Knowledge and Information Systems (2024) 66:765–809
https://doi.org/10.1007/s10115-023-01952-0

REVIEW

A review and evaluation of elastic distance functions for time series clustering

Christopher Holder[1] · Matthew Middlehurst[1] · Anthony Bagnall[1]

Received: 11 May 2023 / Revised: 2 July 2023 / Accepted: 25 July 2023 /
Published online: 7 September 2023

Abstract

Time series clustering is the act of grouping time series data without recourse to a label. Algorithms that cluster time series can be classified into two groups: those that employ a time series specific distance measure and those that derive features from time series. Both approaches usually rely on traditional clustering algorithms such as k-means. Our focus is on partitional clustering algorithms that employ elastic distance measures, i.e. distances that perform some kind of realignment whilst measuring distance. We describe nine commonly used elastic distance measures and compare their performance with k-means and k-medoids clusterer. Our findings, based on experiments using the UCR time series archive, are surprising. We find that, generally, clustering with DTW distance is not better than using Euclidean distance and that distance measures that employ editing in conjunction with warping are significantly better than other approaches. We further observe that using k-medoids clusterer rather than k-means improves the clusterings for all nine elastic distance measures. One function, the move–split–merge (MSM) distance, is the best performing algorithm of this study, with time warp edit (TWE) distance a close second. Our conclusion is that MSM or TWE with k-medoids clusterer should be considered as a good alternative to DTW for clustering time series with elastic distance measures. We provide implementations, extensive results and guidance on reproducing results on the associated GitHub repository.

Keywords Time series clustering · Dynamic time warping · Dynamic barycentre averaging · Move–split–merge · Edit distance with real penalty · Time warp edit distance · Derivative dynamic time warping · Longest common subsequence · Edit distance on real sequences · Weighted dynamic time warping · Weighted derivative dynamic time warping · k-Means · k-medoids clusterer

✉ Anthony Bagnall
ajb@uea.ac.uk

Christopher Holder
c.holder@uea.ac.uk

Matthew Middlehurst
m.middlehurst@uea.ac.uk

[1] School of Computing Sciences, University of East Anglia, Norwich, UK

1 Introduction

Clustering is an unsupervised analysis technique where a set of cases, defined as a vector of continuous or discrete variables, are grouped to create clusters which contain cases considered to be homogeneous, whereas cases in different clusters are considered heterogeneous [12].

Time series clustering is the act of grouping ordered, time series data without recourse to a label. We use the acronym TSCL for time series clustering, to make a distinction between TSCL and time series classification, which is commonly referred to as TSC. There has been a wide range of algorithms proposed for TSCL. Our focus is specifically on clustering using elastic distance measures, i.e. distance measures that use some form of realignment of the series being compared. Our aim is to perform a comparative study of these algorithms that follows the basic structure of bake offs in distance-based TSC [44], univariate TSC [8] and multivariate TSC [60]. A huge number of alternative transformation-based, e.g. [42], deep learning-based clustering algorithms, e.g. [39], and statistical model-based approaches [15] have been proposed for TSCL. These approaches are not the focus of this research. Our primary aim is to provide a detailed description, with examples, of nine commonly used elastic distance measures and conduct extensive experimentation to compare their utility for TSCL. k-means is by far the most popular clustering algorithm for TSCL (for example, [33]). k-medoids clusterer is used much less frequently. Our secondary aim is to compare k-means and k-medoid distance-based clustering for TSCL.

Clustering is often the starting point for exploratory data analysis and is widely performed in all fields of data science [74]. However, clustering is harder to define than, for example, classification. There is debate about what clustering means [31] and no accepted standard definition of what constitutes good clustering or what it means for cases to be homogeneous or heterogeneous. For example, homogeneous could mean generated by some common underlying process or mean it has some hidden variable in common. A clustering of patients based on some medical data might group all male patients in one cluster and all female patients in another. The clusters are from one view homogeneous, but that does not mean it is necessarily a good clustering. The interpretation of the usefulness of clustering a particular dataset requires domain knowledge. This makes comparing algorithms over a range of problems more difficult than performing a bake off of TSC algorithms. Nevertheless, there have been numerous comparative studies (for example, [2, 33] and [39]) which take TSC problems from the UCR archive [18] and then evaluate clusterings based on how well they recreate class labels. We aim to add to this body of knowledge with a detailed description and a reproducible evaluation of clustering with elastic distance measures (described in Sect. 3) following a similar methodology. We do this in the context of partitional clustering algorithms (see Sect. 2). We stress that our findings relate only to performance on the UCR univariate datasets and that our conclusions should be taken in that context. We think that our findings are useful for practitioners wanting a benchmark for a TSCL problem, but there are always use cases for different approaches.

Elastic distance measures are significantly more accurate on average than Euclidean distance for TSC when used with a one nearest neighbour classifier [44]. We evaluate whether this improvement translates to centroid-based clustering. Following experiments presented in Sect. 5.1, we conclude that, somewhat surprisingly, this is not the case with k-means clustering. Using standard default parameters, only one elastic distance measure, move–split–merge (MSM) [67], is significantly better than Euclidean distance, and five of those considered are significantly worse, including the most popular approach, dynamic time warping (DTW). The pattern of results is the same if we cluster the raw data or normalise it first. We repeat our experiments using k-medoids clusterer (see Sect. 5.2) and find that clustering performance is improved for all distances except Euclidean, and DTW is no longer significantly worse than ED. With both k-means and k-medoids clusterer, the move–split–merge distance function [67] performs best. We evaluate whether these results could be an artefact of our clustering algorithm configuration in Sect. 5.3 by comparing alternative cluster initialisation algorithms and finding the same pattern of results in all cases. k-means with DTW is a very popular benchmark for TSCL (for example, see [33]), so the findings in Sect. 5.1 are perhaps counter to the received wisdom in the community. We investigate the performance of DTW under additional scenarios in Sects. 5.4.2 and 5.4. We try alternative parameter settings and use an unsupervised tuning algorithm for the window size. We show that reducing the window size to 5% improves k-means DTW, but only to the point where it is no longer worse than Euclidean distance. We also find that tuning does not improve the 5% window performance. We then explore using dynamic time warping with barycentre averaging (DBA) to improve k-means [55]. DBA finds centroids by aligning cluster members and averaging over values warped to each location. We reproduce the reported improvement that DBA brings to DTW with k-means, but we observe that the improvement is not enough to make it better than Euclidean distance, and it comes with heavy computational overhead. Finally, we assess whether tuning our best performing distance functions and find that we cannot on average improve on using the default parameters. In Sect. 6, we look more closely at the performance of the distance functions for data with different characteristics. We find MSM does relatively better on problems with a large number of classes and a larger training set size. We also observe that it does better in problem domains where we would expect some phase shift within clusters, such as ECG and electric device power usage problems. Our first conclusion in Sect. 7 is that elastic distances perform better on the UCR archive when used with k-medoids clusterer rather than with k-means clustering. Using medoids (data points in the training data) rather than centroids (averaged cluster members) overcomes the mismatch between simple averaging and elastic distances that barycentre averaging is designed to mitigate against, but with far less computational overhead. Our second observation is that MSM is the best performing distance function on the UCR archive. MSM explicitly penalises longer warpings, and we observe that all distance functions that do this perform better on the UCR archive. We hope this work will raise the profile of MSM in the TSCL community.

We have provided optimised scikit-learn compatible Python implementations for the distances and clusterers in the aeon toolkit,[1] and a repository with an associated webpage[2] which provides notebooks on using distances and clusterers and contains all our results with guidance on how to reproduce them.

The remainder of this paper is structured as follows: Sect. 2 describes the general TSCL problem, provides a brief literature review and gives a detailed description of nine elastic distance measures used in our experiments. For more general background on clustering, we direct the reader to [32, 62]. For background into elastic distances for classification, we recommend [1, 65].

The methodology we use in our experiments is outlined in Sect. 4. The results on UCR datasets are presented in Sect. 5, and these are further analysed in Sect. 6. Finally, Sect. 7 concludes and signposts the future direction of our research.

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

[2] https://tsml-eval.readthedocs.io/en/latest/publications/2023/distance_based_clustering/distance_based_clustering.html.

2 Time series clustering background and literature review

A time series $\boldsymbol{x}$ is a sequence of m observations, $(x_1, \ldots, x_m)$. We assume all series are equal in length. For univariate time series, x_i are scalars and for multivariate time series, x_i are vectors. A time series data set, $D = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$, is a set of n time series cases. A clustering is some form of a grouping of cases. Broadly speaking, clustering algorithms are either partitional or hierarchical. Partitional clustering algorithms assign (possibly probabilistic) cluster membership to each time series, usually through an iterative heuristic process of optimising some objective function that measures homogeneity. Given a dataset of n time series, D, the partitional time series clustering problem is to partition D into k clusters, $C = \{C_1, C_2, \ldots, C_k\}$ where k is the number of clusters. It is usually assumed that the number of clusters is set prior to the optimisation heuristic. If k is not set, it is commonly found through some wrapper method. We assume k is fixed in advance for all our experiments.

Clustering algorithms can be split into those that work directly with the time series, and those that employ a transformation to derive features prior to clustering. The focus of this study is on non-probabilistic partitional clustering algorithms that work directly with time series.

2.1 Partitional time series clustering algorithms

Partition-based clustering algorithms share the same basic components. Firstly, the algorithm selects example cases (which we call exemplars) that are meant to characterise a cluster. This is known as the *initialisation* stage. After the initial exemplars are selected, an *update* stage begins where the exemplars are iteratively refined until some convergence condition is met.

One such partition-based algorithm is k-means [47], also known as Lloyd's algorithm [46]. It is the most well-known and popular partitional clustering algorithm, in both standard clustering and time series clustering. The algorithm uses k centroids as exemplars for each cluster. A centroid is a summary of the members of a cluster found through the *update* operation, which for standard k-means involves averaging each time point over cluster members. Each case is assigned to the cluster of its closest centroid, as defined by a distance measure.

Many enhancements to the core algorithm have been proposed. One of the most effective refinements is changing how the exemplars are initialised. By default, the initial centroids for k-means are chosen randomly, either by randomly assigning cluster membership and then averaging or by choosing random instances from the training data as initial clusters. However, this risks choosing centroids that are in fact in the same cluster, making it harder to split the clusters. To address this problem, forms of restart and selection are often employed. For example, [13] propose restarting k-means over subsamples of the data and using the resulting centroids to seed the full run. Another solution is to run the algorithm multiple times and keep the model that yields the best result according to some unsupervised performance measure.

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 [45]. Time series-specific enhancements concern the distance metric used and the averaging technique to recalculate centroids. Averaging series matched with an elastic distance measure will mean that, often, the characteristics that made the series similar are lost in the centroid. [55] describe an alternative averaging method based on pairwise DTW. This is described in detail in Sect. 3.8.

k-medoids clusterer is an alternative clustering algorithm which uses cases, or medoids, as cluster exemplars. One benefit of using instances as cluster centres is that the pairwise distance matrix of the training data is sufficient to fit the model, and this can be calculated before the iterative partitioning. This is particularly important when performing the *update* operation, which is the main difference between k-means and k-medoids clusterer; k-medoids clusterer chooses a cluster member as the exemplar rather than average. The medoid is the case with the minimum total distance to the other members of the cluster. Refinements such as Partition Around Medoids (PAM) [40] avoid the complete enumeration of total distances through a swapping heuristic.

Both of our k-means and k-medoids clusterers extend an implementation of the Lloyds algorithm and employ the same stopping condition. The iterative process stops when cluster membership does not change, or when the *inertia* is below a certain tolerance (convergence) [27]. The inertia is a value that is used to determine how coherent different clusters are and equals the sum of distances between each case and the centre of its assigned cluster. If the difference in inertia between two iterations is below the (very small) threshold, convergence is assumed and the iterations stop.

Partitional clustering algorithms both suffer from the problem of initialising the clusters. The initialisation problem is twofold: defining the accurate cluster numbers to be generated from the given dataset and solving the problem of locating the position of initial centroids [27]. Whist there are numerous proposals on how to select the value of k (for example, [51] use medoids initialisation, [22] PCA initialisation, [29] propose cluster elimination and division initialisation), we follow related comparative studies [33] and assume the number of clusters is known and equal to the number of classes in a classification dataset. The second problem of finding the initial cluster centres is important because poor initial centres can lead to convergence to local optima and worse clustering performance [70]. We consider three commonly used initialisation algorithms in our experimentation.

The most common initialisation technique is random initialisation [47]. This technique consists of choosing the initial centres randomly from the dataset. The rationale behind this is that random selection is likely to pick points from dense regions. Rerunning the model multiple times with random initialisation and taking the best clustering (as measured by inertia) is the most common way of initialising k-means [14] and is used in the majority of experiments. However, we also consider Forgy's method [24] which initialises centres by assigning each data point in the dataset to k clusters uniformly at random. The centres are then given by the centroids of these initial clusters. A drawback of this method is it has no theoretical bases and as such random clusters have no internal homogeneity [5]. Finally, we consider the k-means++ algorithm [7]. k-means++ tries to disperse the clusters by selecting to bias towards separate centres. The first centre is randomly selected from the train data. Subsequent centres are selected with probability inversely proportional to the minimum distance between cases and previously selected centres. If $md(x)$ denotes the minimum distance from a case x and previously selected centres, then the probability of selecting x as the next centre is defined as

$$\frac{md(x)^2}{\sum_{j=1}^{n} md(x_j)^2}. \tag{1}$$

2.2 Literature review

Our focus is on elastic distances-based clustering, but there are many other approaches to TSCL. The most popular TSCL approaches are summarised in Fig. 1.

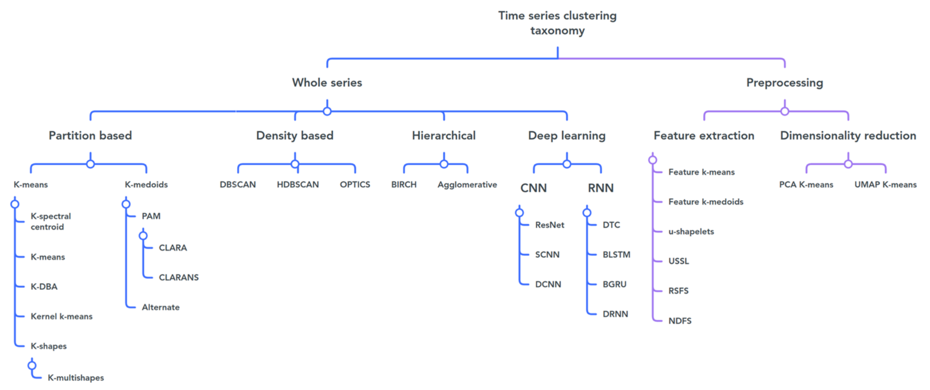


Fig. 1 Time series clustering taxonomy. The following models are included: K-means [47], K-spectral centroid [69], K-DBA [55], Kernel K-means [21], K-shapes [53], K-multishapes [54], PAM [40], CLARA [36], CLARANS [52], Alternate [46], DBSCAN [23], HDBSCAN [49], OPTICS [6], BIRCH [73], Agglomerative [35], Feature K-means [59], Feature K-medoids clusterer [59], U-shapelets [71], USSL [72], RSFS [64], NDFS [43], Deep learning and dimensionality reduction approaches see [39]

The k-Shape algorithm [53] and its extension k-multishapes (k-MS) [54] are a variation of the k-means algorithm that uses cross correlation to compute the similarity between time series. Cross-correlation is a measure of similarity that compares points of time-lagged signals one to one. k-MS algorithm is a extends k-shapes by computing multiple centroids per cluster.

K-spectral centroid (KSc) [69] computes the distance between time series using shifted ranges of lags with k-means. Kernel k-means [21] operates in the Reproducing Kernel Hilbert Space associated with the chosen kernel. Further kernel based techniques include using Fisher kernels with hidden Markov models [68].

Similarity-based techniques for TSCL will compute similarity based on raw time series. However, working with raw time series is a non-trivial task due to the challenges that raw time series present (high dimensionality, missing values, variable length, etc.).

Another common way to cluster time series is to perform some form of feature extraction and then cluster the features. For example, [59] used a standard k-means clusterer on statistical features (skew, standard deviations, mean etc.) extracted from an electricity usage time series dataset. k-medoids clusterer has also been used with features extracted from time series using piecewise aggregate approximation [41]. Further unique feature extraction techniques specifically designed for TSCL include u-shapelets [71], Unsupervised Salient Subsequence Learning (USSL) [72], robust spectral learning for unsupervised feature selection (RSFS) [64] and nonnegative discriminative feature selection (NDFS) [43].

Finally, one of the most recent developments in the TSCL is the use of deep learning. A good review of deep learning-based time series clustering is found in [39].

3 Time series distance measures

Suppose we want to measure the distance between two equal lengths, univariate time series, $\mathbf{a} = \{a_1, a_2, \ldots, a_m\}$ and $\mathbf{b} = \{b_1, b_2, \ldots, b_m\}$. The Euclidean distance d_{ed} is the L2 norm between series,

Table 1 List of the ten distance functions reviewed

Distance	Acronym	Definition
Euclidean distance	ED	Eq. 2
Dynamic time warping [58]	DTW	Sect. 3.1
Derivative DTW [37]	DDTW	Sect. 3.2
Weighted DTW [34]	WDTW	Sect. 3.3
Derivative WDTW	DWDTW	Sect. 3.3
Longest common subsequence	LCSS	Sect. 3.4
Edit distance with real penalty [16]	ERP	Sect. 3.6
Edit distance on real sequences [17]	EDR	Sect. 3.5
Move–split–merge [67]	MSM	Sect. 3.7
Time warp edit [48]	TWE	Sect. 3.7

$$d_{\text{ed}}(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{i=1}^{m}(a_i - b_i)^2}. \tag{2}$$

d_{ed} is a standard starting point for distance-based analysis. It puts no priority on the ordering of the series and, when used in TSC, is a poor benchmark for distance-based algorithms and a very long way from state of the art [50]. Elastic distance measures allow for possible error in offset by attempting to optimally align two series based on distorting indices or editing series. These have been shown to significantly improve k-nearest neighbour classifiers in comparison with d_{ed} [44]. Our aim is to see if we observe the same improvement with clustering. Table 1 lists the ten distance functions we describe.

3.1 Dynamic time warping

Dynamic time warping (DTW) [58] is the most widely researched and used elastic distance measure. It mitigates distortions in the time axis by realigning (warping) the series to best match each other. Let $M(\mathbf{a},\mathbf{b})$ be the $m \times m$ pointwise distance matrix between series $\mathbf{a}$ and $\mathbf{b}$, where $M_{i,j} = (a_i - b_j)^2$. A warping path

$$P =< (e_1, f_1), (e_2, f_2), \ldots, (e_s, f_s) >$$

is a set of pairs of indices that define a traversal of matrix M. A valid warping path must start at location $(1, 1)$, end at point (m, m) and not backtrack, i.e. $0 \le e_{i+1} - e_i \le 1$ and $0 \le f_{i+1} - f_i \le 1$ for all $1 < i < m$. The DTW distance between series is the path through M that minimises the total distance. The distance for any path P of length s is

$$D_P(\mathbf{a},\mathbf{b}, M) = \sum_{i=1}^{s} M_{e_i,f_i}. \tag{3}$$

If $\mathcal{P}$ is the space of all possible paths, the DTW path P^* is the path that has the minimum distance, and hence, the DTW distance between series is

$$d_{\text{dtw}}(\mathbf{a},\mathbf{b}) = D_{P*}(\mathbf{a},\mathbf{b}, M). \tag{4}$$

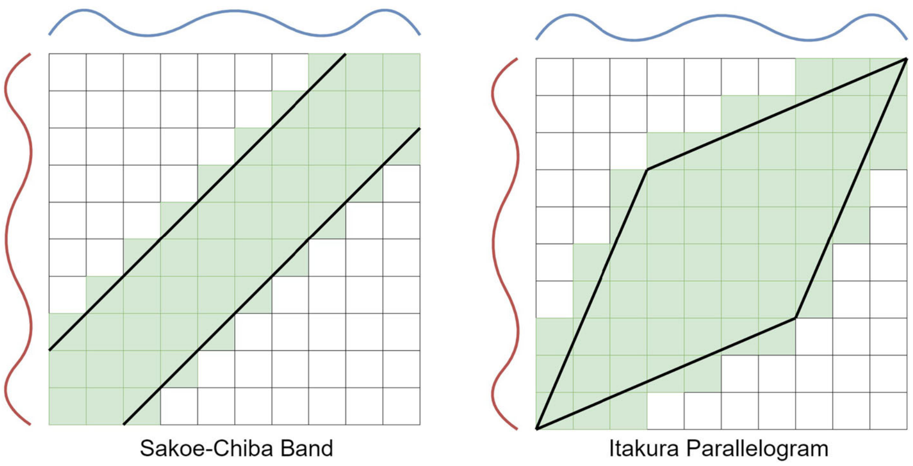


Fig. 2 Two most common bounding techniques [57]

The optimal warping path P^* can be found exactly through a dynamic programming formulation described in Algorithm 1. This can be a time-consuming operation, and it is common to put a restriction on the amount of warping allowed. Figure 2 describes the two most frequently used bounding techniques, the Sakoe–Chiba band [61] and Itakura parallelogram [30]. In Fig. 2, each individual square represents an element of matrix M. Applying a bounding constraint (represented by the darker squares in Fig. 2) reduces the required computation. The Sakoe–Chiba band creates a warping path window that has the same width along the diagonal of M. The Itakura parallelogram allows for less warping at the start or end of the series than in the middle. Algorithm 1 assumes a Sakoe–Chiba band.

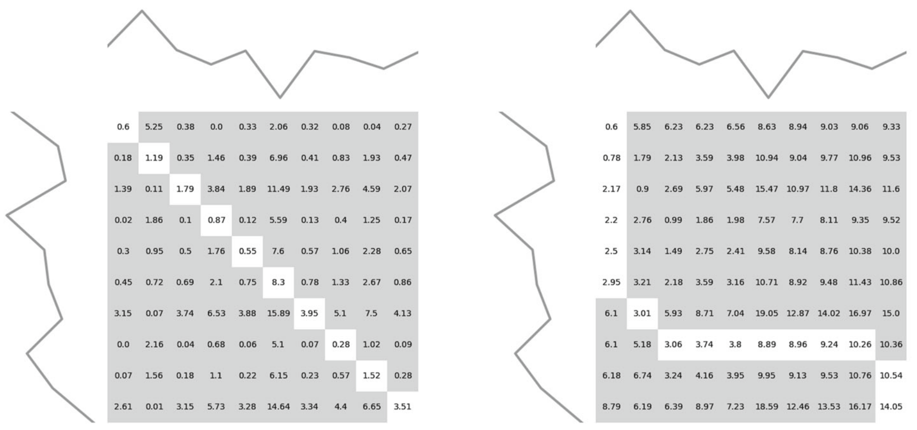
Algorithm 1 DTW ($\mathbf{a}$, $\mathbf{b}$, (*both series of length m*), w (*window proportion*, default value $w \leftarrow 1$), M (*pointwise distance matrix*))

```
1: Let $C$ be an $(m+1) \times (m+1)$ matrix initialised to zero, indexed from zero.
2: for $i \leftarrow 1$ to $m$ do
3:     for $j \leftarrow 1$ to $m$ do
4:         if $|i - j| < w \cdot m$ then
5:             $C_{i,j} \leftarrow M_{i,j} + \min(C_{i-1,j-1}, C_{i-1,j}, C_{i,j-1})$
   return $C_{m,m}$
```

The DTW distance with Sakoe–Chiba window w can be expressed as Eq. 5.

$$d_{\text{dtw}}(\mathbf{a}, \mathbf{b}) = D_{P*}(\mathbf{a}, \mathbf{b}, M) = DTW(\mathbf{a}, \mathbf{b}, w, M). \tag{5}$$

More general bands can be imposed in implementation by setting values outside the band in the matrix, M, to infinity. Figure 3 helps explain how the DTW calculations are arrived at. Euclidean distance is simply the sum of the diagonals of the Matrix M, in Fig. 3a. DTW constructs C using M and previously found values. For example, $C_{1,1} = M_{1,1} = 0.6$ and $C_{1,2}$ is the minimum of $C_{1,1}$, $C_{0,1}$ and $C_{0,2}$ plus $M_{1,2}$. $C_{0,2}$ and $C_{0,1}$ are initialised to infinity, so the best path to get to $C_{1,2}$ has distance $C_{1,1} + M_{1,2}$ which equals $0.6 + 5.25 = 5.85$. Similarly, cell $C_{10,10}$ is the minimum of the three cells $C_{9,9}$, $C_{10,9}$, $C_{9,10}$ plus the pointwise distance $M_{10,10}$. The optimal path is the trace back through the minimum values (shown in



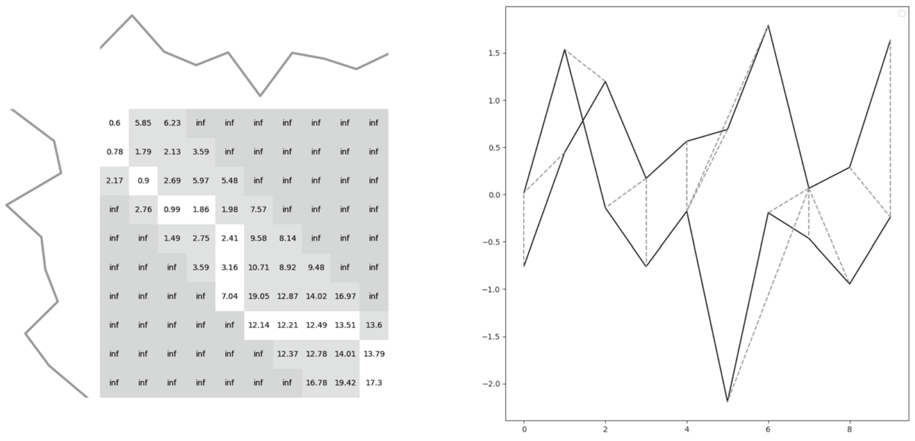
0.6	5.25	0.38	0.0	0.33	2.06	0.32	0.08	0.04	0.27
0.18	1.19	0.35	1.46	0.39	6.96	0.41	0.83	1.93	0.47
1.39	0.11	1.79	3.84	1.89	11.49	1.93	2.76	4.59	2.07
0.02	1.86	0.1	0.87	0.12	5.59	0.13	0.4	1.25	0.17
0.3	0.95	0.5	1.76	0.55	7.6	0.57	1.06	2.28	0.65
0.45	0.72	0.69	2.1	0.75	8.3	0.78	1.33	2.67	0.86
3.15	0.07	3.74	6.53	3.88	15.89	3.95	5.1	7.5	4.13
0.0	2.16	0.04	0.68	0.06	5.1	0.07	0.28	1.02	0.09
0.07	1.56	0.18	1.1	0.22	6.15	0.23	0.57	1.52	0.28
2.61	0.01	3.15	5.73	3.28	14.64	3.34	4.4	6.65	3.51

(a) Pointwise distances matrix M

0.6	5.85	6.23	6.23	6.56	8.63	8.94	9.03	9.06	9.33
0.78	1.79	2.13	3.59	3.98	10.94	9.04	9.77	10.96	9.53
2.17	0.9	2.69	5.97	5.48	15.47	10.97	11.8	14.36	11.6
2.2	2.76	0.99	1.86	1.98	7.57	7.7	8.11	9.35	9.52
2.5	3.14	1.49	2.75	2.41	9.58	8.14	8.76	10.38	10.0
2.95	3.21	2.18	3.59	3.16	10.71	8.92	9.48	11.43	10.86
6.1	3.01	5.93	8.71	7.04	19.05	12.87	14.02	16.97	15.0
6.1	5.18	3.06	3.74	3.8	8.89	8.96	9.24	10.26	10.36
6.18	6.74	3.24	4.16	3.95	9.95	9.13	9.53	10.76	10.54
8.79	6.19	6.39	8.97	7.23	18.59	12.46	13.53	16.17	14.05

(b) Full window DTW in Matrix C

Fig. 3 An example of Euclidean and DTW distance functions for two series. The left hand matrix, (a), is the pointwise distance between the series (matrix M in Equation 5). The Euclidean distance is the sum of the diagonal path. The right hand matrix, (b), shows the DTW distances (matrix C in Equation 5) and the resulting warping path when the window size is unconstrained



0.6	5.85	6.23	inf	inf	inf	inf	inf	inf	inf
0.78	1.79	2.13	3.59	inf	inf	inf	inf	inf	inf
2.17	0.9	2.69	5.97	5.48	inf	inf	inf	inf	inf
inf	2.76	0.99	1.86	1.98	7.57	inf	inf	inf	inf
inf	inf	1.49	2.75	2.41	9.58	8.14	inf	inf	inf
inf	inf	inf	3.59	3.16	10.71	8.92	9.48	inf	inf
inf	inf	inf	inf	7.04	19.05	12.87	14.02	16.97	inf
inf	inf	inf	inf	inf	12.14	12.21	12.49	13.51	13.6
inf	inf	inf	inf	inf	inf	12.37	12.78	14.01	13.79
inf	inf	inf	inf	inf	inf	inf	16.78	19.42	17.3

(a) Constrained DTW with 20% window (b) Constrained DTW alignment.

Fig. 4 An example of constraining the DTW window. Using the same series as Figure 3, (a) shows the DTW distance matrix when using a bounding window that constrains warping to 20% of the series. (b) shows the resulting alignment

white in Fig. 3b). Figure 4 gives a demonstration of the effect of constraining the warping path on DTW using the same two series from Fig. 3. The relatively extreme warping from point 0 to point 5 evident in Fig. 3a is constrained when the maximum warping allowed is 2 places ($w = 0.2$) in Fig. 4.

3.2 Derivative dynamic time warping

Keogh and Pazzani [37] proposed a modification of DTW called derivative dynamic time warping (DDTW) that first transforms the series into a differential series. The difference series of $\mathbf{a}$, $\mathbf{a}' = \{a'_2, a'_3, \ldots, a'_{m-1}\}$ where a'_i is defined as the average of the slopes between a_{i-1} and a_i and a_i and a_{i+1}, i.e.

$$a'_i = \frac{(a_i - a_{i-1}) + (a_{i+1} - a_{i-1})/2}{2} \tag{6}$$

for $1 < i < m$. The DDTW is then simply the DTW of the differenced series,

$$d_{\text{ddtw}}(\mathbf{a}, \mathbf{b}) = d_{\text{dtw}}(\mathbf{a}', \mathbf{b}'). \tag{7}$$

3.3 Weighted dynamic time warping

A weighted form of DTW (WDTW) was proposed by [34]. WDTW adds a multiplicative weight penalty based on the warping distance between points in the warping path. It is a smooth alternative to the cut-off point approach of using a warping window. When creating the distance matrix M, a weight penalty $w(|i-j|)$ for a warping distance of $|i-j|$ is applied, so that

$$M^w_{i,j} = w(|i-j|) \cdot (a_i - b_j)^2. \tag{8}$$

A logistic weight function is proposed in [34], so that a warping of a places imposes a weighting of

$$w(a) = \frac{w_{\max}}{1 + e^{-g\cdot(a-m/2)}} \tag{9}$$

where $w_{\max}$ is an upper bound on the weight (set to 1), m is the series length and g is a parameter that controls the penalty level for large warpings. The larger g is, the greater the penalty for warping. Note that WDTW does not benefit from the reduced search space a window induces. The WDTW distance is then

$$d_{\text{wdtw}}(\mathbf{a}, \mathbf{b}) = D_{P*}(\mathbf{a}, \mathbf{b}, M^w) = DTW(\mathbf{a}, \mathbf{b}, M^w). \tag{10}$$

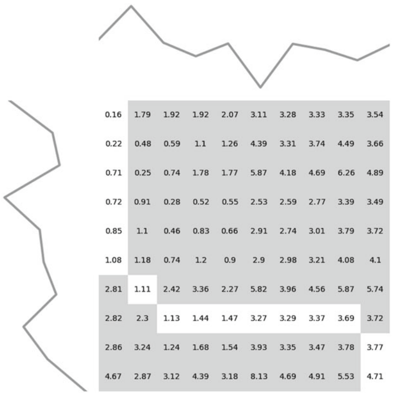
Figure 5 shows the warping resulting from two parameter values. For this example, $g = 0.2$ gives the same warping as full window DTW, but $g = 0.3$ is more constrained.

We also investigate the derivative weighted distance (WDDTW),

$$d_{\text{wddtw}}(\mathbf{a}, \mathbf{b}) = d_{\text{wdtw}}(\mathbf{a}', \mathbf{b}'). \tag{11}$$

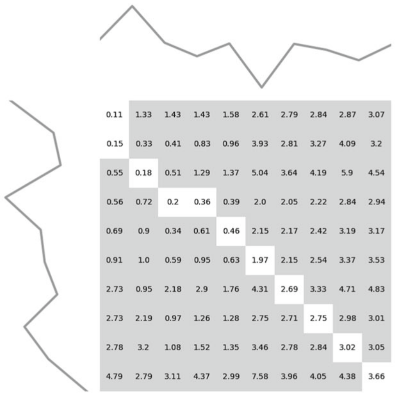
3.4 Longest common subsequence

DTW is usually expressed as a mechanism of finding an alignment so that points are warped onto each other to form a path. An alternative way of looking at the process is as a mechanism of forming a common series between the two input series. With this view, the warping operation can be seen as inserting a gap in one series or removing an element from another series. This way of thinking derives from approaches for aligning sequences of discrete variables, such as strings or DNA. The Longest Common Subsequence (LCSS) distance for time series is derived from a solution to the problem of finding the longest common subsequence between two discrete series through edit operations. For example, if two discrete series are



0.16	1.79	1.92	1.92	2.07	3.11	3.28	3.33	3.35	3.54
0.22	0.48	0.59	1.1	1.26	4.39	3.31	3.74	4.49	3.66
0.71	0.25	0.74	1.78	1.77	5.87	4.18	4.69	6.26	4.89
0.72	0.91	0.28	0.52	0.55	2.53	2.59	2.77	3.39	3.49
0.85	1.1	0.46	0.83	0.66	2.91	2.74	3.01	3.79	3.72
1.08	1.18	0.74	1.2	0.9	2.9	2.98	3.21	4.08	4.1
2.81	1.11	2.42	3.36	2.27	5.82	3.96	4.56	5.87	5.74
2.82	2.3	1.13	1.44	1.47	3.27	3.29	3.37	3.69	3.72
2.86	3.24	1.24	1.68	1.54	3.93	3.35	3.47	3.78	3.77
4.67	2.87	3.12	4.39	3.18	8.13	4.69	4.91	5.53	4.71

(a) Weighted DTW with weight g = 0.2



0.11	1.33	1.43	1.43	1.58	2.61	2.79	2.84	2.87	3.07
0.15	0.33	0.41	0.83	0.96	3.93	2.81	3.27	4.09	3.2
0.55	0.18	0.51	1.29	1.37	5.04	3.64	4.19	5.9	4.54
0.56	0.72	0.2	0.36	0.39	2.0	2.05	2.22	2.84	2.94
0.69	0.9	0.34	0.61	0.46	2.15	2.17	2.42	3.19	3.17
0.91	1.0	0.59	0.95	0.63	1.97	2.15	2.54	3.37	3.53
2.73	0.95	2.18	2.9	1.76	4.31	2.69	3.33	4.71	4.83
2.73	2.19	0.97	1.26	1.28	2.75	2.71	2.75	2.98	3.01
2.78	3.2	1.08	1.52	1.35	3.46	2.78	2.84	3.02	3.05
4.79	2.79	3.11	4.37	2.99	7.58	3.96	4.05	4.38	3.66

(b) Weighted DTW with a weight g = 0.3.

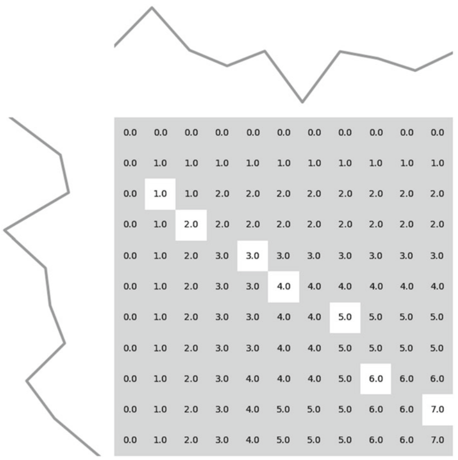
Fig. 5 Examples of the weighted DTW cost matrix C and resulting alignment for two weight parameters

abaacb and *bcacab*, the LCSS is *baab*. Unlike DTW, the LCSS does not provide a path from $(1, 1)$ to (m, m). Instead, it describes edit operations to form a final sequence, and these operations are given a certain cost. So, for example, to edit *abaacb* into the LCSS *baab* requires two deletion operations. For DTW, you can then think of the choice between the three warping paths in line 5 of Eq. 5 as $C_{i-1,j}$ being a deletion in series **b**, $C_{i,j-1}$ as a deletion in series **a** and $C_{i-1,j-1}$ as a match. The warping path shown in Fig. 4 is a sequence of pairs $< (1, 1), (2, 1), (3, 2), (4, 3), (4, 4), (5, 5), (6, 5), (7, 5), (8, 6), (8, 7), (8, 8), (8, 9), (9, 10), (10, 10) >$ can instead be expressed as an edited series $< (1, 1), (3, 2), (4, 3), (5, 5), (8, 6), (9, 10) >$. With this representation the warping operations are in fact deletions (or gaps) in series **a** in positions 2, 6, 7 and 10 and in **b** in positions 4, 7, 8 and 9. With discrete series, the matches in the common subsequence have the same value. Thus, each pair in the subsequence would be, for example, a letter in common for the two series. Obviously, actual matches in real-valued series will be rare. The discrete LCSS algorithm can be extended to consider real-valued time series by using a distance threshold ϵ, which defines the maximum difference between a pair of values that is allowed for them to be considered a match. The length of the LCSS between two series **a** and **b** can be found using Algorithm 2. If two cells are considered the same (line 4), the previously considered LCSS is increased by one. If not, then the LCSS seen so far is carried forward.

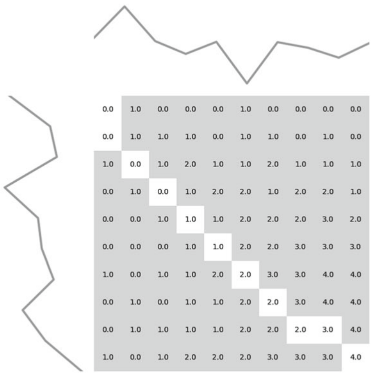
Algorithm 2 LCSS (**a**, **b** , (*both series of length m*), ϵ (*equality threshold*))

```
1: Let $L$ be a $(m + 1) \times (m + 1)$ matrix initialised to zero, indexed from zero.
2: for $i \leftarrow 1$ to $m$ do
3:     for $j \leftarrow 1$ to $m$ do
4:         if $|a_i - b_j| < \epsilon$ then
5:             $L_{i,j} \leftarrow L_{i-1,j-1} + 1$
6:         else
7:             $L_{i,j} \leftarrow \max(L_{i-1,j}, L_{i,j-1})$
   return $L_{m,m}$
```

Fig. 6 An example of the LCSS cost matrix with $\epsilon = 1$, where the white cells are members of the LCSS

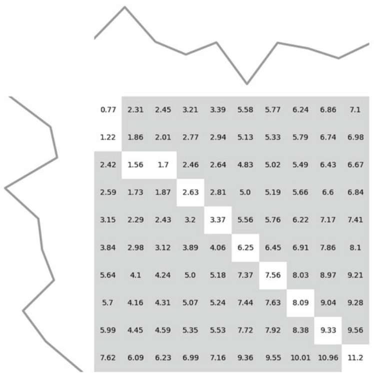


0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.0	1.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
0.0	1.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0	2.0
0.0	1.0	2.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
0.0	1.0	2.0	3.0	3.0	4.0	4.0	4.0	4.0	4.0	4.0
0.0	1.0	2.0	3.0	3.0	4.0	4.0	5.0	5.0	5.0	5.0
0.0	1.0	2.0	3.0	3.0	4.0	4.0	5.0	5.0	5.0	5.0
0.0	1.0	2.0	3.0	4.0	4.0	4.0	5.0	6.0	6.0	6.0
0.0	1.0	2.0	3.0	4.0	5.0	5.0	5.0	6.0	6.0	7.0
0.0	1.0	2.0	3.0	4.0	5.0	5.0	5.0	6.0	6.0	7.0



0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
0.0	1.0	1.0	1.0	0.0	1.0	1.0	0.0	1.0	0.0
1.0	0.0	1.0	2.0	1.0	1.0	2.0	1.0	1.0	1.0
0.0	1.0	0.0	1.0	2.0	2.0	1.0	2.0	2.0	1.0
0.0	0.0	1.0	1.0	1.0	2.0	2.0	2.0	3.0	2.0
0.0	0.0	0.0	1.0	1.0	2.0	2.0	3.0	3.0	3.0
1.0	0.0	1.0	1.0	2.0	2.0	3.0	3.0	4.0	4.0
0.0	1.0	0.0	1.0	1.0	2.0	2.0	3.0	4.0	4.0
0.0	1.0	1.0	1.0	1.0	2.0	2.0	2.0	3.0	4.0
1.0	0.0	1.0	2.0	2.0	2.0	3.0	3.0	3.0	4.0

(a) An example of the EDR cost matrix with $\epsilon = 1$, where the white cells are members of the least cost alignment path.



0.77	2.31	2.45	3.21	3.39	5.58	5.77	6.24	6.86	7.1
1.22	1.86	2.01	2.77	2.94	5.13	5.33	5.79	6.74	6.98
2.42	1.56	1.7	2.46	2.64	4.83	5.02	5.49	6.43	6.67
2.59	1.73	1.87	2.63	2.81	5.0	5.19	5.66	6.6	6.84
3.15	2.29	2.43	3.2	3.37	5.56	5.76	6.22	7.17	7.41
3.84	2.98	3.12	3.89	4.06	6.25	6.45	6.91	7.86	8.1
5.64	4.1	4.24	5.0	5.18	7.37	7.56	8.03	8.97	9.21
5.7	4.16	4.31	5.07	5.24	7.44	7.63	8.09	9.04	9.28
5.99	4.45	4.59	5.35	5.53	7.72	7.92	8.38	9.33	9.56
7.62	6.09	6.23	6.99	7.16	9.36	9.55	10.01	10.96	11.2

(b) An example of ERP distance calculation using our example series, with $g = 0$.

Fig. 7 EDR and ERP example paths

The LCSS distance between **a** and **b** is

$$d_{\text{LCSS}}(\mathbf{a}, \mathbf{b}) = 1 - \frac{\text{LCSS}(\mathbf{a}, \mathbf{b})}{m}. \tag{12}$$

Figure 6 shows the cost match between our two example series. The longest sequence of matches is seven. These are the pairs $< (2, 1), (3, 2), (4, 4), (5, 5), (6, 8), (8, 8), (9, 10) >$.

3.5 Edit distance on real sequences (EDR)

LCSS was adapted in [17], where the edit distance on real sequences (EDR) is described. Like LCSS, EDR uses a distance threshold to define when two elements of a series match. However, rather than count matches and look for the longest sequence, ERP applies a (constant) penalty for non-matching elements where deletions, or gaps, are created in the alignment. Given a distance threshold, ϵ, the EDR distance between two points in series **a** and **b** is given by Algorithm 3. The EDR distance between **a** and **b** is then defined by Eq. 13.

Algorithm 3 EDR (**a**, **b** , (*both series of length m*), ϵ (*equality threshold*)

```
1: Let $E$ be an $(m+1) \times (m+1)$ matrix initialised to zero, indexed from zero.
2: for $i \leftarrow 1$ to $m$ do
3:     for $j \leftarrow 1$ to $m$ do
4:         if $|a_i - b_j| < \epsilon$ then
5:             $c \leftarrow 0$
6:         else
7:             $c \leftarrow 1$
8:         $match \leftarrow E_{i-1,j-1} + c$
9:         $insert \leftarrow E_{i-1,j} + 1$
10:        $delete \leftarrow E_{i,j-1} + 1$
11:        $E_{i,j} \leftarrow \min(match, insert, delete)$
   return $E_{m,m}$
```

$$d_{\text{EDR}}(\mathbf{a}, \mathbf{b}) = \text{EDR}(\mathbf{a}, \mathbf{b}, w, \epsilon) \tag{13}$$

At any step, elastic distances can use one of three costs: diagonal, horizontal or vertical, in forming an alignment. In terms of forming a subsequence from series **a**, we can characterise these as operations such as match (diagonal), deletion (horizontal) and insertion (vertical). Insertion to **a** can also be characterised as deletion from **b**, but we retain the match/delete/insert terminology for consistency and clarity. EDR does not satisfy triangular inequality, as equality is relaxed by assuming elements are equal when the distance between them is less than or equal to ϵ. EDR is very similar to LCSS, but it directly finds a distance rather than simply counting matches, thus removing the need for the subtraction in Eq. 12. The resulting cost matrix shown in Fig. 7a can easily be used to find either an alignment path or a common subsequence. EDR then characterises the three operations in DTW

3.6 Edit distance with real penalty (ERP)

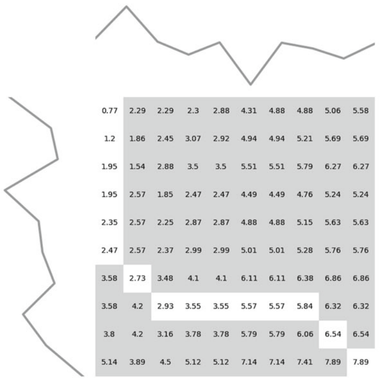
An alternative to EDR was proposed in [16], where edit distance with real penalty (ERP) was introduced. LCSS produces a subsequence that can have gaps between elements. For example, there is a gap between (3,2) and (4,4) in the subsequence shown in Fig. 6. ERP imposes a penalty for when gaps occur based on the distance to a constant parameter g. ERP uses $d(a, b) = \sqrt{(}(a-b)^2)$ for the pointwise distance rather than $d(a, b) = (a-b)^2$ used to find M for DTW. ERP calculates a cost matrix E that is more like DTW than LCSS: it describes a path alignment of two series based on edits rather than warping. The ERP distance between two series is described in Algorithm 4. The edge terms are initialised to a large constant value (lines 5 and 7). The cost matrix E is then either the cost of matching, $E_{i-1,j-1} + d(a_i, b_j)$, where $d(a_i, b_j)$ is the distance between two points or the cost of an inserting/deleting a term on either axis ($E_{i-1,j} + d(a_i, g)$ or $E_{i,j-1} + d(g, b_j)$). The ERP distance between **a** and **b** is given by Eq. 14.

$$d_{\text{ERP}}(\mathbf{a}, \mathbf{b}) = \text{ERP}(\mathbf{a}, \mathbf{b}, g, d) \tag{14}$$

ERP satisfies triangular inequality and is a metric. Figure 7b shows the cost matrix and resulting alignment for our example series with $g = 0$. $E_{1,1}$ is 0.774, which is simply the

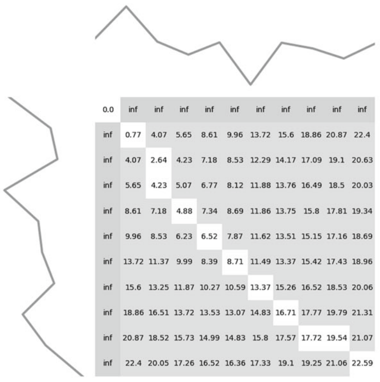
Algorithm 4 ERP (**a**, **b** (*both series of length m*), g, (*penalty value*), d, (*pointwise distance function*))

```
1: Let $E$ be an $(m+1) \times (m+1)$ matrix initialised to zero, indexed from zero.
2: for $i \leftarrow 1$ to $m$ do
3:     for $j \leftarrow 1$ to $m$ do
4:         if i = 0 then
5:             $E_{i,j} \leftarrow \sum_{k=1}^{m} d(b_k, g)$
6:         else if j = 0 then
7:             $E_{i,j} \leftarrow \sum_{k=1}^{m} d(a_k, g)$
8:         else
9:             $match \leftarrow E_{i-1,j-1} + d(a_i, b_j)$
10:            $insert \leftarrow E_{i-1,j} + d(a_i, g)$
11:            $delete \leftarrow E_{i,j-1} + d(g, b_j)$
12:            $E_{i,j} \leftarrow \min(match, insert, delete)$
   return $E_{m,m}$
```



0.77	2.29	2.29	2.3	2.88	4.31	4.88	4.88	5.06	5.58
1.2	1.86	2.45	3.07	2.92	4.94	4.94	5.21	5.69	5.69
1.95	1.54	2.88	3.5	3.5	5.51	5.51	5.79	6.27	6.27
1.95	2.57	1.85	2.47	2.47	4.49	4.49	4.76	5.24	5.24
2.35	2.57	2.25	2.87	2.87	4.88	4.88	5.15	5.63	5.63
2.47	2.57	2.37	2.99	2.99	5.01	5.01	5.28	5.76	5.76
3.58	2.73	3.48	4.1	4.1	6.11	6.11	6.38	6.86	6.86
3.58	4.2	2.93	3.55	3.55	5.57	5.57	5.84	6.32	6.32
3.8	4.2	3.16	3.78	3.78	5.79	5.79	6.06	6.54	6.54
5.14	3.89	4.5	5.12	5.12	7.14	7.14	7.41	7.89	7.89

(a) An example of MSM distance calculation using our example series.



0.0	inf	inf	inf	inf	inf	inf	inf	inf	inf	inf
inf	0.77	4.07	5.65	8.61	9.96	13.72	15.6	18.86	20.87	22.4
inf	4.07	2.64	4.23	7.18	8.53	12.29	14.17	17.09	19.1	20.63
inf	5.65	4.23	5.07	6.77	8.12	11.88	13.76	16.49	18.5	20.03
inf	8.61	7.18	4.88	7.34	8.69	11.86	13.75	15.8	17.81	19.34
inf	9.96	8.53	6.23	6.52	7.87	11.62	13.51	15.15	17.16	18.69
inf	13.72	11.37	9.99	8.39	8.71	11.49	13.37	15.42	17.43	18.96
inf	15.6	13.25	11.87	10.27	10.59	13.37	15.26	16.52	18.53	20.06
inf	18.86	16.51	13.72	13.53	13.07	14.83	16.71	17.77	19.79	21.31
inf	20.87	18.52	15.73	14.99	14.83	15.8	17.57	17.72	19.54	21.07
inf	22.4	20.05	17.26	16.52	16.36	17.33	19.1	19.25	21.06	22.59

(b) An example of TWE distance calculation using our example series.

Fig. 8 MSM and TWE example paths

square root of $M_{1,1}$. $E_{2,1}$ is the minimum of the two edge cases to the left (large constants) and $E_{1,1}+d(b_1, g)$, where $b_1 = 0.42$, so $d(0.17, 1) = 0.97$. Hence, $E_{2,1} = 0.774+0.424 = 1.2$.

3.7 Move–split–merge (MSM)

$$cost(x, y, z, c) = \begin{cases} c \textbf{ if } y \leq x \leq z \textbf{ or } y \geq x \geq z \\ c + min(|x-y|, |x-z|) \textbf{ otherwise}. \end{cases} \quad (15)$$

Move–split–merge (MSM) [67] is a distance measure that is conceptually similar to ERP. The core motivation for MSM is that the cost of insertion/deletion in ERP are based on the distance of the value from some constant value, and thus it prefers inserting and deleting values close to g compared to other values. Algorithm 5 shows that the major difference is in the deletion/insertion operations on lines 10 and 11. The move operation in MSM uses the absolute difference rather than the squared distance for matching in ERP. Insert cost in

Algorithm 5 MSM(**a**, **b** (*both series of length m*), *c* (*minimum cost*), *d*, (*pointwise distance function*))

```
1: Let $D$ be an $m \times m$ matrix initialised to zero.
2: $D_{1,1} = d(a_1, b_1)$
3: for $i \leftarrow 2$ to $m$ do
4:     $D_{i,1} = D_{i-1,1} + cost(a_i, a_{i-1}, b_1, c)$
5: for $i \leftarrow 2$ to $m$ do
6:     $D_{1,i} = D_{1,i-1} + cost(b_i, a_1, b + i - 1, c)$
7: for $i \leftarrow 2$ to $m$ do
8:     for $j \leftarrow 2$ to $n$ do
9:         $match \leftarrow D_{i-1,j-1} + d(a_i, b_j)$
10:        $insert \leftarrow D_{i-1,j} + cost(a_i, a_{i-1}, b_j, c)$
11:        $delete \leftarrow D_{i,j-1} + cost(b_j, b_{j-1}, a_i, c)$
12:        $D_{i,j} \leftarrow \min(match, insert, delete)$
   return $D_{m,m}$
```

ERP $d(a_i, g)$ is replaced by split operation $C(a_i, a_{i-1}, b_j, c)$, where C is the cost function given in Eq. 15. If the value being inserted, b_j, is between the two values a_i and a_{i-1} being split, the cost is a constant value c. If not, the cost is c plus the minimum deviation from the furthest point a_i and the previous point a_{i-1} or b_j. The delete cost of ERP $d(g, b_j)$ is replaced by the merge cost $C(b_j, b_{j-1}, a_i, c)$, which is simply the same operation on the second series. Thus, the cost of splitting and merging values depends on the value itself and adjacent values, rather than treating all insertions and deletions equally as with ERP. The MSM distance between **a** and **b** is given by Eq. 16 and illustrated in Fig. 8b. MSM satisfies triangular inequality and is a metric.

$$d_{\mathrm{MSM}}(\mathbf{a}, \mathbf{b}) = \mathrm{MSM}(\mathbf{a}, \mathbf{b}, c) \tag{16}$$

3.7.1 Time warp edit (TWE)

Introduced in [48], time warp edit (TWE) distance is an elastic distance measure described in Algorithm 6. It encompasses characteristics from both warping and editing approaches. The warping, called *stiffness*, is controlled by a parameter ν. Stiffness enforces a multiplicative penalty on the distance between matched points in a way that is similar to WDTW, where $\nu = 0$ gives no warping penalty. The stiffness is only weighted this way when considering the match option (line 11). For the delete and insert operations (lines 12 and 13), a constant stiffness (ν) and edit (λ) penalty are applied since the warping is considered from consecutive points in the same series. An example is shown in Fig. 8a.

$$d_{\mathrm{TWE}}(\mathbf{a}, \mathbf{b}) = \mathrm{TWE}(\mathbf{a}, \mathbf{b}, \nu, \lambda) \tag{17}$$

A summary of distance function parameters and their default values in our implementations is given in Table 2. DTW is sensitive to the window parameter [19] and large windows can cause pathological warpings. Based on experimental results [58], we set the default warping window to 0.2. WDTW uses a default scale parameter value for g of 0.05, based on results reported in [34]. LCSS and EDR both have an ϵ parameter that is a threshold for similarity. If the difference in two random variables is below ϵ, the observations are considered a match. The variability in parameter effects depending on the values of the series is one of the arguments for normalising all series. We set the default ϵ to 0.05. MSM has a single parameter, c, to represent the cost of the move–split–merge operation. This is set to 1 based

Algorithm 6 TWE($\mathbf{a}$, $\mathbf{b}$ (*both series of length* m), λ (*edit cost*), ν (*warping penalty factor*), d, (*pointwise distance function*))

```
1: Let $D$ be an $m + 1 \times n + 1$ matrix initialised to 0
2: $D_{0,0} = 0$
3: for $i \leftarrow 1$ to $m$ do
4:     $D_{i,0} = \infty$
5:     $D_{0,i} = \infty$
6: for $i \leftarrow 1$ to $m$ do
7:     for $j \leftarrow 1$ to $n$ do
8:         $match = D(i-1, j-1) + d(a_i, b_j) + d(a_{i-1}, b_{j-1}) + 2\nu(|i-j|)$
9:         $delete = D(i-1, j) + d(a_i, a_{i-1}) + \lambda + \nu$
10:        $insert = D(i, j-1) + d(b_j, b_{j-1}) + \lambda + \nu$
11:        $D(i, j) = \min(match, insert, delete)$
   return $D(m, n)$
```

Table 2 Summary of distance functions, their parameters and the default values

Acronym	Metric	Parameters	Default
DTW (Eq. 5)	No	Window $w \in [0, 1]$	$w = 0.2$
DDTW (Eq. 7)	No	Window $w \in [0, 1]$	$w = 0.2$
WDTW (Eq. 10)	No	$g \in [0, \infty)$	$g = 0.05$
DWDTW (Eq. 11)	No	$g \in [0, \infty)$	$g = 0.05$
LCSS (Eq. 12)	No	$x\epsilon \in [0, \infty)$	$\epsilon = 0.05$
ERP (Eq. 14)	Yes	$g \in [0 \ldots 1]$	$g = 0.05$
EDR (Eq. 13)	No	$\epsilon \in [0, \infty)$	$\epsilon = 0.05$
MSM (Eq. 16)	Yes	$c \in [0, \infty)$	$c = 1$
TWE (Eq. 17)	Yes	$\nu, \lambda \in [0, \infty)$	$\nu = 0.05, \lambda = 1$

on the original paper. TWE λ is analogous to the c parameter in MSM, so we also set it to one, whereas ν is related to the weighting parameter of WDTW, so we set it to 0.05.

3.8 Averaging time series

k-means clustering requires characterising a set of time series to form an exemplar. The standard approach for k-means is simply to find the mean of the current members of a cluster over time points. This is appropriate if the distance function is Euclidean distance since the average centroid is the series with the minimal Euclidean distance to members of the cluster. However, if cluster membership is assigned based on an elastic distance measure, the average centroid may misrepresent the elements of a cluster; it is unlikely to be the series that minimises the elastic distance to cluster members. Dynamic time warping with barycentre averaging (DBA) [55] (see Algorithm 7) was proposed to overcome this limitation in the context of DTW.

DBA is a heuristic to find a series that minimises the elastic distance to cluster members rather than the Euclidean distance. Starting with some initial centre, DBA works by first finding the warping path of series in the cluster onto the centre. It then updates the centre, by finding which points were warped onto each element of the centre for all elements of the cluster, then recalculating the centre as the average of these warped points.

Algorithm 7 DTW Barycentre Averaging ($\mathbf{c}$, *the initial average sequence*, $\mathbf{X_p}$, *p time series to average*.

```
1: Let dtw_path be a function that returns the a list of tuples that contain the indexes of the warping path
   between two time series.
2: Let $W$ be a list of empty lists, where $W_i$ stores the values in $\mathbf{X_p}$ of points warped onto centre point $c_i$.
3: for $x \in \mathbf{X_p}$ do
4:     $P \leftarrow dtw\_path(\mathbf{x}, \mathbf{c})$
5:     for $(i, j) \in P$ do
6:         $W_i \leftarrow W_i U x_j$
7: for $i \leftarrow 1$ to $m$ do
8:     $c_i \leftarrow mean(W_i)$
   return $c$
```

Suppose function $f(\mathbf{a}, \mathbf{b})$ returns the warping path of indexes

$$P =< (e_1, f_1), (e_2, f_2), \ldots, (e_s, f_s) >$$

generated by dynamic time warping. Given an initial centre $\mathbf{c} =< c_1, \ldots, c_m >$, DBA is described by Algorithm 7. It warps each series onto c (line 4), then from the warping path associates the value in the series with the value in the barycentre (lines 5 and 6). Once finished for all series, the average of values warped to each index of the centroid is taken. This is meant to better characterise the cluster members in the iterative partitioning of algorithms.

4 Methodology

Guided by related research, we specify the data we use, how we handle the data and the performance measures and hypothesis tests used to compare algorithms.

4.1 Data

We experiment with time series data in the University of California, Riverside (UCR) archive [18].[3] We restrict our attention to univariate time series, and in all experiments, we use 112 of the 128 datasets from the UCR time series archive. We exclude datasets containing series of unequal length or missing values. We also remove the Fungi data, which only provides a single train case for each class label. We report results using the six performance measures described in Sect. 4.2 on both the training sample and the test set.

Using the class labels to assess performance on some of these data may be unfair. For some problems, clustering algorithms naturally find clusters that are independent of the class labels but are nevertheless valid. For example, the GunPoint data set was created by a man and a woman drawing a gun from their belt or pretending to draw a gun. The class labels are Gun/No Gun. However, many clusterers find the Man/Woman clusters rather than Gun/No Gun. Without supervision, this is a perfectly valid clustering, but it will score approximately 50% accuracy since the man and woman cases are split evenly between Gun/No Gun. This is an inherent problem with attempting to evaluate exploratory, unsupervised algorithms by comparing them with what we know to be true a priori: if a clustering simply finds what we already know, its utility is limited. Furthermore, as observed in [39], some of the datasets

[3] https://timeseriesclassification.com/.

Table 3 Six clustering performance measures used in experimentation

Measure	Acronym
Clustering accuracy	CL-ACC
Rand index	RI
Adjusted Rand index	ARI
Mutual information	MI
Adjusted mutual information	AMI
Davies–Bouldin score	DB

have the same time series but with different labels. We aim to mitigate against these problems by using a large number of problems.

There are further problems with using the UCR data for algorithm analysis (see [26]): many of the data have been preprocessed with expert knowledge; the train/test sets have been hand crafted in some instances; there are no missing values in the data we use; all data are sampled at the same frequency; and the train sets are relatively small and the types of problem represent the research interests of the contributors. It is not a perfect model of the real world, nor representative of the type of problem faced by many data scientists in practice. Nevertheless, these faults should be put in context of the wider machine learning research. It is still common to see papers use the same 20 datasets from the UCI archive that have been employed since last century. We believe that bake offs using the UCR data still have value, but that we should continue to look to expand and diversify the archive.

4.2 Clustering metrics

Table 3 summarises the 6 performance measures we use in our evaluation. Clustering accuracy (CL-ACC), like classification accuracy, 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.

$$\text{CL-ACC}(\mathbf{y}, \hat{\mathbf{y}}) = \max_{\mathbf{s} \in \mathbf{S_k}} \frac{1}{|\mathbf{y}|} \sum_{i=1}^{|\mathbf{y}|} \begin{cases} 1, & y_i = \mathbf{s}(\hat{y}_i) \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

Checking every permutation like this is prohibitively expensive, however, and can be done more efficiently using combinatorial optimisation algorithms for the assignment problem. A contingency matrix of cluster assignments and class values is created and turned into a cost matrix by subtracting each value of the contingency matrix from the maximum value. Clusters are then assigned using an algorithm such as the Hungarian algorithm [38] on the cost matrix. If the class value of a case matches the assigned class value of its cluster, the prediction is deemed correct, else it is incorrect. As classes can only have one assigned cluster each, all cases in unassigned clusters due to a difference in a number of clusters and class values are counted as incorrect.

The Rand index (RI) works by measuring the similarity between two sets of labels. This could be between the labels produced by different clustering models (thereby allowing direct comparison) or between the ground truth labels and those the model produced. The rand index is the number of pairs that agree on a label divided by the total number of pairs.

One of the limiting factors of RI is that the score is inflated, especially when the number of clusters is increased. 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 (MI) is a function that measures the agreement of the two clusterings or a clustering and a true labelling, based on entropy. Normalised mutual information (NMI) rescales MI onto [0, 1], and adjusted mutual information (AMI) adjusts the MI to account for the class distribution.

The Davies–Bouldin index is an unsupervised measure we employ for tuning clusterers. It compares the between cluster variation with the inter cluster variation, with a higher score awarded to a clustering where there is good separation between clusters.

4.3 Related clustering comparisons

There have been several reviews and summaries of the TSCL field that compare algorithms on the UCR datasets, and we use these to guide our methodology. [9] compares five DTW and ED clusterers on 5 UCR datasets using the Rand Index. [33] compared eight variants of k-means, k-medoids clusterer and density peaks on the same 112 UCR data we also use. They combined train and test data, used raw series and evaluated algorithms with the adjusted Rand index. [2, 4] and [3] are literature reviews for TSCL that do not include results. [72] compares 10 clusterers on 36 time series from the UCR archive. They use Rand index and normalised mutual information to compare algorithms. They used the provided train and test splits on the raw data. [39] compared a range of deep learning-based algorithms on the UCR data using normalised mutual information, adjusted Rand index and accuracy as assessment criteria.

There are two important decisions to make about experimental design: whether to normalise all series to zero mean and unit variance and whether to merge the train and test data or train and test on separate data samples. The issue of whether to always normalise is an open question for TSC, since some discriminatory features may be in the scale or variance. Some argue that normalisation should always occur. For example, a 2012 paper that has been cited over 1000 times states that "*In order to make meaningful comparisons between two time series, both must be normalised*" [56]. However, in TSC whether to normalise or not is often treated as a parameter. We wish to control as many factors as possible in our experiments, so we perform identical experiments with both raw and normalised data.

Evaluating on unseen test data is essential for any classification comparison. The issue is less clear cut with clustering, which is used more as an exploratory tool than a predictive model. Many comparative studies (e.g. [33]) have combined the train and test data and performed evaluation on a single data set. More recent research has followed the required protocol for classification of evaluating estimators on data not used in training (e.g. [39]). We perform all training on the default train split provided in the archive and present the results on both the train set and test set. We have implemented the distance functions and clustering algorithms in the aeon toolkit. Details of the implementation and guidance on reproducing results are provided in "Appendix A".

To compare multiple clusterers on multiple datasets, we use the rank ordering of the algorithms for any given performance measure. We use an adaptation of the critical difference diagram [20], replacing the post hoc Nemenyi test with a comparison of all classifiers using pairwise Wilcoxon signed-rank tests, and cliques formed using the Holm correction recommended by [11, 25]. We use $\alpha = 0.05$ for all hypothesis tests. Critical difference diagrams such as those shown in Fig. 9 display the algorithms ordered by the average rank of the statistic in question and the groups of algorithms between which there is no significant difference (cliques). So, for example, in Fig. 9c, MSM has the lowest average rank of 3.8973 and is not in a clique, so has significantly better ARI than the other algorithms. TWE, ERP, WDTW and ED are all grouped into a clique, which means there is no pairwise significant difference between them. LCSS, EDR, WDDTW and DTW form another clique, and DDTW is significantly worse than all other algorithms (Fig. 10). For consistency with some of the related research, Fig. 11 shows the same results on train data. The pattern of results is the same as on the test data, although there is less significance in the results.

5 Results

We report a sequence of five sets of experiments designed to detect differences in performance between distance functions and between the two clustering algorithms, *k*-means and *k*-medoids clusterer. Firstly, in Sect. 5.1 we compare *k*-means clustering using 10 different distance functions. Secondly, in Sect. 5.2 we report the equivalent *k*-medoids clusterer results and we address the question of whether the same pattern of performance seen in *k*-means is observable in *k*-medoids clusterer. We also assess whether *k*-medoids clusterer is more effective than *k*-means on the UCR data. We conduct experiments in Sects. 5.1 and 5.2 on both normalised and raw data. Thirdly, in Sect. 5.3 we evaluate the relative effect of the clustering initialisation algorithm on the performance of the distance functions. Next, in Sect. 5.4 we investigate the performance of DTW in more detail, exploring the effect of the warping window on the clustering. Finally, in Sect. 5.5 we see whether we can improve performance through tuning distance parameters.

5.1 Elastic distances with *k*-means

The first set of experiments involves comparing alternative distance functions with *k*-means clustering using the arithmetic mean of series to find the centroids. Our primary aim is to investigate whether there are any significant differences between the measures when used on the UCR univariate classification datasets with both raw and normalised data. For each experiment, we ran *k*-means clustering with the default aeon parameters (random initialisation, maximum 300 iterations, 10 restarts, centroid averaging method is the mean) with Euclidean distance and the nine elastic distance measures, set up with the default parameters listed in Table 2. Figure 9 shows the summarised results on the test data using normalised data. Figure 10 displays the same results found using the raw data. Full results are available on the accompanying website, and implementation details with code examples are provided in “Appendix A” and on the associated GitHub repository.

The two derivative approaches are both significantly worse than their alternatives using the raw data, suggesting that clusters in the time domain better reflect the true classes. LCSS and EDR also perform poorly on all tests. This implies the simple edit thresholding is not sensitive enough to find clustering that represents the class labels. The best overall performing measures are MSM and TWE. These measures are similar, in that they combine elements of both warping and editing.

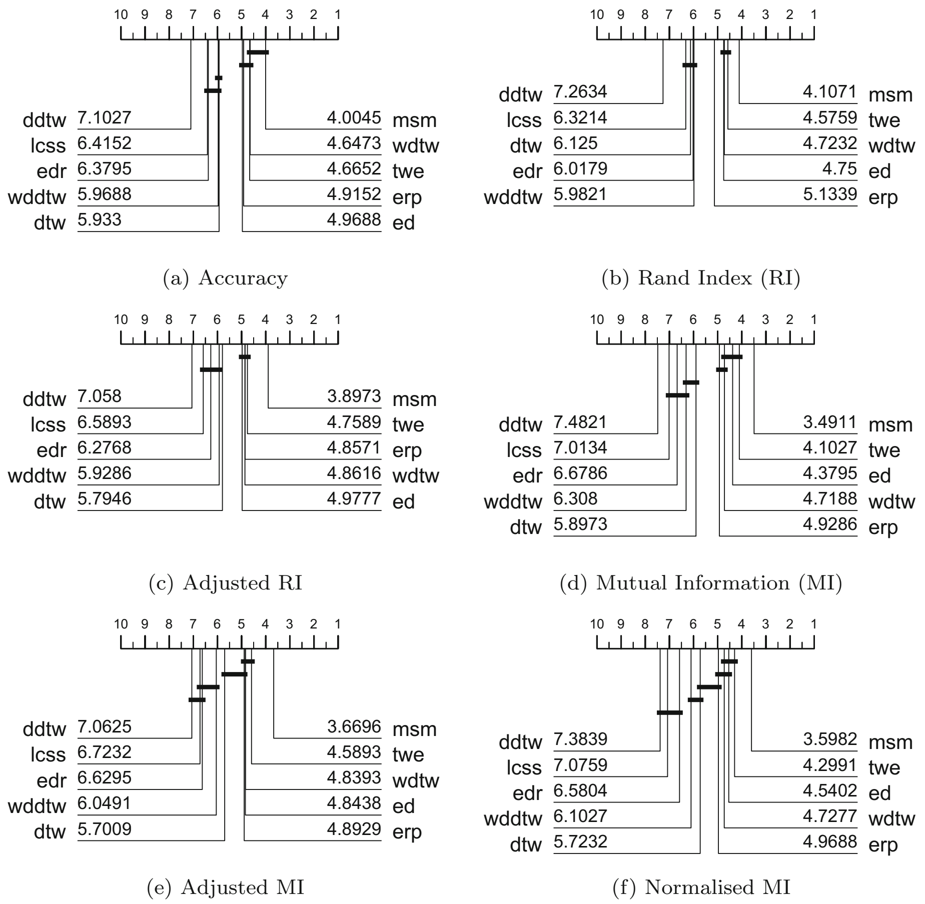


Fig. 9 Critical difference diagrams for k-means clustering using nine different elastic distance functions on 112 normalised UCR problems (normalised data) using six performance measures. Results are on the test data

5.2 Elastic distances with *k*-medoids clusterer

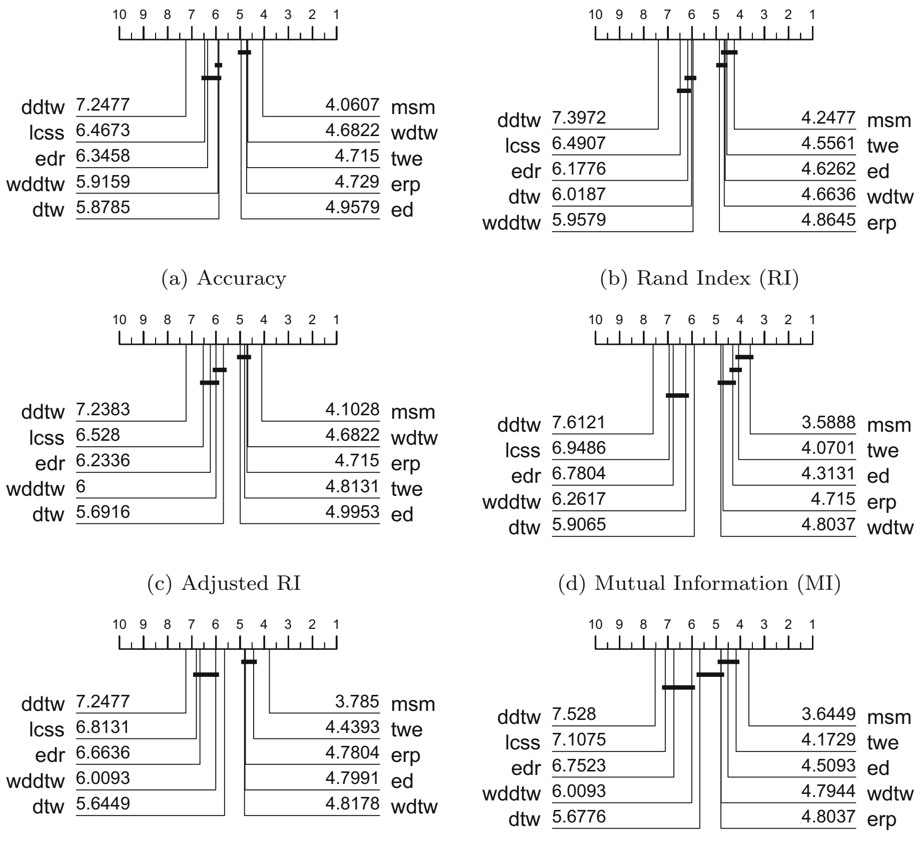
Using standard centroids means that the averaging method is not related to the distance measure used unless employing Euclidean distance. This disconnect between the clustering stages may account for the poor performance of many of the distance measures, in particular relative to k-means with Euclidean distance. Figure 12 shows the ranked performance summary for ten distance measures using k-medoids clusterer rather than k-means. The pattern of performance is broadly the same as with k-means (Fig. 9) with some notable differences: DTW is now no longer worse than Euclidean; ERP performs much better, and there is no overall difference between ERP, TWE and MSM as the top performing algorithms. The top performing distance functions all involve an explicit penalty for warping. As with k-means, this indicates that regularisation on path length produces better clusters on average. Figure 13 shows the equivalent results when using the raw data.

Also of interest is the relative performance between k-means and k-medoids clusterer. Table 4 lists the average accuracy over all problems of k-means, k-medoids clusterer and

Table 4 Accuracy and Davies–Bouldin averaged over 112 problems for *k*-means and *k*-medoids clustering on normalised data

	Accuracy			Davies–Bouldin		
Distance	*k*-Means (%)	*k*-medoids clusterer (%)	Difference (%)	*k*-Means	*k*-medoids clusterer	Difference
MSM	**54.16**	**55.69**	1.54	2.434	4.002	1.568
TWE	52.85	55.63	2.78	2.337	3.526	1.189
ERP	50.89	54.83	3.94	2.811	3.887	1.076
WDTW	52.25	53.58	1.33	2.437	3.447	1.01
DTW	49.08	52.96	3.88	2.602	3.982	1.38
ED	51.78	51.40	**−0.38**	**2.236**	**2.366**	**0.13**
DDTW	42.57	50.22	7.65	3.866	6.073	2.207
WDDTW	46.99	49.55	2.57	4.653	5.551	0.898
LCSS	45.76	49.88	4.13	5.624	6.467	0.843
EDR	45.20	49.70	4.50	6.711	6.936	0.225

Bold in the table are the best performing algorithm



(a) Accuracy

(b) Rand Index (RI)

(c) Adjusted RI

(d) Mutual Information (MI)

(e) Adjusted MI

(f) Normalised MI

Fig. 10 Critical difference diagrams for k-means clustering using nine different elastic distance functions on 112 UCR problems (raw data) using six performance measures. Results are on derived from the test files

single cluster predictions for the ten distance measures. Accuracy increases for all distances except Euclidean. This indicates that k-medoids clusterer is a stronger benchmark for TSCL algorithms than k-means on the UCR data, irrespective of distance measure. Table 4 also shows the aggregated unsupervised Davies–Bouldin (DB) scores for k-means and k-medoids clusterer which we later use in tuning. Small values are better for DB, so the results tell a different picture. We believe this is because DB is closely related to k-means, in that it averages distances within a cluster. k-means is designed to optimise a measure similar to DB. However, Table 4 illustrates this does not always lead to more accurate clustering algorithms.

5.3 Clustering algorithm initialisation

One source of variation could be our implementation and parameterisation of the clustering algorithms. Both implementations of k-means and k-medoids clusterer are simple algorithms based on the scikit-learn implementation of k-means. The three possible parameters that might affect performance are the maximum number of iterations given to the clustering algorithm

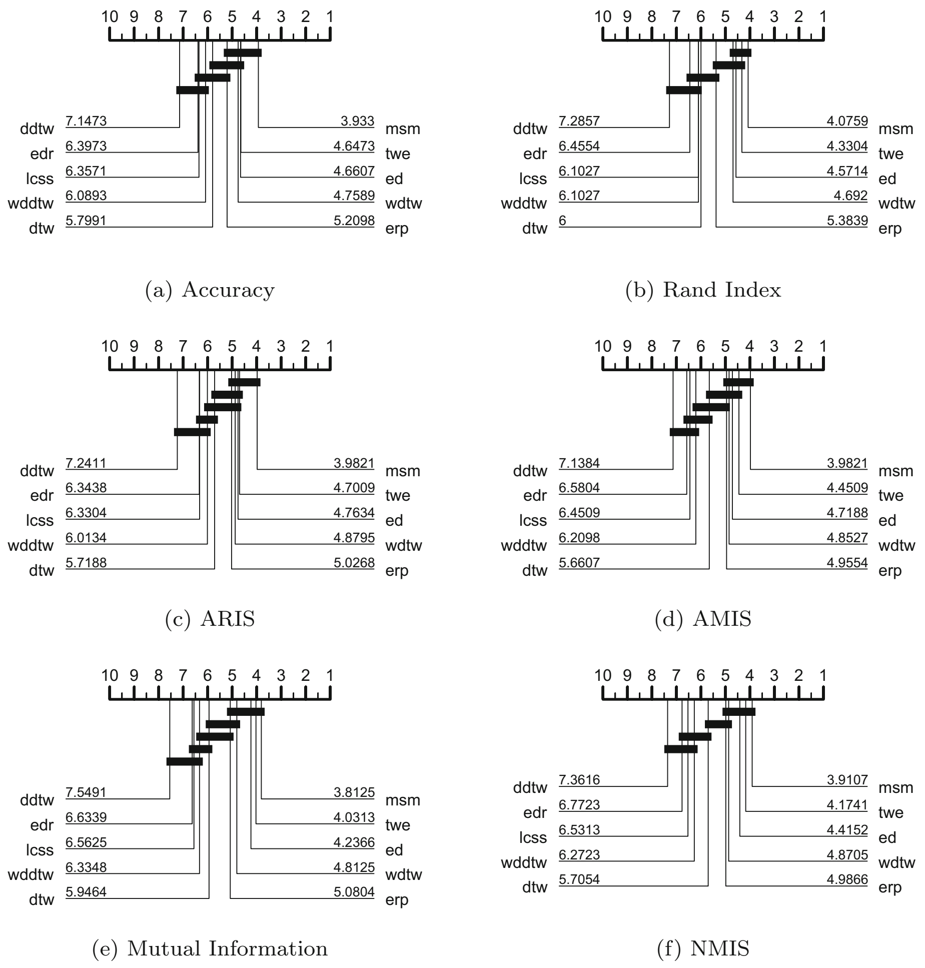
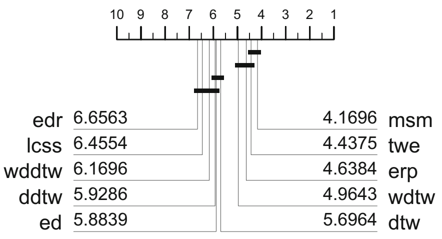
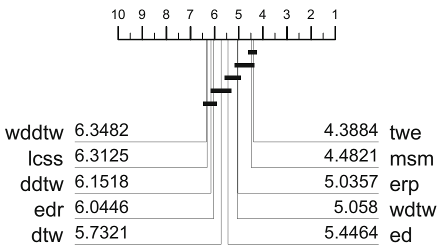


Fig. 11 Train results for k-means clustering with 10 different distance measures using normalised data

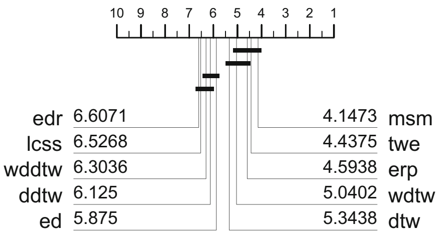
(defaults to 300), the initialisation algorithm (default to random) and the number of initialisation restarts (defaults to 10). Changing the maximum number of iterations is unlikely to improve performance. For most datasets/distances, convergence happens in under 20 iterations. Given tslearn defaults to 30 iterations, increasing beyond 300 is unlikely to have a significant effect. Restarting 10 times is computationally intensive, and increasing this further is also unlikely to improve performance. On the other hand, clustering using the alternating k-medoids clusterer algorithm is known to suffer from the initialisation problem [27] (see Sect. 2.1). We evaluate whether the difference in performance we have observed could have been caused by the initialisation algorithm. We repeat our experiments on the normalised data for DTW, Euclidean and MSM using random, Forgy and $k$means++ initialisation with both k-means and k-medoids clusterer. Our primary interest is the variation caused by the distance function, so we present the relative performance of the three distances for alternative initialisation techniques in Fig. 14. The ordering is the same throughout: MSM is better than ED, which is better than DTW. We conclude that the results observed in Sects. 5.1 and 5.2 are broadly independent of the initialisation algorithm.



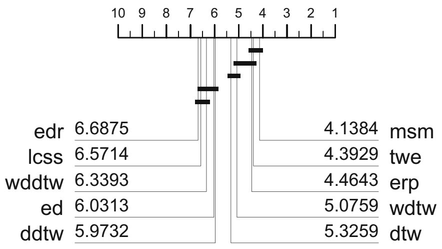
(a) Accuracy



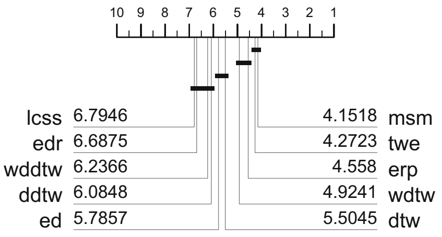
(b) Rand Index



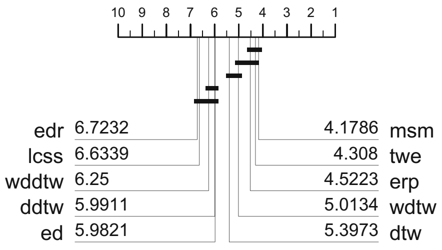
(c) ARIS



(d) AMIS

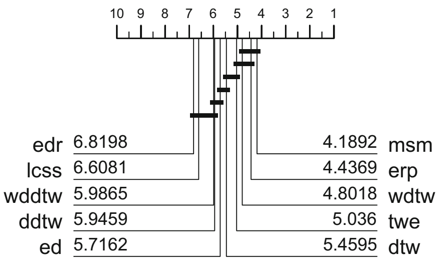


(e) Mutual Information

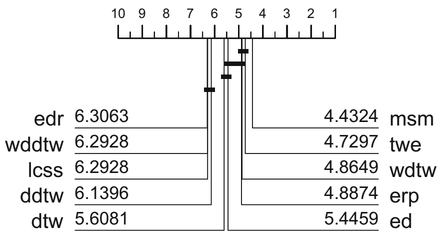


(f) NMIS

Fig. 12 Critical difference diagrams for k-medoids clustering with ten distance measures assessed on the test data (normalised data)

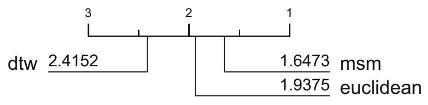


(a) Accuracy

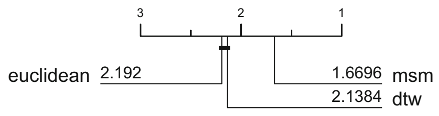


(b) Rand Index

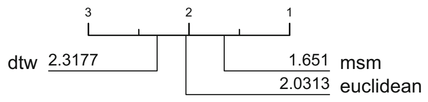
Fig. 13 Critical difference diagrams for k-medoids clustering with ten distance measures assessed on the test data (raw data)



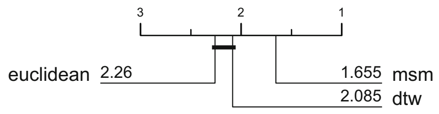
(a) Random k-means



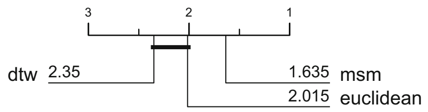
(b) Random k-medoids



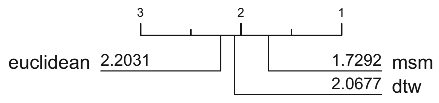
(c) Forgy k-means



(d) Forgy k-medoids



(e) kmeans++ k-means



(f) kmeans++ k-medoids

Fig. 14 Critical difference diagrams for alternative clustering initialisation algorithms

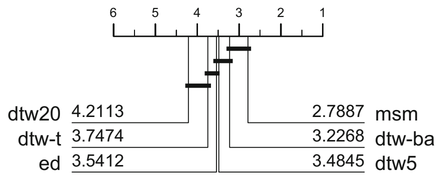
5.4 DTW warping window

In Sect. 5.1, we observed that DTW with a window of 20% of the series length is significantly worse than ED. k-means with DTW is widely used as a base line for time series clustering [33]. ED is a special case of DTW with window size of zero, so the finding is counter to our expectations and merits further investigation. It is even more notable when we observe that WDTW is significantly better than DTW. WDTW, MSM, TWE and ERP all give an explicit penalty for warping. The DTW penalty for warping is implicit (warping means a longer path), and the results indicate that this allows for more warping than is desirable for clustering.

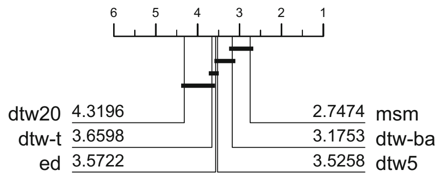
To test whether this result was an artefact of our implementation, we reran the clustering experiments using the Java toolkit tsml[4] version of DTW in conjunction with the WEKA k-means clusterer, and found no significant difference in the results. Furthermore, we have checked that the aeon DTW distance implementation produces the same distances as other implementations listed in Table 15. Begum et al. [10] also found that for clustering, ED outperforms DTW.

If our implementation is correct, then perhaps the poor performance is due to our experimental set up. Window size is the key parameter for k-means DTW. We initially experimented with k-means with full window and found it performed worse than DTW constrained to 20% (often called cDTW [19]). However, a window size of 5% is also popular in the literature [53]. We first repeat our experiments with a 5% window size. We then assess whether tuning the window size improves performance. Finally, we examine whether using an alternative averaging algorithm for DTW makes a significant difference to DTW.

[4] https://github.com/time-series-machine-learning/tsml-java.



(a) Adjusted Rand Index



(b) Normalised Mutual Information

Fig. 15 Critical difference diagrams for k-means clustering with the following DTW variants: DTW 20% window (dtw20); 5% window (dtw5); and a tuned window (dtw-t) and centres found with barycentre averaging (dtw-ba). Euclidean distance (ed) and move–split–merge (msm) are also included for reference

5.4.1 Smaller maximum DTW warping window

Figure 15 demonstrates that using a smaller maximum warping window improves the performance of k-means DTW to the point that it is no longer significantly worse than k-means with ED. Figure 16 shows the scatter plots of the accuracy and rand index of DTW with window size 5% against a 20% window and against ED. A smaller window is better, but using no window is equally as effective. DTW with a small window is a better approximation of ED. These results suggest DTW is not adding much value to the clustering on the UCR data.

5.4.2 Tuning the DTW warping window

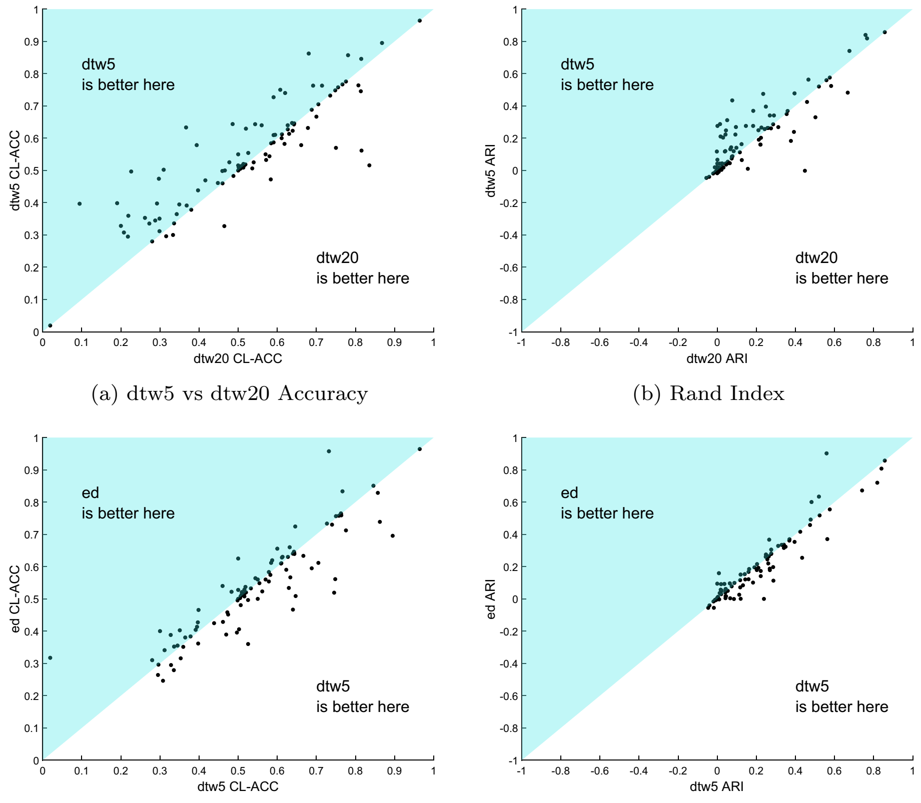
Tuning significantly improves the performance of DTW 1-NN classification, so it is possible that it may also improve DTW-based clustering. To tune a clusterer we need an unsupervised cluster assessment algorithm. We use the Davies–Bouldin score since it is popular and available in scikit-learn. We evaluate DTW for twenty possible windows on even intervals in the range $[0, 1)$ and use the window with the lowest Davies–Bouldin score for our final clustering (favouring small windows if scores are tied).

Figure 17 shows that tuning in this way (dtw-t) is better than using a fixed window of size 20% (dtw20), but does not improve on DTW with a fixed window of size 5% (dtw5). The tuned clusterer is not significantly worse than ED and is significantly worse than MSM. It is possible that a more fine grained tuning would yield better performance. However, tuning takes significant computation and it appears that all the extra effort is simply achieving a better approximation of ED.

5.4.3 *k*-means DBA

Using medoids mitigates the problem of averaging centres with k-means. DTW barycentre averaging (DBA), described in Sect. 3.8, has also been proposed as a means of improving centroid finding for k-means DTW. We have repeated our experiments with the same k-means set up, but centroids were found with the original DBA described in Algorithm 7. Figure 18a shows that DBA does indeed significantly improve DTW, a result that reproduces the findings in [55].

However, Fig. 18a illustrates that it is not significantly different to k-medoids clusterer DTW. Furthermore, Fig. 19 shows that k-means with DBA is significantly worse than the top clique, composed of k-medoids clusterer MSM, k-means MSM and k-medoids clusterer TWE.



(a) dtw5 vs dtw20 Accuracy (b) Rand Index

(c) dtw5 vs ed Accuracy (d) Rand Index

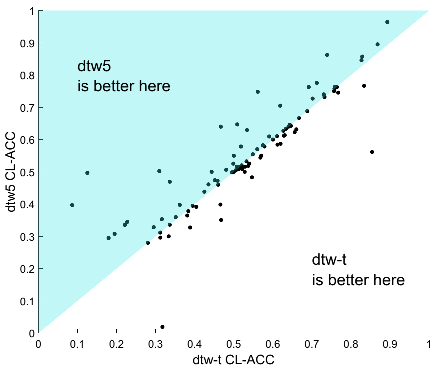
Fig. 16 Scatter plots of accuracy and rand index for 5% window DTW versus 20% window and Euclidean distance

Our implementation of DBA is faithful to the original (Fig. 20). An alternative version of DBA was described in [63]. This version is implemented in the tslearn toolkit. We have wrapped the tslearn *k*-means DBA algorithm into the aeon toolkit and reran this version. Figure 21 compares the performance of two DBA versions with MSM *k*-medoids clusterer and *k*-means.

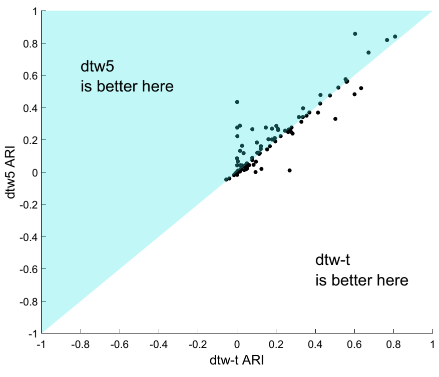
Table 5 summarises the time taken to run experiments. We ran experiments on our HPC cluster, so timing results are indicative only. The differences are fairly small. However, we note that of the two best performing algorithms, *k*-medoids clusterer MSM and TWE, *k*-medoids clusterer MSM is the faster.

5.5 Tuning the distance functions

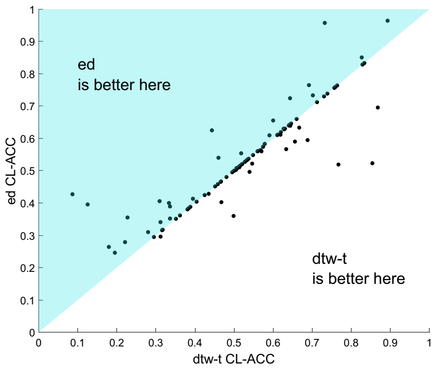
Clustering performance could also be improved by tuning the distance functions (as we did with DTW in Sect. 5.4.2). We tune four distance functions using *k*-medoids clustering on parameter ranges given in Table 6 to determine whether it improved performance. The choice of ranges is taken from [44] which in turn took suggested ranges from the original papers. Tuning is also a way of demonstrating the sensitivity of a distance measure to the parameters:



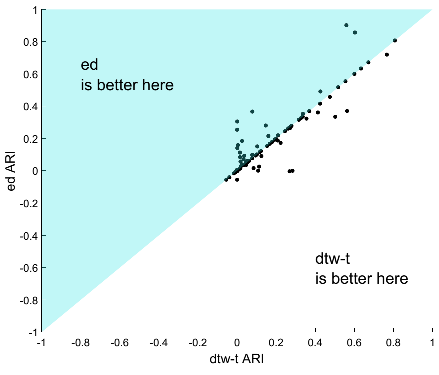
(a) dtw 5% (dtw5) vs dtw tuned (dtw-t) tuned accuracy



(b) Rand Index

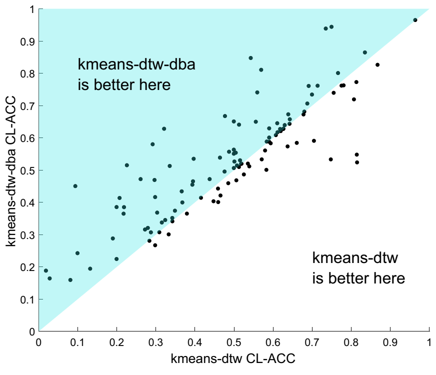


(c) Euclidean distance (ed) vs dtw tuned (dtw-t) tuned accuracy

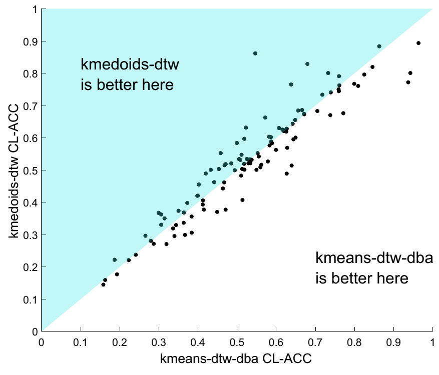


(d) Rand Index

Fig. 17 Scatter plots of accuracy and rand index for tuned DTW versus 5% window and Euclidean distance



(a) DBA wins 68 and loses 36 (8 ties)



(b) DBA wins 57 and loses 44 (11 ties)

Fig. 18 Scatter plots of accuracy for DTW-based k-means, with DBA against standard averaging (**a**) and against k-medoids clusterer (**b**)

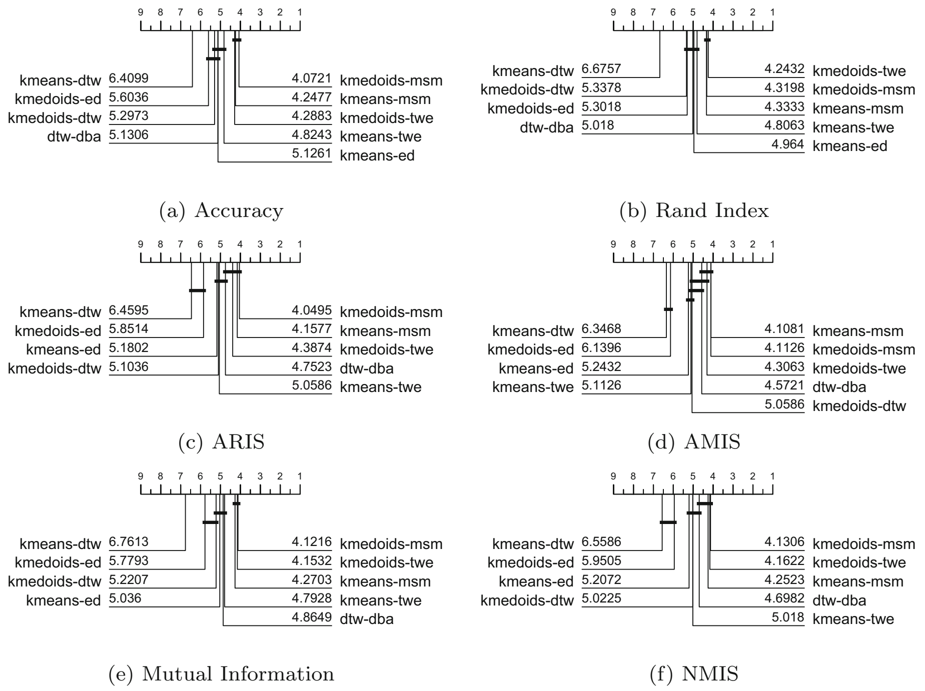
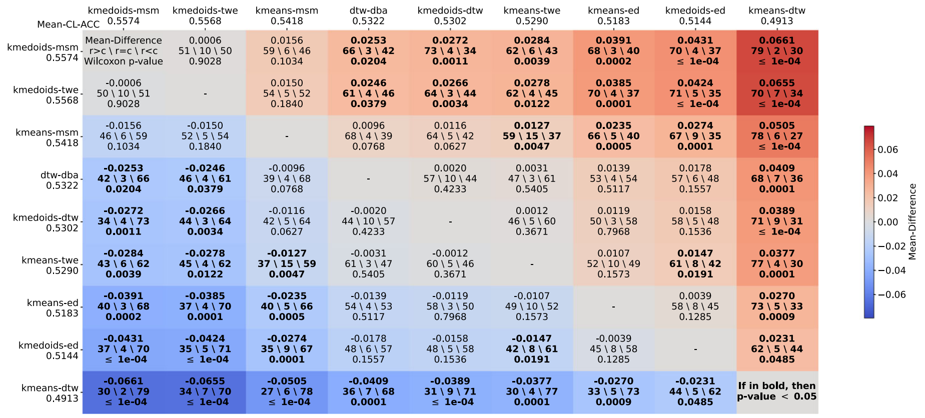


Fig. 19 Critical difference diagrams for the best performing *k*-means and *k*-medoids clusterer distances, and *k*-means with DBA (dtw-dba)



Mean-CL-ACC	kmedoids-msm 0.5574	kmedoids-twe 0.5568	kmeans-msm 0.5418	dtw-dba 0.5322	kmedoids-dtw 0.5302	kmeans-twe 0.5290	kmeans-ed 0.5183	kmedoids-ed 0.5144	kmeans-dtw 0.4913
kmedoids-msm 0.5574	Mean-Difference r>c \ r=c \ r<c Wilcoxon p-value	0.0006 51 \ 10 \ 50 0.9028	0.0156 59 \ 6 \ 46 0.1034	**0.0253 66 \ 3 \ 42 0.0204**	**0.0272 73 \ 4 \ 34 0.0011**	**0.0284 62 \ 6 \ 43 0.0039**	**0.0391 68 \ 3 \ 40 0.0002**	**0.0431 70 \ 4 \ 37 ≤ 1e-04**	**0.0661 79 \ 2 \ 30 ≤ 1e-04**
kmedoids-twe 0.5568	-0.0006 50 \ 10 \ 51 0.9028	-	0.0150 54 \ 5 \ 52 0.1840	**0.0246 61 \ 4 \ 46 0.0379**	**0.0266 64 \ 3 \ 44 0.0034**	**0.0278 62 \ 4 \ 45 0.0122**	**0.0385 70 \ 4 \ 37 0.0001**	**0.0424 71 \ 5 \ 35 ≤ 1e-04**	**0.0655 70 \ 7 \ 34 ≤ 1e-04**
kmeans-msm 0.5418	-0.0156 46 \ 6 \ 59 0.1034	-0.0150 52 \ 5 \ 54 0.1840	-	0.0096 68 \ 4 \ 39 0.0768	0.0116 64 \ 5 \ 42 0.0627	**0.0127 59 \ 15 \ 37 0.0047**	**0.0235 66 \ 5 \ 40 0.0005**	**0.0274 67 \ 9 \ 35 0.0001**	**0.0505 78 \ 6 \ 27 ≤ 1e-04**
dtw-dba 0.5322	**-0.0253 42 \ 3 \ 66 0.0204**	**-0.0246 46 \ 4 \ 61 0.0379**	-0.0096 39 \ 4 \ 68 0.0768	-	0.0020 57 \ 10 \ 44 0.4233	0.0031 47 \ 3 \ 61 0.5405	0.0139 53 \ 4 \ 54 0.5117	0.0178 57 \ 6 \ 48 0.1557	**0.0409 68 \ 7 \ 36 0.0001**
kmedoids-dtw 0.5302	**-0.0272 34 \ 4 \ 73 0.0011**	**-0.0266 44 \ 3 \ 64 0.0034**	-0.0116 42 \ 5 \ 64 0.0627	-0.0020 44 \ 10 \ 57 0.4233	-	0.0012 46 \ 5 \ 60 0.3671	0.0119 50 \ 3 \ 58 0.7968	0.0158 58 \ 5 \ 48 0.1536	**0.0389 71 \ 9 \ 31 ≤ 1e-04**
kmeans-twe 0.5290	**-0.0284 43 \ 6 \ 62 0.0039**	**-0.0278 45 \ 4 \ 62 0.0122**	**-0.0127 37 \ 15 \ 59 0.0047**	-0.0031 61 \ 3 \ 47 0.5405	-0.0012 60 \ 5 \ 46 0.3671	-	0.0107 52 \ 10 \ 49 0.1573	**0.0147 61 \ 8 \ 42 0.0191**	**0.0377 77 \ 4 \ 30 0.0001**
kmeans-ed 0.5183	**-0.0391 40 \ 3 \ 68 0.0002**	**-0.0385 37 \ 4 \ 70 0.0001**	**-0.0235 40 \ 5 \ 66 0.0005**	-0.0139 54 \ 4 \ 53 0.5117	-0.0119 58 \ 3 \ 50 0.7968	-0.0107 49 \ 10 \ 52 0.1573	-	0.0039 58 \ 8 \ 45 0.1285	**0.0270 73 \ 5 \ 33 0.0009**
kmedoids-ed 0.5144	**-0.0431 37 \ 4 \ 70 ≤ 1e-04**	**-0.0424 35 \ 5 \ 71 ≤ 1e-04**	**-0.0274 35 \ 9 \ 67 0.0001**	-0.0178 48 \ 6 \ 57 0.1557	-0.0158 48 \ 5 \ 58 0.1536	**-0.0147 42 \ 8 \ 61 0.0191**	-0.0039 45 \ 8 \ 58 0.1285	-	**0.0231 62 \ 5 \ 44 0.0485**
kmeans-dtw 0.4913	**-0.0661 30 \ 2 \ 79 ≤ 1e-04**	**-0.0655 34 \ 7 \ 70 ≤ 1e-04**	**-0.0505 27 \ 6 \ 78 ≤ 1e-04**	**-0.0409 36 \ 7 \ 68 0.0001**	**-0.0389 31 \ 9 \ 71 ≤ 1e-04**	**-0.0377 30 \ 4 \ 77 0.0001**	**-0.0270 33 \ 5 \ 73 0.0009**	**-0.0231 44 \ 5 \ 62 0.0485**	**If in bold, then p-value < 0.05**

Fig. 20 Heat map, as described in [28], summarising the results used to generate Fig. 19

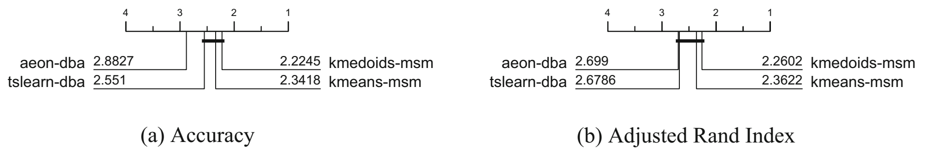


Fig. 21 Comparison of performance of two DBA implementations and *k*-medoids clusterer and *k*-means with MSM distance

Table 5 Run time (in hours) average, maximum and total over 112 problems

Distance	Mean	Max	Total
ED	0.003	0.046	0.341
DTW	3.272	53.579	366.514
WDTW	4.811	95.916	538.815
MSM	4.536	70.406	508.059
EDR	5.044	81.628	564.926
LCSS	5.063	82.803	567.047
DDTW	5.393	85.351	603.973
ERP	5.480	90.075	613.765
WDDTW	5.196	86.864	581.994
TWE	6.417	117.22	718.650
aeon-dba	8.112	116.62	900.514
tslearn-dba	9.724	159.79	1080.43

Table 6 Parameter ranges for tuning distance functions with k-means

Distance	Tuned parameter values
DTW	$w \in \{0.0, 0.01, \ldots, 0.19\}$
WDTW	$g \in \{0.0, 0.05, \ldots, 0.95\}$
MSM	$c \in \{0.0, 0.25, \ldots, 4.75\}$
ERP	$g \in \{0.0, 0.2, \ldots, 1.8\}$

20 parameter values or parameter value ranges are evaluated for each function on an evenly spaced grid

if tuning makes no significant difference to using the default parameters, we would consider the clusterer robust to the parameter and the default value an appropriate one.

6 Performance analysis

The UCR archive is a diverse collection of datasets, and whilst overall performance gives some indication as to good default benchmarks to use, it does not provide insight into the best approaches for data characteristics or the data domain.

The range of the number of classes/clusters in the archive is from 2 to 60. To aid analysis, we group datasets into four groups of datasets with: 2 clusters (Group A, consisting of 40 datasets); 3–5 clusters (Group B with 33 datasets); 6–10 clusters (Group C, 19 datasets); and 11 or more clusters (Group D, 13 datasets). Table 7 shows the average accuracy rank by group for the k-means and k-medoids clusterer results on normalised data presented in Sect. 5. MSM and TWE both improve relatively with more clusters. The two edit distance algorithms EDR and LCSS get worse with more clusters as do the derivative methods, leading to an improvement in ED.

Series length varies from 15 to 3000. We can group these into problems with series lengths less than 200 (Group A, 40 problems), 201–500 (Group B, 31 problems), 501–1000 (Group C, 20 problems) and >1000 (Group D, 21 problems). Table 8 breaks down the ranks by series length. There is no discernable pattern with series length. It does not seem to be a major factor in performance.

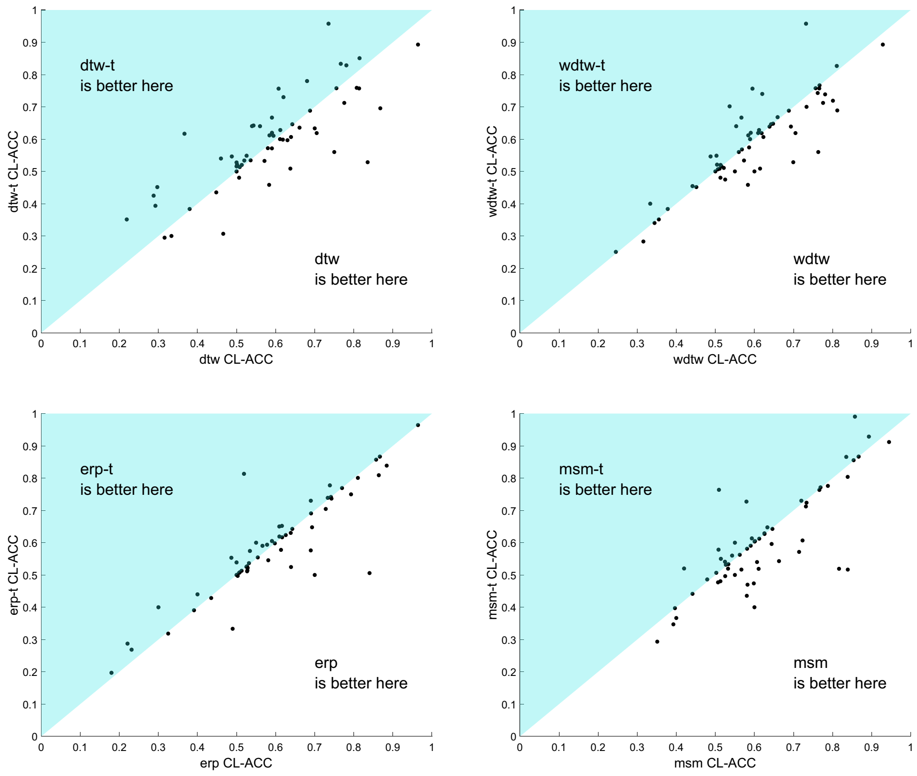


Fig. 22 Scatter plots of tuned versus untuned *k*-means clustering algorithms

Table 7 Average rank performance split by the number of clusters

	k-means				*k*-medoids clusterer			
	A	B	C	D	A	B	C	D
MSM	4.85	4.29	**2.92**	2.9	4.75	**4.32**	3.97	**3.08**
WDTW	5.83	**4.23**	4.39	3.33	4.66	4.52	4.19	4.15
TWE	5.7	4.86	4.22	**2.85**	**4.55**	4.61	4.5	4.98
ERP	**4.78**	5.14	3.94	5.75	6.08	4.82	**3.39**	4.18
ED	5.53	5.21	5	3.33	6.41	5.55	4.83	5.18
DTW	6.04	5.83	5.53	6.15	5.69	6.06	6.42	5.33
WDDTW	5.46	6.45	6.06	5.95	5.44	5.68	6.75	6.58
EDR	5.55	5.79	7.75	7.85	5.65	6.32	6.64	6.5
LCSS	5.69	5.82	7.39	8	5.78	6.5	6.75	7.6
DDTW	5.59	7.38	7.81	8.9	6	6.64	7.56	7.45

Bold in the table are the best performing algorithm

Table 8 Average rank performance split by the length of the series

	k-means				*k*-medoids clusterer			
	A	B	C	D	A	B	C	D
MSM	**3.81**	**3.91**	**4.72**	4.05	4.30	**3.81**	4.58	**4.07**
WDTW	5.30	4.06	4.86	3.90	4.58	5.02	**3.53**	4.19
TWE	5.03	4.47	5.56	**3.45**	**4.01**	4.58	4.40	6.14
ERP	4.79	4.80	4.86	5.50	5.06	4.56	5.90	4.48
ED	5.08	4.89	5.14	4.48	5.61	5.34	5.73	6.36
DTW	5.44	6.42	6.28	5.70	6.01	6.39	5.58	5.19
WDDTW	5.66	6.48	6.67	5.20	5.80	6.19	6.15	5.57
EDR	6.73	6.17	5.03	7.23	5.60	5.92	7.60	6.26
LCSS	6.60	6.18	5.28	7.48	6.85	6.63	5.53	6.33
DDTW	6.58	7.61	6.61	8.03	7.18	6.56	6.03	6.40

Bold in the table are the best performing algorithm

Table 9 Average rank performance split by the length of the number of training cases

	k-means				*k*-medoids clusterer			
	A	B	C	D	A	B	C	D
MSM	**4.45**	**3.94**	**3.31**	**3.83**	4.79	4.20	**3.63**	**3.61**
WDTW	4.84	5.11	4.03	3.88	4.96	**3.92**	4.60	4.04
TWE	5.25	4.86	3.67	4.08	**4.59**	5.04	4.50	4.52
ERP	4.51	5.18	4.97	5.33	5.66	5.02	4.23	4.35
ED	5.44	5.52	3.94	3.98	5.94	6.64	4.73	5.35
DTW	6.40	5.45	5.89	5.70	6.50	5.62	6.25	4.54
WDDTW	6.00	5.29	6.53	6.73	5.85	6.16	5.65	6.30
EDR	5.28	6.97	7.44	6.65	5.45	6.62	6.62	6.72
LCSS	5.83	6.32	7.28	6.80	5.49	5.68	7.15	8.11
DDTW	7.01	6.36	7.94	8.05	5.78	6.10	7.63	7.46

Bold in the table are the best performing algorithm

Train set size may also influence performance. The UCR data range in the train set size from 16 to 8926. We spit data into the following four groups: less than 100 training cases (Group A, 41 datasets); between 100 and 299 train cases (Group B, 24 datasets); 300 to 499 cases (Group C, 25 datasets); and 500 or more cases (Group D, 22 datasets). Table 9 summarises the ranks of the distance functions split into these groups. As the number of training cases increases, the relative difference between MSM, WDTW and TWE increases. This is more pronounced with *k*-medoids clusterer MSM, which is clearly better with 300 or more training instances.

Finally, the datasets have been classified as coming from different problem types. Table 10 describes the seven categories. These categories are useful, but not definitive. For example, there is an overlap between DEVICE, SENSOR and MOTION. However, breaking down by problem type can yield insights. Tables 11 and 12 show the breakdown of ranks by these problem types. MSM performs best on DEVICE, ECG, IMAGE and SIMULATED. These are all categories where some phase shift within the class would be expected. ED is the best at SPECTRO. Spectrograms are ordered series in the frequency domain, and we would expect the complexity of these problems to arise through noise and dimension rather than phase

Table 10 Dataset categories

Category	Number	Meaning
DEVICE	10	Household device electricity usage
ECG	7	Single-channel electrocardiogram
IMAGE	32	Images mapped onto 1-D series
MOTION	18	Single-channel activity recognition
SENSOR	24	Non-human sensor devices
SIMULATED	9	Simulated data
SPECTRO	12	One dimensional spectrogram

Table 11 Average rank performance of *k*-means on problems split by the problem domain

	DEVICE	ECG	IMAGE	MOTION	SENSOR	SIMULATED	SPECTRO
MSM	**3.35**	**2.71**	**3.76**	**3.57**	4.90	**3.33**	5.55
WDTW	4.50	5.29	4.55	4.41	**4.82**	3.56	5.68
TWE	5.00	4.64	4.88	4.00	4.98	4.78	4.91
ERP	3.75	3.57	3.83	5.95	4.86	5.78	5.59
ED	6.10	5.93	5.71	**3.57**	5.66	4.61	**4.32**
DTW	6.65	7.07	6.00	6.26	5.72	4.00	5.64
WDDTW	6.95	6.79	5.83	6.34	5.02	6.78	5.32
EDR	6.40	4.36	6.95	7.05	6.02	7.11	5.00
LCSS	5.00	6.14	7.17	6.88	6.08	5.94	6.36
DDTW	7.30	8.50	6.31	6.97	6.94	9.11	6.64

Bold in the table are the best performing algorithm

shift. ED also does surprisingly well at MOTION problems with *k*-means, but less well with *k*-medoids clusterer.

The results show that MSM is generally the best performing algorithm. Given the prevalence of DTW-based clustering, this is perhaps surprising and the reasons to merit further investigation.

One explanation could be it is an artefact of the clustering algorithm. There are many examples of where hierarchical clustering with DTW seems to produce more intuitive clusterings than Euclidean distance [19]. Partitioning clustering has the advantage over hierarchical clustering in that it can be applied to group unseen test data. This is important because clustering often plays a component role in machine learning (e.g. dimensionality reduction). Evaluation of distance functions for hierarchical clustering is also more complex, and we consider it future work.

Another possible explanation for the results is that they are an artefact of the datasets used in the evaluation. There are many known limitations of the UCR data (see [26]). The data have been preprocessed in ways that may favour Euclidean distance-based clustering. Whilst it is important to acknowledge this, we also note we are following almost every other paper referenced in using the UCR data. By highlighting the fact that standard DTW does not cluster well, we are not arguing that it should never be used. Our aim is discourage its use as a straw man for new algorithms evaluated on UCR data and to highlight that there may be better alternatives.

Table 12 Average rank performance of k-medoids clusterer on problems split by the problem domain

	DEVICE	ECG	IMAGE	MOTION	SENSOR	SIMULATED	SPECTRO
MSM	3.30	**2.86**	**3.64**	**4.07**	5.34	4.06	**4.50**
WDTW	3.00	4.71	4.57	4.12	5.10	4.44	4.64
TWE	**2.80**	4.71	4.48	4.76	**4.82**	4.39	6.05
ERP	6.05	5.07	5.29	4.64	5.18	**3.67**	4.73
ED	5.45	6.57	6.36	5.86	5.78	4.11	4.77
DTW	7.90	6.14	6.62	4.91	5.28	7.39	5.18
WDDTW	6.70	6.71	5.76	5.86	5.48	6.56	5.73
EDR	7.75	5.86	5.07	6.45	6.06	6.00	6.68
LCSS	5.90	6.43	6.10	7.40	5.76	7.17	6.18
DDTW	6.15	5.93	7.12	6.93	6.20	7.22	6.55

Bold in the table are the best performing algorithm

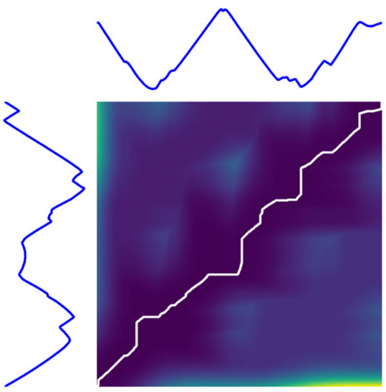
Table 13 Average number of moves on and off diagonal for three distances

Distance	Average path	Diagonal moves	Insert/Delete	Average percentage
DTW (20% window)	863	230	633	145
DTW (5% window)	690	326	363	132
MSM	576	499	77	104.5
TWE	591	484	107	1.08

Results are averaged over all distance calculations and all data sets. The average series length is 551

We think the true cause of the difference between DTW and MSM is in the warping penalty mechanism. All elastic distances implicitly penalise warpings simply because more distances are included in the calculation. However, both MSM and TWE also explicitly penalise for moving off the diagonal with a data driven cost term. We think that this disincentive to warp reduces the pathological warping made more likely in an unsupervised setting such as clustering.

To test this hypothesis, we re-run experiments recording the number of diagonal, horizontal and vertical moves for DTW (5% and 20% warping window), MSM and TWE for each instance to its best cluster centre using k-means. The number of horizontal moves always equals the number of vertical moves since we insist on the final alignment matching at the end. We average moves over all calculations and then over all datasets. Table 13 summarises the number of diagonal and off diagonal moves. DTW is making a surprising number of moves off diagonal. The average series length is 551, but the average DTW warping path is 863 with a 20% window and 690 with a 5% window. On average, the DTW warping path is 145% of the series length. MSM and TWE do far fewer off diagonal moves. There is still much more warping with a 5% window than seen with MSM. This implies DTW is moving back and forward off the diagonal far more than MSM. The notion that *"a little warping is a good thing, but too much warping is a bad thing"* is known for both classification [58] and clustering [19]. These results suggest that too much warping can also mean repeated small warpings rather than single long pathological warpings. Alignments with many gaps seem to result in inferior clustering (Fig. 22).


(a) DTW alignment path

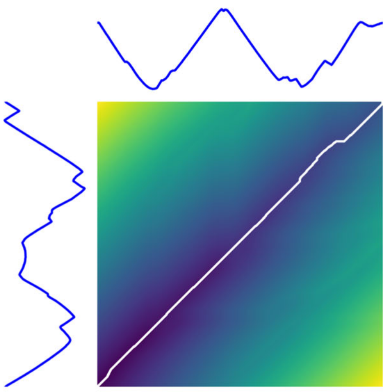

(b) MSM alignment path

Fig. 23 Example warping paths for DTW and MSM on two cases from the Fish dataset

Table 14 Proportion of off diagonal moves in each 10% segment of series across all datasets

Decile (%)	DTW (%)	MSM (%)	TWE (%)
0–10	9.37	18.64	10.53
10–20	9.96	9.45	8.86
20–30	9.85	8.73	9.73
30–40	10.27	9.82	11.15
40–50	10.41	9.88	10.90
50–60	10.42	10.58	11.39
60–70	9.93	12.05	12.53
70–80	10.41	9.08	10.34
80–90	10.38	6.54	7.92
90–100	8.98	5.23	6.66

Warping some of the time is often desirable, but DTW tends to do it too often. An example of this can be found by investigating the Fish dataset. For the Fish dataset, MSM k-means performs better across all metrics compared to DTW k-means. When we investigated the warping paths for the Fish dataset, we found significant differences in the amount of warping occurring in DTW compared to MSM. Figure 23 shows two alignment paths produced (DTW on the left, MSM on the right), between two time series in the Fish dataset. It shows that DTW is making many more small warps off the centre, whereas MSM makes smaller adjustments. The colours represent a heat map for the cost matrix, with lighter values representing a high value and a darker value representing a low value. Previous experiments showed that neither restricting the window nor tuning the window size reduced this over warping in a way that bridged the gap between DTW and MSM. Another possible reason for the poor performance of DTW could be a tendency to over warp series at the beginning or end of the series whilst remaining within the band. This has been shown to effect performance [66].

There are two mechanisms to constrain DTW warping at the end or the beginning. Firstly, the warping band can be made smaller at the beginning or end using, for example, a parallelogram band [30] rather than a uniform band. Secondly, the prefix/suffix invariant technique

[66] can be employed. The latter is designed specifically for data that has not been preprocessed and hence is not suitable for the UCR data. We have run DTW using both PSI and a parallelogram band, but neither significantly improved standard DTW. To find out why, we recorded which decile moves off the diagonal were happening for DTW, MSM and TWE. We already know DTW warps more than MSM, so Table 14 shows the proportion of warps for DTW, MSM and TWE normalised for the total number of shifts. It shows DTW is fairly consistently warping all along the series but surprisingly MSM is proportionately warping much more in the first decile and much less in the ninth and 10th. We are not sure exactly why this happens. It runs counter to the intuition that bad warpings may happen at the beginning.

7 Conclusions

We have described nine elastic time series distance measures and compared them when used to cluster the time series of the UCR archive with both k-means and k-medoids clusterer. There are a wealth of other time series clustering algorithms that we have not evaluated. We have opted to provide an in depth description, with examples and associated code and a tightly constrained bake off, rather than attempt to include all variants of, for example, deep learning and transformation-based time series clusterers. Distance-based approaches are still very popular, and we believe our experimental observations will help practitioners choose distance-based TSCL more effectively.

Our first conclusion is that k-medoids clusterer is more effective than k-means for TSCL with elastic distance measures on the UCR datasets. Standard k-means has the inherent problem that cluster centres are formed by averaging series, which takes no account of distortions that elastic distances are designed to compensate for. DBA does adjust for this problem with k-means, but it is designed specifically for DTW and it comes with a high computational overhead compared to k-medoids clusterer. The improved version of DBA [63] implemented in tslearn is significantly better than the original but is not different to either MSM clusterers, and it takes approximately twice as long to run. There is little difference in run time between the k-means with averaging and k-medoids clusterer: k-medoids clusterer is faster but uses more memory.

Our second conclusion based on these experiments on UCR data is that without any data specific prior knowledge as to the best approach, k-medoids clusterer with either MSM or TWE are good benchmark approaches for distance-based clustering of time series, with MSM preferred because of the lower run time. MSM is the top ranked distance measure on nearly all measures and configurations of both clusterers. We believe it is not widely known and that MSM and TWE should be included when assessing new clustering algorithms, although the tslearn version of DBA is also a good benchmark. TWE is slightly slower than the other distances. However, run time and memory were not a major constraint for these experiments.

More specifically, we conducted the following experiments:

1. In Sect. 5.1, we compare the 10 distance functions listed in Table 1 using k-means clustering and find that MSM produced significantly better clusters with five of the six performance measures and that five of the nine elastic measures perform worse than Euclidean distance. We find the same pattern of results with both normalised data (Fig. 9) and raw data (Fig. 10).
2. We repeat the experiments using k-medoids clusterer in Sect. 5.2. We find that, on the train data, TWE and MSM form a top clique, and that ERP performs much better. We

observe a similar pattern of results on the train data and show that, overall, k-medoids clusterer is better than k-means, independent of elastic distance used.
3. In Sect. 5.3, we assess the impact of using alternative cluster initialisation algorithms and found that MSM still outperforms ED and DTW for three initialisation algorithms with both k-means and k-medoids clusterer.
4. In Sect. 5.4, we explore variants of DTW to determine if we could find out why it performs so poorly. We compare different fixed window sizes and find that whilst smaller windows better approximate ED, they do not improve performance. We find tuning DTW does not improve performance. We then look at barycentre averaging for k-means and show that whilst it significantly improved k-means, it requires a specific refinement described in [63] to gain equivalence.
5. Finally, in Sect. 5.5 we tried tuning the distance functions using the DB measure and found it did not improve performance.

We have released all our results and code in a dedicated repository. We would welcome contributors of new distance functions to aeon and would be happy to extend the evaluation to include them. Our next stage is to extend the bake off to consider clustering algorithms not based on elastic distances and to investigate alternative medoids-based approaches such as partitioning around the medoids [40]. Furthermore, we have not yet investigated the effect of having to set the number of clusters, k. A future experiment will involve testing whether the relative performance remains the same when using standard techniques for setting k. There are also several possible directions for algorithmic advancement highlighted by this research: we will try combining the clusterings through an ensemble in a manner similar to the elastic ensemble for classification [44]. Clustering ensembles are more complex than classification ensembles, requiring some form of alignment of labelling. Nevertheless, it seems reasonably likely that there may be some improvements from doing so. These are just some of the numerous open issues in TSCL research. Our study aims to put future research on a sound basis to facilitate the accurate assessment of algorithmic improvements in a fully reproducible manner.

Acknowledgements This work is supported by the UK Engineering and Physical Sciences Research Council (EPSRC) Grant Number EP/W030756/1. 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. We would like to thank all those donating to and maintaining the time series archives we have used and to everyone who has contributed to open source implementations of the algorithms.

Author Contributions CH implemented the clustering algorithms and distance functions in the aeon toolkit. All authors jointly ran experiments, collated results and drew figures. All authors contributed to writing and reviewing the manuscript.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

A Appendix: Full results and code examples

We have implemented the nine distance functions used in the evaluation in the Python scikit-learn compatible package aeon.[5] The distance functions are all implemented using Numba tool,[6] which uses just in time compilation to dynamically convert Python to C. There are of course numerous open source packages that offer some form of elastic distance functions, most common variants of DTW. Table 15 lists some of the most popular packages with DTW implementations and summarises which other elastic distance measures they contain. We have confirmed the equivalence of DTW results with these packages.

The time taken to perform 200 DTW distance calculations on random series of lengths 1000 to 10,000 on a desktop PC is shown in Table 16. aeon is better than any other package with dtw-python being significantly slower than other packages. aeon offers the widest range of elastic distance measures with equivalent run time to the other packages.

Distance functions can be used directly, either by explicitly importing them or by using the distance factory provided. A Jupyter notebook with example usages is available on the associated repository[7] Listing 1 shows how to calculate the DTW distance and alignment path for the series used in Fig. 3. It also shows how to use the factory to get and call a distance function. The distance factory avoids the repeated Numba compilation of distance functions that can occur with different parameters, so it is our recommended method for creating a distance function. There is also the option to find the pairwise distance matrix.

```python
from aeon.distances import (
    distance, pairwise_distance, alignment_path, dtw_distance, dtw_alignment_path,
    dtw_pairwise_distance
)
import numpy as np

a = np.array(
    [0.018, 1.537, -0.141, -0.761, -0.177, -2.192, -0.193, -0.465, -0.944, -0.240])
b = np.array(
    [-0.755, 0.446, 1.198, 0.171, 0.564, 0.689, 1.794, 0.066, 0.288, 1.634])

# Call a specific distance function directly
d1 = dtw_distance(a, b)
d2 = dtw_distance(a, b, window=0.2)
# Call a distance using a metric string. Valid metrics are:
# 'dtw', 'ddtw', 'wdtw', 'wddtw', 'twe', 'msm', 'lcss', 'erp', 'edr', 'euclidean', 'squared'
d3 = distance(a, b, metric='dtw', window=0.2)
# Call a specific alignment path function directly
p1 = dtw_alignment_path(a, b)
p2 = dtw_alignment_path(a, b, window=0.2)
# Call an alignment path using a metric string. Valid metrics are same as above.
p3 = alignment_path(a, b, metric='dtw', window=0.2)

# Pairwise distance between two time
pair = np.array([a, b])
```

5 https://github.com/aeon-toolkit/aeon.

6 https://numba.pydata.org/.

7 https://tsml-eval.readthedocs.io/en/latest/publications/2023/distance_based_clustering/alignment_and_paths_figures.html.

Table 15 Summary of elastic distance function availability in five Python packages

Package	DTW	WDTW	DDTW	WDDTW	EDR	ERP	MSM	LCSS	TWE	PSI DTW
Univariate										
aeon	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
sktime	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
tslearn	Y	Y	N	N	N	N	N	N	N	N
dtw-python	Y	N	N	N	N	N	N	N	N	N
rust-dtw	Y	N	N	N	N	N	N	N	N	N
Multivariate										
aeon	Y	Y	Y	Y	Y	Y	N	Y	Y	Y
sktime	Y	Y	Y	Y	Y	Y	N	Y	Y	N
tslearn	N	N	N	N	N	N	N	N	N	N
dtw-python	N	N	N	N	N	N	N	N	N	N
rust-dtw	N	N	N	N	N	N	N	N	N	N

Table 16 Time (in seconds) to perform 200 full window DTW distance calculations with random series of length 1000 to 10,000

Length	aeon	sktime	tslearn	dtw-python	rust-dtw
1000	0.82	1.89	2.10	8.09	1.55
2000	3.04	6.50	6.25	31.38	6.00
3000	7.33	14.65	13.54	69.31	13.44
4000	12.68	25.07	25.97	121.33	24.00
5000	20.10	38.96	37.39	191.91	37.42
6000	29.03	57.26	54.44	272.14	55.24
7000	39.83	73.10	73.59	342.45	71.57
8000	52.48	92.25	91.94	451.83	93.49
9000	66.02	123.11	117.81	583.75	119.54
10,000	81.38	175.20	163.55	780.52	167.74

```python
# Call a specific pairwise distance function directly
pw_1 = dtw_pairwise_distance(pair)
pw_2 = dtw_pairwise_distance(pair, window=0.2)
# Call a specific pairwise distance using a metric string.
    Valid metrics are same as above.
pw_dist = pairwise_distance(pair, metric="dtw", window=0.2)
pw_dist_equi = pairwise_distance(pair, pair, metric="dtw")
```

Listing 1 A simple example of finding distances and alignments in aeon

We have implemented the k-means and k-medoids clusterers in aeon. These can be easily used and configured, as shown in Listing 2.

```python
from aeon.clustering.k_means import TimeSeriesKMeans
from aeon.clustering.k_medoids import TimeSeriesKMedoids
from aeon.datasets import load_unit_test
trainX, trainY = load_unit_test(split="test",return_type="
    np2D")
testX, testY = load_unit_test(split="train",return_type="
    np2D")
clst1 = TimeSeriesKMeans()
clst2 = TimeSeriesKMeans(
            averaging_method="dba",
            metric="dtw",
            distance_params={"window":0.1},
            n_clusters=2,
            random_state=1,
        )
clst3 =  TimeSeriesKMedoids()
clst4 =  TimeSeriesKMedoids(
            metric="dtw",
            distance_params={"window": 0.2},
            n_clusters=len(set(trainY)),
            random_state= 1,
        )
clst1.fit(trainX)
pred = clst1.predict(testX)
print(pred)
```

Listing 2 An example of using K-means and K-medoids clusterer in aeon

To conduct an experiment, we generate results in a standard format. This is given below, but additionally, a notebook has been produced for ease of use. [8]

```
from scikit_time.benchmarking.experiments import
    run_clustering_experiment

run_clustering_experiment(
        trainX,
        clst1,
        results_path="_contrib/temp/",
        trainY=trainY,
        testX=testX,
        testY=testY,
        cls_name="kmeans",
        dataset_name="UnitTest",
        resample_id=0,
        overwrite=False,
    )
```

Listing 3 An example of running an experiment in `aeon`. Random state is set to 1 throughout our experiments to faciliate reproducibility. label

Currently, we collate these results files using the associated Java package tsml.[9] An example of this is available on this paper's repository.

References

1. Abanda A, Mori U, Lozano J (2019) A review on distance based time series classification. Data Min Knowl Disc 33(2):378–412
2. Aghabozorgi S, Seyed Shirkhorshidi A, Ying Wah T (2015) Time-series clustering—a decade review. Inf Syst 53:16–38
3. Ali M, Alqahtani A, Jones MW, Xie X (2019) Clustering and classification for time series data in visual analytics: a survey. IEEE Access 7:181314–181338
4. Alqahtani A, Ali M, Xie X, Jones MW (2021) Deep time-series clustering: a review. Electronics
5. Anderberg M (1973) Cluster analysis for applications. Probability and mathematical statistics a series of monographs and textbooks. Academic Press
6. Ankerst M, Breunig MM, Kriegel H-P, Sander J (1999) Optics: ordering points to identify the clustering structure. SIGMOD Rec 28(2):49–60
7. Arthur D, Vassilvitskii S (2007) K-means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, SODA '07, pp. 1027–1035. Society for Industrial and Applied Mathematics
8. Bagnall A, Lines J, Bostrom A, Large J, Keogh E (2017) The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Disc 31(3):606–660
9. Begum N, Ulanova L, Wang J, Keogh E (2015) Accelerating dynamic time warping clustering with a novel admissible pruning strategy. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining, KDD '15, pp 49–58. Association for Computing Machinery, New York
10. Begum N, Ulanova L, Wang J, Keogh E (2015) Accelerating dynamic time warping clustering with a novel admissible pruning strategy. KDD '15, pp 49–58. Association for Computing Machinery, New York
11. 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
12. Bonner RE (1964) On some clustering techniques. IBM J Res Dev 8(1):22–32
13. Bradley PS, Fayyad UM (1998) Refining initial points for k-means clustering. In: Proceedings of the fifth international conference on machine learning, pp 91–99

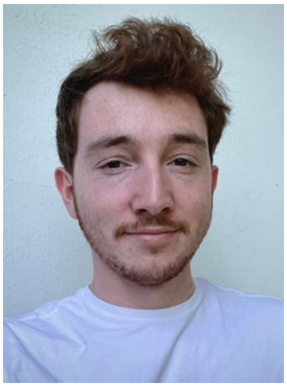
[8] https://tsml-eval.readthedocs.io/en/latest/publications/2023/distance_based_clustering/distance_based_clustering.html.

[9] https://github.com/time-series-machine-learning/tsml-java.

14. Bradley PS, Fayyad UM (1998) Refining initial points for k-means clustering. In: Proceedings of the fifteenth international conference on machine learning, ICML '98, pp 91-99. Morgan Kaufmann Publishers Inc, San Francisco
15. Caiado J, Maharaj E, D'Urso P (2015) Time series clustering. In: Handbook of cluster analysis, pp 241–264
16. Chen L, Ng R (2004) On the marriage of Lp-norms and edit distance. In: Proceedings of the 30th international conference on very large data bases
17. Chen L, Ozsu MT, Oria V (2005) Robust and fast similarity search for moving object trajectories. In: Proceedings of the ACM SIGMOD international conference on management of data
18. Dau H, Bagnall A, Kamgar K, Yeh M, Zhu Y, Gharghabi S, Ratanamahatana C, Chotirat A, Keogh E (2019) The UCR time series archive. IEEE/CAA J Automatica Sinica 6(6):1293–1305
19. Dau H, Silva D, Petitjean F, Forestier G, Bagnall A, Keogh E (2018) Optimizing dynamic time warping's window width for time series data mining applications. Data Min Knowl Disc 32(4):1074–1120
20. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
21. Dhillon IS, Guan Y, Kulis B (2004) Kernel k-means: spectral clustering and normalized cuts. In: Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '04, pp 551–556. Association for Computing Machinery, New York
22. Ding C, He X (2004) K-means clustering via principal component analysis. In: Proceedings of the twenty-first international conference on machine learning, ICML '04, pp 29. Association for Computing Machinery, New York
23. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the second international conference on knowledge discovery and data mining, KDD'96, pp 226-231. AAAI Press
24. Forgy E (1965) Cluster analysis of multivariate data: efficiency versus interpretability of classifications. Biometrics 21:768–769
25. 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
26. Hu B, Chen Y, Keogh E (2016) Classification of streaming time series under more realistic assumptions. Data Min Knowl Disc 30(2):403–437
27. Ikotun AM, Ezugwu AE, Abualigah L, Abuhaija B, Heming J (2023) K-means clustering algorithms: a comprehensive review, variants analysis, and advances in the era of big data. Inf Sci 622:178–210
28. Ismail-Fawaz A, Dempster A, Tan CW, Herrmann M, Miller L, Schmidt D, Berretti S, Weber J, Devanne M, Forestier G, Webb G (2023) An approach to multiple comparison benchmark evaluations that is stable under manipulation of the comparate set. arXiv preprint arXiv:2305.11921
29. Ismkhan H (2018) I-k-means-+: an iterative clustering algorithm based on an enhanced version of the k-means. Pattern Recogn 79:402–413
30. Itakura F (1975) Minimum prediction residual principle applied to speech recognition. IEEE Trans Acoust Speech Signal Process 23(1):67–72
31. Jain AK, Dubes RC (1988) Algorithms for clustering data. Prentice-Hall Inc.
32. Jain AK, Murty MN, Flynn PJ (1999) Data clustering: a review. ACM Comput Surv 31(3):264–323
33. Javed A, Lee BS, Rizzo D (2020) A benchmark study on time series clustering. Mach Learn Appl 1
34. Jeong Y, Jeong M, Omitaomu O (2011) Weighted dynamic time warping for time series classification. Pattern Recogn 44:2231–2240
35. Ward JH Jr (1963) Hierarchical grouping to optimize an objective function. J Am Stat Assoc 58(301):236–244
36. Kaufman L, Rousseeuw PJ (1986) Clustering large data sets. In: Pattern recognition in practice, pp 425–437. Elsevier, Amsterdam
37. Keogh E, Pazzani M (2001) Derivative dynamic time warping. In: Proceedings of the 1st SIAM international conference on data mining
38. Kuhn HW (1955) The Hungarian method for the assignment problem. Naval Res Logist Q 2(1–2):83–97
39. Lafabregue B, Weber J, Gancarski P, Forestier G (2022) End-to-end deep representation learning for time series clustering: a comparative study. Data Min Knowl Disc 36:29–81
40. Leonard Kaufman PJR (1990) Partitioning around medoids (program PAM), chapter 2, pp 68–125. Wiley
41. Li G, Bräysy O, Jiang L, Wu Z, Wang Y (2013) Finding time series discord based on bit representation clustering. Knowl-Based Syst 54:243–254
42. Li X, Lin J, Zhao L (2021) Time series clustering in linear time complexity. Data Min Knowl Disc 35(3):2369–2388
43. Li Z, Yang Y, Liu J, Zhou X, Lu H (2012) Unsupervised feature selection using nonnegative spectral analysis. In Proceedings of the twenty-sixth AAAI conference on artificial intelligence, AAAI'12, pp 1026–1032. AAAI Press

44. Lines J, Bagnall A (2015) Time series classification with ensembles of elastic distance measures. Data Min Knowl Disc 29:565–592
45. Letı R, Ortiz MC, Sarabia LA, Sánchez MS (2004) Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes. Anal Chim Acta 515(1):87–100
46. Lloyd SP (1982) Least squares quantization in pcm. IEEE Trans Inf Theory 28:129–136
47. MacQueen J et al. (1967) Some methods for classification and analysis of multivariate observations. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, vol 1, pp 281–297
48. Marteau P (2009) Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 31(2):306–318
49. McInnes L, Healy J (2017) Accelerated hierarchical density based clustering. In: 2017 IEEE international conference on data mining workshops (ICDMW), pp 33–42
50. Middlehurst M, Large J, Flynn M, Lines J, Bostrom A, Bagnall A (2021) HIVE-COTE 2.0: a new meta ensemble for time series classification. Mach Learn 110:3211–3243
51. Newling J, Fleuret F (2017) K-medoids for k-means seeding. In: Advances in neural information processing systems, vol 30. Curran Associates, Inc
52. Ng R, Han J (2002) CLARANS: a method for clustering objects for spatial data mining. IEEE Trans Knowl Data Eng 14:1003–1016
53. Paparrizos J, Gravano L (2015) k-shape: efficient and accurate clustering of time series. In: Proceedings of the 2015 ACM SIGMOD international conference on management of data, pp 1855–1870
54. Paparrizos J, Gravano L (2017) Fast and accurate time-series clustering. ACM Trans Database Syst (TODS) 42(2):1–49
55. Petitjean F, Ketterlin A, Gancarski P (2011) A global averaging method for dynamic time warping, with applications to clustering. Pattern Recogn 44:678
56. Rakthanmanon T, Bilson J, Campana L, Mueen A, Batista G, Westover B, Zhu Q, Zakaria J, Keogh E (2013) Addressing big data time series: mining trillions of time series subsequences under dynamic time warping. ACM Trans Knowl Discov Data 7(3)
57. Ratanamahatana C, Keogh E (2004) Everything you know about dynamic time warping is wrong. In: Proceedings of the 3rd workshop on mining temporal and sequential data
58. Ratanamahatana C, Keogh E (2005) Three myths about dynamic time warping data mining. In: Proceedings of the 5th SIAM international conference on data mining
59. Räsänen T, Kolehmainen M (2009) Feature-based clustering for electricity use time series data. vol 5495, pp 401–412
60. Ruiz AP, Flynn M, Large J, Middlehurst M, Bagnall A (2021) The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Min Knowl Disc 35(2):401–449
61. Sakoe H, Chiba S (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans Acoust Speech Signal Process 26(1):43–49
62. Saxena A, Prasad M, Gupta A, Bharill N, Patel OP, Tiwari A, Er MJ, Ding W, Lin C-T (2017) A review of clustering techniques and developments. Neurocomputing 267:664–681
63. Schultz D, Jain BJ (2017) Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. CoRR, arXiv:1701.06393
64. Shi L, Du L, Shen Y-D (2014) Robust spectral learning for unsupervised feature selection. In: 2014 IEEE international conference on data mining, pp 977–982
65. Shifaz A, Pelletier C, Petitjean F, Webb G (2023) Elastic similarity and distance measures for multivariate time series. Knowl Inf Syst 65(6)
66. Silva D, Batista G, Keogh E (2016) Prefix and suffix invariant dynamic time warping. In: IEEE International conference on data mining
67. Stefan A, Athitsos V, Das G (2013) The move–split–merge metric for time series. IEEE Trans Knowl Data Eng 25(6):1425–1438
68. van der Maaten L (2011) Learning discriminative fisher kernels. In: Proceedings of the 28th international conference on international conference on machine learning, ICML'11, pp 217–224. Omnipress, Madison
69. 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, WSDM '11, pp 177–186. Association for Computing Machinery, New York
70. Yang J, Wang Y-K, Yao X, Lin C-T (2019) Adaptive initialization method for k-means algorithm
71. 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
72. Zhang Q, Wu J, Zhang P, Long G, Zhang C (2019) Salient subsequence learning for time series clustering. IEEE Trans Pattern Anal Mach Intell 41(9):2193–2207

73. Zhang T, Ramakrishnan R, Livny M (1996) Birch: an efficient data clustering method for very large databases. SIGMOD Rec 25(2):103–114
74. Zolhavarieh S, Aghabozorgi S, Teh YW (2014) A review of subsequence time series clustering. Sci World J 2014



Christopher Holder received his BSc degree in Computer Science in 2020 from the University of East Anglia (UEA), Norwich, UK. He is currently pursuing his PhD, with a primary research focus on time series clustering. He is also involved with the development and open-sourcing of time series machine learning algorithms to facilitate advancements in the field.

Matthew Middlehurst received the BSc degree in Computer Science in 2018 and the PhD degree in Computer Science in 2023 from the University of East Anglia (UEA), Norwich, UK. He is working as a Senior Research Associate at UEA in the time series machine learning group. His research interests are primarily time series classification related, with recent expansion to time series clustering and time series extrinsic regression. He is interested in the provision and maintenance of open-source software for researchers in time-series-based fields.

Anthony Bagnall received the PhD degree in Computer Science from the University of East Anglia (UEA), Norwich, UK, in 2001. He was a Professor of Computer Science at UEA from 2018-2023. He has recently moved to an equivalent position in the Electronics and Computer Science department at the University of Southampton. His primary research interest is in time series machine learning, with a historic focus on classification, but more recently looking at clustering and regression. He has a side interest in ensemble design.