measures in Section 6.

(a) These time series have essentially the same shape but there is a value shift on the intervals $[20,40]$ and $[60,80]$.

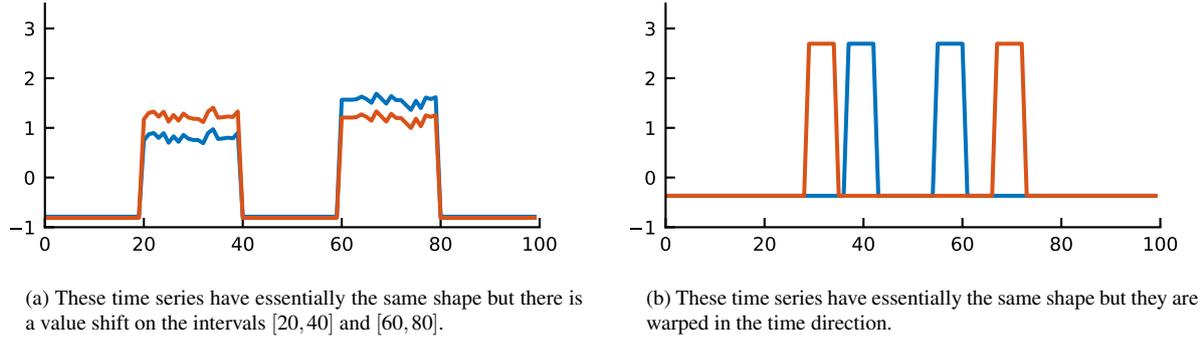(b) These time series have essentially the same shape but they are warped in the time direction.

Fig. 2: The time series in these plots have the same essential shape according to our interpretation. Euclidean distance is a poor measure of shape for (a) and (b), whereas DTW distance is a poor measure of shape for (a). A differencing of the time series in (a) would make DTW a suitable shape distance.

4 Adaptive Brownian bridge-based aggregation

We now introduce ABBA, a symbolic representation of time series where the symbolic length n and the number of symbols k are chosen adaptively. The ABBA representation is computed in two stages.

1. Compression: The original time series T is approximated by a piecewise linear and continuous function, with each linear piece being chosen adaptively based on a user-specified tolerance. The result is a sequence of tuples (`len`, `inc`) consisting of the length of each piece and its increment in value.
2. Digitization: A near-optimal alphabet $\mathbb{A}$ is identified via mean-based clustering, with each cluster corresponding to a symbol. Each tuple (`len`, `inc`) is assigned a symbol corresponding to the cluster in which it belongs.

The reconstruction of a time series from its ABBA representation involves three stages.

1. Inverse-digitization: Each symbol of the symbolic representation is replaced with the center of the associated cluster. The length values of the centers may not necessarily be integers.
2. Quantization: The lengths of the reconstructed segments are re-aligned with an integer grid.
3. Inverse-compression: The piecewise linear continuous approximation is converted back to a pointwise time series representation using a stitching procedure.

Both the computation of the ABBA representation and the reconstruction are inexpensive. It is essential that the digitization process uses incremental changes in value rather than slopes. This way, ABBA consistently works with increments in both the time and value coordinates. Only in this case a mean-based clustering algorithm will identify meaningful clusters in both coordinate directions. As we will explain in Section 5, the error of the ABBA reconstruction behaves like a random walk pinned at zero for both the start and the end point of the time series. But first, we provide a more detailed explanation of the key parts of ABBA. For clarity, we summarize the notation used throughout this section in Table 1.

Table 1: Summary of notation

Original time series:	$T = [t_0, t_1, \ldots, t_N] \in \mathbb{R}^{N+1}$
After compression:	$[(\mathtt{len}_1, \mathtt{inc}_1), (\mathtt{len}_2, \mathtt{inc}_2), \ldots, (\mathtt{len}_n, \mathtt{inc}_n)] \in \mathbb{R}^{2 \times n}$
After digitization:	$S = [s_1, s_2, \ldots, s_n] \in \mathbb{A}^n$ with $\mathbb{A} = \{a_1, a_2, \ldots, a_k\}$
After inverse-digitization:	$[(\widetilde{\mathtt{len}}_1, \widetilde{\mathtt{inc}}_1), (\widetilde{\mathtt{len}}_2, \widetilde{\mathtt{inc}}_2), \ldots, (\widetilde{\mathtt{len}}_n, \widetilde{\mathtt{inc}}_n)] \in \mathbb{R}^{2 \times n}$
After quantization:	$[(\widehat{\mathtt{len}}_1, \widehat{\mathtt{inc}}_1), (\widehat{\mathtt{len}}_2, \widehat{\mathtt{inc}}_2), \ldots, (\widehat{\mathtt{len}}_n, \widehat{\mathtt{inc}}_n)] \in \mathbb{R}^{2 \times n}$
After inverse-compression:	$\widehat{T} = [\widehat{t}_0, \widehat{t}_1, \ldots, \widehat{t}_N] \in \mathbb{R}^{N+1}$

4.1 Compression

The ABBA compression is achieved by an adaptive piecewise linear continuous approximation of T. Given a tolerance `tol`, the method adaptively selects $n+1$ indices $i_0 = 0 < i_1 < \cdots < i_n = N$ so that the time series

$T = [t_0, t_1, \ldots, t_N]$ is approximated by a polygonal chain going through the points (i_j, t_{i_j}) for $j = 0, 1, \ldots, n$. This gives rise to a partition of T into n pieces $P_j = [t_{i_{j-1}}, t_{i_{j-1}+1}, \ldots, t_{i_j}]$, each of length $\texttt{len}_j := i_j - i_{j-1} \geq 1$ in the time direction. We ensure that the squared Euclidean distance of the values in P_j from the straight polygonal line is bounded by $(\texttt{len}_j - 1) \cdot \texttt{tol}^2$. More precisely, starting with $i_0 = 0$ and given an index i_{j-1}, we find the largest possible i_j such that $i_{j-1} < i_j \leq N$ and

$$\sum_{i=i_{j-1}}^{i_j} \left(\underbrace{t_{i_{j-1}} + (t_{i_j} - t_{i_{j-1}}) \cdot \frac{i - i_{j-1}}{i_j - i_{j-1}}}_{\text{straight line approximation}} - \underbrace{t_i}_{\text{actual value}} \right)^2 \leq (i_j - i_{j-1} - 1) \cdot \texttt{tol}^2. \tag{1}$$

Note that the first and the last values $t_{i_{j-1}}$ and t_{i_j} are not counted in the distance measure as the straight line approximation passes exactly through them. If required, one can restrict the maximum length of each segment by imposing an upper bound $i_j \leq i_{j-1} + \texttt{max\_len}$ with a given integer $\texttt{max\_len} \geq 1$.

Each linear piece P_j of the resulting polygonal chain $\widetilde{T}$ is described by a tuple $(\texttt{len}_j, \texttt{inc}_j)$, where $\texttt{inc}_j = t_{i_j} - t_{i_{j-1}}$ is the increment in value (not the slope!). As the polygonal chain is continuous, the first value of a segment can be inferred from the end value of the previous segment. Hence the whole polygonal chain can be recovered exactly from the first value t_0 and the tuple sequence

$$(\texttt{len}_1, \texttt{inc}_1), (\texttt{len}_2, \texttt{inc}_2), \ldots, (\texttt{len}_n, \texttt{inc}_n) \in \mathbb{R}^2. \tag{2}$$

An example of the ABBA compression procedure applied to the time series in Figure 1 is shown in Figure 3. Here a tolerance of $\texttt{tol} = 0.4$ has been used, resulting in $n = 7$ pieces. As the approximation error on each piece P_j satisfies (1), the polygonal chain $\widetilde{T}$ also has a bounded Euclidean distance from T:

$$\begin{aligned}
\texttt{euclid}(T, \widetilde{T})^2 &\leq [(i_1 - i_0 - 1) + (i_2 - i_1 - 1) + \cdots + (i_n - i_{n-1} - 1)] \cdot \texttt{tol}^2 \\
&= (N - n) \cdot \texttt{tol}^2.
\end{aligned} \tag{3}$$

Hence we are sure that the ABBA approximation $\widetilde{T}$ (red dashed curve) in Figure 3 has a Euclidean distance of at most $\sqrt{223} \times 0.4 \approx 6.0$ from the original time series T (black solid curve).
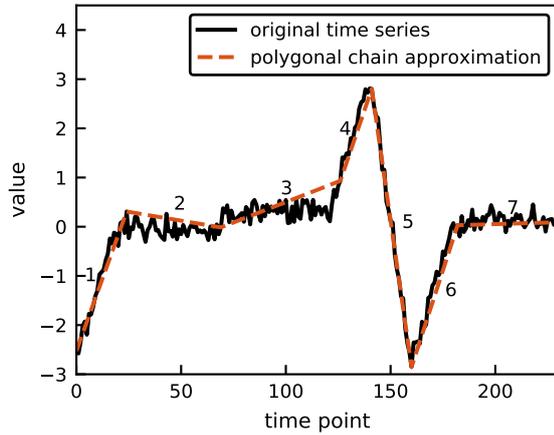

Fig. 3: Result of the ABBA compression. The time series is now represented by $n = 7$ tuples of the form $(\texttt{inc}, \texttt{len})$ and the starting value t_0.
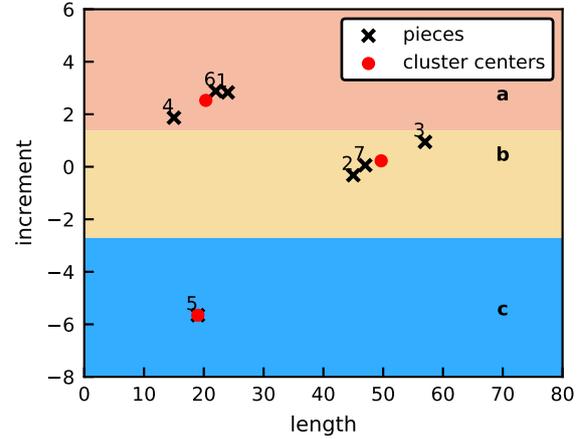
Fig. 4: Result of the ABBA digitization with scaling parameter $\texttt{scl} = 0$. The tuples $(\texttt{len}, \texttt{inc})$ are converted to the symbol sequence abbacab.

4.2 Digitization

Digitization refers to the assignment of the tuples in (2) to k clusters $S_1, S_2, \ldots, S_k$. Before clustering, we separately normalize the tuple lengths and increments by their standard deviations $\sigma_{\texttt{len}}$ and $\sigma_{\texttt{inc}}$, respectively. We use a

further scaling parameter $\texttt{scl}$ to assign different weight ("importance") to the length of each piece in relation to its increment value. Hence, we effectively cluster the *scaled tuples*

$$\left(\texttt{scl}\frac{\text{len}_1}{\sigma_{\text{len}}},\frac{\text{inc}_1}{\sigma_{\text{inc}}}\right),\left(\texttt{scl}\frac{\text{len}_2}{\sigma_{\text{len}}},\frac{\text{inc}_2}{\sigma_{\text{inc}}}\right),\ldots,\left(\texttt{scl}\frac{\text{len}_n}{\sigma_{\text{len}}},\frac{\text{inc}_n}{\sigma_{\text{inc}}}\right)\in\mathbb{R}^2. \tag{4}$$

If $\texttt{scl}=0$, then clustering is performed on the increments alone, while if $\texttt{scl}=1$, we cluster in both the length and increment dimension with equal weighting. The cluster assignment is performed by (approximately) minimizing the within-cluster-sum-of-squares

$$\texttt{WCSS}=\sum_{i=1}^{k}\sum_{(\text{len},\text{inc})\in S_i}\left\|\left(\texttt{scl}\frac{\text{len}}{\sigma_{\text{len}}},\frac{\text{inc}}{\sigma_{\text{inc}}}\right)-\overline{\mu}_i\right\|^2,$$

with each 2d cluster center $\overline{\mu}_i=(\overline{\mu}_i^{\text{len}},\overline{\mu}_i^{\text{inc}})$ corresponding to the mean of the scaled tuples associated with the cluster S_i. In certain situations one may want to cluster only on the lengths of the pieces and ignore their increments, formally setting $\texttt{scl}=\infty$. In this case, the cluster assignment is performed by (approximately) minimizing

$$\texttt{WCSS}=\sum_{i=1}^{k}\sum_{(\text{len},\text{inc})\in S_i}\left|\frac{\text{len}}{\sigma_{\text{len}}}-\overline{\mu}_i^{\text{len}}\right|^2,$$

where $\overline{\mu}_i^{\text{len}}$ is the mean of the scaled lengths in the cluster S_i.

Given a clustering of the n tuples into clusters $S_1,\ldots,S_k$ we use the *unscaled* cluster centers μ_i

$$\mu_i=(\mu_i^{\text{len}},\mu_i^{\text{inc}})=\frac{1}{|S_i|}\sum_{(\text{len},\text{inc})\in S_i}(\text{len},\text{inc})$$

to define the maximal cluster variances in the length and increment directions as

$$\text{Var}_{\text{len}}=\max_{i=1,\ldots,k}\frac{1}{|S_i|}\sum_{(\text{len},\text{inc})\in S_i}\left|\text{len}-\mu_i^{\text{len}}\right|^2,$$

$$\text{Var}_{\text{inc}}=\max_{i=1,\ldots,k}\frac{1}{|S_i|}\sum_{(\text{len},\text{inc})\in S_i}\left|\text{inc}-\mu_i^{\text{inc}}\right|^2,$$

respectively. Here, $|S_i|$ is the number of tuples in cluster S_i. We seek the smallest number of clusters k such that

$$\max(\texttt{scl}\cdot\text{Var}_{\text{len}},\text{Var}_{\text{inc}})\leq\texttt{tol}_s^2 \tag{5}$$

with a tolerance $\texttt{tol}_s$. This tolerance will be specified in Section 5 as a function of the user-specified tolerance $\texttt{tol}$ and is therefore not a free parameter. (In the case of $\texttt{scl}=\infty$, we seek the smallest k such that $\text{Var}_{\text{len}}\leq\texttt{tol}_s^2$.) Once the optimal k has been found, each cluster $S_1,\ldots,S_k$ is assigned a symbol $a_1,\ldots,a_k$, respectively. Finally, each tuple in the sequence (2) is replaced by the symbol of the cluster it belongs to, resulting in the symbolic representation $S=[s_1,s_2,\ldots,s_n]$.

If $\texttt{scl}=0$ or $\texttt{scl}=\infty$, a 1d clustering method can be used which takes advantage of sorting algorithms; see the review [20]. We use the $\texttt{ckmeans}$ algorithm [42], an order $\mathcal{O}(n\log n+kn)$ dynamic programming algorithm which optimally clusters the data by minimizing the WCSS in just one dimension. We have modified the algorithm to choose the smallest k such that the maximal cluster variance is bounded by $\texttt{tol}_s^2$.

For nonzero finite values of $\texttt{scl}$, k-means clustering is used. This algorithm has an average complexity of $\mathcal{O}(kn)$ per iteration (see also [3] for an analysis of the worst case complexity) and might of course result in a suboptimal clustering. In our ABBA implementation the user can specify an interval $[\texttt{min\_k},\ldots,\texttt{max\_k}]$ and we search for the smallest k in that interval such that (5) holds. If no such k exists, we set $k=\texttt{max\_k}$.

By default, we set $\texttt{scl}=0$ as we believe this corresponds most naturally to preserving the up-and-down behavior of the time series. In other words, we ignore the lengths of the pieces and only cluster the value increments. With the value increments represented accurately, the errors in lengths correspond to horizontal stretching in the time direction.

An illustration of the digitization process on the pieces from Figure 3 can be seen in Figure 4 with $\texttt{scl}=0$ (our default parameter choice), Figure 5 with $\texttt{scl}=1$, and Figure 6 with $\texttt{scl}=\infty$.
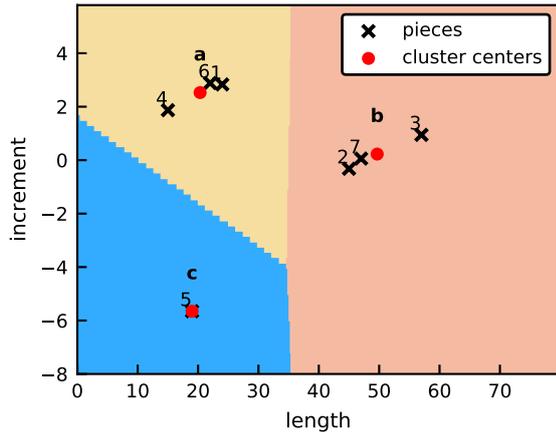
Fig. 5: Result of the ABBA digitization with $\texttt{scl} = 1$. The tuples $(\texttt{len}, \texttt{inc})$ are converted to the symbol sequence $\texttt{abbacab}$.
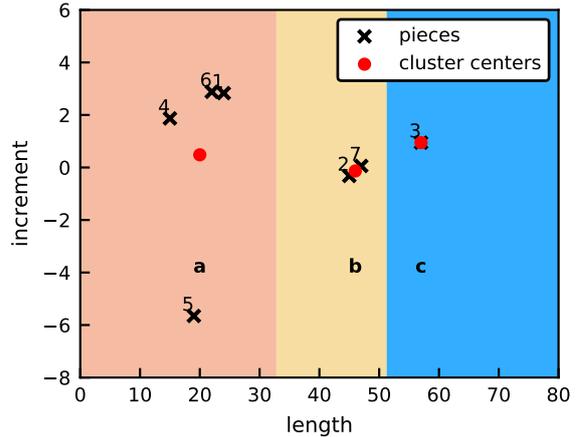
Fig. 6: Result of the ABBA digitization with $\texttt{scl} = \infty$. The tuples $(\texttt{len}, \texttt{inc})$ are converted to the symbol sequence $\texttt{abcaaab}$.

4.3 Inverse digitization and quantization

When reversing the digitization process, each symbol of the alphabet is replaced by the center $(\overline{\texttt{len}}_i, \overline{\texttt{inc}}_i)$ of the corresponding cluster given as

$$(\overline{\texttt{len}}_i, \overline{\texttt{inc}}_i) = \frac{1}{|S_i|} \sum_{(\texttt{len}, \texttt{inc}) \in S_i} (\texttt{len}, \texttt{inc}).$$

Note that the mean-based clustering for digitization is performed on the scaled tuples (4), but the cluster centers used for the inverse digitization are computed with the unscaled tuples (2). The inverse digitization process results in a sequence of n tuples

$$(\widetilde{\texttt{len}}_1, \widetilde{\texttt{inc}}_1), (\widetilde{\texttt{len}}_2, \widetilde{\texttt{inc}}_2), \ldots, (\widetilde{\texttt{len}}_k, \widetilde{\texttt{inc}}_k) \in \mathbb{R}^2,$$

where each tuple is a cluster center, that is $(\widetilde{\texttt{len}}_i, \widetilde{\texttt{inc}}_i) \in \{(\overline{\texttt{len}}_1, \overline{\texttt{inc}}_1), (\overline{\texttt{len}}_2, \overline{\texttt{inc}}_2), \ldots, (\overline{\texttt{len}}_n, \overline{\texttt{inc}}_n)\}$.

The lengths $\widetilde{\texttt{len}}_i$ obtained from this averaging are not necessarily integer values as they were in the compressed representation (2). We therefore perform a simple quantization procedure which realigns the cumulated lengths with their closest integers. We start with rounding the first length, $\widehat{\texttt{len}}_1 := \text{round}(\widetilde{\texttt{len}}_1)$, keeping track of the rounding error $e := \widehat{\texttt{len}}_1 - \widetilde{\texttt{len}}_1$. This error is added to the second length $\widetilde{\texttt{len}}_2 := \widetilde{\texttt{len}}_2 + e$, which is then rounded to $\widehat{\texttt{len}}_2 := \text{round}(\widetilde{\texttt{len}}_2)$ with error $e := \widehat{\texttt{len}}_2 - \widetilde{\texttt{len}}_2$, and so on. As a result we obtain a sequence of n tuples

$$(\widehat{\texttt{len}}_1, \widehat{\texttt{inc}}_1), (\widehat{\texttt{len}}_2, \widehat{\texttt{inc}}_2), \ldots, (\widehat{\texttt{len}}_n, \widehat{\texttt{inc}}_n) \in \mathbb{R}^2 \qquad (6)$$

with integer lengths $\widehat{\texttt{len}}_i$. (The increments remain unchanged but we rename them for consistency: $\widehat{\texttt{inc}}_i := \widetilde{\texttt{inc}}_i$.)

5 Error analysis

During the compression procedure, we construct a polygonal chain $\widetilde{T}$ going through selected points $\{(i_j, t_{i_j})\}_{j=0}^n$ of the original time series T, with a controllable Euclidean distance (3). After the digitization, inverse digitization, and quantization, we obtain a new tuple sequence (6) which can be stitched together to a polygonal chain $\widehat{T}$ going through the points $\{(\widehat{i}_j, \widehat{t}_j)\}_{j=0}^n$, with $(\widehat{i}_0, \widehat{t}_0) = (0, t_0)$. Our aim is to analyze the distance between $\widehat{T}$ and $\widetilde{T}$, and then balance it with the distance between $\widetilde{T}$ and T.

We first note that

$$(\widehat{i}_j, \widehat{t}_{i_j}) = \left(\sum_{\ell=1}^{j} \widehat{\texttt{len}}_\ell, t_0 + \sum_{\ell=1}^{j} \widehat{\texttt{inc}}_\ell \right), \quad j = 0, \ldots, n.$$

As all the lengths $\widehat{\texttt{len}}_\ell$ and increments $\widehat{\texttt{inc}}_\ell$ correspond to cluster centers (averages of all the points in a cluster, consistently rounded during quantization), we have the interesting property that the accumulated deviations from the true lengths and increments exactly cancel out at the right endpoint of the last piece P_n, that is: $(\widehat{i}_n, \widehat{t}_{i_n}) = (i_n, t_{i_n}) = (N, t_N)$. In other words, the polygonal chain $\widehat{T}$ starts and ends at the same values as $\widetilde{T}$ (and hence T).

We now analyze the behavior of $\widehat{T}$ in between the start and endpoints, focusing on the case that $\mathtt{scl} = 0$ and assuming for simplicity that all cluster centers S_i have the same mean length $\mu_i^{\mathtt{len}} = N/n$. (This is not a strong assumption as in the dynamic time warping distance the lengths of the pieces is irrelevant.) We compare $\widehat{T}$ with the polygonal chain $\widetilde{T}$ time-warped to the same regular length grid as $\widehat{T}$, which will give an upper bound on $\mathtt{dtw}(\widehat{T}, \widetilde{T})$. Denoting by $d_\ell := \widehat{\mathtt{inc}}_\ell - \widetilde{\mathtt{inc}}_\ell$ the local deviation of the increment value of $\widehat{T}$ on piece P_ℓ from the true increment of $\widetilde{T}$, we have that

$$\widehat{t}_{i_j} - t_{i_j} = \sum_{\ell=1}^{j} d_\ell =: e_{i_j}, \quad j = 0, \ldots, n.$$

Recall from Section 4.2 that we have controlled the variance of the increment values in each cluster to be bounded by $\mathtt{tol}_s^2$. As a consequence, the increment deviations d_ℓ have bounded variance $\mathtt{tol}_s^2$, and mean zero as they correspond to deviations from their respective cluster center. It is therefore reasonable to model the "global increment errors" e_{i_j} as a random process with fixed values $e_{i_0} = e_{i_n} = 0$, expectation $\mathrm{E}(e_{i_j}) = 0$, and variance

$$\mathrm{Var}(e_{i_j}) = \mathtt{tol}_s^2 \cdot \frac{j(n-j)}{n}, \quad j = 0, \ldots, n.$$

In the case that the d_ℓ are i.i.d. normally distributed, such a process is known as a *Brownian bridge*. See also Figure 7 for an illustration.

Note that so far we have only considered the variance of the global increment errors e_{i_j} at the left and right endpoints of each piece P_j, but we are actually interested in analyzing the error of the reconstruction $\widehat{T}$ on the fine time grid. To this end, we now consider a "worst-case" realization of e_{i_j} which stays s standard deviations away from its zero mean. That is, we consider a realization

$$e_{i_j} = s \cdot \mathtt{tol}_s \cdot \sqrt{\frac{j(n-j)}{n}}, \quad j = 0, \ldots, n.$$

By piecewise linear interpolation of these errors from the coarse time grid $i_0, i_1, \ldots, i_n$ to the fine time grid $i = 0, 1, \ldots, N$ (in accordance with the linear stitching procedure used in ABBA), we find that

$$e_i \leq \sqrt{\frac{n}{N}} \cdot s \cdot \mathtt{tol}_s \cdot \sqrt{\frac{i(N-i)}{N}}, \quad i = 0, \ldots, N,$$

using that the interpolated quadratic function on the right-hand side is concave. We can now bound the squared Euclidean norm of this fine-grid "worst-case" realization as

$$\sum_{i=0}^{N} e_i^2 \leq \frac{n \cdot s^2 \cdot \mathtt{tol}_s^2}{N^2} \cdot \sum_{i=0}^{N} i(N-i) = \frac{n \cdot s^2 \cdot \mathtt{tol}_s^2}{N^2} \cdot \frac{N^3 - N}{6} \leq n \cdot s^2 \cdot \mathtt{tol}_s^2 \cdot \frac{N}{6}.$$

This is a probabilistic bound on squared Euclidean error caused by a "worst-case" realization of the Brownian bridge, and thereby a probabilistic bound on the error incurred from the digitization procedure. Equating this bound with the bound (3) on the accuracy of the compression, we find that we should choose

$$\mathtt{tol}_s = \frac{\mathtt{tol}}{s} \sqrt{\frac{6(N-n)}{Nn}},$$

with the user-specified tolerance $\mathtt{tol}$. We have experimentally determined that $s = 0.2$ typically gives a good balance between the compression accuracy and the number of clusters determined using this criterion.

Example: We now illustrate the above analysis on a challenging real-world example. Consider a time series T ($N = 7127$) consisting of temperature readings off a heat exchanger in an ethylene cracker. We use $\mathtt{tol} = 0.1$ to compress this time series, resulting in a polygonal chain $\widetilde{T}$ with $n = 123$ pieces and an approximation error of $\mathtt{euclid}(T, \widetilde{T}) = 5.3 \leq \sqrt{N-n} \cdot \mathtt{tol} \approx 8.4$. See Figure 8 for a plot of the original time series T and its reconstruction $\widetilde{T}$ after compression.

We then run the ABBA digitization procedure with scaling parameter $\mathtt{scl} = 0$, resulting in a symbolic representation S of length n using $k = 14$ symbols. In Figure 7 we show the "global increment errors" e_{i_j} of the reconstruction $\widehat{T}$ on each piece P_j, that is, the increment deviation of $\widehat{T}$ from T at the endpoints of P_j, $j = 1, \ldots, n$. Note how this error is pinned at zero at $j = 0$ and $j = n$, and how it resembles a random walk in between.

The reconstruction $\widehat{T}$ on the fine time grid is also shown in Figure 8. The reconstruction error measured in the time warping distance is $\mathtt{dtw}(\widetilde{T}, \widehat{T}) = 9.5$ and the overall error is $\mathtt{dtw}(T, \widehat{T}) = 10.8$, both of which are approximately of the same order as $\sqrt{N-n} \cdot \mathtt{tol} \approx 8.4$. Note that the ABBA reconstruction $\widehat{T}$ visually deviates a lot from T due to the rather high tolerance we have chosen for illustration, but nevertheless, the characteristic up-and-down behavior of T is well represented in $\widehat{T}$, despite the high compression rate of $123/7128 \approx 1.7\%$.
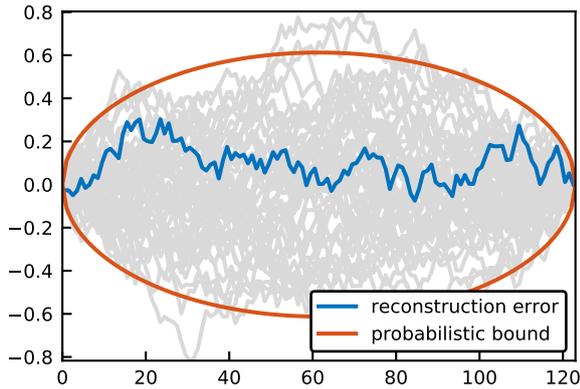
Fig. 7: Example of the ABBA reconstruction error forming a Brownian bridge. The blue line is the actual error, the grey lines are 50 other realizations of the random walk, and the red bounds indicate one standard deviation above and below the zero mean.
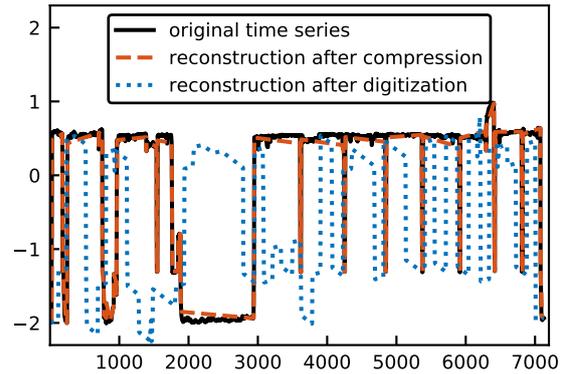
Fig. 8: ABBA representation of a time series from a heat exchanger in an ethylene cracker. With $\mathtt{tol} = 0.1$ and $\mathtt{scl} = 0$, the time series is reduced from 7128 points to 123 tuples using 14 symbols.

6 Discussion and performance comparison

A Python implementation of ABBA, along with codes to reproduce the figures and performance comparisons in this paper, can be found at

$$\mathtt{https://github.com/nla-group/ABBA}$$

When the scaling parameter is $\mathtt{scl} = 0$ or $\mathtt{scl} = \infty$, our implementation calls an adaptation of the univariate k-means algorithm from the R package $\mathtt{Ckmeans.1d.dp}$ written in $\mathtt{C++}$. We use SWIG, the open-source "Simplified Wrapper and Interface Generator", to call $\mathtt{C++}$ functions from Python. If $\mathtt{scl} \in (0, \infty)$, we use the k-means algorithm from the Python $\mathtt{sklearn}$ library [38].

ABBA uses the lengths and increments of a polygonal chain on each segment to construct its symbolic time series representation. Symbolic Polynomial [19] (with $d = 1$) and 1d-SAX [35], on the other hand, use linear regression to fit a polynomial to a window of fixed pre-specified length. As we discussed in Section 2, Symbolic Polynomial provides no dimensional reduction and was specifically designed for time series classification problems. Most other SAX variants increase the length of the symbolic representation by enhancing the string with additional characters to capture shapes and trends. It is not clear whether these representations outperform SAX with a reduced width parameter to compensate for the increased string length. A comparison of this would be interesting but is independent of ABBA's performance and out of the scope of this paper. SMTS [6] and aSAX [40] use machine learning techniques to discretize their representation. SMTS is primarily designed for multivariate time series and provides no dimensional reduction. EN-SAX [5] and aSAX suffer from a loss of the trend information in their compression step.

For these reasons, we focus on profiling the reconstructions errors of the ABBA, SAX [32], and 1d-SAX [35] algorithms, as these are most closely related and easily comparable. Note that none of the representations were primarily designed as compression algorithms. ABBA was designed to be adaptive in both segment length and alphabet cardinality, whereas SAX and 1d-SAX have many other benefits such as being hashable [12], indexable [41], and permitting lower bounding distance measures. Our test set consists of all time series in the UCR Time Series Classification Archive [13] with a length of at least 100 data points. There are $128,978$ such time series from a variety of applications. Although the archive is primarily intended for benchmarking time series classification algorithms, our primary focus in this paper is on the approximation performance of the symbolic representations. Our experiment consists of converting each time series $T = [t_0, t_1, \dots, t_N]$ into its symbolic representation $S = [s_1, \dots, s_n]$, and then measuring the distance between the reconstruction $\widehat{T} = [\widehat{t_0}, \widehat{t_1}, \dots, \widehat{t_N}]$ and T in the (differenced) Euclidean and DTW norms, respectively.

Recall from Section 2 that both SAX and 1d-SAX require a choice for the fixed segment length. In order to provide a fair comparison, we first run the ABBA compression with an initial tolerance $\mathtt{tol} = 0.05$. This returns n, the number of required pieces to approximate T to this tolerance. If n turns out to be larger than $N/5$, we successively increase the tolerance by 0.05 and rerun until a compression rate of at least 20 % is achieved. If a time series cannot be compressed to at least 20 % even at the rather crude tolerance of $\mathtt{tol} = 0.5$, we consider it

as too noisy and exclude it from the test. We also exclude all time series which, after ABBA compression, result in fewer than nine pieces: this is necessary because we want to use $k = 9$ symbols for all compared methods. Table 2 shows how many of the $111,889$ remaining time series were compressed at what tolerance. The table gives evidence that most of these time series can be compressed reasonably well while maintaining a rather high accuracy. The average compression rate is 10.3 %.

Table 2: Tolerance used for the compression and the number of time series to which it was applied

tolerance `tol`	0.05	0.10	0.15	0.20	0.25	0.30	0.35	0.40	0.45	0.50
nr of time series	75417	9247	7786	5855	2972	2236	1910	1670	2146	2650

After the number of pieces n has been specified for a given time series T, we determine the fixed segment length $\texttt{len} = \lfloor (N+1)/n \rfloor$ to be used in the SAX and 1d-SAX algorithms. We then apply SAX and 1d-SAX to the first $n \cdot \texttt{len}$ points of T. This guarantees that all three algorithms (SAX, 1d-SAX, and ABBA) produce a symbolic representation of with n pieces. If $N + 1$ is not divisible by n, SAX and 1d-SAX are applied to slightly shorter time series than ABBA. The number of symbols used for the digitization is $k = 9$ for all three methods. In the case of 1d-SAX this means that three symbols are used for the mean value, and three symbols are used for the slope on each piece. Each algorithm produces a symbolic representation of length n using an alphabet of cardinality $k = 9$. SAX and 1d-SAX requires the value of w and k for the reconstruction, whereas ABBA requires the $2k$ numbers representing the lengths and increments of each cluster. In total, ABBA requires more storage to represent a time series using a string of length n and alphabet of cardinality k, but is able to represent the whole time series more accurately without truncation.

To visualize the results of our comparison we use performance profiles [14]. Performance profiles allow to compare the relative performance of multiple algorithms over a large set of test problems. Each algorithm is represented by a non-decreasing curve in a θ–p graph. The θ-axis represents a tolerance $\theta \geq 1$ and the p-axis corresponds to a fraction $p \in [0,1]$. If a curve passes through a point (θ, p) it means that the corresponding algorithm performed within a factor θ of the best observed performance on $100 \cdot p\,\%$ of the test problems. For $\theta = 1$ one can read off on what fraction of all test problems each algorithm was the best performer, while as $\theta \to \infty$ all curves approach the value $p \to 1$ (unless an algorithm has failed on a fraction of the test problems, which is not the case here).

In Figures 9a–10d we present eight performance profiles for the ABBA scaling parameters $\texttt{scl} = 0$ and $\texttt{scl} = 1$, respectively, and with four different distance measures: Euclidean and DTW distances and their differenced counterparts, respectively. Figure 9a shows the performance profile for $\texttt{scl} = 0$, with the distance between T and $\widehat{T}$ measured in the Euclidean norm. As expected, SAX consistently outperforms ABBA because the Euclidean distance is very sensitive to horizontal shifts in the time direction, which ABBA has completely ignored due to the $\texttt{scl} = 0$ parameter. However, it is somewhat surprising that SAX also outperforms 1d-SAX. It appears that the use of the slope information in 1d-SAX is detrimental to the approximation accuracy and, if the number of symbols is kept constant, they should better be used to represent time series values alone. This observation can also be made in the other performance profiles: irrespective of the distance measure being used, SAX with $k = 9$ symbols performs better than 1d-SAX with $k = 9$ symbols.

The performance changes when we use the DTW distance, thereby allowing for shifts in time. In this case, ABBA outperforms SAX and 1d-SAX significantly; see Figure 9b. This is because ABBA has been tailored to preserve the up-and-down shape of the time series, at the cost of allowing for small errors in the lengths of the pieces which are easily corrected by time warping. The performance gain of ABBA becomes even more pronounced when we difference the data before computing the Euclidean and DTW distances; see Figures 9c and 9d, respectively.

In the next four tests we set $\texttt{scl} = 1$, so the ABBA clustering procedure considers both the increments and lengths equally. Figures 10a and 10b show the resulting performance profiles using the Euclidean and DTW distance measures, respectively. As expected, ABBA becomes more competitive even for the Euclidean distance measure. Computationally, however, this comes at the cost of not being able to use a fast optimal 1d-clustering algorithm. Finally, Figures 10c and 10d show the performance profiles for the Euclidean and DTW distance measures on the differenced data, respectively. As in the case $\texttt{scl} = 0$, differencing helps to improve the performance of ABBA in comparison to SAX and 1d-SAX even further[1].

---

[1] Visual comparisons of the three algorithms on the first time series in each dataset of the UCR Time Series Classification Archive can be found at `https://github.com/nla-group/ABBA/tree/master/paper/performance_profiles/scl0`.
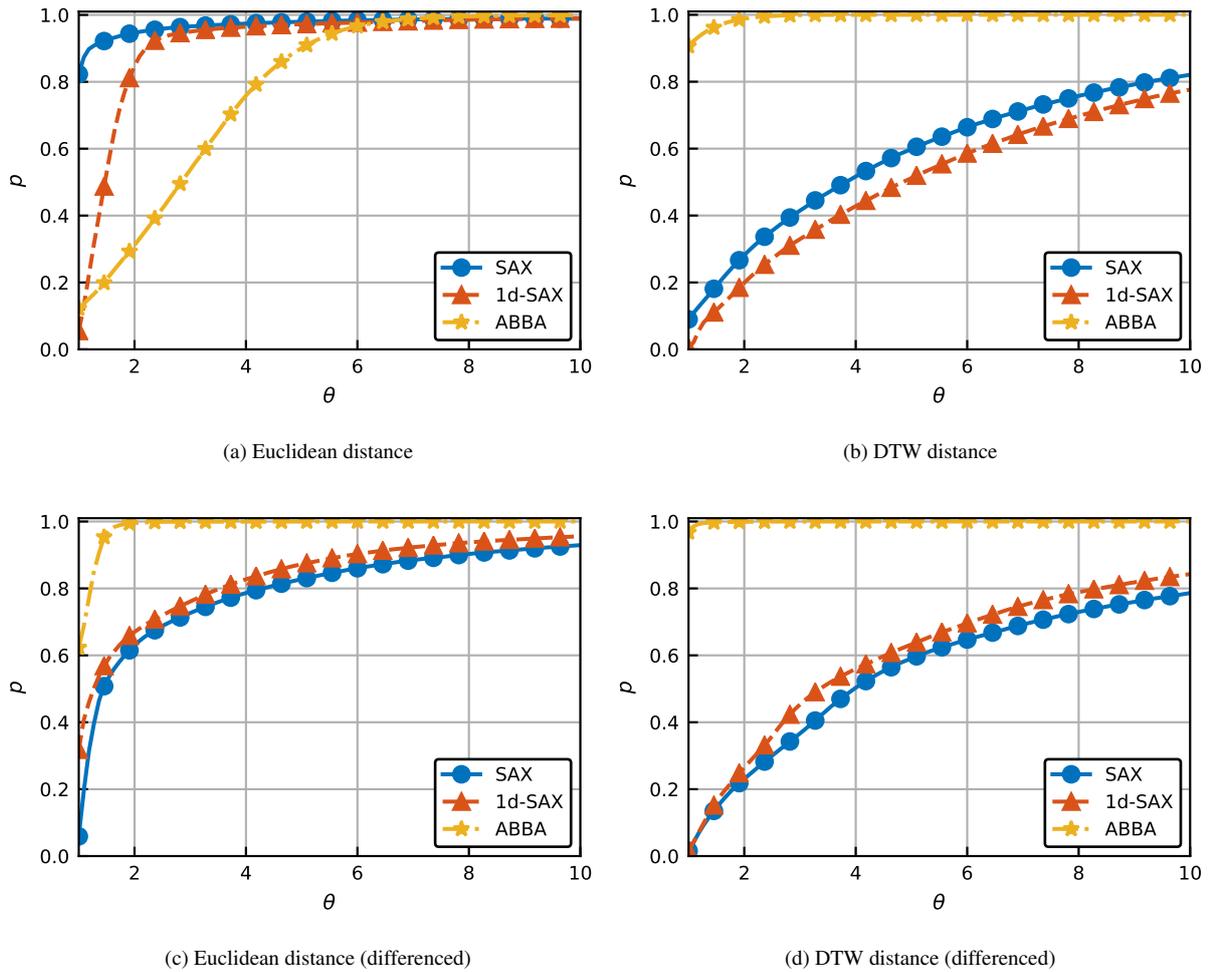
(a) Euclidean distance

(b) DTW distance

(c) Euclidean distance (differenced)

(d) DTW distance (differenced)

Fig. 9: Performance profiles for the reconstruction errors of SAX, 1d-SAX, and ABBA *with scaling parameter* `scl` $= 0$. Figures 9a and 9b compare ABBA (`scl` $= 0$) with SAX and 1d-SAX using Euclidean and Dynamic Time Warping distance, respectively. Figures 9c and 9d compare ABBA (`scl` $= 0$) with SAX and 1d-SAX using Euclidean and Dynamic Time Warping distance of the differenced time series, respectively.

7 Further discussion and applications

Section 6 demonstrated that ABBA provides high compression rates while guaranteeing that the time series reconstruction is still close to the original. The high compression is a consequence of the stitching procedure during the compression stage. Section 5 showed how errors are accumulated piece by piece in the stitching process. We believe that this property prevents ABBA from admitting lower bounding distance measures as are available for SAX. SAX's lower bounding measure and indexability make it suitable for applications where multiple time series have to be compared (like time series classification). ABBA, on the other hand, appears best suited for applications where information has to be extracted from a single time series, such as anomaly detection, motif discovery, and trend prediction. As the output of ABBA is simply a string sequence, it can be combined with existing algorithms that previously used, e.g., a SAX representation. Below we discuss various aspects and applications of ABBA.

In-built differencing. Working with the increments (instead of slopes) allows ABBA to capture linear trends in time series without preprocessing. In Figure 11 we consider the simple test problem of a sine wave with a gradual linear trend in the presence of noise. After normalization, SAX is able to accurately represent the time series as shown in Figure 11(i). If we used the symbolic representation for trend prediction, however, the SAX representation would be unsuitable for continuing the linear trend as new symbols would need to be introduced. Of course, this problem could be overcome by removing the linear trend through differencing the time series. A SAX representation of the differenced time series is shown in Figure 11(ii). Unfortunately, differencing the noisy time series amplifies the noise. Figure 11(iii) compares the original time series against the reconstructed time series from the SAX representation of the differenced data. As we can see, the increased noise level renders the SAX
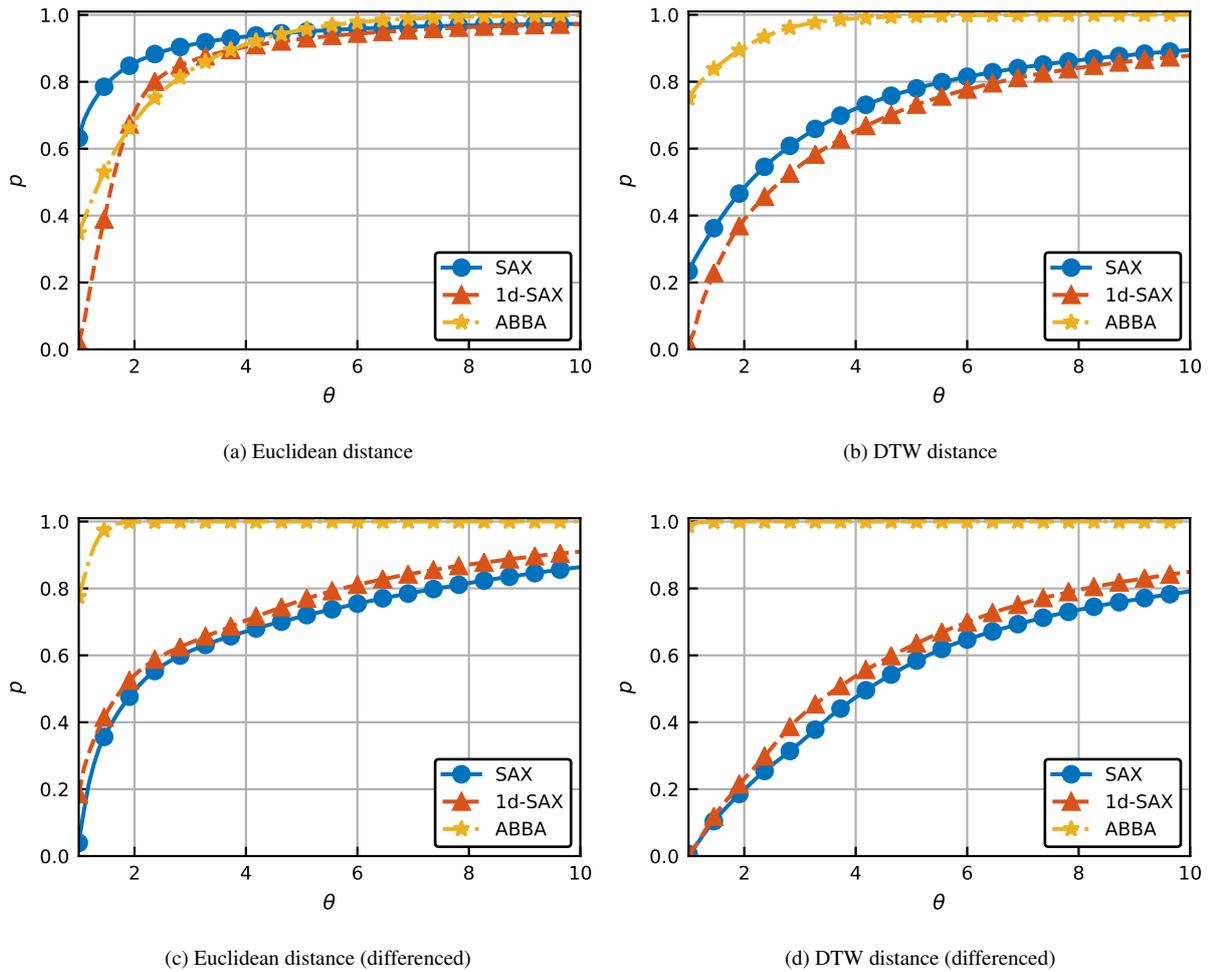
(a) Euclidean distance

(b) DTW distance

(c) Euclidean distance (differenced)

(d) DTW distance (differenced)

Fig. 10: Performance profiles for the reconstruction errors of SAX, 1d-SAX, and ABBA *with scaling parameter* `scl` = 1. Figures 10a and 10b compare ABBA (`scl` = 1) with SAX and 1d-SAX using the Euclidean and Dynamic Time Warping distance, respectively. Figures 10c and 10d compare ABBA (`scl` = 1) with SAX and 1d-SAX using the Euclidean and Dynamic Time Warping distance of the differenced time series, respectively.

representation extremely inaccurate. ABBA, on the other hand, does not require any differencing as it works with increments by default. As a consequence, the ABBA reconstruction shown in Figure 11(iv) stays very close to the original time series, capturing both the gradual linear trend as well as the characteristic up-and-down behavior.

Anomaly detection refers to the problem of finding points or intervals in time series which display surprising or unexpected behavior. Recent literature reviews of existing anomaly detection algorithms are given in [4, 21]. The ABBA representation can be used for anomaly detection in a variety of ways. Trend anomalies can be detected in the digitization procedure via k-means clustering of the lengths and increments. The alphabet is ordered such that `'a'` is the most frequent symbol followed by `'b'` and so forth. If the kth cluster contains very few elements relative to the other clusters, then this might be considered a trend anomaly.

TARZAN [24] is a popular anomaly detection algorithm with linear time and space complexity [39]. The algorithm requires two time series, a reference time series R containing normal behavior and the test time series X. Both time series are converted to a symbolic representation and stored in a suffix tree [36]. An anomaly score is computed by comparing the frequency of a substring in X to an expected frequency computed from R. SAX can be used for the discretization process in TARZAN and has been shown to outperform other symbolic representations with no dimensional reduction [32].

If both symbolic representations are short and X contains a symbol that does not appear in R, then the TARZAN score can suffer through lack of perspective. For example, suppose the expected frequency of the substring `'abc'` is 4.2 and `'abc'` appears 3 times in X, then the anomaly score is $3 - 4.2 = -1.2$. Suppose the symbol `'d'` does not appear in R but `'ada'` appears in X. The expected frequency of the substring `'ada'` is 0 and `'ada'` appears
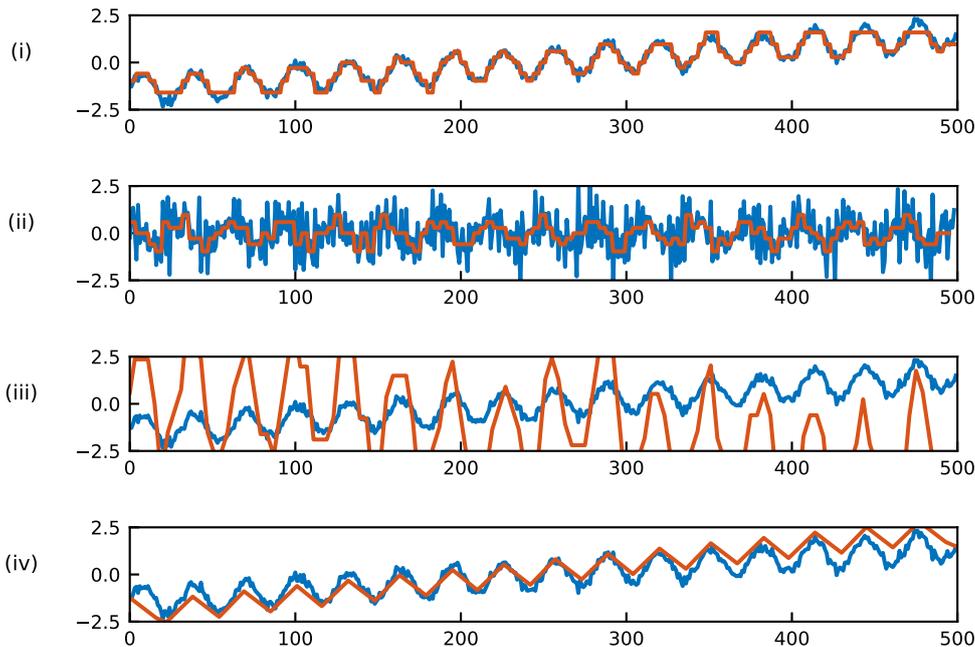
Fig. 11: Comparison of SAX and ABBA on a noisy sine wave with a gradual linear trend. (i) The original time series is shown in blue and the SAX representation is shown in orange. (ii) The differenced version of the original time series is shown in blue and the its SAX representation is shown in orange. (iii) The original time series is given in blue and the cumulative sum of the SAX representation from (ii) is shown in orange. (iv) The original time series is shown in blue and its ABBA representation is shown in orange.

only once, so the anomaly score is $0 - 1 = -1$. This implies that `'abc'` is more of an anomaly than `'ada'`. This issue can be overcome by dividing the anomaly score by the largest of the expected/actual frequency.

In Figures 12 and 13 we consider a simple experiment comparing SAX, 1d-SAX, and ABBA as discretization procedures for TARZAN with the modified anomaly score[2]. The reference time series R is a simple sine wave where each period spans 25 time samples. The time series X has a full wave replaced by a flat line of 22 time points. The SAX and 1d-SAX representations use a window length $w = 5$ and $k = 9$ symbols, whereas ABBA uses a tolerance tuned to give a symbolic representation of equal length and k is bounded by 9. The time series R and X and their symbolic reconstructions are shown in Figure 12. If the length of the anomaly does not align with the window length w, then SAX and 1d-SAX tend to represent the sine wave following the anomaly as a different substring. The adapted TARZAN score is required as certain symbols appear in X that do not appear in R. Figure 13 shows the resulting TARZAN anomaly scores. Both SAX and 1d-SAX suffer from the fixed window length, returning high anomaly scores throughout time following the anomaly, whereas TARZAN using ABBA is able to recover almost immediately after the anomaly due to the adaptive segment lengths.

VizTree. We finally mention the possibility of representing an ABBA output as a VizTree, a time series pattern discovery and visualization tool based on suffix trees [29, 30, 31]. The authors use SAX to discretize the time series before building a suffix tree. Each branch of the suffix tree represents a substring and the thickness of that branch represents the frequency of the substring in the symbolic representation. In principle, SAX pairs well with the visualization as the Gaussian breakpoints should ensure that each symbol appears equally likely. In practice, this is often not the case. One could use ABBA's discretization process instead of SAX by relating the thickness of each line to the frequency of the symbols determined in the clustering procedure. A poor choice of the window length w in the piecewise aggregate approximation in SAX could lead to missing motifs if the distance between is not near a multiple of w. Furthermore, SAX might fail to detect motifs if time warping has occurred, whilst VizTree via ABBA should be able to better capture time-warped motifs as the segment lengths are chosen adaptively. A further exploration of this application will be the subject of future work.

---

[2] A Python implementation of TARZAN which supports the use of SAX, 1d-SAX, and ABBA can be downloaded from `https://github.com/nla-group/TARZAN`.
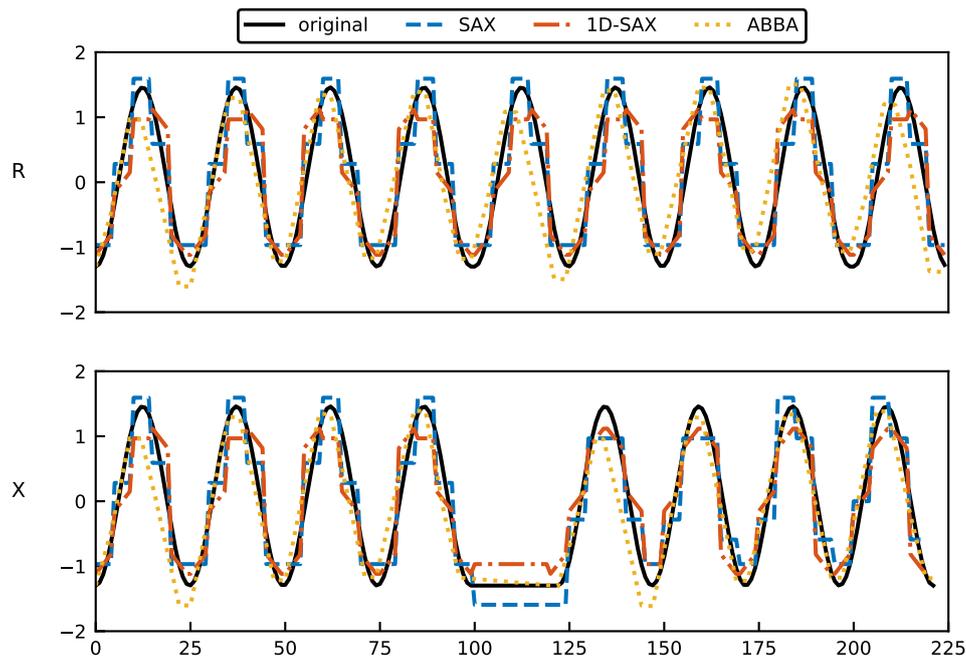
Fig. 12: A visual comparison of the symbolic representations of two time series. Here, R is the reference time series, a simple sine wave, while X is the test time series, a sine wave with a flat region slightly shorter than one wave period.
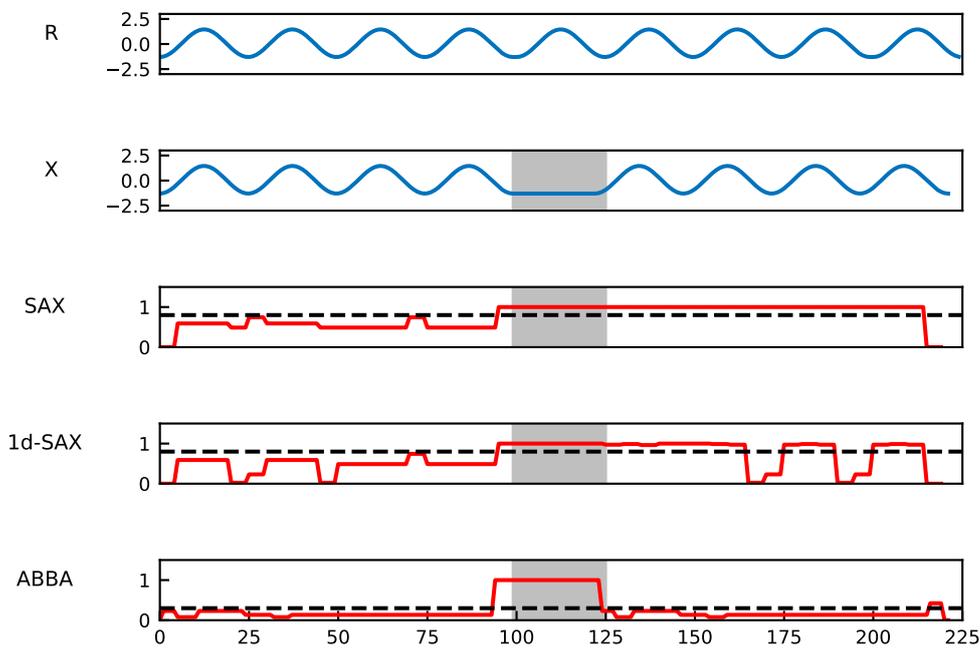

Fig. 13: A comparison of the TARZAN anomaly detection algorithm using the SAX, 1d-SAX, and ABBA representations, respectively. The first time series R is the reference, while the second time series X is to be tested. The final three plots show the adapted TARZAN anomaly scores for the SAX, 1d-SAX, and ABBA representations, respectively. The black dashed lines indicate tolerances that could be used define the anomalies.

8 Conclusions and future work

We introduced ABBA, an adaptive symbolic time series representation which aims to preserve the essential shape of a time series. We have shown that the ABBA representation has favorable approximation properties compared to other popular representations, in particular, when the dynamic time warping distance is used. Furthermore, we demonstrated the use of ABBA in some important data mining applications, including trend prediction and anomaly detection. Future research will be devoted to an online streaming version of ABBA with the necessary adaptations of the the Brownian bridge-based error analysis, as well as a more in-depth study of VizTree visualizations. Our recent work [15] explores ABBA's potential for time series forecasting.

References

1. Abanda, A., Mori, U., Lozano, J.A.: A review on distance based time series classification. Data Min. Knowl. Discov. **33**(2), 378–412 (2019). DOI 10.1007/s10618-018-0596-4. URL https://doi.org/10.1007/s10618-018-0596-4

2. Aghabozorgi, S., Shirkhorshidi, A.S., Wah, T.Y.: Time-series clustering–a decade review. Information Systems **53**, 16–38 (2015). DOI https://doi.org/10.1016/j.is.2015.04.007. URL http://www.sciencedirect.com/science/article/pii/S0306437915000733

3. Arthur, D., Vassilvitskii, S.: How slow is the *k*-means method? In: Symposium on Computational Geometry, vol. 6, pp. 1–10. ACM, New York (2006). URL http://theory.stanford.edu/~sergei/papers/kMeans-socg.pdf

4. Atluri, G., Karpatne, A., Kumar, V.: Spatio-temporal data mining: A survey of problems and methods. ACM Computing Surveys (CSUR) **51**(4), 83 (2018)

5. Barnaghi, P.M., Bakar, A.A., Othman, Z.A.: Enhanced symbolic aggregate approximation method for financial time series data representation. In: 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012), pp. 790–795. IEEE (2012)

6. Baydogan, M.G., Runger, G.: Learning a symbolic representation for multivariate time series classification. Data Mining and Knowledge Discovery **29**(2), 400–422 (2015)

7. Benyahmed, Y., Bakar, A.A., Hamdan, A.R., Abdullah, S.M.S.: A time-weighted average-based PAA representation for time series symbolization. International Journal of Advances in Soft Computing & Its Applications (2015)

8. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: KDD Workshop, vol. 10, pp. 359–370 (1994)

9. Bettaiah, V., Ranganath, H.S.: An analysis of time series representation methods: data mining applications perspective. In: Proceedings of the 2014 ACM Southeast Regional Conference, pp. 16:1–16:6. ACM (2014). DOI 10.1145/2638404.2638475. URL http://doi.acm.org/10.1145/2638404.2638475

10. Bondu, A., Boullé, M., Cornuéjols, A.: Symbolic representation of time series: a hierarchical coclustering formalization. In: International Workshop on Advanced Analysis and Learning on Temporal Data, pp. 3–16. Springer (2016)

11. Boullé, M.: MODL: A Bayes optimal discretization method for continuous attributes. Mach. Learn. **65**(1), 131–165 (2006). DOI 10.1007/s10994-006-8364-x. URL https://doi.org/10.1007/s10994-006-8364-x

12. Chiu, B., Keogh, E., Lonardi, S.: Probabilistic discovery of time series motifs. In: Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 493–498. ACM (2003)

13. Dau, H.A., Keogh, E., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Yanping, Hu, B., Begum, N., Bagnall, A., Mueen, A., Batista, G.: The UCR time series classification archive (2018). URL https://www.cs.ucr.edu/~eamonn/time_series_data_2018/

14. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2, Ser. A), 201–213 (2002). DOI 10.1007/s101070100263. URL https://doi.org/10.1007/s101070100263

15. Elsworth, S., Güttel, S.: Time Series Forecasting Using LSTM Networks: A Symbolic Approach. arXiv EPrint arXiv:2003.05672v1, Manchester Institute for Mathematical Sciences, The University of Manchester, UK (2020). URL https://arxiv.org/abs/2003.05672
16. Esmael, B., Arnaout, A., Fruhwirth, R.K., Thonhauser, G.: Multivariate time series classification by combining trend-based and value-based approximations. In: International Conference on Computational Science and Its Applications, pp. 392–403. Springer (2012)
17. Fu, T.c.: A review on time series data mining. Eng. Appl. Artif. Intell. **24**(1), 164–181 (2011). DOI 10.1016/j.engappai.2010.09.007. URL http://dx.doi.org/10.1016/j.engappai.2010.09.007
18. Ganz, F., Barnaghi, P., Carrez, F.: Information abstraction for heterogeneous real world internet data. IEEE Sensors Journal **13**, 3793–3805 (2013). DOI 10.1109/JSEN.2013.2271562
19. Grabocka, J., Wistuba, M., Schmidt-Thieme, L.: Scalable classification of repetitive time series through frequencies of local polynomials. IEEE Transactions on Knowledge and Data Engineering **27**(6), 1683–1695 (2014)
20. Grønlund, A., Larsen, K.G., Mathiasen, A., Nielsen, J.S.: Fast exact k-means, k-medians and Bregman divergence clustering in 1D. arXiv preprint arXiv:1701.07204 (2017). URL http://arxiv.org/abs/1701.07204
21. Gupta, M., Gao, J., Aggarwal, C.C., Han, J.: Outlier detection for temporal data: A survey. IEEE Transactions on Knowledge and Data Engineering **26**(9), 2250–2267 (2013)
22. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: Proceedings of the 2001 IEEE International Conference on Data Mining, pp. 289–296 (2001). DOI 10.1109/ICDM.2001.989531
23. Keogh, E., Kasetty, S.: On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Min. Knowl. Discov. **7**(4), 349–371 (2003). DOI 10.1023/A:1024988512476. URL https://doi.org/10.1023/A:1024988512476
24. Keogh, E., Lonardi, S., Chiu, B.c.: 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, pp. 550–556. ACM (2002)
25. Keogh, E., Ratanamahatana, C.A.: Exact indexing of dynamic time warping. Knowledge and Information Systems, Springer **7**(3), 358–386 (2005). DOI 10.1007/s10115-004-0154-9. URL https://doi.org/10.1007/s10115-004-0154-9
26. Keogh, E.J., Pazzani, M.J.: Derivative dynamic time warping. In: Proceedings of the 2001 SIAM International Conference on Data Mining, pp. 1–11 (2001)
27. Kim, J.Y.: Detection of change in persistence of a linear time series. J. Econometrics **95**(1), 97–116 (2000). DOI https://doi.org/10.1016/S0304-4076(99)00031-7. URL http://www.sciencedirect.com/science/article/pii/S0304407699000317
28. Li, G., Zhang, L., Yang, L.: TSX: a novel symbolic representation for financial time series. In: PRICAI 2012: Trends in Artificial Intelligence, pp. 262–273. Springer (2012)
29. Lin, J., Keogh, E., Lonardi, S.: Visualizing and discovering non-trivial patterns in large time series databases. Information Visualization **4**(2), 61–82 (2005)
30. Lin, J., Keogh, E., Lonardi, S., Lankford, J.P., Nystrom, D.M.: Visually mining and monitoring massive time series. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 460–469. ACM (2004)
31. Lin, J., Keogh, E., Lonardi, S., Lankford, J.P., Nystrom, D.M.: Viztree: a tool for visually mining and monitoring massive time series databases. In: Proceedings of the 30th International Conference on Very Large Data Bases - Volume 30 (VLDB '04), pp. 1269–1272. VLDB Endowment (2004)
32. Lin, J., Keogh, E., Wei, L., Lonardi, S.: Experiencing SAX: a novel symbolic representation of time series. Data Min. Knowl. Discov. **15**(2), 107–144 (2007). DOI 10.1007/s10618-007-0064-z. URL https://doi.org/10.1007/s10618-007-0064-z
33. Lkhagva, B., Suzuki, Y., Kawagoe, K.: New time series data representation ESAX for financial applications. In: 22nd International Conference on Data Engineering Workshops (ICDEW'06), pp. x115–x115 (2006). DOI doi={10.1109/ICDEW.2006.99}
34. Luo, G., Yi, K., Cheng, S.W., Li, Z., Fan, W., He, C., Mu, Y.: Piecewise linear approximation of streaming time series data with max-error guarantees. In: 2015 IEEE 31st International Conference on Data Engineering, pp. 173–184 (2015). DOI 10.1109/ICDE.2015.7113282
35. Malinowski, S., Guyet, T., Quiniou, R., Tavenard, R.: 1d-SAX: a novel symbolic representation for time series. In: Advances in Intelligent Data Analysis XII, pp. 273–284. Springer (2013)
36. McCreight, E.M.: A space-economical suffix tree construction algorithm. Journal of the ACM (JACM) **23**(2), 262–272 (1976)

37. Mörchen, F., Ultsch, A.: Finding persisting states for knowledge discovery in time series. In: From Data and Information Analysis to Knowledge Engineering, pp. 278–285. Springer (2006)

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

39. Pelkonen, T., Franklin, S., Teller, J., Cavallaro, P., Huang, Q., Meza, J., Veeraraghavan, K.: Gorilla: A fast, scalable, in-memory time series database. Proceedings of the VLDB Endowment **8**(12), 1816–1827 (2015)

40. Pham, N.D., Le, Q.L., Dang, T.K.: HOT aSAX: a novel adaptive symbolic representation for time series discords discovery. In: Proceedings of the Second International Conference on Intelligent Information and Database Systems: Part I, pp. 113–121. Springer (2010). URL `http://dl.acm.org/citation.cfm?id=1894753.1894768`

41. Shieh, J., Keogh, E.: iSAX: indexing and mining terabyte sized time series. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 623–631. ACM (2008)

42. Wang, H., Song, M.: Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming. The R Journal **3**(2), 29–33 (2011). URL `https://journal.r-project.org/archive/2011-2/RJournal_2011-2_Wang+Song.pdf`

43. Zhang, K., Li, Y., Chai, Y., Huang, L.: Trend-based symbolic aggregate approximation for time series representation. In: 2018 Chinese Control And Decision Conference, pp. 2234–2240. IEEE (2018). DOI 10.1109/CCDC.2018.8407498