Data Mining and Knowledge Discovery (2026) 40:21
https://doi.org/10.1007/s10618-026-01189-9

# Rock the KASBA: blazingly fast and accurate time series clustering

**Christopher Holder**[1] · **Anthony Bagnall**[1,2]

Received: 8 September 2025 / Accepted: 19 January 2026 / Published online: 25 February 2026
© The Author(s) 2026

**Abstract**
Time series data has become increasingly prevalent across numerous domains, driving a growing demand for time series machine learning techniques. Among these, time series clustering (TSCL) stands out as one of the most popular machine learning tasks. TSCL serves as a powerful exploratory analysis tool and is also employed as a preprocessing step or subroutine for various tasks, including anomaly detection, segmentation, and classification. The most popular TSCL algorithms are either fast (in terms of runtime) but perform poorly on benchmark problems, or perform well on benchmarks but scale poorly. We present a new TSCL algorithm, the *k*-means (K) Accelerated (A) Stochastic subgradient (S) Barycentre (B) Average (A) (KASBA) clustering algorithm. KASBA is a *k*-means clustering algorithm that uses the Move-Split-Merge (MSM) elastic distance at all stages of clustering, applies a randomised stochastic subgradient descent to find barycentre centroids, links each stage of clustering to accelerate convergence and exploits the metric property of MSM distance to avoid a large proportion of distance calculations. It is a versatile and scalable clusterer designed for real-world TSCL applications. It allows practitioners to balance runtime and clustering performance when similarity is best measured by an elastic distance. We demonstrate through extensive experimentation that KASBA matches the current shape based state of the art clusterers and offers orders of magnitude improvement in runtime over the most performant elastic distance based *k*-means alternatives.

**Keywords** Time series clustering · Elastic distances · Clustering · Barycentre average · Stochastic subgradient · k-means · k-means++ · Move-split-merge · Barycenter · DBA · Elastic barycentre averaging · MBA · KASBA · *k*-means Accelerated stochastic subgradient barycentre average

Extended author information available on the last page of the article

## 1 Introduction

We consider a time series as an ordered sequence of real valued observations. Time series data has become ubiquitous, emerging across numerous domains such as astronomy, biology, engineering, finance, manufacturing, medicine, meteorology, and more (McDowell et al. 2018; Wenig et al. 2024; Gutiérrez et al. 2023). The widespread generation of time series data, coupled with the desire to analyse and derive insights from it, has driven substantial interest in time series machine learning tasks such as anomaly detection, classification, clustering, forecasting, querying, regression, and segmentation. One of the most popular fields is time series clustering (TSCL) (Lafabregue et al. 2022a; Holder et al. 2024b; Paparrizos and Bogireddy 2025). The objective of TSCL is to group time series into clusters where the series within a cluster exhibit homogeneity, while those in different clusters display heterogeneity. TSCL is a common starting point for exploratory data analysis (Zolhavarieh et al. 2014) and is also often used as a step in a supervised pipeline (Dhariyal et al. 2023; Serantoni et al. 2024).

There have been many time series specific clustering algorithms proposed. These can broadly be grouped into deep learning (Alqahtani et al. 2021), feature based (Zhang et al. 2019) and distance based algorithms (Holder et al. 2024b). Historically, distance based algorithms have been the most popular and are the focus of this work. There have been a large number of time series specific distance functions proposed and numerous comparative studies of their application to TSCL (Holder et al. 2024b) and classification (Lines and Bagnall 2015). These have empirically shown that distance measures that ignore temporal ordering (e.g., Euclidean distance) yield significantly worse similarity measures between time series for these tasks. This is because small misalignments can lead to large distance scores for series that are conceptually similar in shape. The most widely used distance measure that compensates for offset is Dynamic Time Warping (DTW) (Ratanamahatana and Keogh 2005), the first in a family of algorithms commonly known as elastic distances (Lines and Bagnall 2015). Elastic distances account for misalignment between time series during distance computation. Another popular approach adopted by the *k*-Shape algorithm (Paparrizos and Gravano 2016) is to employ distance measures based on the cross correlation between series. A recent study (Paparrizos and Bogireddy 2025) comparing 84 clustering algorithms concluded *k*-Shape *"occupies a sweet spot between fast/low-performance and slow/high-performance methods"* and concluded *"Although many methods have been proposed, no algorithm consistently outperforms the decade-old baseline k-Shape"*.

The most common approach to clustering time series data is to use traditional clustering algorithms but to replace the standard distance measure (e.g. Euclidean distance), with a time series distance measure. Among these, DTW has been the most widely used for adapting traditional clustering algorithms such as *k*-means (Petitjean et al. 2011), *k*-medoids (Holder et al. 2023a), Agglomerative Clustering (Almahamid and Grolinger 2022), Density Peaks (Begum et al. 2016), DBSCAN (Javed et al. 2020) or hierarchical methods (Yang and Lin 2024). For many years, DTW combined with traditional clustering models, such as Partitioning Around Medoids (PAM) and Agglomerative Clustering, was considered state-of-the-art (Javed et al. 2020). More

recent algorithms such as *k*-means using DTW barycentre averaging (DBA) (Petitjean et al. 2011) and Soft-DBA (Cuturi and Blondel 2017) improve elastic distance-based TSCL performance within a *k*-means clustering framework. However, these algorithms require significant computation, thereby limiting their applicability for real-world TSCL applications. This underscores that clustering performance is not the sole consideration for TSCL practitioners; the runtime of an algorithm is of similar importance for many applications.

Our focus is on identifying the most effective approach to using an elastic distance measure in *k*-means for TSCL. To this end, we aim to bridge the gap between elastic distance based clustering and shape based algorithms such as *k*-Shape and Shape-DBA (Ismail-Fawaz et al. 2023b) by introducing KASBA: the *k*-means (K) Accelerated (A) Stochastic subgradient (S) Barycentre (B) Average (A) clustering algorithm. KASBA is designed to work seamlessly with any elastic distance that satisfies the properties of a metric, integrating it into every stage of the *k*-means process to create a fully end-to-end solution. While this paper focuses on experiments using the Move-Split-Merge (MSM) (Stefan et al. 2013) elastic distance, which is a metric, the algorithm is flexible. Practitioners who wish to use another elastic distance metric, such as Time Warp Edit (TWE) (Marteau 2009), can easily substitute the appropriate function in the implementation.

Our contributions can be summarised as follows: We present KASBA, a scalable *k*-means approach to time series clustering under a metric elastic distance. KASBA is technically distinguished by:

1. an elastic *k*-means++ initialisation that exposes reusable assignment and distance state;
2. exact triangle-inequality pruning enabled by metricity to reduce assignment-time distance evaluations; and
3. a warm-started stochastic-subgradient barycentre update with early stopping and mini-batch updates that makes elastic centroid estimation practical.

We evaluate KASBA through an extensive experimental evaluation, comparing its clustering performance and runtime against a range of current state-of-the-art clustering algorithms. Our results demonstrate that KASBA delivers state-of-the-art performance in a fraction of the time of the most popular elastic distance based TSCL algorithms. We perform an ablation study to evaluate the contributions of KASBA's structural components and parameters, offering deeper insights into the mechanisms behind its efficiency. We reduce assignment distance calls by approximately 40 percent with identical clusterings, improve accuracy and reduce the number of epochs through our novel form of gradient descent. The integrated end-to-end approach adopted by KASBA yields one to three orders of magnitude speed up over state-of-the-art elastic distance based clustering with no significant difference in accuracy.

This paper is structured as follows. Section 2 provides background into TSCL and reviews related research in TSCL such as elastic distances and barycentre averaging. Section 3 describes the KASBA algorithm in detail and identifies how KASBA combines and extends a range of TSCL *k*-means related research. We describe our experi-

mental setup and present results in Sect. 5 before a closer exploration of KASBA performance in Sect. 6 and concluding in Sect. 7.

## 2 Background

The development of clustering algorithms has a rich history and remains an active area of research. There are a broad range of families of algorithms: For example, density based methods (Ester et al. 1996; Yang and Lin 2025) define clusters as regions of high sample density separated by low-density areas and hierarchical techniques build a tree of merges or splits instead of a single partition (Ward 1963; Yang and Lin 2024). Perhaps the most popular family of algorithms is the partitional approach, which splits the data into non overlapping clusters to optimise an objective function. Once split, clusters are usually characterised by an exemplar, either from the data (a medoid) or created from the cluster members (a centroid).

TSCL methods exploit the unique characteristics of time series. A time series is a sequence of $m$ ordered, real-valued observations of a variable, denoted as $\mathrm{x} = (x_1, \ldots, x_m)$. If each observation $x_i$ is a vector, then x constitutes a multivariate time series. In this work, we focus on the case where each $x_i$ is a scalar; that is, x is a univariate time series. For simplicity, we also assume that all time series are of equal length $m$.

These restrictions are primarily adopted to streamline notation and exposition. In practice, all the distance measures and clustering algorithms we discuss can be extended to multivariate and unequal-length series with relative ease, as demonstrated in recent work (Shifaz et al. 2023).

The goal of TSCL is to take a collection of time series and group them together based on homogeneity. Our focus is on the perennially popular $k$-means clustering algorithm (MacQueen 1967) that forms representative centroids (or prototypes) for each cluster, then assigns series to centroids based on a distance measure. We assume $k$ is known for all experiments (we outline how we identify $k$ in Sect. 5). $k$-means is a partitional clustering algorithm involving the following steps:

1. *initialisation*: create initial centroids (prototypes) for each cluster.
2. *assignment*: assign each instance to the nearest centroid, based on a distance function.
3. *update*: recalculate the centroids based on the new assignment.
4. *stop or repeat*: determine whether to continue with a further iteration based on some stopping condition.

Assignment and update are repeated in an expectation-maximisation style greedy algorithm that minimises the sum of squared error (SSE) objective function:

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n_i} d^2(\mathrm{x}_j^{(i)}, \mathrm{c}_i) \tag{1}$$

where $n_i$ is the number of time series assigned to cluster $i$, $\mathrm{x}_j^{(i)}$ is the $jth$ time series assigned to $ith$ cluster, $\mathrm{c}_i$ is the $ith$ prototype/centroid time series, and $d$ is a distance measure between two series. For tabular clustering, $d$ is assumed to be the Euclidean distance for convergence reasons (Bottou and Bengio 1994). For TSCL, $d$ can be any time series specific distance.

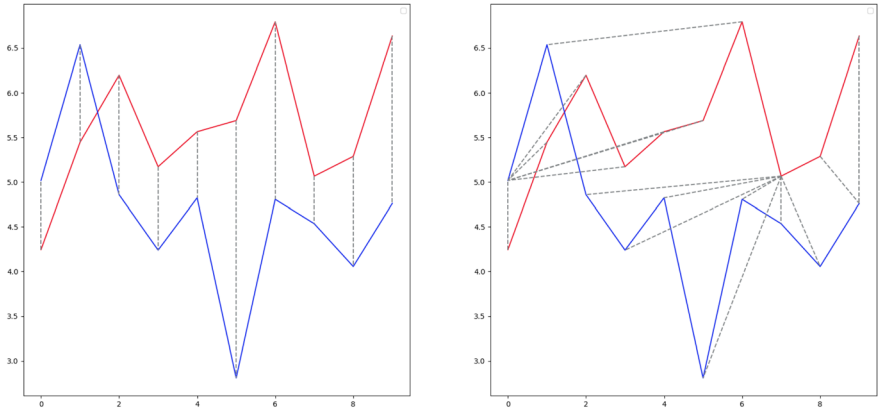
An important consideration when changing the distance measure is the method used to compute centroids. With the Euclidean distance, centroids are calculated as the arithmetic mean, as this minimises the SSE for that specific distance measure. However, when switching to a different distance measure, such as DTW, an alternative averaging technique that minimises the SSE for DTW must be applied to ensure proper convergence. We focus on two elastic distance functions, DTW and Move-Split-Merge (Stefan et al. 2013) and the possible averaging algorithms that could be used with them.

### 2.1 Elastic distances

Measuring the distance between time series is a fundamental operation used across a range of tasks, including classification, clustering, regression, anomaly detection, and query retrieval. Distances that account for temporal misalignment between two time series during comparison are commonly referred to as elastic distances.

Figure 1a illustrates an example where pointwise $L_p$ distances (e.g. Euclidean distance) may fail to capture the true similarity between two series: even a small offset between intrinsically similar sequences can result in a large distance.

Although many elastic distances have been proposed, we focus on two in particular: DTW and MSM. Figure 1b shows how DTW can align two time series to better reflect their similarity. DTW remains the most widely used and extensively researched elastic distance for TSC and TSCL tasks. However, MSM has demonstrated competitive performance in both *k*-means clustering (Holder et al. 2024b)



(a) Euclidean alignment path. (b) DTW alignment path.

**Fig. 1** Examples of alignments between two time series using Euclidean and DTW distances. The dashed grey lines indicate the pointwise comparisons made between the red and blue time series

and *k*-nearest neighbour classification (Lines and Bagnall 2015). Unlike DTW, MSM satisfies the properties of a metric.

For comprehensive surveys of elastic distance measures, we refer readers to Holder et al. (2024b); Shifaz et al. (2023); Abanda et al. (2019), and to Stefan et al. (2013) for a formal proof of the metricity of MSM.

#### 2.1.1 Dynamic time warping (DTW)

Dynamic Time Warping (DTW) was first proposed for time series machine learning by Berndt and Clifford (1994) and has since been employed in thousands of publications. DTW enables one-to-many alignments ("warping") between points in two time series (Shifaz et al. 2023). For time series classification, the one nearest neighbour (1-NN) classifier using a tuned version of DTW was considered state-of-the-art for many years, and it remains a common baseline in time series classification benchmarks (Middlehurst et al. 2024b).

DTW employs dynamic programming to identify the optimal alignment path through a cost matrix, denoted *CM*, that minimises the cumulative distance between two time series. This is achieved by constructing a pairwise matrix in which each entry represents the cost of aligning a point from the first time series with a point from the second. The cost matrix *CM* is of size $m \times m$, where *m* is the length of the series. All entries are initially set to $\infty$, except for the starting point: $CM_{1,1} = 0$.

Once the cost matrix has been initialised, *CM* is incrementally updated according to:

$$CM_{i,j} = (a_i - b_j)^2 + \min \begin{cases} CM_{i-1,j-1} \\ CM_{i-1,j} \\ CM_{i,j-1} \end{cases} \tag{2}$$

where *i* and *j* are integers where $2 \leq i \leq m$ and $2 \leq j \leq m$, and *a* and *b* are the time series.

The alignment path (or warping path) can be viewed as a series of steps through the cost matrix. At each step, an elastic distance algorithm may take one of three paths: diagonal, vertical, or horizontal. DTW imposes no explicit penalty for deviating from the diagonal; instead, it accumulates an implicit penalty by increasing the total path length.

To recover the optimal alignment path, one must backtrack through the cost matrix. Formally, the alignment path is defined as $P = \langle (e_1, f_1), (e_2, f_2), \ldots, (e_s, f_s) \rangle$, where each element represents a position in the cost matrix *CM*. A valid path begins at (1, 1), ends at (*m*, *m*), and must be monotonic, i.e., for all $1 \leq i < s$, it holds that $0 \leq e_{i+1} - e_i \leq 1$ and $0 \leq f_{i+1} - f_i \leq 1$. An example of a DTW alignment path is shown in Fig. 1.

### 2.1.2 Move-split-merge (MSM)

An alternative family of distance functions is based on the concept of edit distances. The MSM distance (Stefan et al. 2013) defines three fundamental operations for transforming one time series into another: *Move*, *Split*, and *Merge*. A diagonal move represents a substitution (or match), a vertical move corresponds to a split (insertion), and a horizontal move corresponds to a merge (deletion).

The *Move* operation modifies the value of a single time series point, with a cost equal to the absolute difference between the original and the new value. This contrasts with DTW, which uses squared Euclidean distance.

The *Split* operation duplicates a value at a given time step, effectively inserting a new point in the series. For instance, given a value $a_i$, applying a split produces two consecutive values equal to $a_i$. The inserted value may subsequently be altered via a Move operation to better align with the other series. The cost of a split is a constant $c$, representing the structural cost of insertion.

The *Merge* operation is the inverse of a split: it removes one of two consecutive equal values by collapsing them into a single value. A merge is only valid when the two adjacent values are equal, ensuring reversibility. Its cost is also $c$, which ensures the symmetry required for metricity.

The combined use of Move, Split, and Merge allows MSM to express insertions and deletions in a context-aware manner. For example, inserting a value between two similar values incurs a lower cost than inserting one that deviates significantly from its neighbours. The formal definitions of these operations and the MSM distance are provided in Eqs. 3 and 4.

MSM satisfies the triangle inequality and is a proper metric (Stefan et al. 2013). In our experiments, we fix the cost parameter to $c = 1$.

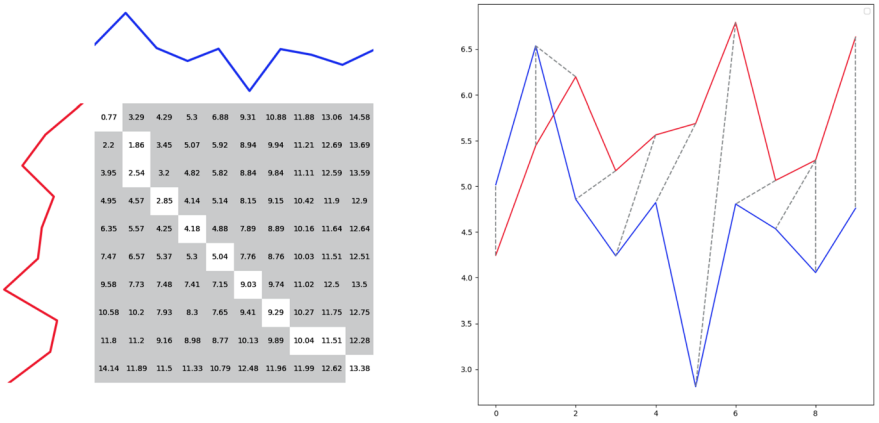
Similar to DTW, MSM computes a pairwise cost matrix in which each entry represents the cost of aligning a point from the first time series with a point from the second. The cost matrix $CM$ is of size $m \times m$, with all values initially set to $\infty$ except for $CM_{1,1}$, which is initialised to the absolute distance between the first elements of each series: $CM_{1,1} = |a_1 - b_1|$.

The matrix is then updated incrementally as follows:

$$CM_{i,j} = \min \begin{cases} CM_{i-1,j-1} + |a_i - b_j| \\ CM_{i-1,j} + \text{msm\_cost}(a_i, a_{i-1}, b_j) \\ CM_{i,j-1} + \text{msm\_cost}(b_j, a_i, b_{j-1}) \end{cases} \tag{3}$$

where:

$$\text{msm\_cost}(x, y, z) = \begin{cases} c, & \text{if } y \le x \le z \text{ or } y \ge x \ge z \\ c + \min\left(|x - y|, |x - z|\right), & \text{otherwise} \end{cases} \tag{4}$$




(a) (b)

**Fig. 2** MSM optimal warping path and cost matrix. Panel **a** shows the cost matrix with the optimal path highlighted; panel **b** illustrates the resulting alignment between the two time series

Here, *CM* denotes the MSM cost matrix; *i* and *j* are integers such that $2 \leq i \leq m$ and $2 \leq j \leq m$; *x* and *y* are the time series; and *c* is the fixed cost of a structural operation (Split or Merge) (Fig. 2).

## 2.2 Time series averaging techniques

The objective of a time series averaging technique is to construct a time series that represents the centre of a collection of time series (Schultz and Jain 2018), often called a prototype (or in the context of *k*-means a centroid). The simplest way to compute the average of a collection of time series is taking the arithmetic mean of all series over time points. This is independent of the ordering, and for most time series machine learning tasks this will result in ignoring important features. Because of this, several time series-specific averaging techniques have been proposed to allow for offset between otherwise similar series.

### 2.2.1 DTW barycentre averaging (DBA)

DTW Barycentre Averaging (DBA) (Petitjean et al. 2011) is an averaging algorithm that uses the alignment path between time series to find an average. A barycentre refers to the time series that minimises the sum of squared distances to a given set of time series (Petitjean et al. 2011). The most straightforward method for computing such a minimum is to use the arithmetic mean of each time point, which gives the time series that minimises the sum of squared Euclidean distances to all time series in the collection.

However, like the Euclidean distance, the arithmetic mean does not attempt to realign time series prior to averaging. Given the effectiveness of realigning time

series before distance computation (as in elastic distances), several methods have been proposed to integrate the DTW alignment directly into the averaging process.

To compute an average that minimises the sum of squared DTW distances, the problem is reformulated as an optimisation problem: finding the sequence $z$ that minimises the DTW Fréchet function (Fréchet 1948). Formally, this is expressed as:

$$F(\mathrm{z}) := \frac{1}{n} \sum_{i=1}^{n} DTW(\mathrm{z}, \mathrm{x}_i)^2 \tag{5}$$

where $F$ is the Fréchet function, z is the candidate barycentre, $n$ is the number of time series in collection X, and $DTW$ denotes the DTW distance.

A polynomial-time algorithm for globally minimising the non-differentiable, non-convex Fréchet function is currently unknown (Schultz and Jain 2018). Although exact averages can be computed (Brill et al. 2019), such methods are computationally infeasible for real-world applications. As a result, approximate algorithms are typically used.

The most widely used approximation is DBA (Petitjean et al. 2011), which begins with an initial prototype (commonly the arithmetic mean) and iteratively refines it. In each iteration, every series is aligned with the current prototype, and the values mapped to each position are collected. An arithmetic mean is then computed over the aligned values, preserving the warping structure.

Algorithm 1 outlines a single refinement iteration of DBA. In each iteration, each time series is aligned to the prototype (line 4), the aligned values are collected (lines 5–6), and the prototype is updated via averaging (lines 7–8). This process is typically repeated for a fixed number of iterations, denoted by $max\_iters$. In all our experiments, we set $max\_iters = 50$.

**Algorithm 1** DBA(c,X)

**Input:** **X**: collection of $n$ time series, each of length $m$
**c**: initial prototype of length $m$
**Output:** **c**: updated prototype
1 Let `dtw_path` be a function that returns a list of index pairs $(i, j)$ representing the DTW alignment path between two time series.
2 Let $W$ be a list of $m$ empty lists, where $W_i$ stores values from **X** that are aligned to prototype index $i$.
3 **foreach** $\mathbf{x} \in \mathbf{X}$ **do**
4 $P \leftarrow$ `dtw_path`$(\mathbf{c}, \mathbf{x})$
**foreach** $(i, j) \in P$ **do**
5 $W_i \leftarrow W_i \cup \{x_j\}$
6 **for** $i \leftarrow 1$ **to** $m$ **do**
7 $c_i \leftarrow$ `mean`$(W_i)$
8 **return c**

An important consideration in DBA is the choice of the initial prototype $c$. In the original publication (Petitjean et al. 2011), the prototype was randomly selected from the collection. Later versions used the DTW medoid (Petitjean et al. 2016), which requires computing a full pairwise DTW distance matrix−computationally expensive for large datasets. A subsequent version (Forestier et al. 2017) proposed using the arithmetic mean as the initial prototype for efficiency.

DBA has been shown to significantly outperform the arithmetic mean in $k$-means clustering tasks (Petitjean et al. 2011; Schultz and Jain 2018; Holder et al. 2023b; Ismail-Fawaz et al. 2023b). The DBA refinement (Algorithm 1) is repeated until the objective function (Eq. 5) no longer improves or a maximum number of iterations is reached (50 in all experiments).

#### 2.2.2 Elastic barycentre average

The Elastic Barycentre Average (Holder et al. 2023b) is a recently proposed generalisation of the DBA algorithm that can be applied with any elastic distance. While Holder et al. (2023b) demonstrated results specifically for the MSM elastic distance, they provided a generalised framework that allows for the use of any elastic distance that computes a complete alignment path through the cost matrix. The Elastic Barycentre Average replaces the DTW alignment path with the alignment path of any other elastic distance in Algorithm 1. It allows the clustering algorithm to easily leverage elastic distances that have been shown to be more effective for clustering than DTW (Holder et al. 2024b). A recent variant of the Elastic Barycentre Average, Shape-DBA (Ismail-Fawaz et al. 2023b), employs shape-DTW (Zhao and Itti 2018) elastic distance measure to form barycentres.

#### 2.2.3 Stochastic subgradient DTW barycentre average (SSG-DBA)

The Stochastic Subgradient DTW Barycentre Average (SSG-DBA) (Schultz and Jain 2018) is similar to DBA, but employs a Stochastic Subgradient (SSG) approach to optimise the Fréchet function rather than the greedy heuristic used in DBA. Both DBA and SSG-DBA are iterative methods that update a candidate solution through successive passes through the collection (epochs). The key difference is that DBA collates alignment paths to the centre then computes the average after the epoch whereas SSG-DBA updates the prototype after every alignment. The update is performed using a decaying learning rate. This means for a single pass or epoch through a collection of $p$ time series, DBA will perform $p$ alignment path computations and make one update to the barycentre, whereas SSG-DBA will perform $p$ alignment path computations and make $p$ updates to the barycentre. Like DBA, SSG-DBA will continue updating in epochs until the stopping criteria of no improvement in the objective function (Eq. 5), or a maximum number of iterations is reached. Because it updates the prototype during the epoch, SSG-DBA requires a further $p$ distance cal-

culations to test the stopping condition. This extra cost is justified through evidence of faster convergence than DBA (Schultz and Jain 2018).

#### 2.2.4 Soft-DTW barycentre average (Soft-DBA)

Soft-DTW (Cuturi and Blondel 2017) is a reformulation of DTW obtained by smoothing its Bellman recursion, making the objective differentiable for any temperature parameter $\gamma > 0$. This modification enables it to be used in gradient-based learning to minimise the Soft-DTW Fréchet function. This contrasts with the Elastic Barycentre Average, where the elastic distance does not have to be differentiable.

Soft-DBA with *k*-means has been shown to significantly outperform versions that use DBA, shape extraction (Paparrizos and Gravano 2016), and shift-invariant averaging (Yang and Leskovec 2011). However, this precision comes at high computational cost.

### 2.3 TSCL Algorithms

The four time series averaging functions described in Sect. 2.2 have historically been used with elastic distances measures. Alternative *k*-means approaches use distances based on lagged or windowed similarity. *k*-Spectral Centroid (*k*-SC) (Yang and Leskovec 2011) is a *k*-means algorithm that bases distance on comparing series over all circular shifts/lags. It finds spectral centroids, obtained from an eigenvalue-based decomposition of a positive semidefinite matrix constructed from the shifted, normalised cluster members. The *k*-Shape clusterer (Paparrizos and Gravano 2016) is a *k*-means algorithm that employs Shape Based Distance (SBD). SBD is derived from the maximum cross correlation between two series. *k*-Shape forms centroids from the first eigenvector from the Gram matrix of the aligned, normalised cluster members. Both employ non-elastic distances, since there is no local warping, only a global shift in time.

Partitional TSCL algorithms can be grouped into distance based algorithms, deep learning clusterers and feature based approaches. Our primary focus is distance based algorithms that use the *k*-means algorithm. A wide range of distance based *k*-means algorithms have been proposed, including, for example, anytime versions (Lin et al. 2004). We focus on the following adaptations of the core Lloyd's *k*-means algorithm (Lloyd 1982):

- *k-means* (MacQueen 1967) uses Euclidean distance assignment and arithmetic mean averaging. The algorithm is the same as used in traditional clustering and treats the time series as tabular data.
- *k-spectral centroids (k-SC)* (Yang and Leskovec 2011) uses windowed distance and eigenvector decomposition centroid construction.
- *k-shape* (Paparrizos and Gravano 2016) uses SBD distance assignment and ei-

genvector decomposition centroid construction.
- *k-DBA* (Petitjean et al. 2011) uses DTW distance in assignment and forms centroids with DTW barycentre averaging.
- *k-MBA* (Holder et al. 2023b) uses MSM distance measure and MSM barycentre averaging (MBA).
- *k-Shape-DBA* (Ismail-Fawaz et al. 2023b) uses shape-DTW distance measure and shape-DTW elastic barycentre average (Shape-DBA).
- *k-Soft-DBA* (Cuturi and Blondel 2017) uses Soft-DTW distance measure and Soft-DBA averaging technique.

Another popular partitional approach for TSCL is *k*-medoids (Leonard Kaufman 1990). *k*-medoids algorithms find a solution to the same optimisation problem as *k*-means but only considers training data instances as cluster prototypes. The prototypes are called medoids rather than centroids. *k*-medoids algorithms are some of the easiest to adapt for TSCL as the distance measure can simply be swapped out from the traditional Euclidean distance to any other without concern of changing the algorithms objective function, since there is no averaging stage. (Holder et al. 2023a) presented a thorough evaluation of 6 different *k*-medoids algorithms including Alternate (Lloyd 1982), Partition Around Medoids (PAM) (Leonard Kaufman 1990), Clustering LARge Applications (CLARA) (Kaufman and Rousseeuw 1990) and CLARA based on raNdomised Search (CLARANS) (Ng and Han 2002) and found that PAM using MSM (**PAM-MSM**) elastic distance outperformed the current state-of-the-art TSCL approaches.

An alternative approach to TSCL is to perform some form of transformation on the time series to form tabular data that can be used with traditional clustering algorithms, which we call feature based TSCL. A common way of doing this is to extract features or perform some form of dimensionality reduction. Some popular approaches include:

- *Rocket clustering (R-clustering)* (Jorge and Cuevas 2024) uses a modified version of the MiniROCKET (Dempster et al. 2021, 2020) feature extraction algorithm. Specifically R-Clustering adjusts the configuration of the hyperparameters to better suit clustering. Once the features have been created, the principal components of the features are then extracted before being passed to a traditional *k*-means clusterer to produce the final clustering results.
- *Unsupervised shapelets (U-shapelets)* (Zakaria et al. 2012) extracts shapelets from the data which are used to cluster the data. Shapelets are time series subsequences that are maximally representative of a class (Ye and Keogh 2009). For unsupervised tasks, unsupervised shapelets (u-shapelets) are extracted by sliding a window of a specified length over every time series in the dataset. Each subsequence is evaluated and ranked based on its utility, which indicates its discriminative power. The ranking process assesses how effectively a subsequence can separate subsets within the dataset, enabling the discovery of the most representative patterns without requiring class labels. Once the shapelets are extracted the distance between each time series and each shapelet is computed. This distance map is treated as a feature vector for each time series which are then clustered

using the traditional *k*-means algorithm.

- *Two-step time series cluster (TTC)* (Aghabozorgi et al. 2014) performs clustering in two steps. First time series are grouped according to similarity in time by applying an affinity search technique. Subsequently, for each cluster a prototype is defined according to the affinity of the time series belonging to it. The second step computes the DTW distances between the subclusters prototypes. The subclusters are merged based on similarity by means of the *k*-medoids standard clustering method to produce the final clustering.

There are numerous other TSCL approaches including deep learning based clustering algorithms e.g. Lafabregue et al. (2022a) and statistical model based approaches (Caiado et al. 2015) and many more, e.g. TADPole (Begum et al. 2016), SOMTimeS (Javed et al. 2024), SPF (Li et al. 2019), multi-view approaches (Han et al. 2022) and approaches designed for a single dataset such as JET (Wenig et al. 2024), *k*-Graph (Boniol et al. 2025) and dim-Sc (Ozer et al. 2020). These are beyond the scope of our study.

## 3 The KASBA clustering algorithm

The *k*-means (K) Accelerated (A) Stochastic subgradient (S) Barycentre (B) Average (A) (KASBA) clusterer delivers competitive clustering performance comparable to the state-of-the-art TSCL algorithms, while reducing runtime by up to three orders of magnitude. KASBA can work with any distance function that is a metric, but is optimised to work with MSM (Stefan et al. 2013).

We have implemented novel adaptations to the initialisation, assignment and update *k*-means components. One significant contribution spans all components: each stage

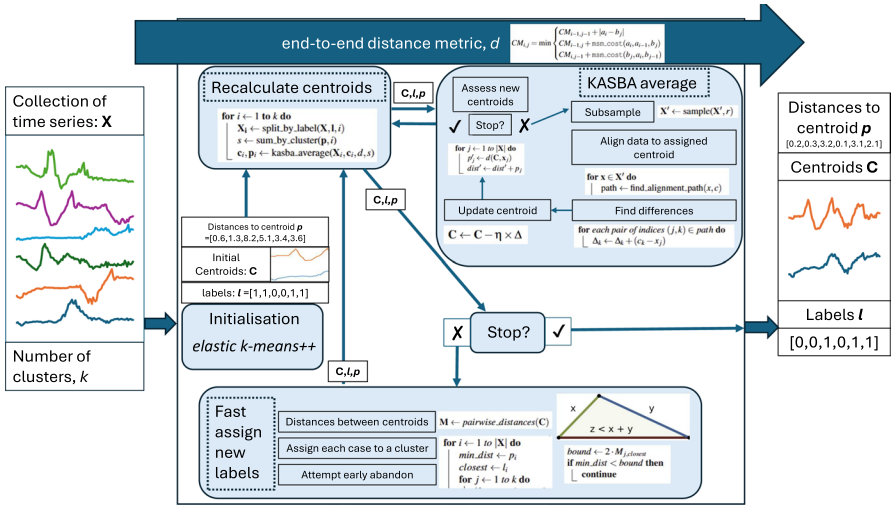



**Fig. 3** Visualisation of the KASBA time series clustering algorithm, described in detail in Algorithm 2

requires a distance function, and KASBA consistently applies the same elastic distance function to each stage. It is not always clear whether this is done in related research (Holder et al. 2024a).

The stages of KASBA, described in Algorithm 2 and visualised in Fig. 3, include initialisation through adapting the k-means++ algorithm (line 2), update through a novel stochastic subgradient barycentre averaging approach to find centroids (line 5) and a fast assignment algorithm that exploits the triangle inequality of the distance measure (line 7) to greatly speed up the algorithm. These operations are described in detail in the subsequent sections. For clarity, we do not include all KASBA parameters in the algorithms. These are described in detail in Sects. 4.2 and 5. We have used standard default parameters and have not performed any tuning. One central component of KASBA is retaining information from each stage to optimise the next iteration. The centroids found in the initialisation (line 2) and calls to recalculate_centroids (line 6) are stored and used in subsequent calls to improve convergence. $\mathrm{p}$ contains the minimum distances to centroids under labelling . These distances may change after centroid recalculation (line 6), and the data in $\mathbf{p}'$ can be used to make the assignment of cases in line 8 more efficient. The assignment returns the new cluster labels and the distances to the assigned centroids.

**Algorithm 2** KASBA (X, $k$, $max\_its$)

**Input:**
**X**: collection of $n$ time series
**k**: number of clusters
**max_its**: maximum number of iterations before the algorithm is forcibly terminated
**Output:**
$\mathbf{C} = [\mathbf{c_1}, \ldots, \mathbf{c_k}]$: collection of $k$ centroids
$\mathbf{l} = [l_1, \ldots, l_n]$ list of initial cluster labels of length $n = |\mathbf{X}|$
$\mathbf{p} = [p_1, \ldots, p_n]$ list of distances to centroids

1 $d \leftarrow$ setup_distance() *configure and return distance function*
2 $\mathbf{C}, \mathbf{l}, \mathbf{p} \leftarrow$ elastic_kmeans_pp$(\mathbf{X}, k, d)$ (Algorithm 3)
3 *continue* $\leftarrow$ **true**
4 *iterations* $\leftarrow \mathbf{1}$
5 **while** *continue* **do**
6 $\quad \mathbf{C}', \mathbf{p}' \leftarrow$ recalculate_centroids$(\mathbf{X}, \mathbf{C}, \mathbf{l}, \mathbf{p}, d, k)$ (Algorithm 4)
7 $\quad \mathbf{C}'$ *contains the new centroids,* $\mathbf{p}'$ *contains the distances to the new centroids*
8 $\quad \mathbf{l}', \mathbf{p}' \leftarrow$ fast_assign$(\mathbf{X}, \mathbf{C}', \mathbf{l}, \mathbf{p}', d, k)$ (Algorithm 7)
9 $\quad$ *iterations* $\leftarrow$ *iterations* $+ 1$
10 $\quad$ **if** $\mathbf{l} = \mathbf{l}' \vee$ *iterations* $=$ *max_its* **then**
11 $\quad\quad$ *continue* $\leftarrow$ **false**
12 $\quad \mathbf{l} \leftarrow \mathbf{l}'$
13 $\quad \mathbf{p} \leftarrow \mathbf{p}'$
14 $\quad \mathbf{C} \leftarrow \mathbf{C}'$
15 **return** $\mathbf{C}, \mathbf{l}, \mathbf{p}$

**Algorithm 3** elastic_kmeans_pp(X, $k$, $d$)

**Input:**
**X**: collection of $n$ time series
**k**: number of clusters
**d**: distance function for two series
**Output:**
$\mathbf{C} = [\mathbf{c_1}, \ldots, \mathbf{c_k}]$: collection of $k$ centroids
$\mathbf{l} = [l_1, \ldots, l_n]$ list of initial cluster labels of length $n = |\mathbf{X}|$
$\mathbf{p} = [p_1, \ldots, p_n]$ list of distances of each time series to its nearest centroid

$a \leftarrow rand(|\mathbf{X}|)$ *Select a random initial centroid index a from* $\{1, 2, \ldots, |X|\}$
$\mathbf{c_1} \leftarrow \mathbf{X}_a$
**for** $i \leftarrow 1$ **to** $|\mathbf{X}|$ **do**
  $p_i \leftarrow d(\mathbf{x_i}, \mathbf{c_1})$
  $l_i \leftarrow 1$
**for** $i \leftarrow 2$ **to** $k$ **do**
  $s = sum(p)$ *Find m, the distribution for sampling the next centroid*
  **for** $j = 1$ **to** $|X|$ **do**
    $m_j \leftarrow p_j / s$
  $a \sim m$ *Randomly select the next centroid index using distribution m*
  $\mathbf{c_i} \leftarrow \mathbf{X}_a$
  **for** $j = 1$ **to** $n$ **do**
    $t \leftarrow d(\mathbf{x_j}, \mathbf{c_i})$
    **if** $t < p_j$ **then**
      $l_j \leftarrow i$ *Reassign to new centroid*
      $p_j \leftarrow t$
**return** $\mathbf{C}, \mathbf{l}, \mathbf{p}$

### 3.1 Elastic k-means++ initialisation

*k*-means++ (Arthur and Vassilvitskii 2007) is one of the most popular and widely recommended methods for cluster initialisation in traditional clustering literature (Celebi et al. 2013). A commonly used alternative to *k*-means++ is to create random initial centroids, either through Forgy initialisation (choose random cases) (Forgy 1965) or random initialisation (create synthetic random cases) (Lloyd 1982). This is used with a fixed number (often 10) restarts of the entire clustering algorithm, with the best clustering to be the restart that minimises some unsupervised objective function the most. For *k*-means this is the SSE (Eq. 1). In tabular clustering, *k*-means++ has been shown to outperform both Forgy and random initialisation with restarts, and is also significantly faster (Celebi et al. 2013). However, for TSCL random or Forgy initialisation with 10 restarts has been shown

to outperform the traditional *k*-means++ using the Euclidean distance (Holder et al. 2024a).

Restarting *k*-means 10 times with different initial centres is computationally expensive. Moreover, the random placement of initial centroids significantly increases the time required for the algorithm to converge. This issue is even more pronounced in TSCL due to the added computational complexity of elastic distance functions. To address this, we propose the elastic *k*-means++ algorithm, which extends *k*-means++ to work with any elastic distance. Algorithm 3 provides a detailed outline of this approach.

In Sect. 6, we demonstrate that the elastic *k*-means++ algorithm produces significantly better initial centroids than *k*-means++ with Euclidean distance. Furthermore, it outperforms both the Forgy method and random initialisation with 10 restarts for TSCL. Notably, it also leads to significantly faster convergence.

Algorithm 3 begins by randomly selecting the first prototype from the training data (line 1) and calculating distances to this initial centroid (lines 3–5). Subsequent prototypes are chosen based on a probability proportional to their distance from the nearest already-selected prototype (lines 7–9). In standard *k*-means++, Euclidean distance is used for these calculations. However, experiments have shown that for TSCL, using Euclidean distance performs significantly worse than employing restart schemes (Holder et al. 2024a).

One of the key computational improvements we introduce for KASBA is the adaptation of *k*-means++ to elastic distances. Specifically, we incorporate the global end-to-end distance function into the *k*-means++ algorithm (lines 4 and 13). For clarity, minor optimisations are omitted from the pseudocode (lines 11–15). For instance, if a series has already been selected as a centroid, its distance to other centroids does not need to be recalculated. Additionally, the algorithm outputs the distances to the initial centroids, eliminating the need for recalculating these during the update stage.

### 3.2 Update: recalculating centroids

Finding centroids/prototypes with arithmetic means whilst assigning with an elastic distance creates poor clusterings (Holder et al. 2024b). This is because using an averaging technique that does not minimise the SSE for the distance used can lead to unexpected convergence. DBA and similar barycentre averaging techniques (BA) have been proposed to minimise time series specific distance functions (see Sect. 2.2). However, whilst they significantly improve cluster quality compared to standard *k*-means, they are very slow. The BA algorithm is iterative and requires a distance calculation for all cluster members at each iteration. This can take a significant amount of time, particularly if there are a large number of instances. One approach to reducing the number of iterations is to use a search heuristic such as stochastic subgradient descent in the internal fitting (described in Sect. 2.2.3).

**Algorithm 4** recalculate_centroids(X, C, l, p, $d$, $k$)

**Input:** **X**: collection of time series
**C**: current cluster centroids
**l**: current cluster labels
**p**: current distances to centroid
**d**: distance function
**k**: number of clusters
**Output:** **C**: list of new cluster centroids
**p**: list of distances of each time series to its nearest centroid

**for** $i \leftarrow 1$ **to** $k$ **do**
  $\mathbf{X_i} \leftarrow$ split_by_label($\mathbf{X}, \mathbf{l}, i$) *find cluster i*
  $s \leftarrow$ sum_by_cluster($\mathbf{p}, \mathbf{X_i}$) *find current sum of distances to centroid*
  $\mathbf{c}_i, \mathbf{p}_i \leftarrow$ kasba_average($\mathbf{X}_i, \mathbf{c}_i, d, s$) (Algorithm 5)
  $\mathbf{p} \leftarrow$ merge($\mathbf{p}, \mathbf{p}_i$) *overwrite old p values whilst maintaining indices*
return $\mathbf{C}, \mathbf{p}$

**Algorithm 5** kasba_average(X, c, $d$, $dist$, $s$)

**Input:**
**X**: collection of time series in one cluster
**C**: current centroid
**d**: distance function
**dist**: sum of distances to old centroid
**max_its**: maximum number of iterations before the algorithm is forcibly terminated
**s**: proportion of the collection to use each iteration after the first
**Output:**
**C**: new centroid for **X**
**p**: distance from each series to new centroid **C**

1 $r \leftarrow \min(|X|, \max(10, (s \times |X|)))$ *Proportion to sample*
2 $\eta \leftarrow 0.05$ *Gradient descent learning rate*
3 $max\_iters \leftarrow 50$
4 **for** $i \leftarrow 1$ *to* $max\_iters$ **do**
5 $\quad \mathbf{X}' \leftarrow \mathbf{X}$
6 $\quad$ **if** $i > 1$ **then**
7 $\quad\quad \mathbf{X}' \leftarrow \text{sample}(\mathbf{X}', r)$
8 $\quad \mathbf{C}' \leftarrow \text{kasba\_update\_centroid}(\mathbf{X}', \mathbf{C}, d, \eta)$ (Algorithm 6)
9 $\quad dist' \leftarrow 0$
10 $\quad \mathbf{p}' = []$ *New distances to centroid*
11 $\quad$ **for** $j \leftarrow 1$ *to* $|\mathbf{X}|$ **do**
12 $\quad\quad p'_j \leftarrow d(\mathbf{C}, \mathbf{x}_j)$
13 $\quad\quad dist' \leftarrow dist' + p_j$
14 $\quad$ **if** $dist \leq dist'$ **then**
15 $\quad\quad$ **return C**, **p** *Current centroid no better than previous, return previous*
16 $\quad dist \leftarrow dist'$ *Update* $dist, \mathbf{p}, \eta$ *and* **C** *for the next iteration*
17 $\quad \mathbf{p} \leftarrow \mathbf{p}'$
18 $\quad \eta = 0.05 \cdot e^{-0.1 \cdot i}$
19 $\quad \mathbf{c} \leftarrow \mathbf{c}'$
20 **return c**, **p**

Our update function recalculate_centroids is described in Algorithm 4. This is called from line 6 of KASBA (Algorithm 2). recalculate_centroids simply calls kasba_average (Algorithm 5) on each current cluster. Algorithm 5 is passed the current centroid $c_i$. and the current sum of distances $s$. These are used to initialise the search. It returns the new centroid and $\mathbf{p}'$, the distance to the new centroid for the current assignment . These values are retained, overwriting the previous distances to the old centroids (line 5). This is important to optimise the assignment stage, described in Sect. 3.3.

Function kasba_average (Algorithm 5) is a barycentre stochastic gradient descent inspired by the version described in Schultz and Jain (2018). It is adapted to work

with any elastic distance, uses the previous centroid and sum of distances as the initial starting point, introduces a subsampling randomisation to reduce the number of distance calls each epoch and uses a learning rate with exponential decay.

Algorithm 5 is presented with the hard coded parameters we used in experimentation for clarity. On each iteration except the first, a proportion $s$ of the series are selected (line 5-7) and the centroid is updated with these series (line 8) using function kasba_update_centroid (Algorithm 6). The algorithm iterates until either the total distances to the centroid does not change (lines 11-15), or $max\_iters$ iterations are performed. Initialising the search with the centroid c from the previous round and the sum of distances to this centroid (*dist*) dramatically reduces the number of epochs. However, since *dist* is calculated over all cluster members, we do not subsample on the first iteration to avoid premature convergence. We experimentally assess these changes in Sect. 6.

The batch update function kasba_update_centroid (Algorithm 6) aligns series in the subsample with the current centroid (line 2), then incrementally adjusts the centroid towards the alignments. It uses a learning rate (line 6) which is constant for the batch.

**Algorithm 6** kasba_update_centroid( $\mathrm{X}, \mathrm{C}, d, \eta$)

**Input:**
**X**: sample of cluster members
**C**: current centroid
η: learning rate
**d**: distance function
**c**: MSM distance cost
**Output: C**: Updated centroid

1 **for x** $\in$ **X do**
2 path $\leftarrow$ find_alignment_path$(x, d, c)$
3 Let Δ be a zero array
4 **for** *each pair of indices* $(j,k) \in$ *path* **do**
5 $\Delta_k \leftarrow \Delta_k + (c_k - x_j)$
6 $\mathbf{C} \leftarrow \mathbf{C} - \eta \times \Delta$
7 **return C**

### 3.3 Assignment of series to clusters

**Algorithm 7** fast_assign(X, C, l, p, $d$)

**Input:** **X**: collection of $n$ time series
**C**: collection of $k$ centroids
**l**: previous cluster labels
**p**: distance to assigned centroid
$d$: distance function
**Output:** **l**: list of new cluster labels **p**: distance from series to their new assigned cluster

```
M ← pairwise_distances(C)
for i ← 1 to |X| do
    min_dist ← p_i                    Distance to current assignment for x_i
    closest ← l_i
    for j ← 1 to k do
        if j == closest then
            continue
        bound ← 2 · M_{j,closest}     Apply triangle inequality, try skip this distance
        if min_dist < bound then
            continue
        temp ← d(x_i, c_j)
        if temp < min_dist then
            min_dist ← temp
            closest ← j
    l_i ← closest
    p_i ← min_dist
return l, p
```

Once new centroids are found, the assignment stage involves calculating the distance from each series to all centroids. This requires $nk$ distance calculations and represents a significant proportion of the time taken when clustering with elastic distances. The distance function we use, MSM, has a property that DTW does not: it is a metric, and hence satisfies the triangle inequality, i.e. if we have three series x, y and z, then

$$d(\mathrm{x}, \mathrm{z}) \leq d(\mathrm{x}, \mathrm{y}) + d(\mathrm{y}, \mathrm{z}).$$

This property means we can exploit ideas proposed in Elkan (2003) that use the triangle inequality to skip some distance calculations. Suppose a series currently belongs to a cluster with centroid $\mathbf{c_a}$ and we want to see if it is closer to centroid $\mathbf{c_b}$. When the triangle inequality holds, it is shown in Elkan (2003) that

$$\text{If } d(\mathbf{c_a}, \mathbf{c_b}) \geq 2d(\mathrm{x}, \mathbf{c_a}) \text{ then } d(\mathrm{x}, \mathbf{c_b}) \geq d(\mathrm{x}, \mathbf{c_a}).$$

This inequality states that if the distance between centroids is twice that between a series and its current centroid, then the distance between the series and the candidate must be greater than the distance to the current centroid. Informally, if a series could move to its current centroid and back in less distance that it takes to move centroid to centroid, it will be further to the new centroid and it can be discounted.

If we store the distance from a series to its centroid, $d(\mathrm{x}, \mathbf{c_a})$ and have calculated the pairwise centroid distance function $d(\mathbf{c_a}, \mathbf{c_b})$ at the start of an assignment stage, we know that we will not need to move $\mathbf{x}$ to a different cluster if the distance between the series and its current centroid is less than half the distance between centroids. We have stored the distance to the new centroids ($\mathbf{p}'$) as a side effect of the recalculation of centroids. The function fast_assign (Algorithm 7) exploits this to assign labels based on centroids. It starts by finding the pairwise distance between the new centroids (line 1). We show in Sect. 6 that this dramatically reduces the number of distance calculations required despite introducing $k \cdot (k+1)/2$ new distance calls at the start.

## 4 Experimental setup

### 4.1 Datasets

We conduct our experiments using the UCR archive. Our focus is on univariate time series, and for all experiments, we utilise 112 of the 128 datasets available in the UCR archive. We exclude datasets that contain series of unequal length or missing values.

In supervised machine learning fields, such as TSC, datasets are typically divided into a training split and a test split. This approach helps prevent overfitting. However, in unsupervised tasks such as TSCL, labels are not provided to the learning algorithm. This means that overfitting is not a concern, and any patterns or structures learned by the model must come from the model's inherent understanding of the training data itself, rather than from predefined labels (Javed et al. 2020). Consequently, a common approach in clustering is to provide the model with all available data during training, since a separate prediction step is not required.

However, there are problems with comparing clustering algorithms on data used in training. Often, clustering forms an unsupervised component of a supervised pipeline for tasks such as channel selection. For example, TSCL has been used as: a component of gesture and handwriting recognition (Jang et al. 2011); cluster-then-forecast frameworks that train models per cluster (Bandara et al. 2020); traffic prediction that profiles detectors before modelling (Zou and Chung 2024); clinical risk pipelines that cluster patient trajectories then score incoming patients (Barnes et al. 2024); and behaviour or fitness profiling that maps new sessions into prelearned pattern types (Tselentis and Papadimitriou 2023). Mixing train and test during clustering would leak information from evaluation data into the prototypes, biasing results and overstating out-of-sample performance. We follow the recent trend in TSCL (Lafabregue et al. 2022b; Holder et al. 2024b) and evaluate on the unseen predefined test

split in the UCR archive. We use the labels only in the cluster evaluation. One such evaluation is to understand how well clusterers can extrapolate meaningful general patterns from the data. A model that performs well on unseen data has probably learned more about the underlying properties of the data, potentially making its clusterings more valuable. As such our primary form of evaluation will focus on training a clusterer on the predefined train split for the UCR archive, followed by evaluation using the predefined test split. However, we acknowledge that a large body of the TSCL literature evaluate by combining the training and test subsets into one large dataset for reasons described above. Hence, we also provide results and evaluation for KASBA using the combined training and test split. We believe this approach showcases clusterers utility for many more use cases and improves the evaluation of TSCL models.

Finally, we z-normalise all datasets prior to clustering. We are aware that some argue "*any improvement resulting from pre-processing (normalisation) should not be attributed to the clustering method itself*" (Javed et al. 2020), others contend that "*in order to make meaningful comparisons between two time series, both must be normalised*" (Rakthanmanon et al. 2013). Therefore, the decision to normalise or not remains an ongoing research question. Ideally, with unlimited time and computational resources, we would run both normalised and un-normalised experiments to compare the results. However, this is not feasible given our constraints, and we are forced to make a choice. Following the recommendation of Keogh and Kasetty (2003) and Rakthanmanon et al. (2013) we choose to normalise our data.

All of our experiments were run on a single thread of an Intel Xeon Gold 6138 processors processor on a shared high-performance computer (HPC) cluster.

### 4.2 Clusterer configuration

All the clusterers we consider in this paper use the parameter $k$ which defines the number of clusters that should be formed. For simplicity we set a value that is equal to the number of unique class labels defined by the UCR archive for a given dataset. $k$-means assumes the number of clusters $k$ is set a priori. There are a range of methods of finding $k$. These often involve iteratively increasing the number of clusters until some stopping condition is met. This can involve some form of elbow finding or unsupervised quality measure, such as the silhouette value (Rousseeuw 1987).

#### 4.2.1 Evaluation

We measure performance on the 112 UCR datasets using four common clustering performance measures. Clustering accuracy (CLACC) is the number of correct predictions divided by the total number of cases. To determine whether a cluster prediction is correct, each cluster has to be assigned to its best matching class value. This can be done naively, taking the maximum accuracy from every permutation of cluster and class value assignment $S_k$ or more efficiently using a combinatorial optimisation algorithm.

$$CLACC(y, \hat{y}) = \max_{s \in S_k} \frac{1}{|y|} \sum_{i=1}^{|y|} \begin{cases} 1, & y_i = s(\hat{y}_i) \\ 0, & \text{otherwise} \end{cases}$$

The Rand index measures the similarity between two clusterings by comparing predictions of pairs of instances with the ground truth. If the predicted values are the same for a pair and the ground truth of the pair is also the same, this is a positive count for the Rand index. The Rand index is popular and simple. However, it is not comparable across problems with different number of clusters. The adjusted Rand index (ARI) compensates for this by adjusting the RI based on the expected scores on a purely random model.

The mutual information is a function that measures the agreement of a clustering and a true labelling, based on entropy. Normalised mutual information (NMI) rescales mutual information onto [0, 1], and adjusted mutual information (AMI) adjusts the MI to account for the class distribution.

We use a range of tools to summarise and compare multiple clusterers on multiple datasets. We compare ranks using an adaptation of the critical difference (CD) diagram (Demšar 2006), replacing the post-hoc Nemenyi test with a pairwise comparison using the Wilcoxon signed-rank tests, and cliques formed using the Holm correction recommended by García and Herrera (2008) and Benavoli et al. (2016). We use $\alpha = 0.05$ for all hypothesis tests. Critical difference diagrams such as those shown in Fig. 4 display the average ranks of estimators and cliques of clusterers as solid lines. A clique represents a group where there is no significant difference between clusterers. CD diagrams are useful, but when presented alone they can mask similarities. We also present summary performance statistics in a heat map proposed in Ismail-Fawaz et al. (2023a) and use scatter plots for pairwise comparison.

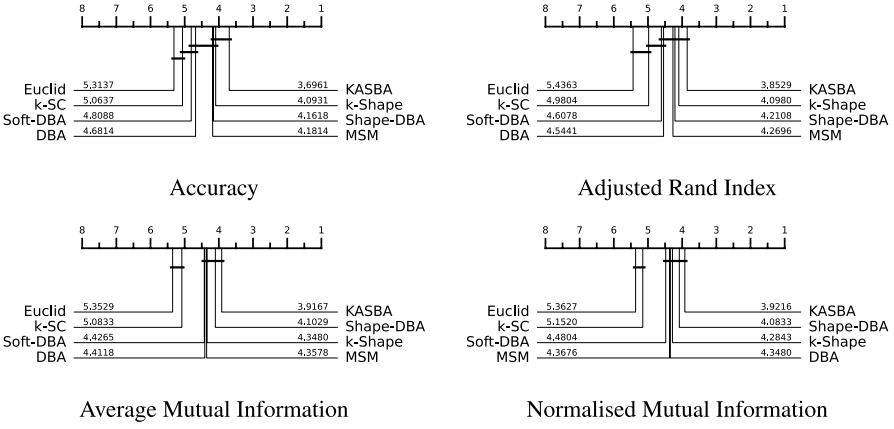



**Fig. 4** KASBA against seven benchmark algorithms on test datasets, averaged over the 102 UCR archive data completed by all algorithms using the default train-test split

### 4.3 Reproducibility

We conducted all experiments using the `aeon` open source time series machine learning toolkit.[1] All results were evaluated with the `tsml-eval` open source package. Additionally, we have provided a notebook that outlines the steps required to run and reproduce our results.[2] Furthermore, we have included all the results reported in this paper as separate CSV files for each evaluation metric alongside the notebook.

## 5 Results

We organise our results into four sections to enhance clarity. Section 5.1 compares KASBA with elastic and shape distance based *k*-means variants. Section 5.2 evaluates KASBA and its variants without optimisations and related medoid-based approaches. Section 5.3 assesses KASBA's performance relative to a broader range of TSCL algorithms. Finally, Sect. 5.4 contains a detailed comparison of KASBA to *k*-shape, the current SOTA over all representations.

For our experiments, we use the clusterers described in Sect. 2.3. All the clusterers assume the value of $k$ is known beforehand and set to the true number of clusters. In all cases, we use the clusterer parameters recommended in the literature. Table 1 summarises the configuration of various distance-based clusterers.

We are missing some results for specific datasets for certain clusterers. In this case the algorithm repeatedly formed empty clusters and failed to converge. We explicitly indicate the number of datasets used in each set of results. A complete list of the missing results, along with the corresponding reasons, is presented in Tables 6 and 7.

**Table 1** Configuration of distance-based clustering algorithms

| Algorithm | Distance | Assignment | Centroid | Parameters |
|---|---|---|---|---|
| KASBA | MSM | KASBA assign | KASBA average | $c = 1.0$ |
| DBA | DTW | Lloyd's assign | DBA | – |
| Shape-DBA | shape-DTW | Lloyd's assign | Shape-DBA | $reach = 15$ |
| Soft-DBA | Soft-DTW | Lloyd's assign | Soft-DBA | $\gamma = 1.0$ |
| MBA | MSM | Lloyd's assign | MBA | $c = 1.0$ |
| Euclid | Euclidean | Lloyd's assign | Arithmetic mean | – |
| MSM | MSM | Lloyd's assign | Arithmetic mean | $c = 1.0$ |
| *k*-Shape | SBD | Lloyd's assign | Shape extraction | – |
| *k*-SC | *k*-SC dist | Lloyd's assign | *k*-SC average | *max_shift* = m |
| PAM-MSM | MSM | PAM-SWAP | MSM medoids | $c = 1.0$ |

All clusterers are run for a maximum of 300 iterations and use *k*-means++ initialisation with the corresponding distance. During the assignment phase, each algorithm uses the distance measure specified in the table, and centroids (or medoids) are computed using the listed centroid computation method

[1] https://github.com/aeon-toolkit/aeon.

[2] https://github.com/time-series-machine-learning/tsml-eval/tree/main/tsml_eval/publications/clustering/kasba/kasba.ipynb.

### 5.1 KASBA against distance based k-means algorithms

The most widely used approach for *k*-means clustering is Lloyd's algorithm (Lloyd 1982). For TSCL, Lloyd's algorithm remains the standard, though it is adapted by modifying the distance measure and averaging technique. For this section we use 7 other clusterers: DBA, Shape-DBA, Soft-DBA, Euclid, MSM, *k*-Shape and *k*-SC. The parameters used are presented in Table 1. Lloyd's algorithm is sensitive to initial centroids. We initialise all clusterers with the *k*-means++ using the corresponding distance measure specific to the algorithm.

Figure 4 shows the average ranks of KASBA against these seven *k*-means based clusterers using the default train/test split. KASBA is the top ranked algorithm for all four evaluation metrics. Three algorithms appear in the top clique for all four measures: KASBA, *k*-Shape and Shape-DBA. KASBA is significantly better than the commonly used benchmarks DBA, Soft-DBA and *k*-SC.

The relative performance in terms of clustering accuracy is summarised in the heat map in Fig. 5 (described in Ismail-Fawaz et al. (2023a)).

Figure 5 indicates that when we order by average values rather than average ranks, Shape-DBA is ranked second, and *k*-Shape third. MSM is significantly worse than KASBA in a pairwise test. The average accuracy rank and log total runtimes are shown together in Fig. 6.

Figure 7 shows the scatter plots for accuracy of KASBA against the algorithms closest to it in design: DBA and and MSM.

Where KASBA really excels is in runtime. Table 2 summarises the runtimes for completing 102 datasets for the train/test experiments.

KASBA completes the 102 problems in under two hours in total, whereas Shape-DBA takes nearly nine days and Soft-DBA would take almost two months (59 days). KASBA is around fifteen times slower than standard *k*-means with Euclidean distance but performs significantly better. KASBA is one to three orders of magnitude faster than all the slowest algorithms on the train/test data (*k*-SC, Shape-DBA, and Soft-DBA). All algorithms show a skew between mean and median. This is due to the larger number of small problems in the archive. However, the ratio of mean to median identifies a problem with some of these algorithms failing to converge in a reason-

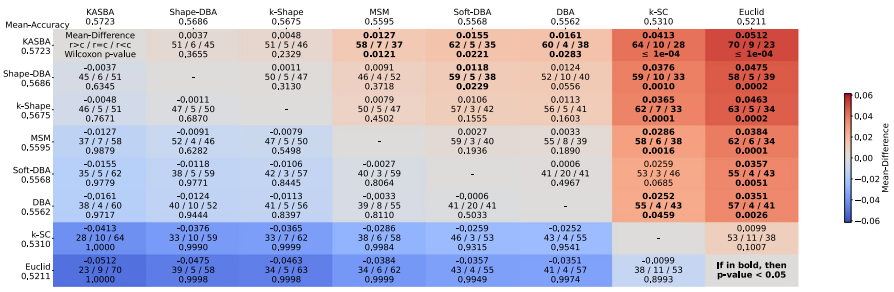



**Fig. 5** Summary performance measures for eight clustering algorithms for clustering accuracy over 102 UCR archive datasets completed by all algorithms using the default train-test split, including mean difference (top), wins/ties/losses (middle) and *p*-value for a p-value for a one-sided Wilcoxon signed-rank test, unadjusted for multiple testing (bottom)

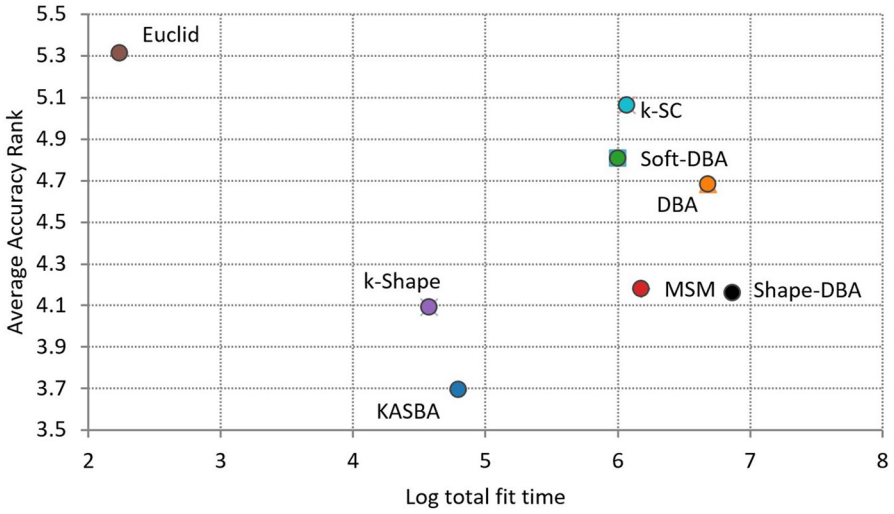
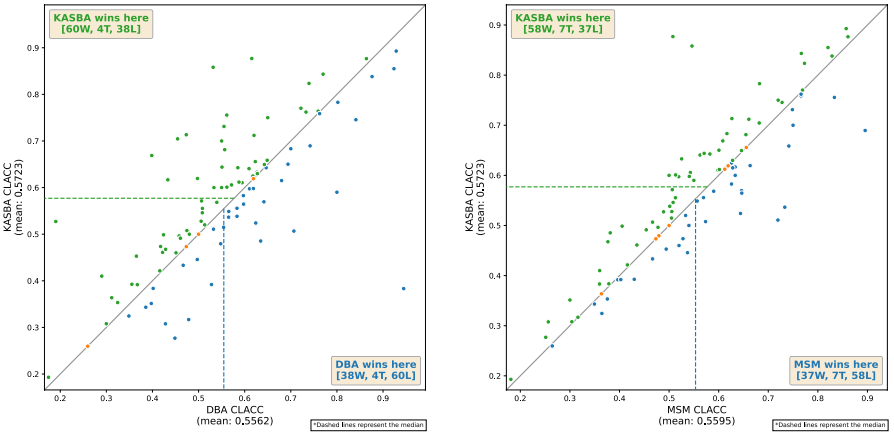



**Fig. 6** Average rank against log runtime for eight clusterers




Accuracy Adjusted Rand Index

**Fig. 7** Scatter plots of KASBA against DBA and MSM for clustering accuracy and ARI

able time. The mean for KASBA is about eleven times the median. The mean for MSM is approximately 100 times the median. The single slowest problem for MSM is PigCVP. This is a strange dataset: it has 50 classes and only 100 training instances. Clearly, this is unusual and MSM is running for its maximum iterations. Conversely, KASBA converges in under half an hour. This highlights how robust KASBA is. Shape-DBA does well in terms of performance, but is slow and also has convergence issues. Soft-DBA performs poorly in terms of performance and runtime. This is surprising. We also ran it with 10 restarts and it did better, but it was so slow to make it

**Table 2** Summary of training times for eight clusterers on 102 UCR datasets completed

Timings are for sequential training runs for 102 datasets that all algorithms completed within the seven day limit

| Estimator | Median (mins) | Mean (mins) | Max (hrs) | Total (hrs) |
|---|---|---|---|---|
| Euclid | 0.0 | 0.0 | 0.0 | 0.1 |
| KASBA | 0.04 | 1.04 | 0.33 | 1.77 |
| *k*-Shape | 0.04 | 0.62 | 0.28 | 1.06 |
| DBA | 5.41 | 79.55 | 22.88 | 135.24 |
| MSM | 0.07 | 25.11 | 19.94 | 42.68 |
| *k*-SC | 0.33 | 19.56 | 8.65 | 33.25 |
| Shape-DBA | 1.30 | 121.92 | 56.93 | 207.27 |
| Soft-DBA | 7.2 | 870.4 | 165.5 | 1421.7 |

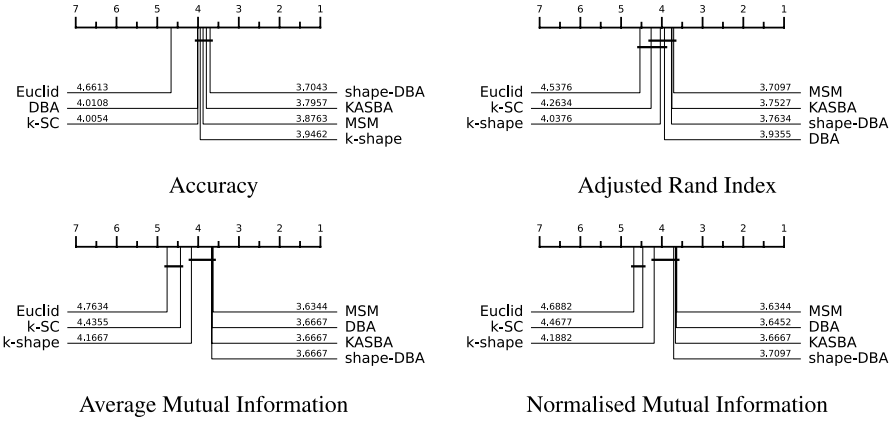



**Fig. 8** KASBA against seven benchmark algorithms using combined train and test data, averaged over the 93 UCR archive datasets completed by all algorithms

impractical. *k*-Shape is not significantly worse than KASBA on any metric and it is marginally faster. We compare KASBA to *k*-Shape in more detail in Sect. 5.4.

We repeat the same experiment evaluating the algorithms on the combined train/test data. Figure 8 shows the results for our four performance statistics. There is now no significant difference between the top five algorithms.

We believe the differences are simply harder to detect using the combined data, and that it gives a potentially skewed view of performance. However, if you did believe there is no difference in these algorithms, then your secondary criteria would probably be runtime. KASBA is 30-50 times faster than DBA and MSM and 400 times faster than Shape-DBA. Runtimes are obviously implementation and hardware dependent but *k*-Shape and KASBA are orders of magnitude faster than the others. There is little to split the two in terms of performance on data used to train the algorithm. We compare KASBA to *k*-Shape in more detail in Sect. 5.4.

### 5.2 KASBA against other partitional clusterers

KASBA is an extension of the MSM with Barycentre Averaging (MBA) algorithm first presented in Holder et al. (2023b). As discussed in Sect. 2.3, the original MBA clusterer followed Lloyd's algorithm, employing the MSM distance metric and the

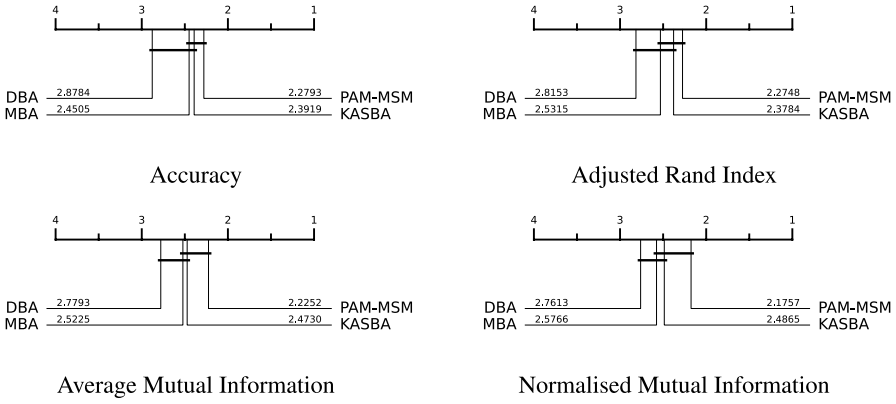



**Fig. 9** KASBA against *k*-means with DTW barycentre averaging (DBA), MSM barycentre averaging (MBA) and Partitioning Around the Medoid with MSM (pam-msm), ranks averaged over the 111 UCR archive data completed by all algorithms using the default train-test split

**Table 3** Summary of training times for four MSM based clustering algorithms on 111 UCR datasets completed

| Estimator | Median (mins) | Mean (mins) | Max (hrs) | Total (hrs) |
|---|---|---|---|---|
| KASBA | 0.1 | 3.0 | 1.2 | 5.6 |
| MBA | 0.0 | 32.2 | 13.7 | 60.2 |
| DBA | 1.2 | 55.0 | 22.9 | 102.7 |
| PAM-MSM | 0.0 | 228.7 | 234.3 | 426.9 |

Timings are for sequential training runs for all 111 datasets

MSM elastic barycentre average. Additionally, it utilised Forgy initialisation with 10 restarts.

Restarting 10 times obviously makes the algorithm 10 times slower. For our experiments with an MBA clusterer, we replace the initialisation algorithm with the elastic MSM *k*-means++ algorithm. This modification ensures that any improvements in clustering performance or runtime arise independently of the initialisation method.

One of the primary motivations of KASBA was to achieve similar clustering performance to MBA while significantly reducing the runtime. An effective elastic distance based alternative to *k*-means that circumvents the challenges of averaging is the medoid-based PAM algorithm (see Sect. 2.3). When combined with MSM, PAM has been shown to be a highly effective clustering approach (Holder et al. 2023a).

Figure 9 shows the performance of the three MSM variants, with DBA included for reference. KASBA is not significantly worse than the other two MSM based clusterers, but it is significantly faster. Table 3 summarises the runtimes. KASBA is 10 to 75 times faster. PAM requires the calculation of the pairwise distance matrix prior to clustering, which can add a huge time and memory cost for large problems; PAM-MSM takes seven days to cluster the largest problem in the archive, Crop. If we combine the train and test set, CROP has 240,000 unique time series. KASBA completes the clustering in minutes. PAM-MSM did not finish within our runtime limit. PAM would require 288,000,000 unique elastic distance function calls and need over 2GB of memory to store the matrix.

### 5.3 KASBA against other TSCL algorithms

For context we compare KASBA to three clustering algorithms that are not distance based. We have selected these particular algorithms because they were published in good venues and we have implementations of them readily available.

Figure 10 shows that KASBA is significantly better or not significantly worse using all four measures, further supporting its case as a strong benchmark.

Results for 300 different deep learning clustering algorithms on the same UCR datasets we use were presented in Lafabregue et al. (2022a). We have not recreated these results due to computational constraints. However, the associated repository[3] provides NMI results for over 300 different clustering algorithms on the same default train/test splits of the UCR datasets. These are not directly comparable, since they are averaged over five runs and there may be other experimental differences. Despite this, they can give some indication of relative performance. The best deep learning approach was a convolutional neural network with joint pretext loss and without clustering loss (key in their results is res_cnn_joint_None). It achieved an average NMI of 0.3292 over the 112 problems we use. KASBA average NMI is 0.3135. The deep learning NMI are much higher on two datasets, skewing the average. Overall, KASBA was better on 57 problems, the best deep learner better on 55. Given selecting the best of 300 is likely to bias the results, we believe this demonstrates that KASBA is competitive with deep learning TSCL algorithms while being significantly faster.

### 5.4 KASBA and *k*-Shape

Since we released KASBA in the `aeon` toolkit, it has been used in a comparative study of clusterers published at the VLDB conference (Paparrizos and Bogireddy

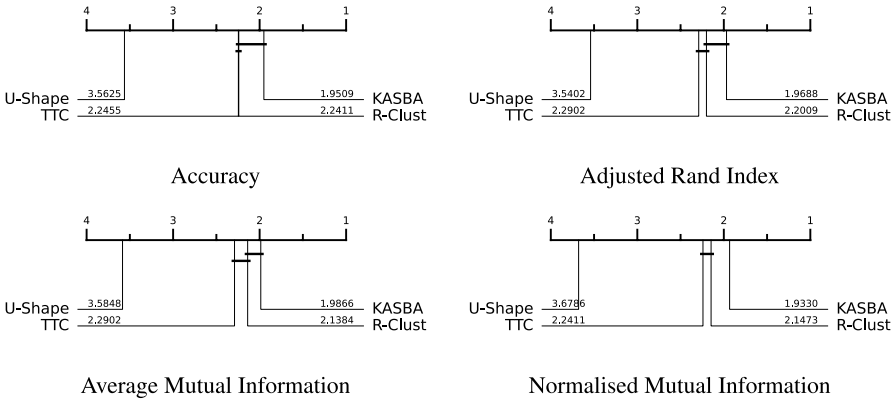



**Fig. 10** KASBA against alternative TSCL algorithms U-Shapelets, R-Clustering and TTC, with ranks averaged over the 112 UCR archive data completed by all algorithms using the default test-train split

[3] https://github.com/blafabregue/TimeSeriesDeepClustering.

2025). This paper reports that *k*-Shape significantly outperforms KASBA using the unadjusted Rand index measure.

Given the results presented in Paparrizos and Bogireddy (2025) give a different impression to those in Fig. 8, it merits further investigation as to the cause. Firstly, it is important to observe that all experiments in Paparrizos and Bogireddy (2025) were conducted on the combined train/test datasets. Secondly, their experimental set up has differences to ours, such as not using a restart algorithm (we use k-means++) and setting the maximum number of iterations to 100 (we use 300). Thirdly, we do not use the unadjusted Rand index in our analysis. Finally, we do not perform the post-hoc Nemenyi test for difference in ranks (for reasons discussed in Sect. 4.2).

We repeat their experiment as faithfully as we can. We compare *k*-AVG, *k*-Shape and KASBA using a maximum of 100 iterations using the original *k*-Shape implementation. We normalise all series (as we have done in all experiments) and average over 10 runs on the combined train/test data with different random seeds. We do not include the 16 missing value and unequal length data to avoid and bias caused by preprocessing differences.

Figure 11 shows the relative performance of the three algorithms with four performance measures, including unadjusted Rand index.

Our results for unadjusted Rand index (Fig. 11 top left) closely match those in Fig. 2a of Paparrizos and Bogireddy (2025). The critical difference between *k*-Shape and KASBA is 0.25. This is not significant with the Nemenyi test, but it is with the Wilcoxon signed rank test. However, there is no difference when comparing adjusted Rand index, clustering accuracy or mutual information. It is widely accepted that unadjusted Rand index (RI) is a poor performance measure for detecting differences in relative performance of clusterers (Hubert and Arabie 1985; Warrens and Van der Hoef 2019). Our results for accuracy, ARI and NMI closely tally with the results presented on combined train/test in Fig. 8. It also reflects our train/test results presented in Fig. 4 that show there is no significant difference between KASBA and *k*-Shape. We believe there is no overall significant difference between *k*-Shape and KASBA. Table 2 indicates that *k*-Shape is faster than KASBA. So why do we need KASBA?

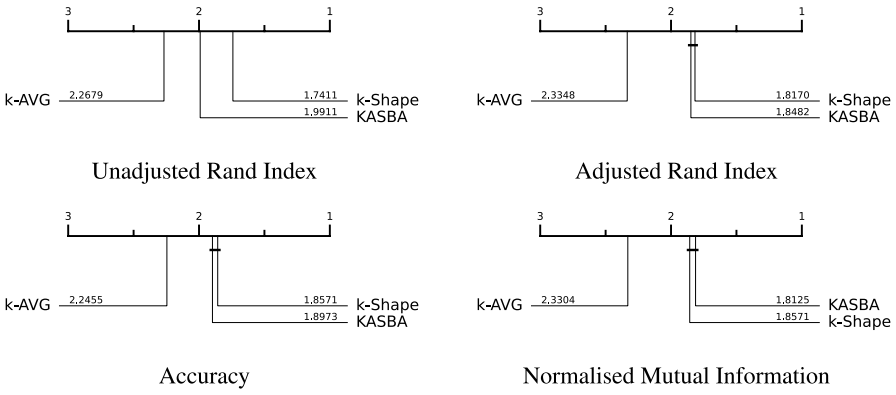



**Fig. 11** Results for three clusterers used in Fig. 2 of Paparrizos and Bogireddy (2025) using four performance metrics

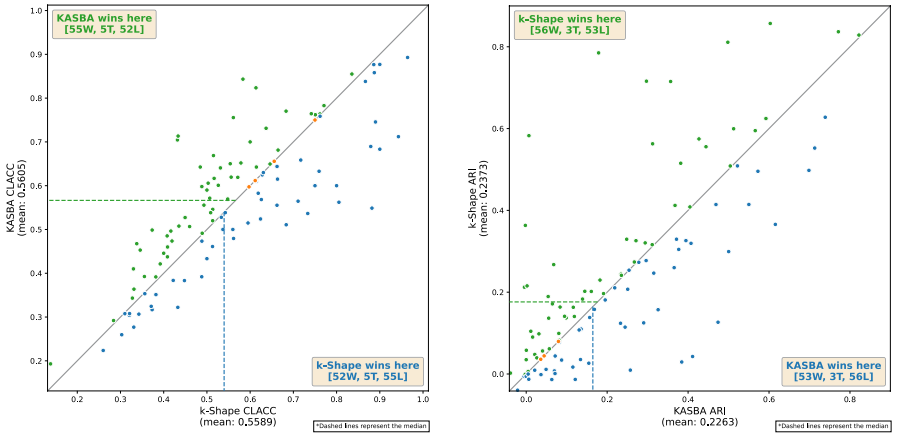



**Fig. 12** Scatter plots of KASBA against *k*-Shape for clustering accuracy and ARI

Firstly, they are built on fundamentally different types of distance function. KASBA is based on elastic distances, *k*-Shape on global alignment, and they will detect different kinds of similarity between series. Figure 12 shows the two algorithms have a wide dispersal in quality of clustering.

There is also significant divergence in runtime, particularly on larger problems, where it is more of an issue. For example, *k*-Shape is over 30 times faster on the dataset SemgHandGenderCh2, whereas KASBA is 30 times faster on Crop. Experiments indicate that KASBA is faster with large collections of shorter series. This would seem reasonable: KASBA has a slower worse case distance function, but has fast convergence. To test this hypothesis, and quantify scalability, we compare KASBA and *k*-Shape on datasets in the new MONSTER archive (Dempster et al. 2025). MONSTER is an archive of 27 large time series classification problems. Some are very large (over one million) collections of relatively short series. Others are collections of thousands of longer series (over 1000 time points). Only three clusterers in our evaluation could possibly be trained on these data within our time constraints: standard *k*-means, KASBA and *k*-Shape. We trained these clusterers on MONSTER, using a single channel for multivariate problems, to assess scalability. Euclid finished 23, KASBA 18 and *k*-Shape 14 within our time constraints. There was divergence between the problems completed and the time taken between KASBA and *k*-Shape. For example, the Traffic data has 1,460, 968 time series of length 24. KASBA and *k*-Shape have similar accuracy on this data, but KASBA fits the model in 11 min, whereas *k*-Shape takes 11 h (and Euclid in 5 mins). Conversely, the CornellWhaleChallenge data has 30,000 series of length 4,000. KASBA takes 40 h to converge, rather than 1 h for *k*-Shape (and under a minute for Euclid). Table 4 summarises the data characteristics and the time to fit each algorithm for the MONSTER data that both *k*-Shape and KASBA completed.

This supports our hypothesis that KASBA is faster for large collections of shorter series, whereas *k*-Shape is faster for longer series such as CornellWhaleChallenge. These results are only indicative since run time is implementation and hardware

**Table 4** Summary of the time to fit (in minutes) clustering algorithms for the MONSTER datasets on the default train split

| Dataset | Instances | Length | Channels | Classes | Euclid | KASBA | k-Shape |
|---|---|---|---|---|---|---|---|
| CornellWhaleChallenge | 30,000 | 4,000 | 1 | 2 | 0.1 | 2373.6 | 51.4 |
| DreamerA | 170,246 | 256 | 14 | 2 | 1.5 | 27.3 | 53.1 |
| DreamerV | 170,246 | 256 | 14 | 2 | 1.65 | 27.3 | 53.6 |
| InsectSound | 50,000 | 600 | 1 | 10 | 0.3 | 137.4 | 323.1 |
| LakeIce | 129,280 | 161 | 1 | 3 | 0.3 | 5.1 | 204.0 |
| Opportunity | 17,386 | 100 | 113 | 5 | 0.1 | 0.6 | 9.9 |
| PAMAP2 | 38,856 | 100 | 52 | 12 | 0.1 | 1.9 | 18.1 |
| STEW | 28,512 | 256 | 14 | 2 | 0.1 | 4.4 | 9.9 |
| Tiselac | 99,687 | 23 | 10 | 9 | 0.6 | 1.0 | 119.8 |
| Traffic | 1,460,968 | 24 | 1 | 7 | 5.2 | 11.7 | 703.8 |

dependent: we have run experiments with a single thread on a CPU and both *k*-Shape and KASBA could be made much faster through threading and/or using a GPU implementation. We believe KASBA complements *k*-Shape: they find different kinds of clusters and are efficient on different types of data. Clustering quality is ultimately problem specific and subjective, and the practitioner should choose the tool they feel best suited to the task.

## 6 Analysis of KASBA

To investigate the applicability of KASBA and analyse the importance of its design components, we conduct additional comparisons under controlled conditions. This section explores the impact of various factors, including initialisation techniques, distance measures, barycentre initialisation methods, average subset sizes for barycentre computation, and the utilisation of the metric property in conjunction with KASBA. Our goal is to highlight the importance of each component.

### 6.1 Exploratory analysis

Clustering is most commonly used as an exploratory tool, where a training and test split typically does not exist. When considering the combined train/test split we observe that similar algorithms such as DBA and Shape-DBA are consistently in the top clique with KASBA. However, Soft-DBA underperformed compared to our expectations. In previous studies (Holder et al. 2024a), Soft-DBA, when paired with Forgy initialisation and 10 restarts, significantly outperformed the current state-of-the-art algorithms (e.g., DBA and Shape-DBA). However, when using elastic *k*-means++, Soft-DBA appears to perform significantly worse.

Initially, we hypothesised that the $\gamma$ parameter might explain this behaviour. The original Soft-DBA paper (Cuturi and Blondel 2017) states: "low smoothing parameter $\gamma$ yields barycentres that are spurious. On the other hand, a descent on the Soft-DTW loss with sufficiently high $\gamma$ converges to a reasonable solution." We used $\gamma = 1.0$ because it is the value most likely to ensure convergence (other values were shown to fail to converge on multiple datasets (Cuturi and Blondel 2017)). To ver-

ify this hypothesis, we conducted additional experiments with a different $\gamma$ value: $\gamma = 0.001$. However, we found no significant differences in overall performance across the tested $\gamma$ values.

To better understand and explain results, such as the poor performance of Soft-DBA, and to highlight the exploratory power of KASBA, we compared various algorithms on the classic GunPoint problem (Ye and Keogh 2011). This dataset records the movement along the x-axis of two actors performing two actions: drawing a gun from a holster at their belt and mimicking the same action without a gun, instead pointing their finger. One time series represents one actor doing one action.

We applied various averaging methods to instances from each class using techniques from standard *k*-means (mean average), DBA, KASBA, Soft-DBA, and Shape-DBA. The central column of Fig. 13 presents the prototype for each class overlaid: Class 1 (gun) is shown in blue, and Class 2 (point) in red. The mean average produces similar shapes for both classes, with a notable difference between points 20–50, where the gun is being drawn from the holster in Class 1. The DBA prototypes exhibit peaks in the middle, albeit at different positions for each class, with Class 1 displaying a higher plateau. However, these prototypes are challenging to interpret in terms of motion. Soft-DBA appears similar to the mean average when $\gamma = 1.0$, but resembles DBA when $\gamma = 0.01$. Shape-DBA, on the other hand, inverts the peak visible in both DBA and Soft-DBA, resulting in distinct centroids.

KASBA highlights distinctive variations around time points 40–60 and 100–120, which correspond to the moments when the hand is reaching for the gun and drawing it from the holster. These intervals are known to be the most discriminative for this dataset (Ye and Keogh 2011). This example demonstrates how the centroids produced by KASBA can guide exploratory analysis toward key areas of interest in the time series. We also extracted the actual centroids from a clustering run for these algorithms, as shown on the right in Fig. 13. While all centroids resemble the class prototypes depicted in the middle of Fig. 13, they appear more attenuated. Notably, KASBA preserves the variation around the gun-drawing motion, maintaining its discriminatory power.

### 6.2 Ablative study

We introduce several novel features into the *k*-means clustering process to improve performance, hasten convergence and/or reduce runtime. In this section we assess the impact of each component on KASBA performance in terms of quality and speed of clustering.

#### 6.2.1 Using the elastic distance in kmeans++ initialisation

KASBA introduces elastic kmeans++ initialisation, described in Algorithm 3. We have compared this to the standard approach of using Forgy with 10 restarts. The scatter plots for the ARI are shown Fig. 14a. There is no significant difference in performance, but random restarts is obviously an order of magnitude slower. We also tried restarting kmeans++ 10 times. Figure 14b shows there is a small benefit from

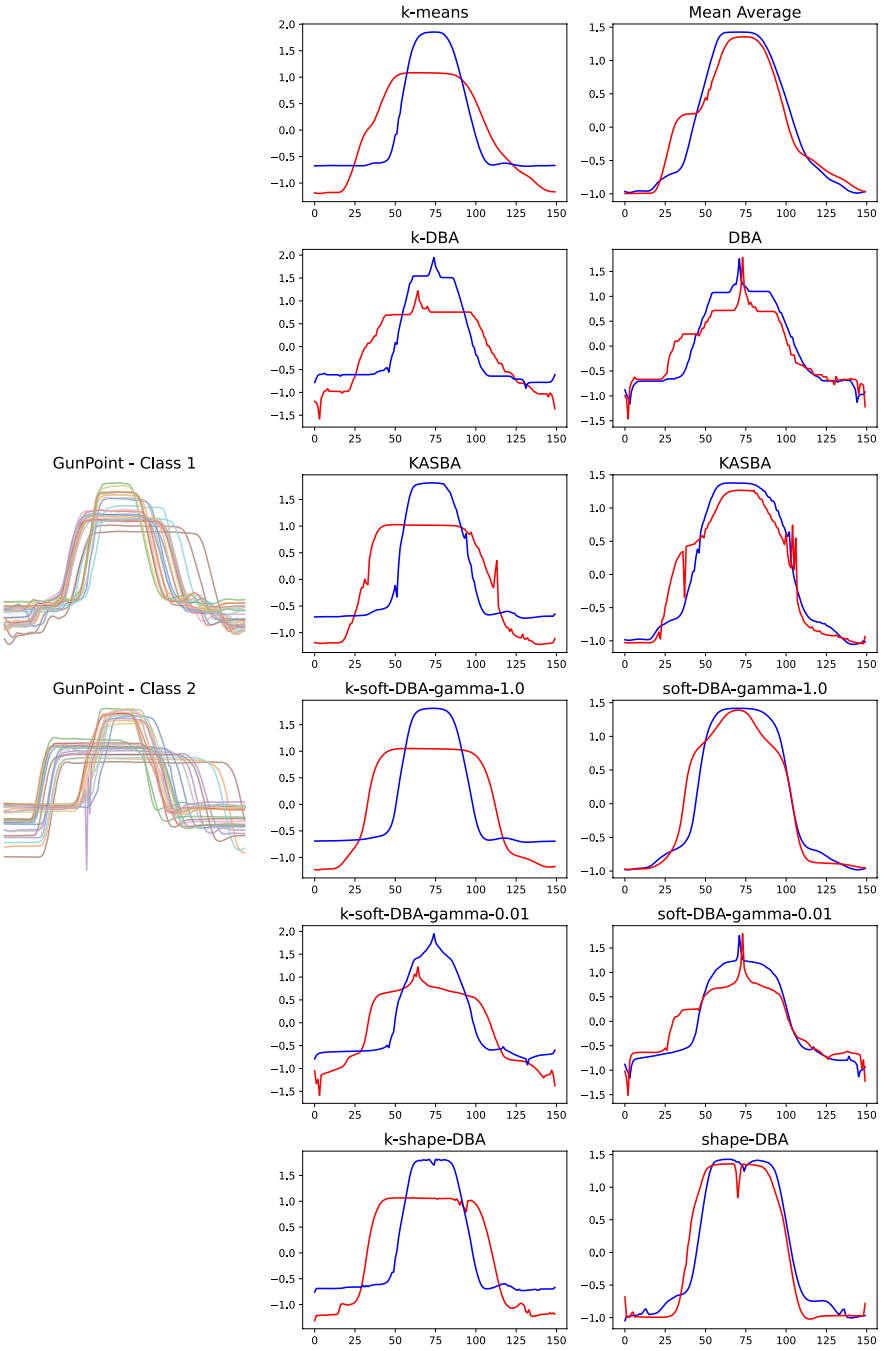



**Fig. 13** Different prototypes for GunPoint data. The left hand plots show the data, the middle column shows the prototypes formed using the class labels when using different averaging techniques. The column on the right shows the actual cluster centroids found by the respective algorithms

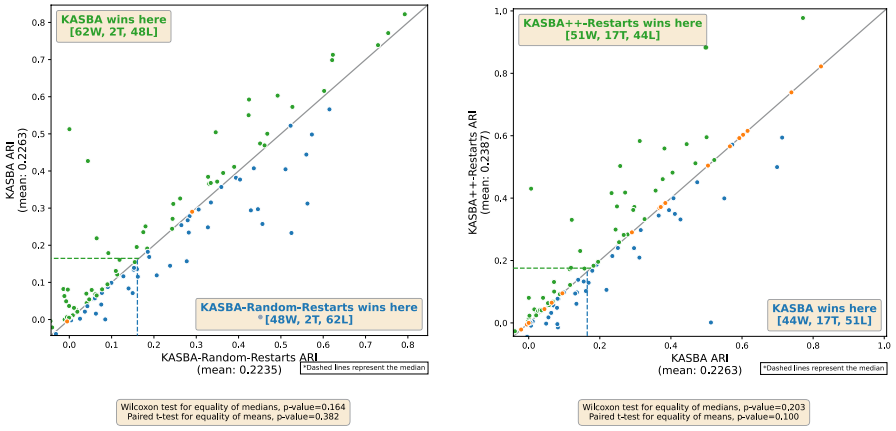



**Fig. 14** Scatter plot of ARI for single run KASBA against 10 random restarts and 10 kmeans++ restarts

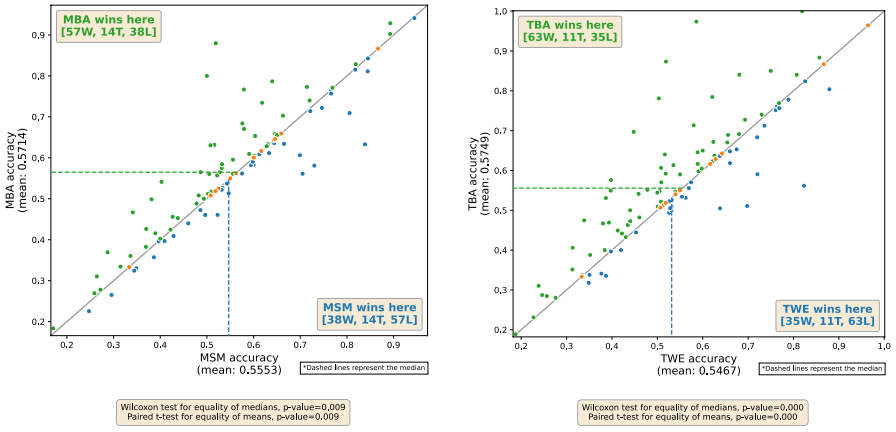



**Fig. 15** Scatter plots of MSM/TWE with barycentre averaging (MBA/TBA) against *k*-means using MSM/TWE for assignment only (MSM/TWE) over 109 UCR archive

restarting kmeans++, but it is not significant and we feel not worth the extra computation. It is easy to configure KASBA to do so if desired.

### 6.2.2 Elastic barycentre averaging

The *k*-means algorithm uses distances in both its update and assignment stages. By contrast, DBA employs DTW in both stages, consistently outperforming *k*-means that uses DTW for assignment only (Petitjean et al. 2011), a result corroborated in Holder et al. (2024b).

The idea of incorporating MSM for barycentre averaging (MBA), alongside its use for assignment, was introduced by Holder et al. (2023b). While our results focus on the MSM distance, an alternative elastic distance function that is also a metric

is the Time-Warp Edit (TWE) distance (Marteau 2009). We compare KASBA with MSM and TWE in more detail in Sect. 6.2.6. Elastic barycentre averaging can also be applied with TWE, yielding a clusterer we denote TBA. We refer to *k*-means algorithms that use an elastic distance solely during assignment as DTW, TWE, or MSM, depending on the specific distance used. Figure 15 presents scatter plots comparing MBA against MSM and TBA against TWE, demonstrating that both barycentre-based approaches outperform *k*-means with elastic distance applied only during assignment.

MBA and TWE also significantly outperform DTW and DBA, as illustrated in Fig. 16, which displays the critical difference diagram for all eight clusterers. TBA, MBA, KASBA-TWE and KASBA-MSM consistently rank in the top clique across all evaluation metrics. Specifically, for AMI, NMI, and ARI, they are significantly better than MSM, DBA, and TWE. While MBA and TBA consistently share the top clique with KASBA-MSM and KASBA-TWE (see Sect. 5), they are significantly slower than KASBA.

### 6.2.3 Seeding of the stochastic subgradient descent with the previous centroid

We use the centroid found during the last iteration (or at initialisation) as a starting point for the search for a new centroid rather than the arithmetic mean proposed in Petitjean et al. (2016) (see Algorithm 5 line 8). This refinement is only apparent when you consider the clustering algorithm as a whole process rather than averaging in isolation: it improves both runtime and performance. Seeding makes KASBA take on average 80% of time taken when initialising with the average. This is achieved through making 60% of the distance calls in update (4237 calls compared to 7210). It also results in better clusters. Figure 17 shows the performance of KASBA with and without seeding.

Seeding the centroid allows us to reuse the sum of distances to the centroid (parameter *dist* in Algorithms 5), which saves distance calculations and makes the algorithm

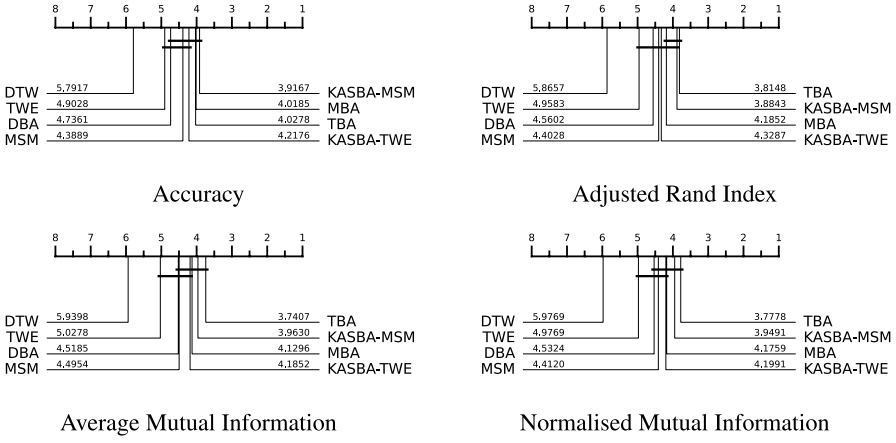



**Fig. 16** Elastic distance based *k*-means with ranks averaged over the 108 UCR archive data completed by all algorithms using the default test-train split

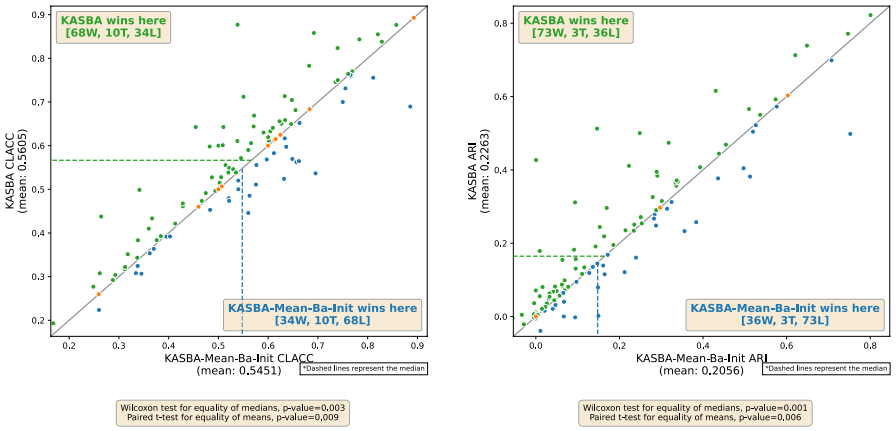



**Fig. 17** Scatter plot for clustering accuracy (CLACC) and Adjusted Rand Index (ARI) of KASBA using centroid seeding to initialise gradient descent (KASBA) and KASBA using arithmetic mean (KASBA-MEAN)

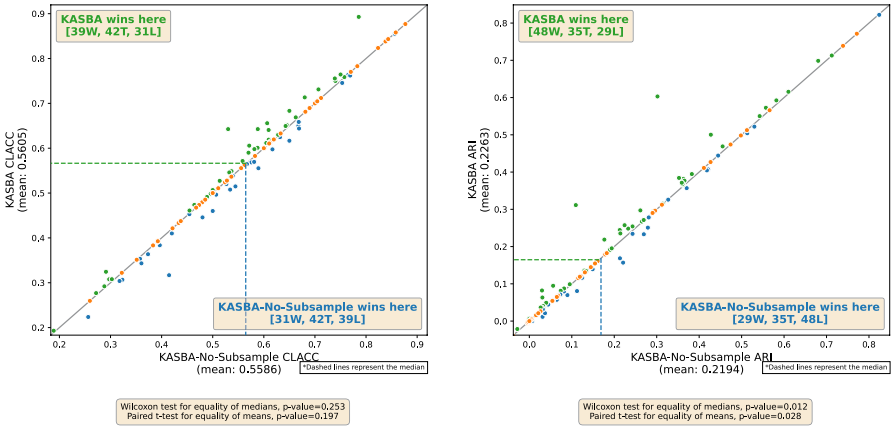



**Fig. 18** Scatter plot of KASBA using subsampling in the gradient descent against using the full cluster membership for each epoch (KASBA-Full)

more stable. To demonstrate this, we reran KASBA where instead of passing the distance sum, we always performed the first epoch of the gradient descent. We found this massively increased the number of epochs it took to reach convergence, rising from an average of 3.4 to 16.5, with a commensurate huge increase in distance calls and hence runtime.

### 6.2.4 Random subset subgradient descent

When finding a barycentre with batch stochastic subgradient descent (Algorithm 5) we take a random subset of series in the cluster to update the centroid on all epochs after the first. This is meant to reduce the number of distance calls on any epoch.

However, it is also meant to decrease the likelihood of premature convergence by injecting some diversity into the process. We would expect subsampling to increase the number of epochs. This is desirable, to counteract the seeding risking premature convergence, but it does also increase the number of distance calls.

To assess the impact of these competing factors, we reran KASBA using the whole dataset on every barycentre iteration. Figure 18 shows the scatter plot of using the full sample on all iterations vs random subsamples. In terms of accuracy (left plot), there is no significant difference. However, sampling produces significantly better ARI (right plot).

Overall, KASBA with subsampling is only marginally faster than using the full dataset in every iteration, even though it makes half as many distance calls per epoch. This indicates that it requires more epochs to converge than the whole dataset every iteration. To confirm this, we reran the experiments and tracked both the number of distance calls and epochs. Subsampling resulted in a total of 85% of the distance calls during updates but required 10% more epochs to reach convergence.

Although we initially hypothesised that subsampling would significantly reduce runtime, the actual reduction was modest. However, it did lead to improved performance. We hypothesise that this improvement arises because using a random subset helps prevent premature convergence.

#### 6.2.5 Triangle inequality in assignment

We exploit the triangle inequality to speed up KASBA in Algorithm 7, lines 8–10. To assess the impact of this, and to verify it has no impact on the actual final clustering, we rerun experiments with the test of whether to skip turned off. Rather than measure runtime, we count the number of distance function calls. This has the added benefit of profiling the algorithm, since the vast majority of the computation is in the distance calls. Overall, most of the distance calculations happen in the assignment stage. KASBA makes on average 38,130 distance calls per problem, and these are split between initialisation: 5780 calls (15%); update: 4237 calls (11%); and assignment: 28113 calls (74%). If we do not apply the triangle inequality in assignment, we perform on average 66,005 distance calls: on average, the triangle inequality results in about 40% of the distance calculations. Overall, KASBA is approximately twice as fast when using the triangle inequality, with identical results.

#### 6.2.6 KASBA sensitivity and functionality

We have used the default configuration of KASBA throughout experiments. These values, summarised in Table 5 were set prior to any experimentation based on the default values from the literature and implementations. The implementation of KASBA in `aeon` is configurable through the constructor and could be tuned with standard scikit-learn mechanisms.

The most significant hyper-parameter is the distance function. This must be a metric in order to exploit the triangle inequality speed up. Currently the only elastic distance metrics are MSM and TWE. Figure 19 shows there is no significant difference in performance of KASBA with the two distance functions.

**Table 5** KASBA parameters and hyper-parameters

| Parameter | Type/default | Description |
|---|---|---|
| `distance` | default `'msm'` | The distance measure use must be a metric |
| `distance_params` | default `c=1.0` | Dictionary of distance parameters |
| `ba_subset_size` | default `0.5` | The proportion of the data to use with BA |
| `initial_step_size` | float, default `0.05` | BA step size for SGD |
| `max_iter` | int, default `300` | Maximum number of iterations for k-means |
| `decay_rate` | float, default `0.1` | decay rate for BA |

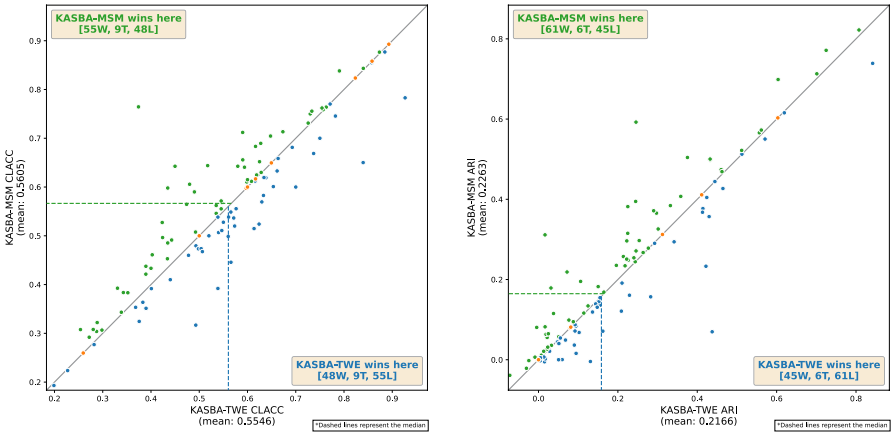



**Fig. 19** Performance of KASBA using TWE and MSM with default parameter settings for clustering accuracy (left) and ARI (right)

The runtimes are very similar. MSM is preferred because it has one parameter, $c$, rather than two, and this parameter is more intuitive: $c$ is the minimum cost of moving off diagonal. The value of $c$ has been set to 1 for all the experiments here and across numerous publications (Holder et al. 2023b, 2024b, 2023a). A diagonal move will be selected if the absolute difference between the two diagonal values is less than $c$. The probability of two independent observations of normally distributed random variables with zero mean and unit variance being greater than one is approximately 0.5. If the series have not been normalised, setting $c$ becomes more critical, but all our experiments are run with normalised series. Experiments with a range of $c$ values, summarised in Fig. 20, show KASBA is robust to different values of $c$. When $c$ is 0 or 2.0, performance is no different to standard $k$-means. For other values, 0.75 is highest ranked, but there is no significant difference between 0.25 to 1.75.

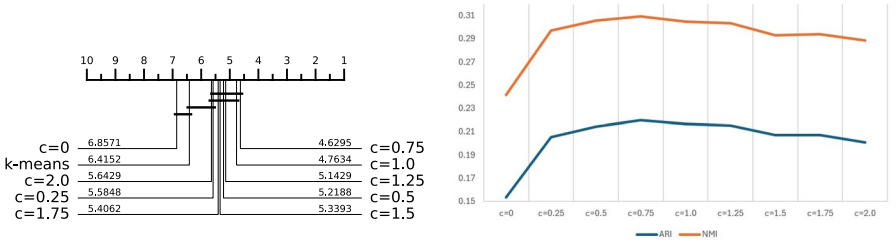



**Fig. 20** KASBA performance for different values of the MSM penalty parameter $c$: left, relative average ranks for accuracy; right, mean ARI and NMI

KASBA can cluster unequal length and multivariate time series. Examples of both use cases are provided on the website.

## 7 Conclusion

We have presented the *k*-means (K) Accelerated (A) Stochastic subgradient (S) Barycentre (B) Average (A) (KASBA) clustering algorithm. Our extensive experimentation demonstrates that KASBA performs at least as well as the current state-of-the-art clusterers, if not better, whilst requiring orders of magnitude less computation: we can cluster all 112 UCR datasets with KASBA in under two hours on an Apple M4 Max CPU.

TSCL research with *k*-means often focusses on a single element of the partitional clustering such as the update/averaging stage. KASBA is designed with bespoke versions of the initialisation, update and assignment phases and their linkage is one of the reasons KASBA is so fast. We employ the same distance measure, MSM, throughout the algorithm, including initialisation with elastic *k*-means++. We propose a novel random subset stochastic subgradient descent averaging algorithm and exploit the metric property of MSM to make assignment faster. We improve convergence by retaining information between iterations.

Our experiments demonstrate that KASBA's performance is comparable to both PAM-MSM and MBA whilst being two orders of magnitude faster. We also recreated results for three non distance based clusterers and compared to published results to the best deep learning algorithm. None of these outperformed KASBA.

Currently, the `aeon` KASBA implementation is not threaded or compatible with GPU. KASBA is a sequential iterative algorithm, but the barycentre averaging step is performed independently on each cluster, and this could easily be threaded. Within a thread, the distance calculations could usefully be performed on a GPU. This is part of our future work.

KASBA has a small memory footprint and is very fast. It has few parameters and works well without tuning. We believe KASBA is a valuable addition to the TSCL canon, a solid benchmark for future research and will be useful for practitioners.

## Appendix A: Missing results

Results missing from the train/test results by clusterer are listed in Table 6. Those missing from the combined results are listed in Table 7. An x means the dataset is missing for the model. If a model is not included as a column it means all 112 datasets completed.

**Table 6** Missing datasets for the train/test split experiments. A total of 14 datasets are excluded. Missing results for MSM were due to repeated empty cluster formation. Missing datasets for Shape-DBA and Soft-DBA were due to the runtime exceeding our 7-day limit

| Dataset | MSM | Shape-DBA | Soft-DBA |
|---|---|---|---|
| EOGHorizontalSignal | | x | x |
| EOGVerticalSignal | | x | x |
| EthanolLevel | | x | x |
| HandOutlines | | x | x |
| InlineSkate | | x | x |
| NonInvasiveFetalECGThorax1 | | x | x |
| NonInvasiveFetalECGThorax2 | | | x |
| Phoneme | | | x |
| PigAirwayPressure | | | x |
| PigArtPressure | x | | |
| PigCVP | x | | |
| UWaveGestureLibraryAll | | | x |
| Total missing | 2 | 6 | 10 |

**Table 7** Missing datasets for the combined train/test split experiments. A total of 19 datasets are excluded. Missing results for MSM and DBA were due to repeated empty cluster formation. Missing datasets for Shape-DBA and *k*-SC were due to the runtime exceeding our 7-day limit

| Dataset | DBA | MSM | k-SC | Shape-DBA |
|---|---|---|---|---|
| CinCECGTorso | | | | x |
| EOGHorizontalSignal | | | | x |
| EthanolLevel | | | | x |
| FordA | | | | x |
| FordB | | | | x |
| HandOutlines | x | | | |
| InlineSkate | | | | x |
| MixedShapesRegularTrain | | | | x |
| MixedShapesSmallTrain | | | | x |
| NonInvasiveFetalECGThorax1 | | | | x |
| Phoneme | | | | x |
| PigArtPressure | | x | | |
| PigCVP | | x | | |
| SemgHandGenderCh2 | | | | x |
| SemgHandMovementCh2 | | | | x |
| SemgHandSubjectCh2 | | | | x |
| StarLightCurves | | | | x |
| UWaveGestureLibraryAll | | | | x |
| UWaveGestureLibraryZ | | | x | |
| Total missing | 1 | 2 | 1 | 15 |

**Author contributions** CH ran the majority of the experiments and AB wrote more of the paper, but both CH and AB worked jointly and closely on this project.

**Funding** This work has been supported by the UK Research and Innovation Engineering and Physical Sciences Research Council (grant reference EP/W030756/2). Some of the experiments were carried out on the High Performance Computing Cluster supported by the Research and Specialist Computing Support service at the University of East Anglia.

**Data availability** In line with the FAIR principles (Findable, Accessible, Interoperable, and Reusable), all datasets used in this study are publicly available from the the University of California, Riverside (UCR) Time Series Archive https://timeseriesclassification.com. All datasets are also available from zenodo https://zenodo.org/communities/tsml/ and can be downloaded as a single zip file https://zenodo.org/records/11198697. The source code and experimental workflows are implemented using the open-source aeon toolkit https://aeon-toolkit.org and the tsml-eval evaluation suite https://tsml-eval.readthedocs.io/en/stable/. KASBA is available as scikit-learn compatible estimator in aeon release 1.1 as are most of the other clustering algorithms we use. All experiments, results, and reproduction scripts for this paper are available in the repository associated with this paper https://github.com/aeon-toolkit/aeon-benchmark. All code is released under the BSD 3-Clause License, and full documentation is provided to support reuse, extension, and validation by other researchers.

## Declarations

**Conflict of interest** The authors have no relevant financial or non-financial interests to disclose. The authors have no Conflict of interest to declare that are relevant to the content of this article. All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript. The authors have no financial or proprietary interests in any material discussed in this article.

**Ethical approval** In line with the FAIR principles (Findable, Accessible, Interoperable, and Reusable), all datasets used in this study are publicly available from the University of California, Riverside (UCR) Time Series Archive (Dau et al. 2018)(https://timeseriesclassification.com). All datasets are also available from zenodo(https://zenodo.org/communities/tsml/) and can be downloaded as a single zip file (https://zenodo.org/records/11198697). The source code and experimental workflows are implemented using the open-source `aeon` toolkit (Middlehurst et al. 2024a)(https://aeon-toolkit.org) and the tsml-eval evaluation suite(https://tsml-eval.readthedocs.io/en/stable/). KASBA is available as scikit-learn compatible estimator in `aeon` release 1.1 as are most of the other clustering algorithms we use. All experiments, results, and reproduction scripts for this paper are available in the repository associated with this paper(https://github.com/aeon-toolkit/aeon-benchmark). All code is released under the BSD 3-Clause License, and full documentation is provided to support reuse, extension, and validation by other researchers.

**Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

## References

Abanda A, Mori U, Lozano J (2019) A review on distance based time series classification. Data Min Knowl Disc 33(2):378–412

Aghabozorgi S, Ying Wah T, Herawan T et al (2014) A hybrid algorithm for clustering of time series data based on affinity search technique. Sci World J 1:562194. https://doi.org/10.1155/2014/562194
Almahamid F, Grolinger K (2022) Agglomerative hierarchical clustering with dynamic time warping for household load curve clustering. https://doi.org/10.1109/CCECE49351.2022.9918481
Alqahtani A, Ali M, Xie X et al (2021) Deep time-series clustering: a review. Electronics. https://doi.org/10.3390/electronics10233001
Arthur D, Vassilvitskii S (2007) k-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms. Society for industrial and applied mathematics, USA, SODA '07, p 1027–1035
Bandara K, Bergmeir C, Smyl S (2020) Forecasting across time series databases using recurrent neural networks on groups of similar series: a clustering approach. Expert Syst Appl 140:112896
Barnes T, Werner E, Clark JN et al (2024) Towards personalised patient risk prediction using temporal hospital data trajectories. In: Shaban-Nejad A, Michalowski M, Bianco S (eds) AI for health equity and fairness, studies in computational intelligence, vol 1164. p 9–18
Begum N, Ulanova L, Dau HA et al (2016) A general framework for density based time series clustering exploiting a novel admissible pruning strategy. ArXiv abs/1612.00637. https://api.semanticscholar.org/CorpusID:7950039
Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? J Mach Learn Res 17:1–10
Berndt DJ, Clifford J (1994) Using dynamic time warping to find patterns in time series. In: Proceedings of the 3rd international conference on knowledge discovery and data mining, AAAIWS'94, pp 359–370
Boniol P, Tiano D, Bonifati A et al (2025) $k$-graph: a graph embedding for interpretable time series clustering. IEEE Trans Knowl Data Eng 37(5):2680–2694. https://doi.org/10.1109/TKDE.2025.3543946
Bottou L, Bengio Y (1994) Convergence properties of the k-means algorithms. In: Tesauro G, Touretzky D, Leen T (eds) Advances in neural information processing systems
Brill M, Fluschnik T, Froese V et al (2019) Exact mean computation in dynamic time warping spaces. Data Min Knowl Disc 33:252–291
Caiado J, Maharaj E, D'Urso P (2015) Time series clustering. In: Handbook of cluster analysis, pp 241–264
Celebi ME, Kingravi HA, Vela PA (2013) A comparative study of efficient initialization methods for the k-means clustering algorithm. Expert Syst Appl 40(1):200–210. https://doi.org/10.1016/j.eswa.2012.07.021https://www.sciencedirect.com/science/article/pii/S0957417412008767
Cuturi M, Blondel M (2017) Soft-DTW: a differentiable loss function for time-series. In: Proceedings of the 34th international conference on machine learning, Vol 70, ICML'17, p 894–903
Dau H, Bagnall A, Kamgar K et al (2018) The UCR time series archive. arXiv preprint arXiv:1810.07758
Dempster A, Petitjean F, Webb G (2020) ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. Data Min Knowl Disc 34:1454–1495
Dempster A, Schmidt DF, Webb GI (2021) Minirocket: a very fast (almost) deterministic transform for time series classification. In: Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining. Association for computing machinery, New York, NY, USA, KDD '21, p 248–257. https://doi.org/10.1145/3447548.3467231
Dempster A, Mohammadi Foumani N, Tan CW et al (2025) Monster: monash scalable time series evaluation repository. arXiv preprint arXiv:2502.15122
Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
Dhariyal B, Nguyen TL, Ifrim G (2023) Scalable classifier-agnostic channel selection for multivariate time series classification. Data Min Knowl Disc 37:1010–1054
Elkan C (2003) Using the triangle inequality to accelerate k-means. In: Proceedings of the twentieth international conference on international conference on machine learning, pp 147–153
Ester M, Kriegel HP, Sander J et al (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, pp 226–231
Forestier G, Petitjean F, Dau HA et al (2017) Generating synthetic time series to augment sparse datasets. In: 2017 IEEE international conference on data mining (ICDM), pp 865–870. https://doi.org/10.1109/ICDM.2017.106
Forgy EW (1965) Cluster analysis of multivariate data : efficiency versus interpretability of classifications. Biometrics 21:768–769. https://api.semanticscholar.org/CorpusID:118110564
Fréchet M (1948) Les éléments aléatoires de nature quelconque dans un espace distancié. Annales de l'institut Henri Poincaré 10(4):215–310. http://eudml.org/doc/79021

García S, Herrera F (2008) An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J Mach Learn Res 9:2677–2694

Gutiérrez AP, Delfa JLV, López MV (2023) Time series clustering using trend, seasonal and autoregressive components to identify maximum temperature patterns in the "iberian" peninsula. Environ Ecol Stat 30(3):421–442. https://doi.org/10.1007/s10651-023-00572-9

Han J, Xu J, Nie F et al (2022) Multi-view k-means clustering with adaptive sparse memberships and weight allocation. IEEE Trans Knowl Data Eng 34(2):816–827. https://doi.org/10.1109/TKDE.2020.2986201

Holder C, Guijo-Rubio D, Bagnall A (2023) Clustering time series with k-medoids based algorithms. In: Ifrim G, Tavenard R, Bagnall A et al (eds) Advanced analytics and learning on temporal data. Springer Nature Switzerland, Cham, pp 39–55

Holder C, Guijo-Rubio D, Bagnall AJ (2023b) Barycentre averaging for the move-split-merge time series distance measure. In: International conference on knowledge discovery and information retrieval, https://api.semanticscholar.org/CorpusID:265356576

Holder C, Bagnall A, Lines J (2024a) On time series clustering with k-means. arXiv preprint arXiv:2410.14269https://doi.org/10.48550/arXiv.2410.14269, cs.LG

Holder C, Middlehurst M, Bagnall A (2024) A review and evaluation of elastic distance functions for time series clustering. Knowl Inf Syst 66(2):765–809. https://doi.org/10.1007/s10115-023-01952-0

Hubert L, Arabie P (1985) Comparing partitions. J Classif 2(1):193–218

Ismail-Fawaz A, Dempster A, Tan CW et al (2023a) An approach to multiple comparison benchmark evaluations that is stable under manipulation of the comparate set. arXiv preprint arXiv:2305.11921

Ismail-Fawaz A, Ismail Fawaz H, Petitjean F et al (2023) Shapedba: generating effective time series prototypes using shapedtw barycenter averaging. In: Ifrim G, Tavenard R, Bagnall A et al (eds) Advanced analytics and learning on temporal data. Springer Nature Switzerland, Cham, pp 127–142

Jang M, Han MS, Kim Jh et al (2011) Dynamic time warping-based k-means clustering for accelerometer-based handwriting recognition. In: Developing concepts in applied intelligence, studies in computational intelligence, vol 363. Springer, Berlin, Heidelberg, p 21–26. https://doi.org/10.1007/978-3-642-21332-8_3

Javed A, Lee BS, Rizzo DM (2020) A benchmark study on time series clustering. Machine learning with applications 1. https://doi.org/10.1016/j.mlwa.2020.100001. https://www.sciencedirect.com/science/article/pii/S2666827020300013

Javed A, Rizzo DM, Lee BS et al (2024) Somtimes: self organizing maps for time series clustering and its application to serious illness conversations. Data Min Knowl Disc 38(3):813–839. https://doi.org/10.1007/s10618-023-00979-9

Jorge MB, Cuevas R (2024) Time series clustering with random convolutional kernels. Data Min Knowl Disc 38(4):1862–1888. https://doi.org/10.1007/s10618-024-01018-x

Kaufman L, Rousseeuw PJ (1990) Clustering large applications (Program "CLARA"), chap 3, pp 126–163. https://doi.org/10.1002/9780470316801.ch3

Keogh E, Kasetty S (2003) On the need for time series data mining benchmarks: a survey and empirical demonstration. Data Min Knowl Disc 7(4):349–371. https://doi.org/10.1023/A:1024988512476

Lafabregue B, Weber J, Gancarski P et al (2022) End-to-end deep representation learning for time series clustering: a comparative study. Data Min Knowl Disc 36:29–81

Lafabregue B, Weber J, Gançarski P et al (2022) End-to-end deep representation learning for time series clustering: a comparative study. Data Min Knowl Disc 36(1):29–81. https://doi.org/10.1007/s10618-021-00796-y

Leonard Kaufman PJR (1990) Partitioning around medoids (Program PAM), chap 2, pp 68–125

Li X, Lin J, Zhao L (2019) Linear time complexity time series clustering with symbolic pattern forest. In: Proceedings of the twenty-eighth international joint conference on artificial intelligence (IJCAI-19). International joint conferences on artificial intelligence organization, pp 2930–2936. https://doi.org/10.24963/ijcai.2019/406

Lin J, Vlachos M, Keogh E et al (2004) Iterative incremental clustering of time series. In: Advances in database technology, pp 106–122

Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Disc 29:565–592

Lloyd S (1982) Least squares quantization in "pcm". IEEE Trans Inf Theory 28(2):129–137. https://doi.org/10.1109/TIT.1982.1056489

MacQueen JB (1967) Some methods for classification and analysis of multivariate observations

Marteau PF (2009) Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 31(2):306–318. https://doi.org/10.1109/TPAMI.2008.76
McDowell IC, Manandhar D, Vockley CM et al (2018) Clustering gene expression time series data using an infinite gaussian process mixture model. PLoS Comput Biol 14(1):1–27. https://doi.org/10.1371/journal.pcbi.1005896
Middlehurst M, Ismail-Fawaz A, Guillaume A et al (2024a) Aeon: a python toolkit for learning from time series. J Mach Learn Res 25(289):1–10. http://jmlr.org/papers/v25/23-1444.html
Middlehurst M, Schäfer P, Bagnall A (2024) Bake off redux: a review and experimental evaluation of recent time series classification algorithms. Data Min Knowl Disc. https://doi.org/10.1007/s10618-024-01022-1
Ng R, Han J (2002) CLARANS: a method for clustering objects for spatial data mining. IEEE Trans Knowl Data Eng 14:1003–1016. https://doi.org/10.1109/TKDE.2002.1033770
Ozer M, Sapienza A, Abeliuk A et al (2020) Discovering patterns of online popularity from time series. Expert Syst Appl 151:113337. https://doi.org/10.1016/j.eswa.2020.113337
Paparrizos J, Bogireddy SPTR (2025) Time-series clustering: a comprehensive study of data mining, machine learning, and deep learning methods. Proceedings of the VLDB Endowment 18(11), 4380–4395. https://doi.org/10.14778/3749646.3749700
Paparrizos J, Gravano L (2016) k-shape: efficient and accurate clustering of time series. SIGMOD Rec 45(1):69–76. https://doi.org/10.1145/2949741.2949758
Petitjean F, Ketterlin A, Gançarski P (2011) A global averaging method for dynamic time warping, with applications to clustering. Pattern Recogn 44(3):678–693. https://doi.org/10.1016/j.patcog.2010.09.013https://www.sciencedirect.com/science/article/pii/S003132031000453X
Petitjean F, Forestier G, Webb GI et al (2016) Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. Knowl Inf Syst 47:1–26
Rakthanmanon T, Campana B, Mueen A et al (2013) Addressing big data time series: mining trillions of time series subsequences under dynamic time warping. ACM Trans Knowl Discov Data 7(3). https://doi.org/10.1145/2500489,
Ratanamahatana C, Keogh E (2005) Three myths about dynamic time warping data mining. In: proceedings of the 5th SIAM international conference on data mining, pp 506–510
Rousseeuw PJ (1987) Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65. https://doi.org/10.1016/0377-0427(87)90125-7. https://www.sciencedirect.com/science/article/pii/0377042787901257
Schultz D, Jain B (2018) Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. Pattern Recogn 74:340–358. https://doi.org/10.1016/j.patcog.2017.08.012https://www.sciencedirect.com/science/article/pii/S0031320317303163
Serantoni C, Riente A, Abeltino A et al (2024) Integrating dynamic time warping and k-means clustering for enhanced cardiovascular fitness assessment. Biomed Signal Process Control 97:106677
Shifaz A, Pelletier C, Petitjean F et al (2023) Elastic similarity and distance measures for multivariate time series. Knowl Inf Syst 65(6):2665–2698
Stefan A, Athitsos V, Das G (2013) The Move-Split-Merge metric for time series. IEEE Trans Knowl Data Eng 25(6):1425–1438
Tselentis DI, Papadimitriou E (2023) Time-series clustering for pattern recognition of speed and heart rate while driving: a magnifying lens on the seconds around harsh events. Trans Res F Traffic Psychol Behav 98:254–268
Ward J (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244. https://doi.org/10.1080/01621459.1963.10500845
Warrens MJ, Van der Hoef H (2019) Understanding partition comparison indices based on counting object pairs. J Classif 36(1):127–150
Wenig P, Höfgen M, Papenbrock T (2024) Jet: fast estimation of hierarchical time series clustering. Engineering proceedings 68(1). https://doi.org/10.3390/engproc2024068037, https://www.mdpi.com/2673-4591/68/1/37
Yang J, Leskovec J (2011) Patterns of temporal variation in online media. In: Proceedings of the fourth ACM international conference on web search and data mining. Association for computing machinery, New York, NY, USA, WSDM '11, p 177–186. https://doi.org/10.1145/1935826.1935863,
Yang J, Lin CT (2024) Enhanced adjacency-constrained hierarchical clustering using fine-grained pseudo labels. IEEE Trans Emerg Topics Comput Intell 8(3):2481–2492
Yang J, Lin CT (2025) Autonomous clustering by fast find of mass and distance peaks. IEEE Trans Pattern Anal Mach Intell 47(7):5336–5349

Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining. Association for computing machinery, New York, NY, USA, KDD '09, p 947–956. https://doi.org/10.1145/1557019.1557122,

Ye L, Keogh E (2011) Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. Data Min Knowl Disc 22(1–2):149–182

Zakaria J, Mueen A, Keogh E (2012) Clustering time series using unsupervised-shapelets. In: 2012 IEEE 12th international conference on data mining, pp 785–794. https://doi.org/10.1109/ICDM.2012.26

Zhang Q, Wu J, Zhang P et al (2019) Salient subsequence learning for time series clustering. IEEE Trans Pattern Anal Mach Intell 41(9):2193–2207. https://doi.org/10.1109/TPAMI.2018.2847699

Zhao J, Itti L (2018) shapeDTW: shape dynamic time warping. Pattern Recogn 74:171–184

Zolhavarieh S, Aghabozorgi S, Teh YW (2014) A review of subsequence time series clustering. Sci World J 2014:312521. https://doi.org/10.1155/2014/312521

Zou X, Chung E (2024) Traffic prediction via clustering and deep transfer learning with limited data. Comput-Aided Civil Infrastruct Eng. https://doi.org/10.1111/mice.13207

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

## Authors and Affiliations

**Christopher Holder[1] · Anthony Bagnall[1,2]**

✉ Anthony Bagnall
a.j.bagnall@soton.ac.uk

Christopher Holder
c.l.holder@soton.ac.uk

[1] School of Electronics and Computer Science, University of Southampton, University Rd, Southampton, Hampshire SO17 1BJ, England

[2] School of Computing Sciences, University of East Anglia, Norwich, Norfolk NR4 7TJ, England