# Time series clustering using elastic distances

**Christopher Lawrence Holder**

School of Computing Sciences

University of East Anglia

This thesis is submitted for the degree of

*Doctor of Philosophy*

June 2025

# Abstract

In recent years, time series data has become increasingly ubiquitous, emerging across numerous domains such as astronomy, biology, engineering, finance, manufacturing, medicine, meteorology, and more. A time series is an ordered sequence of real valued observations. The most common form of ordering is in the time domain. Although the concept of a time series is not limited to time-based ordering, the fact that human experience is inherently bound to the passage of time makes time-domain data particularly prevalent. As a result, nearly any recorded phenomenon can be represented as a time series.

The widespread generation of time series data, coupled with the desire to analyse and derive insights from it, has sparked significant interest in time series machine learning tasks. Among these, time series clustering (TSCL) has emerged as one of the most prominent fields. TSCL aims to group time series into clusters where the series within a cluster exhibit homogeneity, while those outside the cluster are heterogeneous. As an unsupervised task, TSCL requires no manual labelling, making it versatile and applicable to a wide range of time series datasets. It is often employed as a key tool for exploratory data analysis.

One of the most common approaches to clustering time series data is to adapt traditional clustering algorithms (e.g., *k*-means, *k*-medoids, DBSCAN, agglomerative clustering) by replacing conventional distance measures with elastic ones. Elastic distances account for misalignment between time series during distance computation—when similar events occur but are recorded at different time intervals in each series. By accounting for misalignment, elastic distances significantly improve the quality of the similarity measure between time series.

Dynamic Time Warping (DTW) has become the most widely used elastic distance in TSCL literature. However, other elastic distances have demonstrated superior performance in related fields, such as time series classification. Despite this, the TSCL community has been slow to adopt these alternatives. This thesis addresses this gap by conducting the most comprehensive review of elastic distances for TSCL. We evaluate 12 different elastic distances, nine of which had not been previously applied to TSCL. Our empirical analysis reveal that many of these unconsidered elastic distances significantly outperform DTW for TSCL tasks.

Building on these findings, we propose novel elastic distance-based algorithms, including the Elastic Barycentre Average, the Elastic Unsupervised Proportional Weighting (EUPW) ensemble scheme, the Elastic Clustering Ensemble (ECE), and the $k$-means end-to-end Elastic Stochastic subgradient Barycentre Average (KESBA) clusterer.

This thesis demonstrates the benefits of incorporating previously unexplored elastic distances into established TSCL algorithms, introduces new elastic-based averaging techniques, and presents the development of state-of-the-art elastic-based partition and ensemble clustering methods. Together, these contributions advance TSCL performance and lay the foundation for future innovations in the field.

# Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Prof. Anthony Bagnall. His constant guidance, unwavering support, and patience throughout this PhD have been invaluable. This thesis would not have been possible without it. I am especially grateful for his continued support even after his move to a new university. I would also like to extend my sincere thanks to Dr. Jason Lines, for his continuous support throughout my time at UEA and for stepping up to become my primary supervisor in my final year after Tony's departure. Thank you both.

I am also grateful to everyone in the time series machine learning group at UEA. In particular, I want to thank Dr. Matthew Middlehurst for his tremendous help during my PhD journey, offering guidance and developing invaluable open-source tools such as tsml-eval, which played a significant role in my work.

To my friends and family, thank you for your unwavering love and support. This PhD would not have been possible without you. To my Dad, you are my greatest role model, and who I strive to be. To my Mum, your love and encouragement have been my constant foundation. To my siblings, Imogen, Alexander, and Kieran, you continually inspire me to grow; your talent, drive, and determination push me to become better. Finally to Larissa, your endless patience, understanding, encouragement and support have been my anchor. Without each of you, I would not be the person I am today. Thank you.

# Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Christopher Lawrence Holder

June 2025

# Publications

## As First Author

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

- Holder, C., Guijo-Rubio, D., Bagnall, A. (2023). Clustering Time Series with k-Medoids Based Algorithms. In: Ifrim, G., et al. Advanced Analytics and Learning on Temporal Data. AALTD 2023. Lecture Notes in Computer Science(), vol 14343. Springer, Cham. https://doi.org/10.1007/978-3-031-49896-1_4

- Holder, C., Bagnall, A., Lines, J. (2024). On time series clustering with *k*-means. In: under review at Advances in Data Analysis and Classification

- Holder, C.; Guijo-Rubio, D. and Bagnall, A. (2023). Barycentre Averaging for the Move-Split-Merge Time Series Distance Measure. In Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KDIR; ISBN 978-989-758-671-2; ISSN 2184-3228, SciTePress, pages 51-62. DOI: 10.5220/0012164900003598

# Table of contents

# List of figures

# List of tables

# List of Algorithms

# Chapter 1

# Introduction

In recent years, time series data has become increasingly ubiquitous, emerging across numerous domains such as astronomy, biology, engineering, finance, manufacturing, medicine, meteorology, and more [98, 80, 127, 109, 33, 67, 42]. A time series is an ordered sequence of real valued observations [84]. The most common form of ordering is in the time domain. Although the concept of a time series is not limited to time-based ordering, the fact that human experience is inherently bound to the passage of time makes time-domain data particularly prevalent. As a result, nearly any recorded phenomenon can be represented as a time series.

The widespread generation of time series data, coupled with the desire to analyse and derive insights from it, has driven substantial interest in time series data mining tasks such as anomaly detection, classification, clustering, forecasting, querying, regression, and segmentation [105, 84, 47, 109, 27, 122]. Interest in these tasks continues to grow. Figure 1.1 illustrates the increasing number of published papers across various time series data mining fields. The graph was generated using data from the Web of Science [1], querying the number of papers with titles containing the phrase “Time Series” followed by a specific field such as “Clustering”.

[1] https://clarivate.com/

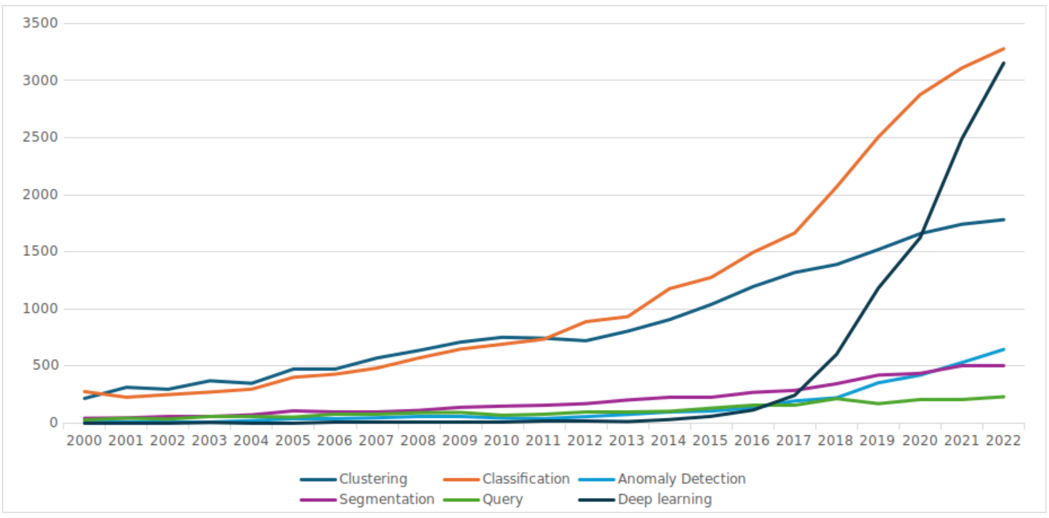


Fig. 1.1 Number of time series-related published papers over time. The data was obtained from the Web Of Science, filtering papers that start with “Time Series” followed by a field of interest.

One of the most popular fields is time series clustering (TSCL) [89]. The objective of TSCL is to group time series into clusters where the series within a cluster exhibit homogeneity, while those outside the cluster display heterogeneity [14]. As an unsupervised learning task, TSCL does not require human supervision or labour-intensive data annotation, making it highly versatile and applicable to almost any time series dataset without prerequisites. This flexibility makes TSCL a common starting point for exploratory data analysis [133].

However, a key challenge in clustering is defining what constitutes a “good” cluster. There is no universally accepted definition of a good cluster [50]. For instance, homogeneity could be defined as time series generated by a common underlying process or based on shared hidden variables [47]. Consider clustering patients using medical data: one approach might group patients by gender, while another might cluster by age. There are countless ways to group patients, each valid depending on the chosen criteria for similarity.

A central concept in clustering is “similarity”. What makes one time series more similar to another? These are critical questions in TSCL. In traditional clustering (i.e., clustering static, non-sequential tabular data), one of the most common methods to measure similarity is through a distance measure.

A distance measure is a mathematical function that quantifies the similarity or dissimilarity between data points. It assigns a numeric value representing how close or far apart two points are in a feature space. Hundreds, if not thousands, of clustering algorithms have been proposed [31], many of which use distance functions to define similarity.

Measuring the distance between time series is significantly more complex than for tabular data. While a time series can be represented in feature space, this approach disregards the temporal ordering inherent in the data. Extensive research has evaluated various distance measures for time series, and numerous studies [74, 47, 111, 99, 76] have empirically shown that distance measures that ignore temporal ordering (e.g., Euclidean distance) yield significantly less accurate similarity measures between time series.

The most widely used distance measure to improve time series comparison is Dynamic Time Warping (DTW) [10], the first in a family of algorithms known as elastic distances. Elastic distances account for misalignment between time series during distance computation. Misalignment occurs when the same or similar events happen but are recorded at different time intervals in each series.

For example, consider recording the weather over two days: on the first day, it only rains in the morning, while on the second day, it only rains in the afternoon. Though both days experienced rain, the time (or temporal ordering) of this event differs. A traditional distance measure that does not account for alignment would fail to detect this similarity, as it does not account for the misalignment of temporal events. However, an elastic distance would recognise this misalignment, determining that rain occurred on both days and thus identifying the two time series as similar.

Consequently, the most common approach to clustering time series data is to use traditional clustering algorithms (e.g., *k*-means [75], *k*-medoids [69], DBSCAN [29], Agglomerative [55]) but replace the Euclidean or other traditional distance measures with an elastic distance. Numerous traditional clustering algorithms have been adapted to work specifically with DTW. Examples include *k*-means [94, 21, 47, 93], *k*-medoids [46, 47, 91], Agglomerative [2, 63, 51], Density Peaks [9, 51], and DBSCAN [51].

In recent years, many new elastic distances have been proposed, showing superior performance to DTW in supervised tasks such as time series classification (TSC) [74, 111, 76]. Despite this, the TSCL community has been slow to adopt these new elastic distances, potentially missing opportunities to improve clustering performance. Prior to this thesis, we identified only three instances in the TSCL literature where alternative elastic distances to DTW were used: soft-DTW [21], the Edit Distance on Real sequences (EDR) [85], and the Longest Common Subsequence LCSS [118].

This thesis seeks to address this gap in the research. Our objective is to conduct the most extensive review of elastic distances for TSCL to date. We will review and implement 12 different elastic distances, nine of which have not been considered for TSCL before. These distances will be benchmarked using existing TSCL algorithms previously tested only with DTW. Moreover, based on our findings, we will develop new and novel TSCL algorithms that significantly advance the state-of-the-art in TSCL while also being considerably faster in terms of runtime.

## 1.1 Thesis Contributions

1. **Chapter 4 presents a robust Lloyd's-based clustering algorithm for TSCL.** Many popular TSCL algorithms adapt *k*-means (Lloyd's algorithm) with time series distance functions to account for temporal dependencies.

However, our survey of the TSCL literature reveals that nearly every Lloyd's-based algorithm is configured differently, making meaningful comparisons difficult due to the algorithm's sensitivity to configuration. To address this, we propose a standardised Lloyd's-based model for TSCL, consistently applying a specialised distance function across the initialisation, assignment, and stopping condition. This unified approach allows us to attribute performance differences to the distance function itself, rather than to varying configurations. Using this model, we benchmark five of the most commonly used Lloyd's-based TSCL algorithms, which serve as our point of comparison throughout the thesis.

2. **Chapter 5 presents a comprehensive review of 12 different elastic distances using the *k*-means clustering algorithm.** Building on our standardised Lloyd's configuration from Chapter 4, we conduct an extensive review of 12 different elastic distances with the *k*-means algorithm. Our findings challenge several common misconceptions within the TSCL community and identify multiple new distances capable of achieving state-of-the-art performance, which had been previously overlooked. Additionally, we analyse the shared characteristics of the top-performing elastic distances and conduct a detailed investigation into the unexpectedly poor performance of DTW.

3. **Chapter 6 presents a detailed evaluation of 12 elastic distances across four *k*-medoids clustering algorithms.** In Chapter 5, we address a key limitation of *k*-means when using elastic distances: centroid computation. Since most elastic distances lack a specialised averaging technique (except for DTW), *k*-means relies on the arithmetic mean, leading to suboptimal minimisation of the objective function and resulting in unexpected clusterings. In contrast, *k*-medoids can fully leverage elastic distances in both the assignment and centroid computation stages without requiring any modifications. We evaluate

four *k*-medoids variants, including two never before benchmarked with elastic distances, and show that using elastic distances in centroid computation significantly improves clustering performance. Additionally, our experiments reveal that PAM with newly introduced elastic distances surpasses the current state-of-the-art, establishing a new best-in-class TSCL approach.

4. **Chapter 7 proposes a new Elastic Barycentre Averaging technique tailored for TSCL.** In Chapter 6, we demonstrate that *k*-medoids, which utilises elastic distances for centroid computation, significantly outperforms methods that rely on the arithmetic mean. However, when comparing PAM-DTW (medoids) to *k*-means-ba-DTW (which uses a DTW-specific averaging technique), we observed that *k*-means-ba-DTW achieved superior clustering performance. This led us to hypothesise that developing an averaging technique for the best-performing elastic distances could further enhance state-of-the-art clustering. To this end, we propose the Elastic Barycentre Average, a generalised version of the Dynamic Time Warping Barycentre Average (DBA), applicable to any elastic distance that computes a full alignment path through a cost matrix. Our empirical analysis show that using the Elastic Barycentre Average for all elastic distances significantly improves clustering performance compared to both the arithmetic mean and medoids. Moreover, the best-performing elastic distances with this technique surpass the current state-of-the-art.

5. **Chapter 8 proposes the *k*-means end-to-end Elastic Stochastic subgradient Barycentre Average (KESBA) clusterer: a state-of-the-art, versatile, and highly scalable clustering algorithm for real-world TSCL applications.** In Chapter 7, we introduced a new elastic averaging technique that achieved state-of-the-art TSCL performance. Additionally, in Chapter 6, we found that PAM with certain elastic distances also exceeded the state-of-the-

art. However, both PAM and the Elastic Barycentre Average were shown to have prohibitively high computational costs, rendering them impractical for large-scale TSCL applications. To address this, we develop KESBA, designed specifically for large-scale TSCL. KESBA incorporates a novel extension of the Elastic Barycentre Average, called the Random Subset Elastic Stochastic Subgradient Barycentre Average, along with several optimisations to the baseline Lloyd's algorithm outlined in Chapter 4. These improvements enable KESBA to achieve state-of-the-art clustering performance while being significantly faster than other high-performing algorithms.

6. **Chapter 9 proposes the Elastic Clustering Ensemble (ECE): a state-of-the-art elastic PAM ensemble created using a novel Elastic Unsupervised Proportional Weighting (EUPW) ensemble scheme.** Building on the best-performing PAM clusterers identified in Chapter 6, we introduce a new EUPW ensemble scheme to create the Elastic Clustering Ensemble (ECE). Our empirical evaluation demonstrates that the ECE clusterer, leveraging the EUPW ensemble scheme, achieves state-of-the-art clustering performance, surpassing six other commonly recognised ensemble schemes from the literature.

## 1.2 Thesis Outline

This thesis is organised into ten chapters. In the following, we outline the contents of the remaining chapters.

Chapter 2 offers a detailed background on TSCL. It begins by outlining the foundational concepts of time series data mining, including key background information and notation, and situates TSCL within this context. We then present a general overview of TSCL, defining our specific research focus within the field. Following this, we provide an in-depth description of 12 elastic distance measures, complete with formal notation and pseudocode for each. Finally, we review TSCL models that incorporate specialised time series distance measures to enhance clustering performance.

Chapter 3 outlines the experimental methodology used throughout this thesis. We address the complexities involved in evaluating clustering performance and propose a robust methodology to overcome these challenges. The chapter includes details on statistical methods and comparison techniques, as well as an overview of the open-source software packages utilised and contributed to, ensuring the reproducibility of our research and results.

Chapter 4 surveys Lloyd's-based TSCL algorithms and identifies significant variation in configuration across different studies. Since Lloyd's algorithm is highly sensitive to its configuration, comparing results from different papers is challenging. To address this, we propose a standardised Lloyd's-based model for TSCL and conduct benchmark experiments to establish a baseline for comparison, which is used throughout this thesis.

Chapters 5 and 6 present an extensive review of *k*-means and *k*-medoids clustering using 12 different elastic distances. We identify superior elastic distances that had not previously been considered for TSCL and highlight specific models that outperform the current state-of-the-art clustering algorithms. Additionally, we analyse the attributes of the best-performing elastic distances to uncover common traits that explain their superior performance.

Chapter 7 introduces a new averaging technique called the Elastic Barycentre Average, a generalised version of the DBA algorithm that can be applied to any elastic distance with a complete alignment path. We demonstrate that using the Elastic Barycentre Average with *k*-means as the centroid computation algorithm exceeds the current state-of-the-art performance.

Chapter 8 presents a new clustering model called KESBA. To create KESBA, we develop the Random Subset Elastic Stochastic Subgradient Barycentre Average, an extension of the Elastic Barycentre Average, incorporating optimisations from the Stochastic Subgradient Dynamic Barycentre Average and random subsampling inspired by CLARA. Further enhancements to the standard Lloyd's baseline clusterer enable KESBA to achieve state-of-the-art performance while significantly reducing computational runtime compared to similar high-performing algorithms.

Chapter 9 introduces a new elastic PAM ensemble, the Elastic Clustering Ensemble (ECE), developed using a novel Elastic Unsupervised Proportional Weighting (EUPW) ensemble scheme. Our empirical analysis demonstrates that the ECE clusterer, leveraging the EUPW scheme, outperforms six widely used ensemble schemes from the literature, all using the same PAM models. Additionally, we show that ECE surpasses the performance of each individual PAM model that comprises it, further highlighting its effectiveness.

Chapter 10 concludes this thesis by summarising the key contributions and discussing potential future directions for elastic distance TSCL research.

# Chapter 2

# Background and Related Work

This chapter introduces the relevant background material for this thesis. We begin by outlining the foundational concepts of time series data mining, including key background information and notation, and position TSCL within this context. Next, we present a general overview of traditional clustering methods and explain how TSCL builds upon these approaches. Following this, we provide an in-depth description of 12 elastic distance measures, including formal notation and pseudocode for each. Finally, we review specific TSCL models relevant to this thesis that incorporate specialised time series distance measures and averaging techniques to enhance clustering performance.

## 2.1 Time series data mining

A time series is any data that has discriminatory features dependent on its ordering [8]. The most common form of ordering is in the time domain. Although the concept of a time series is not limited to time-based ordering, the fact that human experience is inherently bound to the passage of time makes time-domain data particularly prevalent.

Formally, we define a time series as:

$$T = (t_1, \ldots, t_i, t_m) \tag{2.1}$$

where $t_i \in \mathbb{R}^{CH}$ is the observed value at the $i$-th time point, $CH$ are the number of channels for a time point, and $m$ as the number of time points. When $CH = 1$, we consider the time series to be univariate as each time point represent one value. When $CH > 1$, we consider the time series to be multivariate as each time point represents more than one value.

Time series can also be regularly or irregularly sampled. We consider a time series to be irregularly sampled when the interval between observations are not consistent. For example assume we have a time series $T = (t_1, t_2, t_3, t_4)$, which has four observations representing a temperature reading at a given time. $T$ would be considered regularly sampled if each reading was taken at the same time apart, say exactly one hour. However, we would consider $T$ to be irregularly sampled if the first ($t_1$) and second ($t_2$) time point were recorded two hours apart but the third ($t_3$) and fourth ($t_4$) observation were recorded thirty minutes apart. The interval they were recorded at is not consistent therefore it is irregularly sampled.

Time series data exhibits unique characteristics not found in other data. Time series machine learning techniques attempt to exploit these unique characteristics and by doing so can achieve better results than traditional approaches. These unique characteristics will now be outlined.

### 2.1.1 Time series unique characteristics

#### Temporal dependency

Time series data exhibits dependencies on its ordering. In other words, a given time point may correlate to previous time points. In the context of time series machine learning, given an input time point $t_i$ a model may predict $y_i$. However, if the same value is observed at a later time point, say $t_{i+n}$ where $n$ is some number of observations in the future, the value of $y_{i+n}$ may be different to $y_i$. This is due to the model having observed more values since the last prediction and therefore, the prediction changes due to the temporal dependency.

### High dimensionality

In real world scenarios it is very common to encounter multivariate time series where there could be tens if not hundreds of additional channels to consider. High dimensionality poses numerous problems when considering algorithmic and computational complexity, but also presents a complex data mining challenges. Multivariate data mining techniques must consider the inter-relationship between observations across channels. The discovery, and understanding of inter-relationship between channels is critical to multivariate time series data mining.

### Noise

Noise in time series is very common and can arise in many ways such as measurement error with faulty sensors, rounding errors in the collection of the data or other errors introduced by human error. An important challenge for data mining techniques to overcome is determining what data is relevant and what is noise.

### Missing values

Time series, especially in the real world scenarios, can have values missing. We consider a time series to contain missing values if there is no reading for a given time point where it is expected to have one for. The reason a time series may have missing values could be due to faulty sensors, human error or data being unavailable for collection at certain time points.

#### Diverse semantics

In other data mining fields such as image or text data; patterns and trends learned in one dataset to some extent can be transferred over to other datasets (namely word or image embeddings). However, time series datasets are generally unique and observations learned for one dataset will not be applicable to any other due to its highly diverse and unique semantics.

### 2.1.2 Time series machine learning

The goal of a time series machine learning model is to leverage the unique characteristics of time series data to generate a desired output. This output varies depending on the specific discipline within time series machine learning. Some of the main disciplines include:

#### Time series clustering (TSCL)

The objective of time series clustering (TSCL), is to group time series into clusters where the series within each cluster exhibit homogeneity, while those outside the cluster display heterogeneity [14]. As an unsupervised learning task, TSCL does not require human supervision or labour-intensive data annotation, making it highly versatile and applicable to almost any time series dataset without prerequisites. This flexibility makes TSCL a popular starting point for exploratory data analysis [133].

#### Time series classification (TSC)

The objective of time series classification (TSC) is to assign predefined class labels to a set of time series. TSC trains a classifier on a dataset by learning temporal patterns and features that distinguish different classes. As a supervised learning task, TSC requires the training data to be annotated with class labels that define the observations. Once a classifier has been trained, new, unseen, and unlabelled time series data can be inputted into the model, which will predict class labels for the unseen data.

### Time series forecasting (TSF)

Time series forecasting (TSF) aims to predict the future values of a time series by explicitly modelling the dynamics and dependencies among historical observations [121].

### Time series extrinsic regression (TSER)

Time series extrinsic regression (TSER) is similar to TSC but instead of predicting a discrete class labels, TSER predicts a continuous target variable. For instance, TSC might classify an ECG signal as arrhythmia or normal, while TSER could be used to predict a quantitative value such as the heart rate or respiratory rate of a patient based on patterns in the ECG signal [117].

### Time Series Segmentation (TSS)

Time series segmentation (TSS) aims to divide a time series into multiple subsequences and assign labels to each subsequence. This segmentation is useful for identifying regime changes, anomaly detection, and trend analysis.

### Time Series Anomaly Detection (TSAD)

Time series anomaly detection (TSAD) focuses on identifying unusual or abnormal patterns in time series data. Anomalies can indicate significant temporal events. TSAD methods typically aim to detect points, segments, or trends that deviate significantly from the expected behavior of a time series [13].

The above list is not intended to be an exhaustive list of all time series machine learning disciplines but is instead meant to highlight the diverse time series machine learning field and its huge range of real world applications.

The rest of the literature review will set out the specific literature that is the focus of this thesis, namely TSCL and elastic distances. For simplicity, all examples and pseudocode will assume the time series used are univariate of equal length $m$. Additionally all arrays will be assumed to be indexed from 0.

## 2.2 Clustering

Before delving into specific TSCL approaches, we first outline traditional clustering techniques to demonstrate how they have been adapted for TSCL. Over the years, hundreds of clustering algorithms have been proposed [31] to solve various clustering problems. Broadly, these techniques can be divided into two main categories: hierarchical-based and partition-based [32].

In this thesis, we focus primarily on partition-based clustering, as it is the most commonly used and widely implemented approach in the TSCL literature. Specifically, our research centres on crisp, squared error clustering [32], a method that forms partitions by optimising a criterion function—typically minimising the sum of squared distances within clusters. We focus on this type of clustering because it is the most prevalent form of partition-based clustering in both the traditional [32] and TSCL literature. While other partition-based methods, such as density-based or model-based clustering, could be considered, these approaches often pursue different clustering objectives, making direct comparisons and result evaluations particularly challenging.

### 2.2.1 Hierarchical-based

Hierarchical clustering refers to a family of algorithms that build nested clusters by successively merging or splitting them [92]. Clusters are formed iteratively in either a top-down or bottom-up manner, resulting in a dendrogram that depicts the hierarchical structure of the clusters [104]. Agglomerative clustering, a bottom-up approach, starts with individual objects and successively merges them to form larger clusters. Conversely, divisive clustering, a top-down approach, begins with all objects in a single cluster, which is then split into smaller clusters until each object is isolated. A visualisation of hierarchical clustering is shown in Figure 2.1.

Popular examples of hierarchical clustering algorithms include Agglomerative [55], Balanced Iterative Reducing and Clustering Using Hierarchies (BIRCH) [131], Clustering Using Representatives (CURE) [41], and CHAMELEON [56]. Many of these algorithms have been adapted for TSCL by incorporating time series distance measures [63, 2, 51] and time series-specific averaging techniques [66].

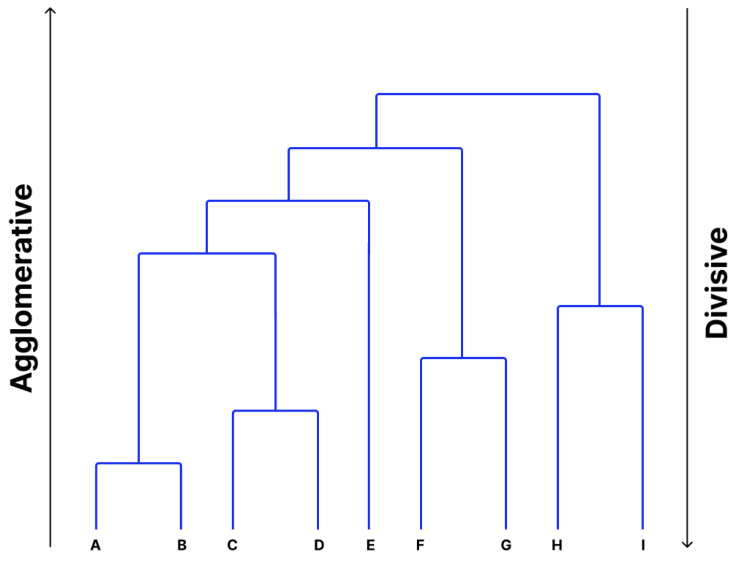


Fig. 2.1 Hierarchical clustering dendrogram.

### 2.2.2 Partition-based

Partition-based clustering is the opposite to hierarchical clustering [104]. Its primary goal is to generate clusters that capture the natural groupings inherent in a dataset [32]. While partition-based clustering can be divided into several subcategories, we will outline those most relevant to the TSCL literature. For a more comprehensive overview of every partition-based subcategory, we refer interested readers to [32]. In this thesis, we define three key subcategories of partition-based clustering extensively used in TSCL: Squared Error, Density-based, and Model-based clustering.

#### Squared error partition clustering

Squared error clustering is the most widely used form of partition-based clustering [32]. It divides a dataset into $k$ clusters without any hierarchical structure by optimising a criterion function, typically the squared error of a distance measure [65]. The main objective is to define $k$ centroids (also called exemplars or cluster centres) that represent each cluster. Instances are assigned to clusters based on their similarity to the centroids. A visual example of squared error clustering is shown in Figure 2.2. Figure 2.2 depicts three distinct clusters, each with a centroid that minimises a given error function. All other instances are assigned to one of the three clusters based on their proximity to the three centroids.

Examples of squared error clustering in the traditional clustering literature include $k$-means [75] and $k$-medoids algorithms, such as Partition Around Medoids (PAM) [69]. Notably, $k$-means is one of the most well-known and widely used clustering algorithms in the traditional literature.

Owing to its popularity in traditional clustering literature and its straightforward adaptability to time series-specific distance functions (by replacing the traditional distance function with a time series-specific one), squared error clustering algorithms like *k*-means have become the most widely used clustering approaches in the TSCL literature. Numerous TSCL algorithms have adapted traditional square error partition clustering algorithms by incorporating time series-specific distance and averaging techniques to enhance clustering performance [89, 94, 113, 128, 21, 46, 47]. Later in this chapter we will provide a detailed outline of each of the referenced algorithms.

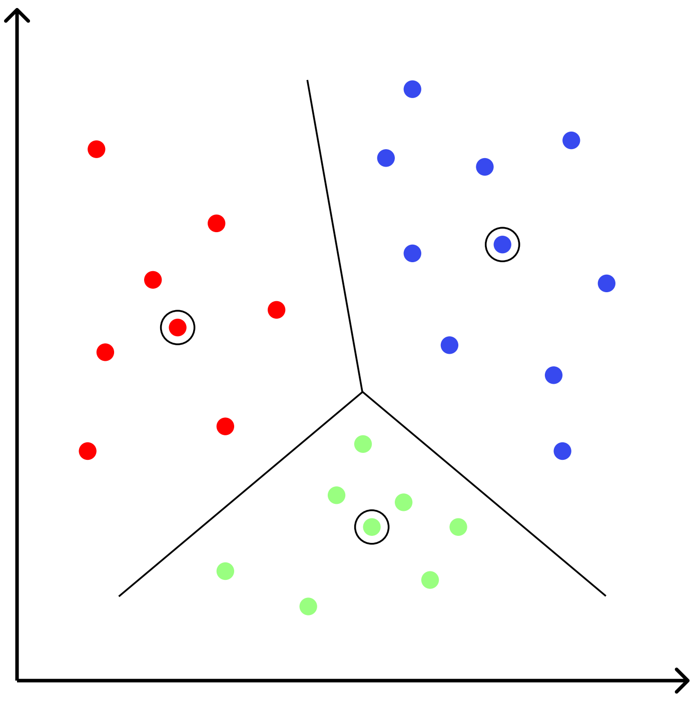


Fig. 2.2 Partition-based clustering example.

### Density-based

Density-based clustering identifies clusters by locating regions in the data where the density of points is higher than in other areas. Clusters are defined as dense areas separated by regions of lower density, and points outside of these dense regions are often considered noise [29]. The definition of a "dense" region depends on how neighborhoods of data points are established. Typically, a distance measure is used to assess the similarity between time series in a dataset. A common approach is to define neighborhoods based on a distance threshold, where two instances are considered to be in the same neighborhood if their distance is within this threshold.

A distinctive feature of some density-based methods, such as Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [29], is that instances not located near dense regions are not assigned to any cluster and are instead labelled as noise. Figure 2.3 illustrates an example of density-based clustering.

Although Figure 2.3 resembles the squared error clustering example in Figure 2.2, it differs in that some instances are marked as noise (depicted in black). Additionally, each cluster is represented by a ellipse that defines dense regions in the data, forming the clusters. Overall, the figure highlights the differing objectives between density-based and squared error clustering algorithms when attempting to partition the data into clusters.

Examples of popular density-based clustering algorithms include DBSCAN [29], Density Peaks (DP) [100], Hierarchical DBSCAN (HDBSCAN) [16], Ordering Points to Identify the Clustering Structure (OPTICS) [5], and Mean-Shift [20]. In the context of TSCL, various density-based models have been adapted for TSCL using time series specific distance measures [9, 51].

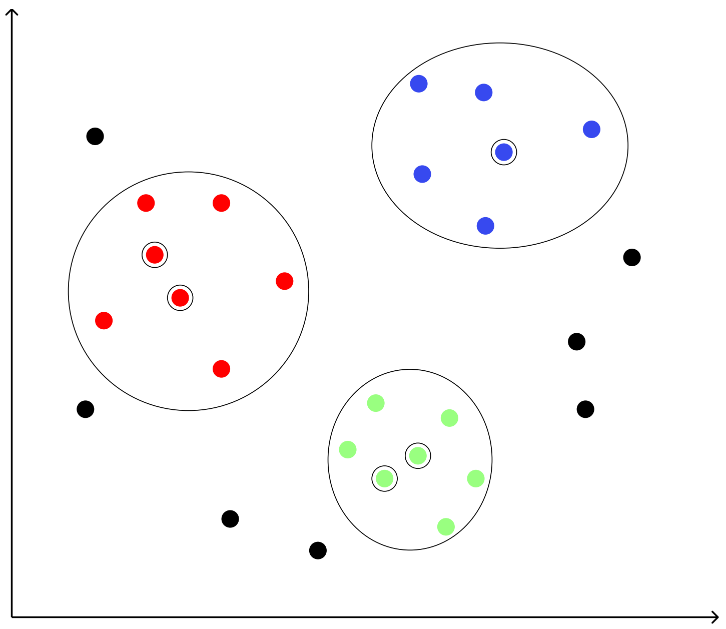


Fig. 2.3 Density clustering example.

### Model-based

Model-based clustering assumes that the data is generated from an underlying probabilistic model, where each cluster corresponds to a component of the model. The most common approach is to represent the data as a mixture of distributions, with each distribution representing a cluster [36]. The goal of model-based clustering is to find the parameters of these distributions that best fit the data. Once the model is established, data points are assigned to clusters based on the likelihood that they were generated by each distribution.

Figure 2.4 illustrates the model-based clustering algorithm Self-Organising Maps (SOM), where the neural network learns the distribution of input data by mapping input neurons to output neurons (clusters). The weights between these neurons are iteratively updated during training, allowing SOM to adapt to the data by learning topological relationships. SOM organises data points into clusters based on their likelihood of belonging to the same underlying distribution.

Examples of popular model-based partition clustering algorithms in the traditional literature include Expectation-Maximisation (EM) [81] and SOM [124], both of which have been applied in TSCL. Several SOM models have been proposed that incorporate time series-specific distance measures [113, 52]. Additionally, EM has been employed to estimate parameters for mixture models, improving clustering performance [123].

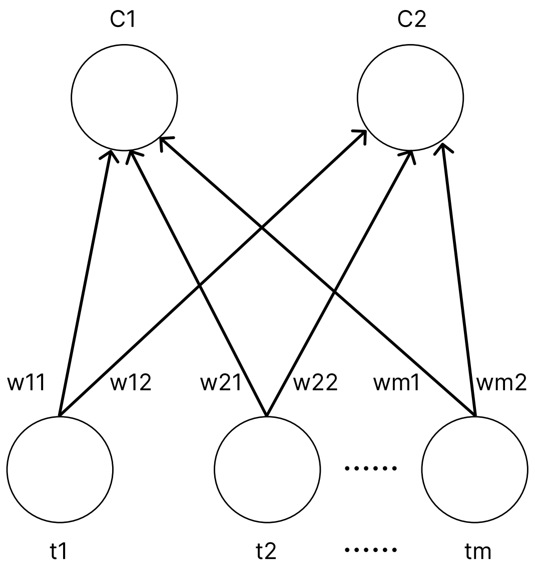


Fig. 2.4 An example of a single-layered neural network clusterer. A time series $T = \{t_1, t_2, \ldots t_m\}$ of length $m$ is mapped to $m$ input neurons (one for each time point). Each input neuron is connected to the output neurons with weights denoted by $w_{n1}$, where $n$ is the input neuron number and the second value represents the corresponding output neuron. The output neuron layer represents the clusters. In this example there are two clusters ($C_1$ and $C_2$).

## 2.3 Time series clustering (TSCL)

TSCL shares the same objective as traditional clustering: to divide data into clusters where instances within the same cluster are homogeneous, and those outside are heterogeneous. The key distinction lies in the type of data being clustered—time series data. As previously established, time series data has unique temporal characteristics that must be exploited to produce “good” clusterings.

Figure 2.5 provides a high-level overview of various clusterers proposed for TSCL. The clusterers highlighted in blue represent traditional clustering models that have been adapted for TSCL, typically through the incorporation of time series-specific distance measures and/or averaging technique. We have only included models that literature specifically for TSCL has been published for.

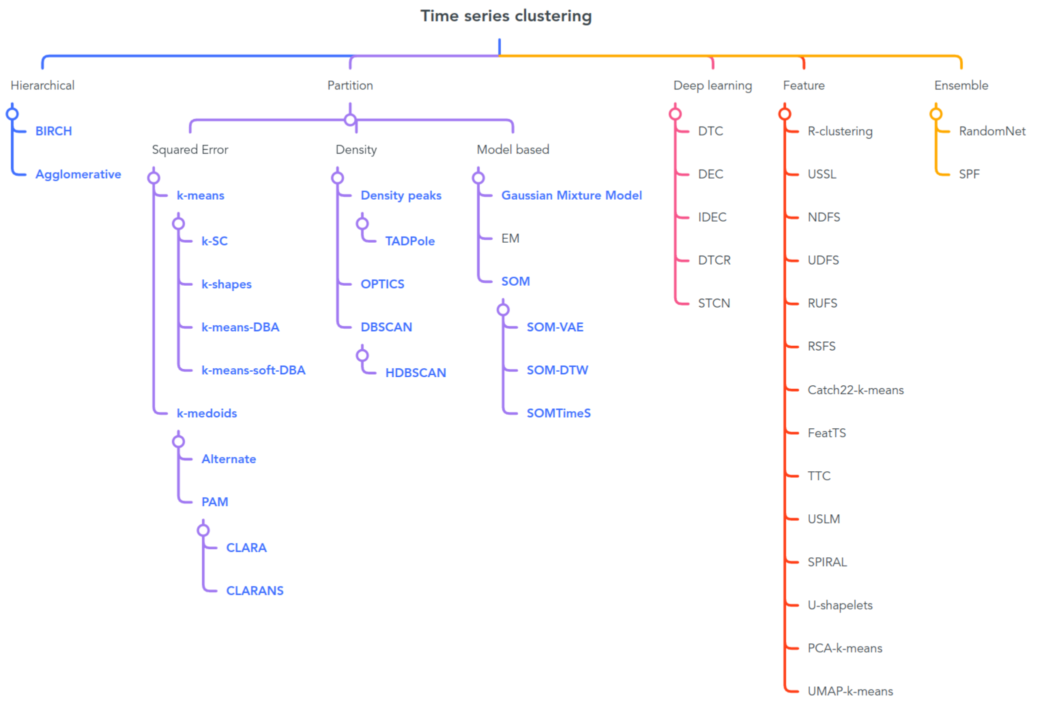


Fig. 2.5 TSCL taxonomy.

In Figure 2.5, alongside the previously defined Hierarchical and Partition-based categories of clustering, we have included three additional categories: “Deep Learning,” “Feature-based,” and “Ensemble”. We separate Deep Learning because it encompasses a wide range of clustering models that could potentially fit into multiple other categories. For simplicity, we place them in their own category.

Feature-based time series machine is another popular time series analysis field. In traditional clustering, input vectors are often considered “features”, which means feature-based methods could theoretically be classified as traditional methods. However, a key aspect of feature-based TSCL is the selection and generation of unsupervised features, typically paired with a specific clustering model. Given the extensive literature on unsupervised feature generation and selection, we assign it its own category.

Lastly, Ensemble TSCL models are placed in a separate category because ensembles can be composed of models from various other categories. To simplify, we treat them as distinct. While Figure 2.5 provides an overview of existing feature-based and deep learning TSCL approaches, we will not explore them further in this thesis, as they are not the focus of our research.

In Figure 2.5, nearly all the models listed under Hierarchical and Partition-based clustering are traditional models that have been adapted with specialised time series optimisations. Of the 20 clusterers highlighted in blue (meaning they are traditional clusterers with time series specific optimisations), 17 have been adapted through the use of a time series-specific distance measure and/or averaging technique. This highlights that adapting traditional models with a distance and averaging method is the primary approach for developing TSCL algorithms.

One of the most commonly used distance measures is Dynamic Time Warping (DTW) [10], which belongs to a family of algorithms called elastic distances. Among the 20 clusterers highlighted in blue, 13 (*k*-means-DBA, *k*-means-soft-DBA, alternate, PAM, CLARA, CLARANS, Density Peaks, TADPole, OPTICS, DBSCAN, HDBSCAN, SOM-DTW, SOMTimeS) have been adapted specifically using an elastic distance. Of those, 12 specifically utilise DTW (*k*-means-soft-DBA uses soft-DTW). This demonstrates that the most common—and as will be shown, one of the most successful—ways of adapting traditional clustering algorithms is through the use of elastic distances.

Since DTW's introduction to time series data mining in 1994 [12], numerous new elastic distances have been proposed [111]. Many of these newer distances have been shown to significantly outperform DTW in supervised time series machine learning tasks, such as TSC [74, 111]. However, none of these "better" elastic distances for TSC have been explored in the context of TSCL. Therefore, the primary question this thesis seeks to answer is: how do these other elastic distances perform in TSCL?

We will now outline the 12 different elastic distances that will be explored throughout this thesis to test this hypothesis. Following this, we will provide detailed reviews of 10 of the most commonly used TSCL approaches that incorporate a time series-specific distance measure. These will serve as the core clusterers for our analysis and development of new elastic distance TSCL models.

## 2.4 Elastic distances

Measuring the distance between time series is a primitive operation that can be used for a range of tasks such as classification, clustering, extrinsic regression, anomaly detection and retrieval. The simplest way to calculate the distance between two time series is to use the $L_p$ distance also known as the Minkowski distance [111].

$$L_p(a,b) = \left( \sum_{i}^{m} |a_i - b_i|^p \right)^{\frac{1}{p}} \tag{2.2}$$

where $p$ is the order of the norm, $a$ and $b$ are time series, $m$ is the length of the time series and $i$ is the current index in the time series that are being considered.

Equation 2.2 shows the $L_p$ distance which takes a parameter $p$. The $L_1$ $(p = 1)$ distance is known as the Manhattan distance and the $L_2$ $(p = 2)$ distance is known as the Euclidean distance. The $L_p$ distance is of limited application for time series machine learning since it does not perform any alignment between two given time series. This forces distance calculations to only consider point-to-point comparisons between two series and does not allow for any misalignment between series in the $x$ axis, as shown in an example in Figure 2.6.

Distances that account for the misalignment between two time series during the distance computation, can be considered as elastic distances. While there have been numerous elastic distances proposed since their first conception, this thesis will only consider a subset which are the most popular in recent literature.

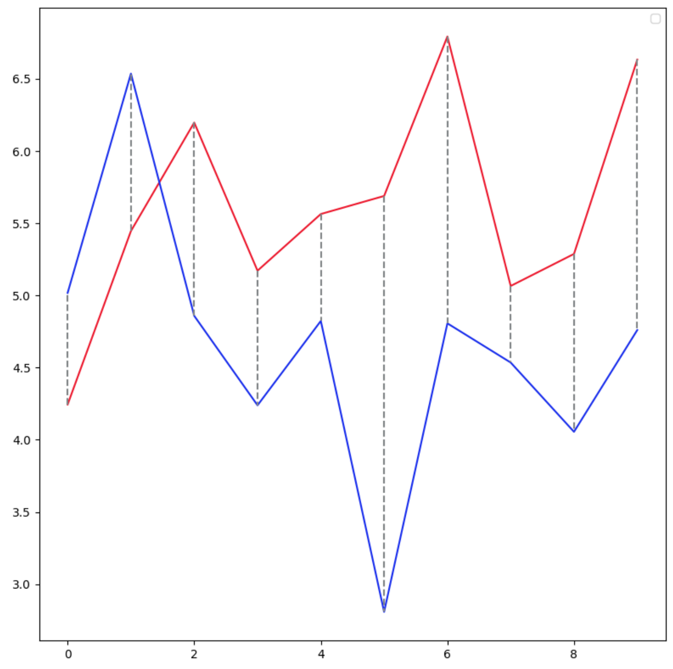


Fig. 2.6 Example of alignment between two time series when using a $L_p$ distance. The dashed grey lines represents which points in the red time series are compared to in the blue time series.

### 2.4.1 Dynamic time warping (DTW)

Dynamic time warping (DTW) [10] was first proposed for time series machine learning by [12]. DTW works by allowing one-to-many alignments ("warping") of points between two time series [111]. For disciplines such as TSC the one nearest neighbor (1-NN) classifier using a tuned version of DTW was considered state-of-the-art for many years and is still used as a baseline comparison for TSC benchmarks [84].

DTW uses dynamic programming to find the optimal path through a cost matrix ($CM$) that minimises the cumulative distance between two time series. It achieves this by constructing a pairwise matrix where each index computes the cost of aligning a point from the first time series with a point from the second time series. Formally, the cost matrix for DTW is denoted as $CM_{dtw}$. Let $CM_{dtw}$ be an $(m_1+1)\times(m_2+1)$ cost matrix with indices starting from $i=1$ and $j=1$, where $m_1$

is the number of time points in first time series and $m_2$ is the number of time points in the second time series. The matrix $CM_{dtw}$ is initialised with $CM_{dtw}(1,1) = 0$ and $+\infty$ for all other indexes.

Once the $CM_{dtw}$ has been initialised, for $i$ from 1 to $m_1$ and $j$ from 1 to $m_2$, the values are incrementally updated such that:

$$CM_{dtw}(i+1, j+1) = (a_i - b_j)^2 + \min \begin{cases} CM_{dtw}(i,j) \\ CM_{dtw}(i+1,j) \\ CM_{dtw}(i,j+1) \end{cases} \tag{2.3}$$

where $a$ is a time series of length $m_1$, and $b$ is a time series of length $m_2$.

Once all values in $CM_{dtw}$ have been computed the DTW distance between time series $a$ and $b$ is given in Equation 2.4.

$$d_{dtw}(a,b) = CM_{dtw}(m_1+1, m_2+1) \tag{2.4}$$

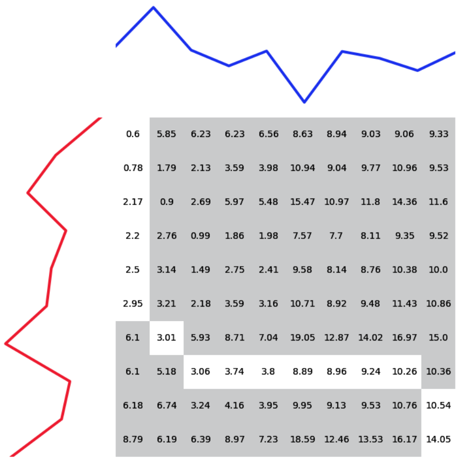
Algorithm 1 shows the algorithmic implementation of DTW outlined in Equations 2.3 and 2.4.

**Algorithm 1:** DTW ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$)

```
1 Input: a (time series of length $m_1$), b (time series of length $m_2$), w
        (window proportion, default value $w \leftarrow 1$)
  Output: DTW distance between a and b
2 Let $CM$ be a $(m_1+1)$ by $(m_2+1)$ matrix initialised with all values $\infty$.
3 $CM_{1,1} \leftarrow 0$.
4 for $i \leftarrow 1$ to $m_1$ do
5     for $j \leftarrow 1$ to $m_2$ do
6         if $|i-j| < w \cdot \max(m_1, m_2)$ then
7             $CM_{i+1,j+1} \leftarrow (a_i - b_j)^2 + \min(CM_{i,j}, CM_{i+1,j}, CM_{i,j+1})$
8 return $CM_{m_1+1,m_2+1}$
```

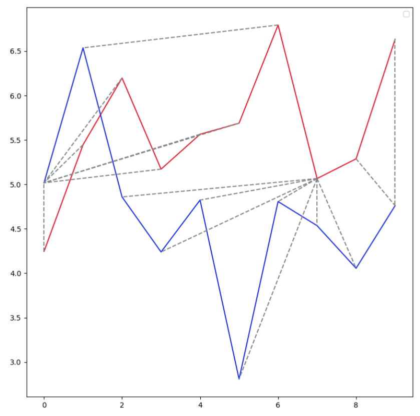
In addition to a distance being extracted from the CM, a "warping path" can also be extracted. A warping path is defined as $P =< (e_1, f_1), (e_2, f_2), \ldots, (e_s, f_s) >$ where each value in $P$ is a pair of indices that define a traversal of matrix $CM$. A valid warping path must start at location $(1,1)$, end at point $(m_1, m_2)$ and should not backtrack.

The optimal warping path through a $CM$ is $P*$ which is defined by creating a path that minimises the cumulative distance through a $CM$. Algorithm 2 shows the process to extract $P*$ from a $CM$. Figure 2.7 (a) shows the optimal path through the $CM_{dtw}$ extracted using Algorithm 2 between the red and the blue time series. Figure 2.7 (b) uses dashed grey lines show which time points in the red time series are aligned to which time points in the blue time series.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

(a) Optimal warping path (white squares) through $CM_{dtw}$ between the red and blue time series.



(b) A visualisation of the DTW alignment between the red and blue time series.

Fig. 2.7 Optimal DTW warping path through $CM_{dtw}$ and a visualisation of DTW alignment between the two time series.

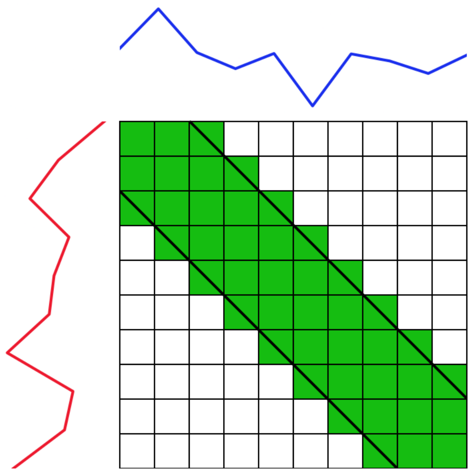
DTW is susceptible to pathological warping where one time point in the first time series is mapped to a large number of time points in the second time series [24]. This can be observed in Figure 2.7 (b) where the eighth time point in the red time series maps to seven time points in the blue time series. Visually we can inspect

**Algorithm 2:** optimal_warping_path (**CM**)

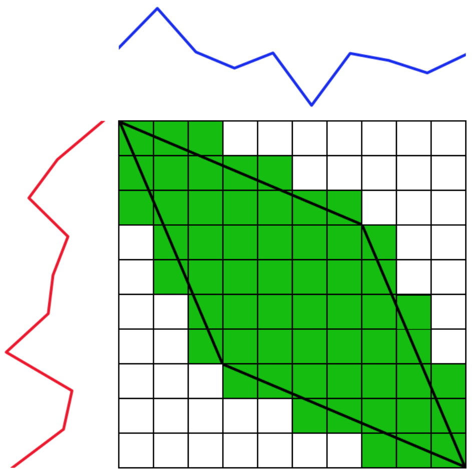
```
Input: CM (cost matrix of shape $m_1 \times m_2$)
Output: List of indices that make up the optimal warping path through CM
Let $(x\_size, y\_size)$ be the dimensions of $CM$
$i \leftarrow x\_size$
$j \leftarrow y\_size$
$P\_star \leftarrow []$
while $i > 1$ or $j > 1$ do
    $P\_star.append((i, j))$
    if $i == 1$ then
        $j \leftarrow j - 1$
    else
        if $j == 1$ then
            $i \leftarrow i - 1$
        else
            $min\_index \leftarrow \text{argmin}([CM_{i-1,j-1}, CM_{i-1,j}, CM_{i,j-1}])$
            if $min\_index == 0$ then
                $i, j \leftarrow i - 1, j - 1$
            else
                if $min\_index == 1$ then
                    $i \leftarrow i - 1$
                else
                    $j \leftarrow j - 1$
$P\_star.append((1, 1))$
$reversed\_P\_star = P\_star.reverse()$
return $reversed\_P\_star$
```

this alignment and see this is not a sensible realignment. To combat pathological warping and improve DTWs time complexity which is $O(m^2)$, bounding windows which constrain the optimal warping path have been proposed. A bounding window limits how many time points can be mapped to a single time point in the other time series.

In the literature the two most common bounding windows are the Sakoe-Chiba bounding window [103] and Itakura Parallelogram bounding window [49]. Figure 2.8 gives a visual representation of the area in which each bounding technique allows DTW to consider an optimal warping path through. Any points in the *CM* highlighted in green can be aligned to however, any in white cannot be aligned to.



(a) Visualisation of Sakoe-chiba bounding bounding where $w = 0.2$.



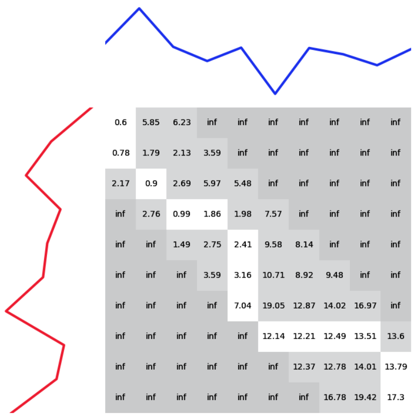
(b) Visualisation of Itakura Parallelogram bounding where $max\_w = 0.2$.

Fig. 2.8 Visualisation of two $CM$ bounding algorithms. Green squares represents within bounds and white squares represent out of bounds.

The Sakoe-Chiba bounding window applies a bounding window of constant width. This width is determined by the parameter $w$ where $0 \leq w \leq 1$. $w$ represents the percentage of the time series of length $m$ the bounding matrix should allow warping through.

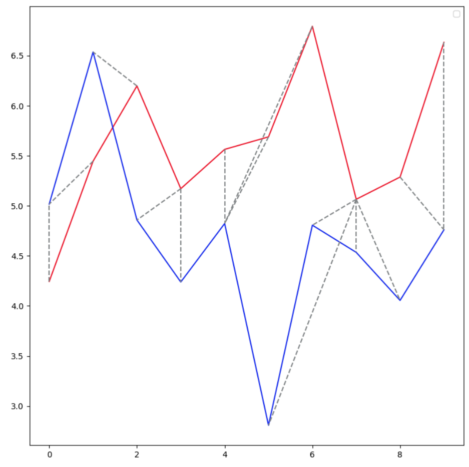
The Itakura Parallelogram bounding window changes the amount of warping allowed based on the location within the $CM$. At the start and end of the window very little warping is allowed. In the middle of the window a large amount of warping is allowed. When visualised this bounding window forms a parallelogram shape over the $CM$ (Figure 2.8 (b)). The maximum width for Itakura Parallelogram is determined by the parameter $max\_w$ where $0 \leq max\_w \leq 1$. $max\_w$ represents the percentage of the time series of length $m$ the parallelogram will allow warping through at its max width.

Figure 2.9 shows the use of a Sakoe-Chiba bounding window of $w = 0.2$ on the $CM_{dtw}$ between the red and the blue time series. Figure 2.9 (b) shows that when a bounding window is applied there is less pathological warping. Specifically the seventh time point in the red time series in Figure 2.9 only warps to 4 other time points in blue.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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) Optimal warping path (white squares) through $CM_{dtw}$ between the red and blue time with a Sakoe-Chiba bounding window of $w = 0.2$.



(b) A visualisation of the DTW alignment between the red and blue time series with a Sakoe-Chiba bounding window of $w = 0.2$.

Fig. 2.9 Optimal DTW warping path through $CM_{dtw}$ using a Sakoe-Chiba bounding window and a visualisation of DTW alignment between the two time series using a Sakoe-Chiba bounding window.

The Sakoe-Chiba bounding window will be assumed as the default window for elastic distances (if a bounding window is specified). Consequently, any pseudocode provided for our elastic distances will include an additional parameter, $w$, which controls the Sakoe-Chiba bounding window. The default value of $w$ is set to 1, indicating that no bounding window is applied. The Sakoe-Chiba bounding window is enforced by the condition $\mathbf{if}(|\mathbf{i}-\mathbf{j}| < \mathbf{w} \cdot \max(\mathbf{m_1}, \mathbf{m_2}))$ in the pseudocode (or a similar line depending on how the $CM$ is initialised). For example, in the DTW algorithm presented in Algorithm 1, the bounding window is applied on line 6.

### 2.4.2 Derivative dynamic time warping (DDTW)

Derivative DTW (DDTW) [60] proposes a modification to DTW that first transforms the input time series into a their first-order derivative form. The motivation for taking the derivative is DTW only considers alignment in the time axis (i.e. $x$ axis). This means that when two time series differ in alignment in $y$ axis (known as "shape") DTW cannot find these difference. By taking the first derivative of the time series this extracts the shapes of the time series allowing DTW to consider the alignment in both the $x$ and the $y$ axis. For example if we had two time points $a_i$ and $b_j$ which have identical values, but $a_i$ is part of a rising trend and $b_j$ is part of a falling trend. DTW would consider a mapping between these two points ideal, although intuitively we would prefer not to map a rising trend to a falling trend [60]. If we were to use DDTW instead these two points would not be mapped together as the trend (or shape) is considered.

The differential series of $\mathbf{a}$ is $\mathbf{a}' = (a'_1, a'_2, \ldots, a'_{m-1})$ where $a'_i$ defines the average of the slopes between $a_{i-1}$, $a_i$, $a_i$ and $a_{i+1}$. Formally this is defined in Equation 2.5.

$$a'_i = \frac{(a_i - a_{i-1}) + \frac{(a_{i+1} - a_{i-1})}{2}}{2} \tag{2.5}$$

where $a$ is a time series and $1 < i < m$.

Using the derived series from Equation 2.5 the DDTW distance can be defined in Equation 2.6.

$$d_{ddtw}(a,b) = d_{dtw}(a',b'). \tag{2.6}$$

Algorithm 3 and 4 show the pseudocode for DDTW described in Equations 2.5 and 2.6. In Algorithm 3, line 2, the loop starts from index 2 and ends at $m-2$. The reason for this is there is no time point before index 1 and no time points after index $m-2$. Due to this the first and last values are removed hence the final array $a'$ being of size $m-2$.

Figure 2.10 visualises the alignment path between the red and the blue time series. When compared to the warping path in Figure 2.7, the warping path in

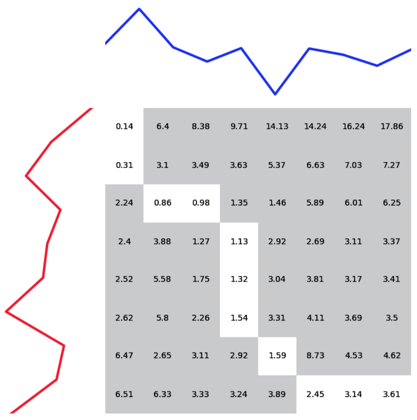
**Algorithm 3:** average_of_slope (**a**)

```
   Input: a (time series of length m)
   Output: time series of length (m − 2)
1  let $a'$ be an array of size $(m-2)$ initialised to zero
2  for $i \leftarrow 2$ to $m-2$ do
3      $a'_{i-1} \leftarrow \frac{(a_i - a_{i-1}) + \frac{(a_{i+1} - a_{i-1}))}{2}}{2}$
4  return $a'$
```

**Algorithm 4:** DDTW (**a**, **b**, **w**)

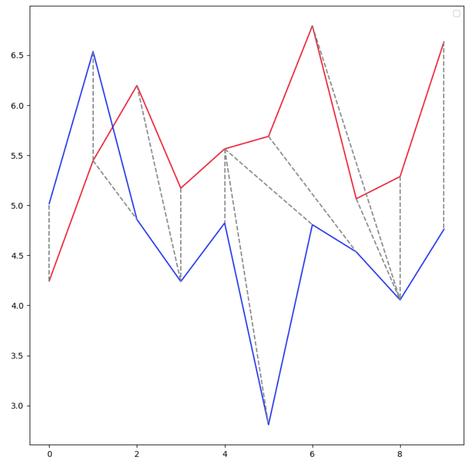
```
   Input: a (time series of length $m_1$), b (time series of length $m_2$), w (window
          proportion, default value $w \leftarrow 1$)
   Output: DDTW distance between a and b
1  $a' \leftarrow$ average_of_slope($a$)
2  $b' \leftarrow$ average_of_slope($b$)
3  return DTW($a'$, $b'$, $w$)
```

Figure 2.10 is much more constrained as a result of accounting for the alignment in the y axis as well as in the *x* axis.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.14 | 6.4 | 8.38 | 9.71 | 14.13 | 14.24 | 16.24 | 17.86 |
| 0.31 | 3.1 | 3.49 | 3.63 | 5.37 | 6.63 | 7.03 | 7.27 |
| 2.24 | 0.86 | 0.98 | 1.35 | 1.46 | 5.89 | 6.01 | 6.25 |
| 2.4 | 3.88 | 1.27 | 1.13 | 2.92 | 2.69 | 3.11 | 3.37 |
| 2.52 | 5.58 | 1.75 | 1.32 | 3.04 | 3.81 | 3.17 | 3.41 |
| 2.62 | 5.8 | 2.26 | 1.54 | 3.31 | 4.11 | 3.69 | 3.5 |
| 6.47 | 2.65 | 3.11 | 2.92 | 1.59 | 8.73 | 4.53 | 4.62 |
| 6.51 | 6.33 | 3.33 | 3.24 | 3.89 | 2.45 | 3.14 | 3.61 |

(a) Optimal warping path (white squares) through $CM_{ddtw}$ between the red and blue time series.



(b) A visualisation of the DDTW alignment between the red and blue time series.

Fig. 2.10 Optimal DDTW warping path through $CM_{ddtw}$ and a visualisation of DDTW alignment between the two time series.

## 2.4.3 Weighted dynamic time warping (WDTW)

Weighted DTW (WDTW) [53] adds a multiplicative penalty to DTW for warping off the diagonal. As observed in Figure 2.7 DTW is susceptible to pathological warping. While a bounding window like the Sakoe-Chiba bounding window offers some remedy to this problem, bounding windows act as hard cut offs to stop warping. To find an appropriate window size for a given problem and additionally for this size to be appropriate throughout the entire bounding window is very challenging. As such WDTW proposes an alternative control for warping by adding a multiplicative weighted penalty which acts as a “soft warping window”.

WDTWs takes a parameter $g$ which controls the level of penalisation that occurs for warping. Formally, Equation 2.7 defines the WDTW penalty.

$$weight_{|i-j|} = \frac{1}{1+e^{-g\cdot(\frac{|i-j|-m}{2})}} \tag{2.7}$$

where $weight_{|i-j|}$ is the multiplicative weight value for position $|i-j|$ indexes off the diagonal warping, $g$ is the parameter that controls the penalty level and $m$ is the length of the time series.

To incorporate this weight penalty into DTW, it is multiplied with the Squared Euclidean distance. This is shown in Equations 2.8 and 2.9. The $CM$ should be initialised with $CM_{wdtw}(1,1) = 0$ and $+\infty$ for all other entries.

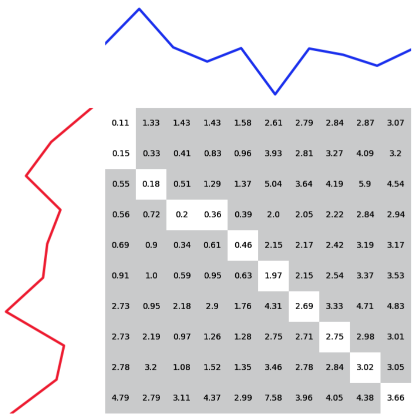
$$CM_{wdtw}(i,j) = weight_{|i-j|} \cdot (a_i - b_j)^2 + \min \begin{cases} CM_{wdtw}(i-1, j-1) \\ CM_{wdtw}(i-1, j) \\ CM_{wdtw}(i, j-1) \end{cases} \tag{2.8}$$

$$d_{wdtw}(a,b) = CM_{wdtw}(m_1+1, m_2+1) \tag{2.9}$$

Algorithm 5 shows the algorithmic implementation of WDTW outlined in Equations 2.7, 2.8 and 2.9.

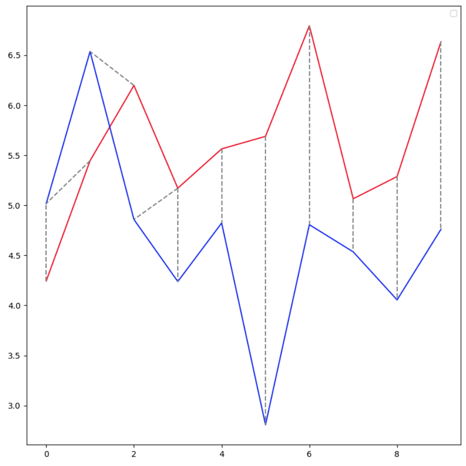
**Algorithm 5:** WDTW (**a**, **b**, **w**, **g**)

```
1 Input: a (time series of length $m_1$), b (time series of length $m_2$), w
         (window proportion, default value $w \leftarrow 1$), g (float that controls the
         penalty level, default value $g \leftarrow 0.05$)
  Output: WDTW distance between a and b
2 Let $CM$ be a $(m_1+1)$ by $(m_2+1)$ matrix initialised with all values $\infty$.
3 $CM_{1,1} \leftarrow 0$.
4 for $i \leftarrow 1$ to $m_1$ do
5     for $j \leftarrow 1$ to $m_2$ do
6         if $|i-j| < w \cdot \max(m_1, m_2)$ then
7             $weight \leftarrow \frac{1}{1+e^{-g\cdot(\frac{|i-j|-\max(m_1,m_2)}{2})}}$
8             $CM_{i+1,j+1} \leftarrow weight \cdot (a_i - b_j)^2 + \min(CM_{i,j}, CM_{i+1,j}, CM_{i,j+1})$
9 return $CM_{m_1+1,m_2+1}$
```



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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 |

(a) Optimal warping path (white squares) through $CM_{wdtw}$ between the red and blue time series where $g = 0.3$.



(b) A visualisation of the WDTW alignment between the red and blue time series where $g = 0.3$.

Fig. 2.11 Optimal WDTW warping path through $CM_{ddtw}$ and a visualisation of WDTW alignment between the two time series where $g = 0.3$.

Figure 2.11 shows a WDTW warping path between the red and blue time series. When compared to the warping path in Figure 2.7 the warping path it is much more constrained. This is the result of the multiplicative penalty.

### 2.4.4 Weighted derivative dynamic time warping (WDDTW)

Weighted derivative DTW (WDDTW) [53] is an extension to WDTW that takes the derivative of the input time series before WDTW is performed. WDDTW attempts to combines the benefits of DDTW and WDTW. This means WDDTW applies a multiplicative warping penalty while taking advantage of shape information gained by taking the derivative of the time series.

WDDTW will first take the derivative of each series shown in Equation 2.5 which will then be passed to Equation 2.9. This process is defined in Equation 2.10.

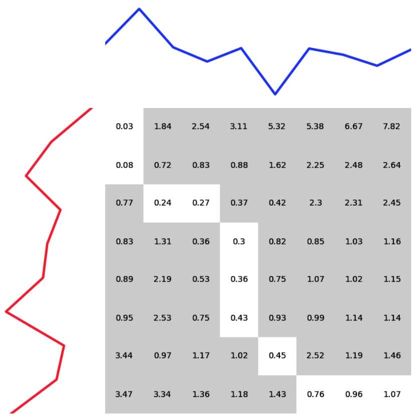
$$d_{wddtw}(a,b) = d_{wdtw}(a',b') \tag{2.10}$$

The pseudocode for WDDTW is defined in Algorithm 6.

**Algorithm 6:** WDDTW ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$, $\mathbf{g}$)

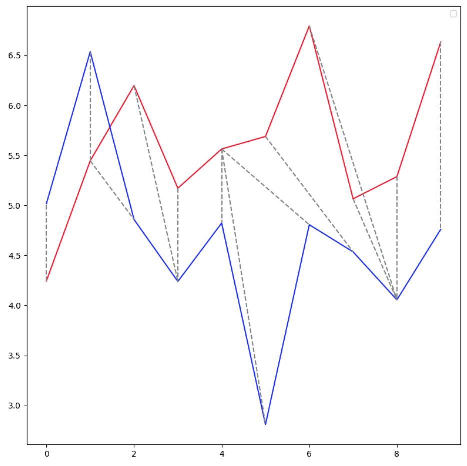
```
1 Input: a (time series of length $m_1$), b (time series of length $m_2$), w
        (window proportion, default value $w \leftarrow 1$), g (float that controls the
        penalty level, default value $g \leftarrow 0.05$)
  Output: WDDTW distance between a and b
2 $a' \leftarrow$ average_of_slope($a$)
3 $b' \leftarrow$ average_of_slope($b$)
4 return WDTW($a'$, $b'$, $w$, $g$)
```

Figure 2.12 illustrates the WDDTW alignment between the red and blue time series. While it forms the same alignment as DDTW, as shown in Figure 2.10, the parameter $g$ controls the level of constraint on the warping path. Increasing the value of $g$ results in a more constrained warping path, while a lower value makes it less constrained.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0.03 | 1.84 | 2.54 | 3.11 | 5.32 | 5.38 | 6.67 | 7.82 |
| 0.08 | 0.72 | 0.83 | 0.88 | 1.62 | 2.25 | 2.48 | 2.64 |
| 0.77 | 0.24 | 0.27 | 0.37 | 0.42 | 2.3 | 2.31 | 2.45 |
| 0.83 | 1.31 | 0.36 | 0.3 | 0.82 | 0.85 | 1.03 | 1.16 |
| 0.89 | 2.19 | 0.53 | 0.36 | 0.75 | 1.07 | 1.02 | 1.15 |
| 0.95 | 2.53 | 0.75 | 0.43 | 0.93 | 0.99 | 1.14 | 1.14 |
| 3.44 | 0.97 | 1.17 | 1.02 | 0.45 | 2.52 | 1.19 | 1.46 |
| 3.47 | 3.34 | 1.36 | 1.18 | 1.43 | 0.76 | 0.96 | 1.07 |

(a) Optimal warping path (white squares) through $CM_{wddtw}$ between the red and blue time series where $g = 0.3$.



(b) A visualisation of the WDDTW alignment between the red and blue time series where $g = 0.3$.

Fig. 2.12 Optimal WDDTW warping path through $CM_{wddtw}$ and a visualisation of WDDTW alignment between the two time series where $g = 0.3$.

## 2.4.5 Amercing dynamic time warping (ADTW)

Amercing DTW (ADTW) [44] is a recently proposed variant of DTW which penalises warping with a fixed additive cost ($\omega$). The motivation for ADTW is to address weaknesses of multiplicative penalties (e.g. WDTW, WDDTW) in that a multiplicative weight is relative to the distances between aligned points along a warping path, rather than being a direct function of the amount of warping [44]. Furthermore, ADTW addresses the limitation of using a bounding window to constrain warping which only looks at potential alignment within a specific bound and ignores anything outside of it.

ADTW takes a parameter $\omega$ which is a constant that is an additive penalty for warping off the diagonal. Assuming a $CM$ initialised with $CM_{adtw}(1,1) = 0$ and $+\infty$ for all other entries, Equation 2.11 shows how to construct the $CM_{adtw}$ and Equation 2.12 shows how to extract a distance from the $CM_{adtw}$.

$$CM_{adtw}(i,j) = (a_i - b_j)^2 + \min \begin{cases} CM_{adtw}(i-1,j-1) \\ CM_{adtw}(i-1,j) + \omega \\ CM_{adtw}(i,j-1) + \omega \end{cases} \tag{2.11}$$

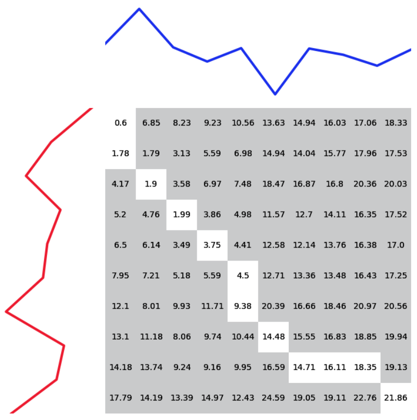
$$d_{adtw}(a,b) = CM_{adtw}(m_1+1, m_2+1) \tag{2.12}$$

The pseudocode for ADTW is given in Algorithm 7.

**Algorithm 7:** ADTW ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$, $\omega$)

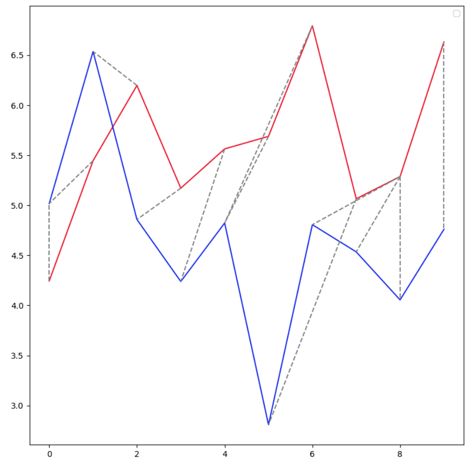
```
1 Input: a (time series of length $m_1$), b (time series of length $m_2$), w
        (window proportion, default value $w \leftarrow 1$) , $\omega$ (additive cost of
        warping, default $\omega \leftarrow 1$)
  Output: ADTW distance between a and b
2 Let $CM$ be a $(m_1+1)$ by $(m_2+1)$ matrix initialised with all values $\infty$.
3 $CM_{1,1} \leftarrow 0$.
4 for $i \leftarrow 1$ to $m_1$ do
5     for $j \leftarrow 1$ to $m_2$ do
6         if $|i-j| < w \cdot \max(m_1, m_2)$ then
7             $CM_{i+1,j+1} \leftarrow$
                $(a_i - b_j)^2 + \min(CM_{i,j}, CM_{i+1,j} + \omega, CM_{i,j+1} + \omega)$
8 return $CM_{m_1+1,m_2+1}$
```

Figure 2.13 shows the ADTW warping path between the red and blue time series. The warping path produced is very similar to DTW using a bounding matrix in Figure 2.9. This shows the effectiveness of the constant to constrain pathological warping.



|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 0.6 | 6.85 | 8.23 | 9.23 | 10.56 | 13.63 | 14.94 | 16.03 | 17.06 | 18.33 |
| 1.78 | 1.79 | 3.13 | 5.59 | 6.98 | 14.94 | 14.04 | 15.77 | 17.96 | 17.53 |
| 4.17 | 1.9 | 3.58 | 6.97 | 7.48 | 18.47 | 16.87 | 16.8 | 20.36 | 20.03 |
| 5.2 | 4.76 | 1.99 | 3.86 | 4.98 | 11.57 | 12.7 | 14.11 | 16.35 | 17.52 |
| 6.5 | 6.14 | 3.49 | 3.75 | 4.41 | 12.58 | 12.14 | 13.76 | 16.38 | 17.0 |
| 7.95 | 7.21 | 5.18 | 5.59 | 4.5 | 12.71 | 13.36 | 13.48 | 16.43 | 17.25 |
| 12.1 | 8.01 | 9.93 | 11.71 | 9.38 | 20.39 | 16.66 | 18.46 | 20.97 | 20.56 |
| 13.1 | 11.18 | 8.06 | 9.74 | 10.44 | 14.48 | 15.55 | 16.83 | 18.85 | 19.94 |
| 14.18 | 13.74 | 9.24 | 9.16 | 9.95 | 16.59 | 14.71 | 16.11 | 18.35 | 19.13 |
| 17.79 | 14.19 | 13.39 | 14.97 | 12.43 | 24.59 | 19.05 | 19.11 | 22.76 | 21.86 |

(a) Optimal warping path (white squares) through $CM_{adtw}$ between the red and blue time series where $\omega = 1$.



(b) A visualisation of the ADTW alignment between the red and blue time series where $\omega = 1$.

Fig. 2.13 Optimal ADTW warping path through $CM_{adtw}$ and a visualisation of ADTW alignment between the two time series where $\omega = 1$.

### 2.4.6 Shape dynamic time warping (shapeDTW)

Shape DTW (shapeDTW) [132] transforms input time series to extract local structures (shapes) before applying DTW. ShapeDTW consists of two main steps. First, each time point is transformed into a *shape descriptor* that encodes structural information from its temporal neighborhood. Second, DTW is used on the two transformed sequences of shape descriptors to extract an alignment path. Finally, using the computed alignment path, the distance between the two time series is calculated.

While multiple shape descriptors are described in [132], many require prior knowledge of the data characteristics. Since unsupervised learning tasks, such as TSCL, assume no prior knowledge of data characteristics, a shape descriptor named "identity" is used. The identity shape descriptor utilises the raw subsequences as the descriptor and as such assumes no prior knowledge of the data.

shapeDTW takes a parameter called *reach* which is the length of the subsequence to consider. The subsequences are extracted using a sliding window. The size of each window is given in Equation 2.13.

$$windows\_size = 2 \times reach + 1 \tag{2.13}$$

As a consequence of using a sliding window approach, the input time series must be padded on both ends to ensure subsequences are well-defined.

To pad the time series $a = (a_1, \ldots, a_i, a_m)$ of length $m$, the start of the time series is padded with the value $a_1$, *reach* times, and the end of the time series is padded with the value $a_m$, *reach* times. The resulting padded time series (*padded_a*) will be of length *padded_m* given in Equation 2.14.

$$padded\_m = m + 2 \times reach \tag{2.14}$$

Algorithm 8 defines the process described above to pre-process a given time series to be used in shapeDTW.

**Algorithm 8:** extract_sliding_window (**a**, **reach**)

```
1 Input: a (time series of length m), reach (length of a subsequence)
  Output: Matrix of subsequences
2 $padded\_a \leftarrow pad\_edges(a)$ // pad_edges assumes process described
    above
3 $padded\_m \leftarrow m + 2 \times reach$
4 $window\_size \leftarrow 2 \times reach + 1$
5 Let subsequences be a matrix of shape window_size by m matrix initialised
    with values 0.
6 for $i \leftarrow 1$ to $m$ do
7     $subsequences_{(:,i)} = a_{(i:i+window\_size)}$
8 return subsequences
```

Once the subsequences have been extracted using Algorithm 8, the subsequences are passed to the DTW *CM* function given in Equation 2.3 and Algorithm 1 (assuming the whole *CM* is returned rather than just the last element of *CM*). Using the $CM_{dtw}$, an optimal warping path can be extracted, and the distance can be computed using Algorithm 9.

**Algorithm 9:** shapeDTW_from_cm (**a**, **b**, **reach**, **CM**)

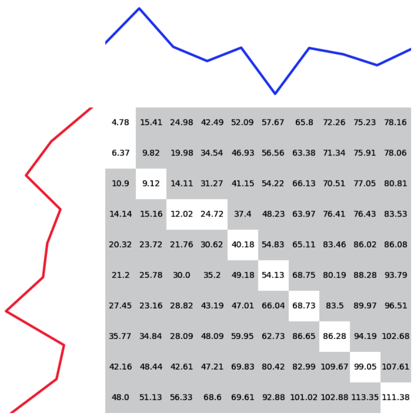
```
1  Input: a (time series of length $m_1$),b (time series of length $m_2$), reach
        (length of a subsequence), CM (CM of shape $m_1 \times m_2$)
   Output: shapeDTW distance
2  $i \leftarrow m_1$
3  $j \leftarrow m_2$
4  $distance \leftarrow 0$
5  while $i \geq 1$ and $j \geq 1$ do
6      $distance \leftarrow distance + (a_{reach+i-1} - b_{reach+j-1})^2$ $min\_a \leftarrow CM_{i-1,j-1}$
7      $min\_b \leftarrow CM_{i,j-1}$
8      $min\_c \leftarrow CM_{i-1,j}$
9      if $min\_a < min\_b$ and $min\_a < min\_c$ then
10         $i = i - 1$ ;
11         $j = j - 1$ ;
12     else
13         if $min\_b < min\_c$ then
14             $j = j - 1$ ;
15         else
16             $i = i - 1$ ;
17 return $distance$ ;
```

Algorithm 10 demonstrates the full algorithm to compute shapeDTW.

**Algorithm 10:** shapeDTW (**a**, **b**, **w**, **reach**)

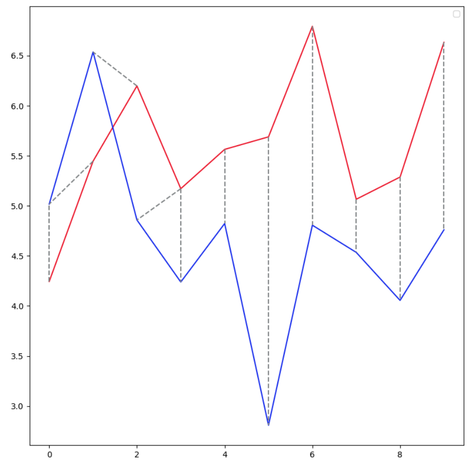
```
1  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
        (window proportion, default value $w \leftarrow 1$) , reach (length of a
        subsequence)
   Output: shapeDTW distance between a and b
2  $window\_a \leftarrow extract\_sliding\_window(a, reach)$
3  $window\_b \leftarrow extract\_sliding\_window(b, reach)$ $CM \leftarrow$
     $DTW\_CM(window\_a, window\_b, w)$ // Algorithm 1 but returns the entire CM
     return $shapeDTW\_from\_cm(a, b, reach, CM)$
```

Figure 2.14 shows the shapeDTW path between the red and blue time series. In Figure 2.14 the first three time points in the blue time series are warped to the first four time points in the red time series. When we visually inspect this alignment in Figure 2.14 (b) it shows the first four time points in the red and blue time series are of very similar shape but are offset by one time point. shapeDTW identifies this offset and realigns the two considering their shape.



|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| 4.78 | 15.41 | 24.98 | 42.49 | 52.09 | 57.67 | 65.8 | 72.26 | 75.23 | 78.16 |
| 6.37 | 9.82 | 19.98 | 34.54 | 46.93 | 56.56 | 63.38 | 71.34 | 75.91 | 78.06 |
| 10.9 | 9.12 | 14.11 | 31.27 | 41.15 | 54.22 | 66.13 | 70.51 | 77.05 | 80.81 |
| 14.14 | 15.16 | 12.02 | 24.72 | 37.4 | 48.23 | 63.97 | 76.41 | 76.43 | 83.53 |
| 20.32 | 23.72 | 21.76 | 30.62 | 40.18 | 54.83 | 65.11 | 83.46 | 86.02 | 86.08 |
| 21.2 | 25.78 | 30.0 | 35.2 | 49.18 | 54.13 | 68.75 | 80.19 | 88.28 | 93.79 |
| 27.45 | 23.16 | 28.82 | 43.19 | 47.01 | 66.04 | 68.73 | 83.5 | 89.97 | 96.51 |
| 35.77 | 34.84 | 28.09 | 48.09 | 59.95 | 62.73 | 86.65 | 86.28 | 94.19 | 102.68 |
| 42.16 | 48.44 | 42.61 | 47.21 | 69.83 | 80.42 | 82.99 | 109.67 | 99.05 | 107.61 |
| 48.0 | 51.13 | 56.33 | 68.6 | 69.61 | 92.88 | 101.02 | 102.88 | 113.35 | 111.38 |

(a) Optimal warping path (white squares) through $CM_{shapeDTW}$ between the red and blue time series where $reach = 2$.



(b) A visualisation of the shapeDTW alignment between the red and blue time series where $reach = 2$.

Fig. 2.14 Optimal shapeDTW warping path through $CM_{shapeDTW}$ and a visualisation of shapeDTW alignment between the two time series where $reach = 2$.

### 2.4.7 Soft dynamic time warping (soft-DTW)

Soft Dynamic Time Warping (soft-DTW) [21] is an adaptation of DTW that uses a soft minimisation function in place of the discrete minimisation function. The soft minimisation function computes a smooth approximation of the minimum by considering a soft-minimum over all possible alignment paths between two time series. As a result, the output of soft-DTW changes smoothly as the input changes, ensuring that the gradient exists and is continuous. This means soft-DTW is differentiable everywhere in the cost matrix.

The soft-minimum function is defined in Equation 2.15:

$$\text{soft\_min}(X) = -\gamma \log \sum_{x_i \in X} e^{\frac{-x_i}{\gamma}} \tag{2.15}$$

where $X$ is a set of values over which the minimum is computed, $\gamma > 0$ controls the smoothness of the approximation, and $x_i$ represents each individual value in $X$.

This soft-minimum function is incorporated into the DTW cost matrix function. This is shown in Equation 2.16. The cost matrix is initialised similarly to DTW.

$$CM_{soft\_dtw}(i,j) = (a_i - b_j)^2 + \text{soft\_min}\begin{pmatrix} CM_{soft\_dtw}(i-1,j-1), \\ CM_{soft\_dtw}(i-1,j), \\ CM_{soft\_dtw}(i,j-1) \end{pmatrix} \tag{2.16}$$

$$d_{soft\_dtw}(a,b) = CM_{soft\_dtw}(m_1,m_2) \tag{2.17}$$

Algorithm 11 illustrates the implementation of the soft-minimum function for three variables $x$, $y$, and $z$, as described in Equation 2.15. Algorithm 12 then shows how this soft_min_three function is integrated into the computation of the soft-DTW distance between two time series $a$ and $b$.

**Algorithm 11:** soft_min (**x**, **y**, **z**, $\gamma$)

```
1  Input: x, y, z (Input floating-point values), $\gamma$ (smoothing parameter)
   Output: Soft-minimum of three input floats with respect to $\gamma$
2  if $\gamma = 0$ then
3      return $\min(x,y,z)$
4  $max\_val \leftarrow \max\left(\frac{x}{-\gamma}, \frac{y}{-\gamma}, \frac{z}{-\gamma}\right)$
5  $summed\_exp \leftarrow e^{\left(\frac{x}{-\gamma} - max\_val\right)} + e^{\left(\frac{y}{-\gamma} - max\_val\right)} + e^{\left(\frac{z}{-\gamma} - max\_val\right)}$
6  return $-\gamma \cdot (\log(summed\_exp) + max\_val)$
```

To compute a single optimal alignment path through the soft-DTW cost matrix, Algorithm 2 can be applied similarly to DTW. However, unlike DTW, the soft-DTW cost matrix is differentiable everywhere, enabling the computation of a gradient with respect to the soft-DTW cost matrix. This gradient matrix can then be interpreted as the expected alignment path under a Gibbs distribution [21]. In other words, the

**Algorithm 12:** soft_DTW ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$, $\gamma$)

```
1  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
          (window proportion, default value $w \leftarrow 1$), $\gamma$ (smoothing
          parameter)
   Output: soft-DTW distance between a and b
2  Let $CM$ be a $(m_1 + 1)$ by $(m_2 + 1)$ matrix initialised with all values $\infty$.
3  $CM_{1,1} \leftarrow 0$.
4  for $i \leftarrow 2$ to $m_1$ do
5      for $j \leftarrow 2$ to $m_2$ do
6          if $|i - j| < w \cdot \max(m_1, m_2)$ then
7              $CM_{i,j} \leftarrow (a_{i-1} - b_{j-1})^2 +$
                 soft_min_three$(CM_{i-1,j}, CM_{i-1,j-1}, CM_{i,j-1}, \gamma)$
8  return $CM_{m_1+1,m_2+1}$
```

gradient matrix represents the contribution of each possible alignment to the final distance.

The soft-DTW gradient matrix can be utilised in various downstream time series tasks. In particular, this thesis focuses on its application to time series averaging. We will outline how a gradient matrix can be used to compute the Soft Dynamic Barycentre Average (soft-DBA) in Section 2.5.1.

To compute the gradient through the soft-DTW cost matrix efficiently, [21] proposed a quadratic runtime algorithm based on backpropagation. The key insight is that the final soft-DTW value ($CM_{m_1,m_2}$) depends on all previous computations in the matrix through the recursive soft-min operations. Therefore, using the chain rule, we can compute the gradient with respect to every value in the cost matrix by recursively propagating derivatives backwards. This backward recursion computes $\frac{\partial CM_{soft\_dtw}(m_1,m_2)}{\partial CM_{soft\_dtw}(i,j)}$ for all cells $(i, j)$. Programmatically the gradient matrix ($E$) can be computed using Algorithm 13.

Figure 2.15 visualises this process. The figure illustrates: (i) the initial cost matrix computed using soft-DTW and (ii) the gradient matrix ($E$), obtained by applying the chain rule to the cost matrix with respect to the distance matrix.

**Algorithm 13:** Soft_DTW_Gradient ($D$, $CM$, $\gamma$)

```
Input: D (pairwise squared Euclidean distance matrix of shape $m_1 \times m_2$),
       CM (accumulated soft-DTW cost matrix of shape $m_1 \times m_2$),
       $\gamma$ (smoothing parameter)
Output: Gradient matrix
1  Let $E$ be an $m_1 \times m_2$ matrix initialised with zeros
2  $E_{m_1,m_2} \leftarrow 1.0$
3  for $i \leftarrow m_1 - 1$ to $1$ step $-1$ do
4      for $j \leftarrow m_2 - 1$ to $1$ step $-1$ do
5          if $i + 1 < m_1$ then
6              $w_h \leftarrow e^{\frac{CM_{i+1,j} - CM_{i,j} - D_{i+1,j}}{\gamma}}$
7              $E_{i,j} \leftarrow E_{i,j} + E_{i+1,j} \cdot w_h$
8          if $j + 1 < m_2$ then
9              $w_v \leftarrow e^{\frac{CM_{i,j+1} - CM_{i,j} - D_{i,j+1}}{\gamma}}$
10             $E_{i,j} \leftarrow E_{i,j} + E_{i,j+1} \cdot w_v$
11         if $(i + 1 < m_1)$ and $(j + 1 < m_2)$ then
12             $w_d \leftarrow e^{\frac{CM_{i+1,j+1} - CM_{i,j} - D_{i+1,j+1}}{\gamma}}$
13             $E_{i,j} \leftarrow E_{i,j} + E_{i+1,j+1} \cdot w_d$
14 return $E$
```

Previously, when using Algorithm 2 with DTW, a single optimal alignment path was obtained, as shown in Figure 2.7(a).

In contrast, the gradient matrix shown in Figure 2.15 provides a more nuanced representation of alignment probabilities, capturing the contribution of every possible alignment. This comparison highlights the advantage of deriving a gradient matrix rather than a single alignment path.

Additionally, in Figure 2.15 (ii), the matrix is presented with a heatmap overlay, where colours closer to purple indicate a lower likelihood of belonging to the optimal alignment path, while indexes with more yellow and green hues are more likely to be in the optimal alignment path. For example, in Figure 2.15 (ii), index $(1,1)$ and index $(5,5)$ are marked in yellow with a value of 1.0, indicating a 100% probability of being part of the alignment path.

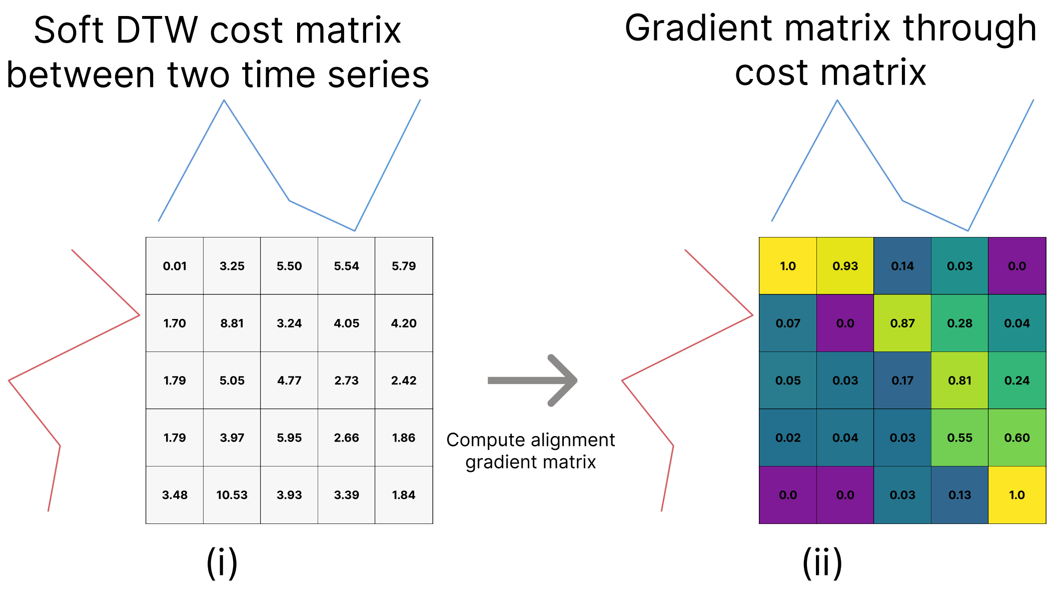


Fig. 2.15 Flow diagram to show how to compute a gradient matrix ($E$) from a cost matrix. For the gradient matrix a heatmap is also used where purple is low values to yellow and green value which represent high values.

## 2.4.8 Longest common subsequence (LCSS)

Longest Common Subsequence (LCSS) [126] is based on the edit distance algorithm which is used for string matching [45]. Figure 2.16 shows an example of LCSS used on a string matching problem. The figure demonstrates the process of adding gaps (denoted by "-") to each series to enable the LCSS "BAAB" to be extracted.

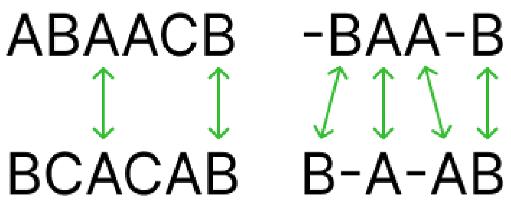


Fig. 2.16 Example of string matching using LCSS. The left image shows a direct pairwise match (i.e. similar to using euclidean distance). The right image shows gaps being allowed, denoted by "-" to find the longest common subsequence between the two words. This allows matching with *elasticity* similar to DTW.

LCSS, is considered an edit distance. Edit distances evaluate the similarity between two time series by counting the number of operations required to make them identical. Each operation has an associated cost, and the total cost represents the distance between the two time series.

For example consider Figure 2.16. To make the two identical it would take 4 operations (removals) to make the two series identical. Therefore the distance between the two time series could be considered 4.

LCSS builds on this concept of edit distances and specifically [126] adapts the original algorithm to allow it to match real-valued data. This is done by adding a floating point threshold $\varepsilon$ where if the difference between two values is less than $\varepsilon$, then they are considered a match. This allows the LCSS algorithm described in [126] to be used for time series data.

In addition as LCSS allows "gaps" if there is no logical match between the series making it less sensitive to noise. However, the quality of result is highly sensitive to the parameterised value $\varepsilon$.

To extract the LCSS distance between two time series, LCSS generates a *CM*. To begin the *CM* is initialised as a matrix of zeros. The *CM* is then generated using Equation 2.18.

$$CM_{lcss}(i,j) = \begin{cases} 1 + CM_{lcss}(i-1,j-1) & \textit{if}\,(|a_i - b_j| \leq \varepsilon) \\ \max \begin{cases} CM_{lcss}(i-1,j) \\ CM_{lcss}(i,j-1) \end{cases} & \textit{otherwise} \end{cases} \tag{2.18}$$

where $CM_{lcss}$ is the LCSS *CM* initialised, $i$ is a integer where $0 \leq i < m$, $j$ is a integer where $0 \leq j < m$, $\varepsilon$ is the matching threshold and $a$ and $b$ are time series of length $m$.

Once all values in $CM_{lcss}$ have been computed the LCSS distance between time series $a$ and $b$ can be extracted using Equation 2.19.

$$d_{lcss}(a,b) = 1 - \frac{CM_{lcss}(m_1+1, m_2+1)}{\min(m_1,m_2)} \tag{2.19}$$

In Equation 2.19 the $CM$ value is subtracted from 1 because, in traditional LCSS, a perfect match between two time series yields a value of 1, while two infinitely dissimilar series yield a value of 0. To ensure consistency with other distance measures in this thesis, where higher values indicate greater dissimilarity, the LCSS value is adjusted accordingly.

Algorithm 14 shows the pseudocode for Equations 2.18 and 2.19

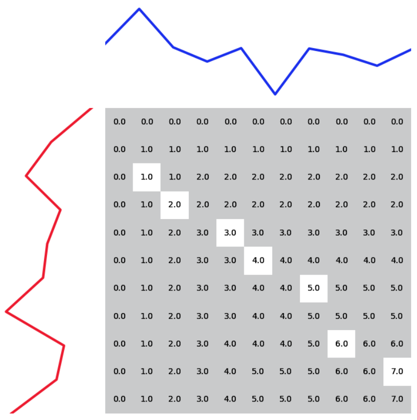
**Algorithm 14:** LCSS ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$, $\varepsilon$)

```
1  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
         (window proportion, default value $w \leftarrow 1$), $\varepsilon$ (matching threshold to
         determine if two subsequences are considered matching, default
         value $\varepsilon \leftarrow 1$.))
   Output: LCSS distance between a and b
2  Let $CM$ be a $(m_1+1)$ by $(m_2+1)$ matrix initialised with all values 0.
3  for $i \leftarrow 2$ to $m_1+1$ do
4      for $j \leftarrow 2$ to $m_2+1$ do
5          if $|i-j| < w \cdot \max(m_1,m_2)$ then
6              if $||a_i - b_j|| \leq \varepsilon$ then
7                  $CM_{i,j} \leftarrow 1 + CM_{i-1,j-1}$
8              else
9                  $CM_{i+1,j+1} \leftarrow \max(CM_{i,j-1}, CM_{i-1,j})$
10 return $1 - (CM_{m_1+1,m_2+1} / \min(m_1,m_2))$
```

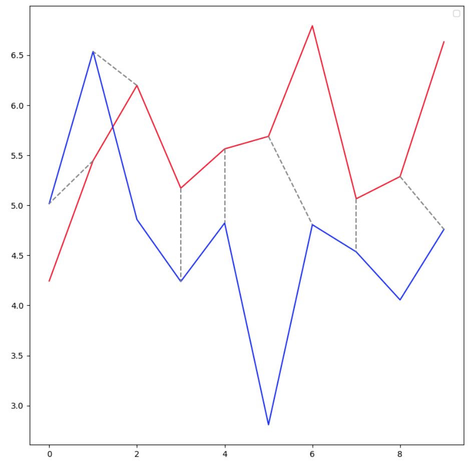
As an optimal LCSS warping path can have gaps this means the algorithm to extract the optimal warping path for DTW and similar elastic distances (Algorithm 2) will not work for LCSS. As such LCSS has its own bespoke algorithm to extract the optimal LCSS warping path. This is given in Algorithm 15.

```
Algorithm 15: optimal_lcss_warping_path ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$, $\varepsilon$, CM)
  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
         (window proportion, default value $w \leftarrow 1$), $\varepsilon$ (matching threshold to
         determine if two subsequences are considered matching, default
         value $\varepsilon \leftarrow 1.$)), CM (CM for LCSS, of shape $(m_1+1, m_2+1)$)
  Output: List of indices that make up the optimal warping path through CM
1 $i \leftarrow m_1 + 1$
2 $j \leftarrow m_2 + 1$
3 $P_s tar \leftarrow []$
4 while $i > 1$ and $j > 1$ do
5     if $|(i-1)-(j-1)| < w \cdot \max(m_1, m_2)$ then
6         if $|a_{i-1} - b_{j-1}| \leq \varepsilon$ then
7             $P_s tar.append((i-1, j-1))$
8             $i \leftarrow i-1$
9             $j \leftarrow j-1$
10        else if $CM[i-1, j] > CM[i, j-1]$ then
11            $i \leftarrow i-1$
12        else
13            $j \leftarrow j-1$
14 $reversed\_P\_star = P\_star.reverse()$
15 return $reversed\_P\_star$
```

Figure 2.17 shows the LCSS alignment path between the red and blue time series. When compared to other warping paths (such as Figures 2.7, 2.10, 2.13 etc.) the most obvious difference is the LCSS optimal warping path has gaps. Figure 2.17 (b) shows it may be logical to have gaps in our alignment paths. For example the fifth index in the blue time series isn't aligned to any index in the red time series. This would seem ideal as there is no similar time points in the red time series therefore it shouldn't be aligned.



(a) Optimal warping path (white squares) through $CM_{lcss}$ between the red and blue time series where $\varepsilon = 1$.



(b) A visualisation of the LCSS alignment between the red and blue time series where $\varepsilon = 1$.

Fig. 2.17 Optimal LCSS warping path through $CM_{lcss}$ and a visualisation of LCSS alignment between the two time series where $\varepsilon = 1$.

### 2.4.9 Edit Distance on Real Sequences (EDR)

Edit Distance on Real Sequences (EDR) [19] builds on LCSS whereby it uses a distance threshold $\varepsilon$ to determine if two time points are considered a match, but in addition it applies a constant penalty for non-matching time point occurs. This is described in Equations 2.20, 2.21 and 2.22.

$$edr\_cost(a_i, b_j) = \begin{cases} 0 & if\ (|a_i - b_j| \leq \varepsilon) \\ 1 & otherwise \end{cases} \tag{2.20}$$

$$CM_{edr}(i,j) = \min \begin{cases} CM_{edr}(i-1, j-1) + edr\_cost(a_i, b_j) \\ CM_{edr}(i-1, j) + 1 \\ CM_{edr}(i, j-1) + 1 \end{cases} \tag{2.21}$$

$$d_{edr}(a,b) = \frac{CM_{edr}(m_1+1, m_2+1)}{\max(m_1, m_2)} \tag{2.22}$$

where $CM_{edr}$ is the EDR $CM$ initialised to zeros, $i$ is a integer where $0 \leq i < m$, $j$ is a integer where $0 \leq j < m$, $\varepsilon$ is the matching threshold and $a$ and $b$ are time series of length $m$.

By applying a constant penalty to gaps it makes the distance more accurate than LCSS [19] and more robust to noise than DTW as it quantises the distance (values increments of 0 or 1).

Finally EDR prescribes a more robust way to select the value of $\varepsilon$ by using a quarter of the maximum standard deviation of the input time series. This is given in Equation 2.24.

$$\sigma(c) = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}(c_i - \bar{c})^2} \tag{2.23}$$

$$\varepsilon = \frac{\max(\sigma(a), \sigma(b))}{4} \tag{2.24}$$

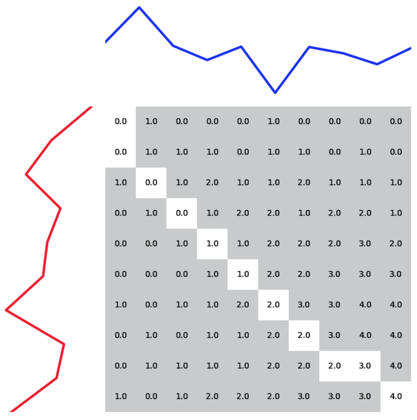
where $a$ and $b$ are time series of length $m$.

Algorithm 16 outlines EDR.

Figure 2.18 shows the EDR alignment path between the red and blue time series. The EDR warping path is much more constrained than the DTW path (Figure 2.7). Specifically the fifth index in the blue time series, which could be considered noise, doesn't throw off the EDR alignment as is does in the DTW $CM$. This highlights EDRs ability to deal with noise by quantising the distance rather than using the Squared Euclidean distance. This quantisation can be clearly seen in Figure 2.18 (a) with the $CM$ values.

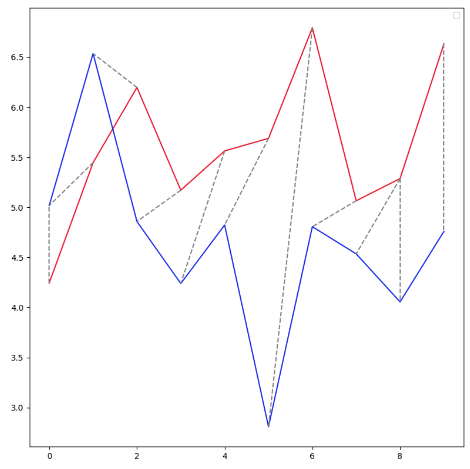
**Algorithm 16:** EDR (**a**, **b**, **w**)

```
1  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
         (window proportion, default value $w \leftarrow 1$)
   Output: EDR distance between a and b
2  $\varepsilon \leftarrow \max(\sigma(a), \sigma(b))/4$
3  Let $CM$ be a $(m_1 + 1)$ by $(m_2 + 1)$ matrix initialised with all values $\infty$.
4  $CM_{1,:} \leftarrow 0$
5  $CM_{:,1} \leftarrow 0$
6  for $i \leftarrow 2$ to $m_1 + 1$ do
7      for $j \leftarrow 2$ to $m_2 + 1$ do
8          if $|(i-1)-(j-1)| < w \cdot \max(m_1, m_2)$ then
9              if $||a_{i-1} - b_{j-1}|| \leq \varepsilon$ then
10                 $cost \leftarrow 0$
11             else
12                 $cost \leftarrow 1$
13             $CM_{i,j} \leftarrow \min(CM_{i-1,j-1} + cost, CM_{i-1,j} + 1, CM_{i,j-1} + 1)$
14 return $CM_{m_1+1,m_2+1}$
```



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 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) Optimal warping path (white squares) through $CM_{edr}$ between the red and blue time series.



(b) A visualisation of the EDR alignment between the red and blue time series.

Fig. 2.18 Optimal EDR warping path through $CM_{edr}$ and a visualisation of EDR alignment between the two time series.

### 2.4.10 Edit Distance with Real Penalty (ERP)

Edit Distance with Real Penalty (ERP) [18] is an edit distance that attempts to align time series by considering how indexes are carried forward through the $CM$. Usually in $CM_{dtw}$, if an alignment cannot be found, the previous value is carried forward. ERP allows gaps or sequences of points that have no matches (similar to LCSS). However, instead of using a threshold, like EDR or LCSS to determine matches, ERP uses the Squared Euclidean distance between elements to measure a match. Additionally the Squared Euclidean distance is used to apply a penalty to warping which changes based on the time point in question. As ERP doesn't use a threshold for a match but instead uses the Squared Euclidean distance, this means that it satisfies triangle inequality which can yield advantages in tasks such as indexing and TSCL.

The penalty of warping ($g$) is set to a constant floating point which is the standard deviation $\sigma$ (Equation 2.23) of the training data.

Equations 2.25 and 2.26 define how the $CM_{erp}$ is generated and how a distance extracted from it.

$$CM_{erp}(i,j) = \min \begin{cases} CM_{erp}(i-1,j-1) + (a_i - b_j)^2 \\ CM_{erp}(i-1,j) + (a_i - g)^2 \\ CM_{erp}(i,j-1) + (b_j - g)^2 \end{cases} \tag{2.25}$$

$$d_{erp}(a,b) = CM_{erp}(m_1+1, m_2+1) \tag{2.26}$$

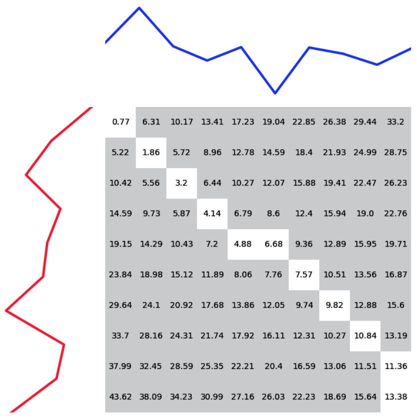
where $CM_{erp}$ is the ERP $CM$ initialised to zeros, $i$ is a integer where $0 \le i < m$, $j$ is a integer where $0 \le j < m$, $g$ is the penalty for moving off the diagonal and $a$ and $b$ are time series of length $m$.

Algorithm 17 defines the implementation of ERP.

**Algorithm 17:** ERP ($\mathbf{a}, \mathbf{b}$, **w**, **g**)

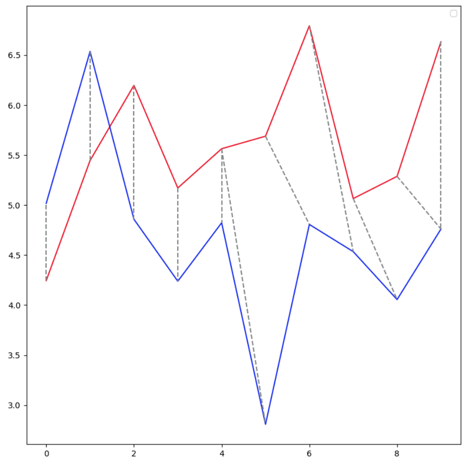
```
1  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
          (window proportion, default value $w \leftarrow 1$), g (penalty for warping
          off the diagonal)
   Output: ERP distance between a and b
2  Let $CM$ be a $(m_1+1)$ by $(m_2+1)$ matrix initialised with all values $\infty$.
3  $CM_{1,1} \leftarrow 0$
4  for $i \leftarrow 2$ to $m_1+1$ do
5      for $j \leftarrow 2$ to $m_2+1$ do
6          if $|(i-1)-(j-1)| < w \cdot \max(m_1, m_2)$ then
7              $match \leftarrow CM_{i-1,j-1} + ||a_{i-1} - b_{j-1}||$
8              $insert \leftarrow CM_{i-1,j} + (a_{i-1} - g)$
9              $delete \leftarrow CM_{i,j-1} + (b_{j-1} - g)$
10             $CM_{i,j} \leftarrow \min(match, insert, delete)$
11 return $CM_{m_1+1,m_2+1}$
```

Figure 2.19 shows the ERP warping path between the red and the blue time series. The warping path is constrained due to the penalty applied for warping off the diagonal.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.77 | 6.31 | 10.17 | 13.41 | 17.23 | 19.04 | 22.85 | 26.38 | 29.44 | 33.2 |
| 5.22 | 1.86 | 5.72 | 8.96 | 12.78 | 14.59 | 18.4 | 21.93 | 24.99 | 28.75 |
| 10.42 | 5.56 | 3.2 | 6.44 | 10.27 | 12.07 | 15.88 | 19.41 | 22.47 | 26.23 |
| 14.59 | 9.73 | 5.87 | 4.14 | 6.79 | 8.6 | 12.4 | 15.94 | 19.0 | 22.76 |
| 19.15 | 14.29 | 10.43 | 7.2 | 4.88 | 6.68 | 9.36 | 12.89 | 15.95 | 19.71 |
| 23.84 | 18.98 | 15.12 | 11.89 | 8.06 | 7.76 | 7.57 | 10.51 | 13.56 | 16.87 |
| 29.64 | 24.1 | 20.92 | 17.68 | 13.86 | 12.05 | 9.74 | 9.82 | 12.88 | 15.6 |
| 33.7 | 28.16 | 24.31 | 21.74 | 17.92 | 16.11 | 12.31 | 10.27 | 10.84 | 13.19 |
| 37.99 | 32.45 | 28.59 | 25.35 | 22.21 | 20.4 | 16.59 | 13.06 | 11.51 | 11.36 |
| 43.62 | 38.09 | 34.23 | 30.99 | 27.16 | 26.03 | 22.23 | 18.69 | 15.64 | 13.38 |

(a) Optimal warping path (white squares) through $CM_{erp}$ between the red and blue time series.



(b) A visualisation of the ERP alignment between the red and blue time series.

Fig. 2.19 Optimal ERP warping path through $CM_{erp}$ and a visualisation of ERP alignment between the two time series.

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

Move-split-merge (MSM) [114] is an edit distance that assigns different "costs" depending on the type of edit and its location within $CM_{msm}$. This differs from similar edit distances such as ERP that for any given insertion or deletion operation assigns a cost that is the absolute magnitude of the value that was inserted or deleted [114].

For example when using ERP the cost to insert a 1 between 1$s$ would yield a cost of 1. However, the cost to insert a 10 between two 10$s$ would yield a cost of 10. This means for the same edit operation, using ERP, the cost is dependent on the value itself. MSM instead recognises these two edit operations are the same by considering the $CM$ neighborhood, and therefore the cost should be constant.

MSM computes the distance between two time series based on the number and type of edit operations required to transform one series to the other [111]. MSM has three types of edits: move, split and merge. A move operation substitutes one value into another value, a split operation inserts a copy of the value immediately after itself and a merge operation removes a value if it is directly followed by an identical value. Additionally MSM applies a cost (penalty) for warping off the diagonal (merge and split) set by the parameter $c$. Figure 2.20 visually demonstrates each operation.

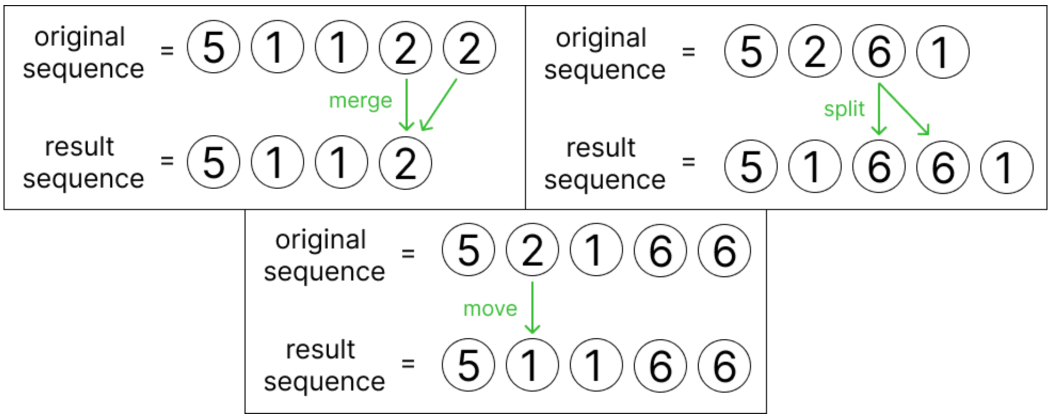

original sequence = 5 1 1 2 2
merge
result sequence = 5 1 1 2
original sequence = 5 2 6 1
split
result sequence = 5 1 6 6 1
original sequence = 5 2 1 6 6
move
result sequence = 5 1 1 6 6

Fig. 2.20 Example of different MSM edit operations. The top left box shows the *merge* operation, the top right box shows the *split* operation and the bottom box shows the *move* operation. The *original sequence* refers to the series before MSM has applied a operation. The *result sequence* refers to the series after the MSM operation has been applied.

MSM satisfies triangle inequality and is formally defined in Equations 2.27, 2.28 and 2.29

$$msm\_cost(a_i, a_{i-1}, b_j, c) = \begin{cases} c & if\ (a_{i-1} \leq a_i \leq b_j) \\ c & if\ (a_{i-1} \geq a_i \geq b_j) \\ c + \min \begin{cases} |a_i - a_{i-1}| \\ |a_i - b_j| \end{cases} & otherwise \end{cases} \tag{2.27}$$

$$CM_{msm}(i, j) = \min \begin{cases} CM_{msm}(i-1, j-1) + |a_i - b_j| \\ CM_{msm}(i-1, j) + msm\_cost(a_i, a_{i-1}, b_j, c) \\ CM_{msm}(i, j-1) + msm\_cost(b_j, a_i, b_{i-1}, c) \end{cases} \tag{2.28}$$

$$d_{msm}(a,b) = CM_{msm}(m_1,m_2) \tag{2.29}$$

where $CM_{msm}$ is the MSM *CM* initialised to zeros, $i$ is a integer where $0 \leq i < m$, $j$ is a integer where $0 \leq j < m$, $c$ is the cost for moving off the diagonal and $a$ and $b$ are time series of length $m$.

MSM has a more sophisticated cost function (Equation 2.27) than other elastic distances such as EDR (Equation 2.20). A move edit does not use the cost function and is the absolute difference between the values. A split edit does use the cost function: $msm\_cost(a_i, a_{i-1}, b_j, c)$. A split operation considers if the value being inserted, $b_j$, is between two values $a_i$ and $a_{i-1}$. If true, the cost is constant $c$. However, if false, the cost $c$ is added to the minimum deviation from the further point $a_i$ and the previous point $b_i$ or $a_{i-1}$. A merge edit also uses the MSM cost function: $msm\_cost(b_j, a_i, b_{i-1}, c)$. The merge edit is similar to the split edit however, it computes the cost on the second series ($b$). By using this more sophisticated cost function, MSM can compute a more context aware warping path while satisfying triangle inequality.

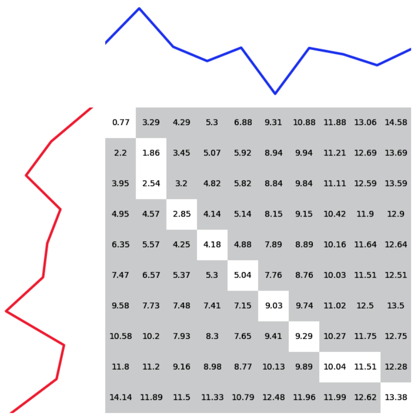
Algorithms 18 and 19 define MSMs implementation.

```
Algorithm 18: cost_msm(x, y, z, c)
  Input: x (first time point), y (second time point), z (third time point), c cost
         of warping off the diagonal
  Output: Cost value
1 if ($y \leq x \leq z$) or ($y \geq x \geq z$) then
2     return $c$
3 return $c + \min(|x - y|, |x - z|)$
```

Figure 2.21 shows the MSM warping path between the red and blue time series. Figure 2.21 has a very constrained warping path compared to DTWs warping path in Figure 2.7. The MSM warping path is more constrained due to its sophisticated cost function penalising pathological warping.

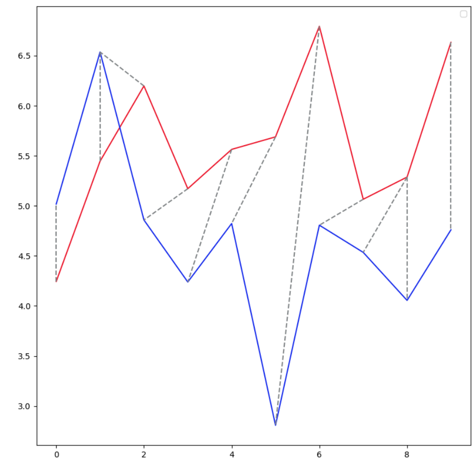
**Algorithm 19:** MSM ($\mathbf{a}, \mathbf{b}$, $\mathbf{w}$, $\mathbf{c}$)

```
1  Input: a (time series of length $m_1$), b (time series of length $m_2$), w
         (window proportion, default value $w \leftarrow 1$), c (cost of warping off
         the diagonal)
   Output: MSM distance between a and b
2  Let $CM$ be an $m_1 \times m_2$ matrix initialised to $\infty$.
3  $CM_{1,1} \leftarrow |a_1 - b_1|$
4  for $i \leftarrow 2$ to $m_1$ do
5      if $|(i-1)-0| < w \cdot \max(m_1, m_2)$ then
6          $split \leftarrow CM_{i-1,1} + cost\_msm(a_i, a_{i-1}, b_1, c)$
7          $CM_{i,1} \leftarrow split$
8  for $j \leftarrow 2$ to $m_2$ do
9      if $|0-(j-1)| < w \cdot \max(m_1, m_2)$ then
10         $merge \leftarrow CM_{1,j-1} + cost\_msm(b_j, b_{j-1}, a_1, c)$
11         $CM_{1,j} \leftarrow merge$
12 for $i \leftarrow 2$ to $m_1$ do
13     for $j \leftarrow 2$ to $m_2$ do
14         if $|(i-1)-(j-1)| < w \cdot \max(m_1, m_2)$ then
15             $move \leftarrow CM_{i-1,j-1} + |a_i - b_j|$
16             $split \leftarrow CM_{i-1,j} + cost\_msm(a_i, a_{i-1}, b_j, c)$
17             $merge \leftarrow CM_{i,j-1} + cost\_msm(b_j, a_i, b_{j-1}, c)$
18             $CM_{i,j} \leftarrow \min(move, split, merge)$
19 return $CM_{m_1,m_2}$
```



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.77 | 3.29 | 4.29 | 5.3 | 6.88 | 9.31 | 10.88 | 11.88 | 13.06 | 14.58 |
| 2.2 | 1.86 | 3.45 | 5.07 | 5.92 | 8.94 | 9.94 | 11.21 | 12.69 | 13.69 |
| 3.95 | 2.54 | 3.2 | 4.82 | 5.82 | 8.84 | 9.84 | 11.11 | 12.59 | 13.59 |
| 4.95 | 4.57 | 2.85 | 4.14 | 5.14 | 8.15 | 9.15 | 10.42 | 11.9 | 12.9 |
| 6.35 | 5.57 | 4.25 | 4.18 | 4.88 | 7.89 | 8.89 | 10.16 | 11.64 | 12.64 |
| 7.47 | 6.57 | 5.37 | 5.3 | 5.04 | 7.76 | 8.76 | 10.03 | 11.51 | 12.51 |
| 9.58 | 7.73 | 7.48 | 7.41 | 7.15 | 9.03 | 9.74 | 11.02 | 12.5 | 13.5 |
| 10.58 | 10.2 | 7.93 | 8.3 | 7.65 | 9.41 | 9.29 | 10.27 | 11.75 | 12.75 |
| 11.8 | 11.2 | 9.16 | 8.98 | 8.77 | 10.13 | 9.89 | 10.04 | 11.51 | 12.28 |
| 14.14 | 11.89 | 11.5 | 11.33 | 10.79 | 12.48 | 11.96 | 11.99 | 12.62 | 13.38 |

(a) Optimal warping path (white squares) through $CM_{msm}$ between the red and blue time series where $c = 1$.



(b) A visualisation of the MSM alignment between the red and blue time series where $c = 1$.

Fig. 2.21 Optimal MSM warping path through $CM_{msm}$ and a visualisation of MSM alignment between the two time series.

### 2.4.12 Time Warp Edit (TWE)

Time Warp Edit (TWE) [79] is another edit distance adapted to the time series domain. TWE combines an $L_p$ distance technique with an edit based algorithm that allows warping while remaining a metric (satisfies triangle inequality). The objective of TWE is to remove time points from either time series $a$ or $b$ until they are equal. To do so TWE defines three edit operations: $match$, $delete_a$, $delete_b$.

Furthermore, TWE introduces the concept of "stiffness". Stiffness ($\nu$) is a multiplicative penalty that is applied to all three edit operations. Stiffness controls the elasticity of the algorithm (how much it can warp). When the value of $\nu = 0$ TWE becomes more stiff like an $L_p$ distance. When the value of $\nu = \infty$ TWE becomes less stiff and more elastic like DTW. In addition TWE has another parameter, $\lambda$, which is a constant penalty for warping off the diagonal (applied to $delete_a$ and $delete_b$). By having two parameters to control many aspects of how a warping path is generated, TWE can be one of the most versatile elastic distances.

Equations 2.30 and 2.31 formally define TWE.

$$CM_{twe}(i,j) = \min \begin{cases} CM_{twe}(i-1,j-1) + (a_i - b_j)^2 + (a_{i-1} - b_{j-1})^2 + 2 \cdot \nu \\ CM_{twe}(i-1,j) + (a_i - a_{i-1})^2 + \nu + \lambda \\ CM_{twe}(i,j-1) + (b_j - b_{j-1})^2 + \nu + \lambda \end{cases} \tag{2.30}$$

$$d_{twe}(a,b) = CM_{twe}(m_1+1, m_2+1) \tag{2.31}$$

where $CM_{twe}$ is the TWE $CM$ initialised with $CM_{twe}(1,1) = 0$ and $+\infty$ for all other entries, $i$ is a integer where $0 \leq i < m$, $j$ is a integer where $0 \leq j < m$, $\nu$ is the stiffness determining elasticity, $\lambda$ is the constant penalty for warping off the diagonal and $a$ and $b$ are time series of length $m$.

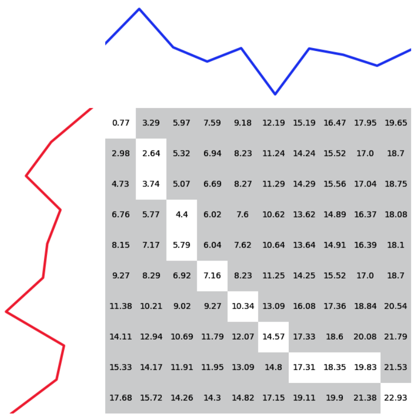
Equation 2.30 shows that the “match” condition: $CM_{twe}(i-1, j-1) + (a_i - b_j)^2 + (a_{i-1} - b_{j-1})^2 + 2 \cdot \nu$ is more sophisticated than other edit distances. Specifically the current and previous time point values are considered $(a_i - b_j)^2 + (a_{i-1} - b_{j-1})^2$ to give greater context to the match allowing smoother alignments to be generated while being less sensitive to noise.

Algorithm 20 defines TWE from Equations 2.30 and 2.31.

**Algorithm 20:** TWE(**a**, **b**, **w**, $\lambda$, $\nu$)

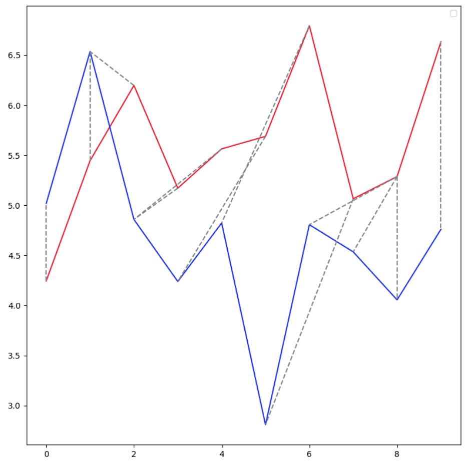
```
Input: a (time series of length $m_1$), b (time series of length $m_2$), w
       (window proportion, default value $w \leftarrow 1$) $\lambda$ (cost of warping off
       the diagonal, default value $\lambda \leftarrow 1.0$) $\nu$ (stiffness of warping, default
       value $\nu \leftarrow 0.001$)
Output: TWE distance between a and b
1  Let $CM$ be a $m_1$ by $m_2$ matrix initialised with all values $\infty$.
2  $CM_{1,1} \leftarrow 0$
3  for $i \leftarrow 2$ to $m_1$ do
4      for $j \leftarrow 2$ to $m_2$ do
5          if $|(i-1)-(j-1)| < w \cdot \max(m_1, m_2)$ then
6              $match = CM_{i-1,j-1} + ||a_i - b_j|| + ||a_{i-1} - b_{j-1}|| + \nu \times (|i-j| + |(i-1)-(j-1)|)$
7              $delete = CM_{i-1,j} + ||a_i - a_{i-1}|| + \nu + \lambda$
8              $insert = CM_{i,j-1} + ||b_j - b_{j-1}|| + \nu + \lambda$
9              $CM_{i,j} = \min(match, insert, delete)$
10 return $CM_{m_1,m_2}$
```

Figure 2.22 shows the TWE warping path between the red and blue time series. The warping path allows some warping while remaining somewhat constrained. Figure 2.22 warps more than other edit distances (Figures 2.18, 2.19 and 2.21). This is because the stiffness parameter is set very low: $\nu = 0.001$. To make the path more constrained we could increase the value of $\nu$ or increase the penalty $\lambda = 1$. TWEs additional parameters makes it very versatile.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.77 | 3.29 | 5.97 | 7.59 | 9.18 | 12.19 | 15.19 | 16.47 | 17.95 | 19.65 |
| 2.98 | 2.64 | 5.32 | 6.94 | 8.23 | 11.24 | 14.24 | 15.52 | 17.0 | 18.7 |
| 4.73 | 3.74 | 5.07 | 6.69 | 8.27 | 11.29 | 14.29 | 15.56 | 17.04 | 18.75 |
| 6.76 | 5.77 | 4.4 | 6.02 | 7.6 | 10.62 | 13.62 | 14.89 | 16.37 | 18.08 |
| 8.15 | 7.17 | 5.79 | 6.04 | 7.62 | 10.64 | 13.64 | 14.91 | 16.39 | 18.1 |
| 9.27 | 8.29 | 6.92 | 7.16 | 8.23 | 11.25 | 14.25 | 15.52 | 17.0 | 18.7 |
| 11.38 | 10.21 | 9.02 | 9.27 | 10.34 | 13.09 | 16.08 | 17.36 | 18.84 | 20.54 |
| 14.11 | 12.94 | 10.69 | 11.79 | 12.07 | 14.57 | 17.33 | 18.6 | 20.08 | 21.79 |
| 15.33 | 14.17 | 11.91 | 11.95 | 13.09 | 14.8 | 17.31 | 18.35 | 19.83 | 21.53 |
| 17.68 | 15.72 | 14.26 | 14.3 | 14.82 | 17.15 | 19.11 | 19.9 | 21.38 | 22.93 |

(a) Optimal warping path (white squares) through $CM_{twe}$ between the red and blue time series where $\nu = 0.001$ and $\lambda = 1$.



(b) A visualisation of the TWE alignment between the red and blue time series where $\nu = 0.001$ and $\lambda = 1$.

Fig. 2.22 Optimal TWE warping path through $CM_{twe}$ and a visualisation of TWE alignment between the two time series where $\nu = 0.001$ and $\lambda = 1$.

## 2.5 TSCL Algorithms

Thus far, we have defined time series data mining, positioned the TSCL discipline within it, and provided a high-level overview of traditional clustering methods. We have also highlighted how many TSCL approaches adapt these traditional algorithms by incorporating time series-specific distance measures and/or averaging techniques. Additionally, we have outlined 12 elastic distances for potential use with these traditional clustering algorithms.

To benchmark our proposed elastic distance TSCL algorithms, we must first establish a point of comparison by reviewing the existing TSCL approaches. Therefore, we outline 10 of the most popular and widely cited partition-based TSCL algorithms. By evaluating and implementing these algorithms, we can better situate our new methods within the current literature and state-of-the-art.

As previously mentioned, we do not consider deep learning-based or feature-based TSCL algorithms, as they do not adapt traditional clustering methods using distance and/or averaging techniques. For readers interested in these areas, there are several comprehensive reviews on deep learning-based TSCL [3, 64], as well as an extensive body of literature on unsupervised feature selection and generation [130, 54, 110, 102, 129, 73, 95].

Furthermore, many existing TSCL approaches follow a similar methodology to what we plan to implement: adapting traditional clustering algorithms with time series-specific distances and/or averaging techniques. By reviewing how this has been done in previous research, we can refine the design and implementation of our proposed clustering algorithms.

We will begin this section by clearly defining the objectives of partition-based TSCL algorithms, using visualisations for clarity. Following this, we will outline 10 TSCL algorithms, offering detailed descriptions, formal notation, and pseudocode for each.

### 2.5.1 Partition-based TSCL

As previously outlined, the primary objective of partition-based clustering is to define *k* centroids (or exemplars or cluster centres) that represent each cluster. A new time series is assigned to a cluster based on its similarity to these centroids.

In traditional clustering, the concept of a centroid, with multiple points assigned to it, can easily be visualised, as shown in Figure 2.2. However, this concept is less intuitive for TSCL. Therefore, we begin by visually demonstrating how partition-based TSCL algorithms partition a dataset of time series.

Figure 2.23 presents a visualisation of partition-based clustering for time series data. Clusters of similar time series are formed (Cluster 1, 2, and 3), and a centroid—represented by the coloured time series—is derived from the clustered data.

Numerous partition-based models have been proposed for TSCL. However, at their core, partition-based TSCL models aims to answer solve two problems:

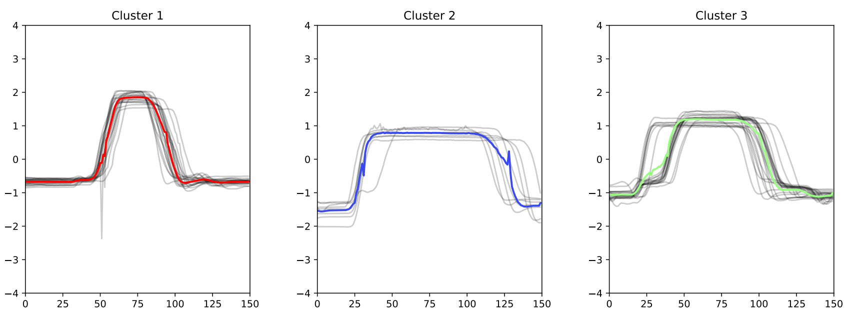


Fig. 2.23 Example of partition-based clustering using k-means on the *Gunpoint* time series dataset. k-means employs the Euclidean distance and arithmetic mean to form clusters. The black time series represent individual series assigned to Clusters 1, 2, and 3, while the colored lines indicate the *centroids* of each cluster.

1. How do you measure the similarity between two time series?
2. How do you form a centroid from a cluster of time series?

With these objectives in mind, specific partition-based TSCL approaches will now be outlined.

## *k*-means

*k*-means [78] describes an objective function that aims to minimise the sum-of-squared errors (Equation 2.32). The most common algorithm to optimise this objective function is Lloyd's algorithm [75]. *k*-means, using Lloyd's algorithm, is one of the most well-known and popular partition-based TSCL algorithm.

Given a dataset of time series $D = \{T_1, T_2, \ldots, T_n\}$ of length $n$, the goal of $k$-means is to form $k$ clusters such that the sum-of-squared errors (also known as inertia) is minimised. Formally, the $k$-means objective function is defined in Equation 2.32.

$$SSE = \sum_{i=1}^{k} \sum_{j=1}^{n_i} ||T_j^{(i)} - c_i||^2 \tag{2.32}$$

where $k$ is the number of clusters, $n_i$ is the number of time series assigned to cluster $i$, $T_j^{(i)}$ is the $jth$ time series assigned to $ith$ cluster, $c_i$ is the $ith$ centroid, and $||T_j^{(i)} - c_i||^2$ is a distance measure between $T_j^{(i)}$ and $c_i$.

### Lloyds *k*-means

Lloyd's [75] solution to the *k*-means problem is one of the most famous and popular clustering algorithms [32]. All of the *k*-means variants outlined in this thesis use a form of Lloyd's algorithm. As such, "*k*-means" and "Lloyd's algorithm" will be referred to interchangeably.

To begin the Lloyd's algorithm randomly selects $k$ time series from dataset $D$ to act as initial centroids. These initial centroids are defined in Equation 2.33

$$C = \{c_1, c_2, \dots, c_k\} \tag{2.33}$$

where $C$ is an array of centroids, $c_i$ is the centroid for $ith$ cluster, and $k$ is the number of clusters.

Once the initial centroids are defined, the *k*-means iterative optimisation begins. First, each time series is assigned to its closest cluster using a distance measure. This is known as the "assignment stage" and is formally defined in Equation 2.34.

$$\arg\min_S \sum_{i=1}^{k} \sum_{j=1}^{n_i} ||T_j - c_i||^2 \tag{2.34}$$

where $S$ is the set of cluster assignments, $k$ is the number of clusters, $n_i$ is the number of time series assigned to cluster $i$, $T_j$ is the $j$th time series in the dataset $D$, $c_i$ is the $i$th centroid in the set of centroids $C$, and $||T_j - c_i||^2$ is the squared Euclidean distance between $T_j$ and $c_i$.

Once each time series has been assigned to its closest centroid, the centroids are updated by computing the arithmetic mean of each cluster. The arithmetic mean of cluster $i$ is formally defined in Equation 2.35.

$$\mu_i = \frac{1}{|C_i|} \sum_{j=1}^{n_i} T_j^{(i)} \tag{2.35}$$

where $\mu_i$ is the arithmetic mean of the $ith$ cluster, $n_i$ is the number of time series assigned to cluster $i$, and $T_j^{(i)}$ is the $jth$ time series assigned to the $ith$ cluster.

Once the arithmetic mean of each cluster is computed, the centroid of each cluster is updated to its respective $\mu_i$. This process is referred to as the “update” or “centroid computation” stage. The completion of this step marks the end of a single iteration of *k*-means.

The *k*-means algorithm continues iterating until the assignment of each time series to its closest centroid does not change between iterations, thus the centroid computed does not change between iterations.

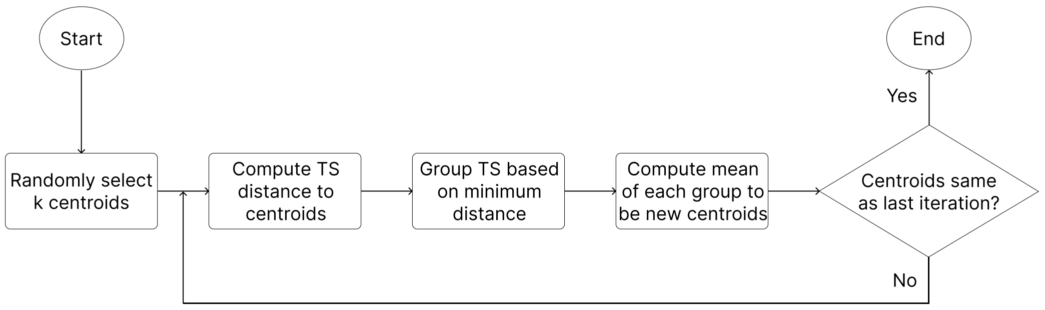

Start
End
Yes
Randomly select k centroids
Compute TS distance to centroids
Group TS based on minimum distance
Compute mean of each group to be new centroids
Centroids same as last iteration?
No

Fig. 2.24 Flow diagram of *k*-means algorithm

Figure 2.24 shows a flow diagram of the described *k*-means algorithm.

The above describes the simplest implementation of *k*-means. However, there are many open issues and challenges with *k*-means.

*Initialisation problem*: - The initialisation problem in *k*-means is twofold: defining the correct value of *k*, and defining the position of initial centroids [48]. Defining the correct value of *k* is an on going research question but techniques such as the Elbow method, Silhouette coefficient [58], Canopy algorithm [30] and the Gap statistic algorithm [119] have been proposed to find a appropriate value of *k*. Selecting the position of the initial centroids is also an on going research question, however, several techniques have been proposed in the literature, including random selection, furthest point heuristic, sorting heuristic, density-based, projection-base and splitting techniques. The most comprehensive review of initialisation techniques was done by [17] on 32 real and 12,228 synthetic datasets. Their results do not clearly point out a single technique that would be consistently better than others. As such for both TSCL and traditional clustering there is no clear state-of-the-art approach for initialisation problem [37].

*Distance measure*: - The *k*-means convergence condition, sum-of-squared errors (Equation 2.32), uses the Squared Euclidean distance. Due to the time series unique characteristics, such as their temporal alignment, the Squared Euclidean distance has been shown to perform poorly when measuring similarity between time series [74]. As such in the context of TSCL it is desirable to use a different similarity measure. Finding a strategy to integrate other distance measures into *k*-means while still conforming to its convergence criteria is challenging. In the context of TSCL algorithms such as DBA [94] (see Section 2.5.1) have been proposed to allow DTW (see Section 2.4.1) to be used with *k*-means.

*Sensitivity to outliers*: - A dataset with many outliers produces unstable clusters with several *k*-means clustering algorithms runs [48]. This is due to outliers increasing the sum-of-squared errors within clusters, thus affecting the final accuracy of the clustering results [112]. This problem becomes even more pronounced in TSCL because generally outliers exist within the individual time series as subseqeunces. Detection of these outliers is thus very challenging and an on going point of research.

**$k$-Spectral Centroid ($k$-SC)**

$k$-Spectral Centroid ($k$-SC) [128] is a TSCL specific algorithm that adapts $k$-means to use a novel distance and averaging measure that is scale and shift invariant. Formally the $k$-SC distance measure is defined in Equation 2.36 and 2.37.

$$\alpha = \frac{L_2(a, b_{(q)})^2}{L_2(b_{(q)}, b_{(q)})^2} \tag{2.36}$$

$$\hat{d}(a,b) = \min_{\alpha,q} \frac{L_2(a, \alpha b_{(q)})}{L_2(a,a)} \tag{2.37}$$

where $b_{(q)}$ is the result of shifting time series $b$ by $q$ time points, $L_2$ distance is given in Equation 2.2, and $a$ and $b$ are time series of length $m$.

$k$-SC aims to minimise the sum-of-squared errors (Equation 2.32). However, instead of using the Euclidean distance like $k$-means, $k$-SC minimises over the $\hat{d}(a,b)$ distance. Therefore, it would be inappropriate to perform the same average as $k$-means [128] (Equation 2.35). Instead $k$-SC uses an averaging technique that integrates the $\hat{d}(a,b)$ distance shift operation to compute a more accurate centroid.

The centroid computation in $k$-SC involves using the $\hat{d}(a,b)$ distance to find the optimal shift $q_j$ that minimises the distance between each time series $T_j \in C_i$ and the current centroid $c_i$. Using these optimal shifts, a covariance matrix $M$ is constructed. Eigen decomposition of $M$ is then performed to find the principal component. The eigenvector corresponding to the smallest eigenvalue of $M$ is used as the new centroid. Formally, this process is defined in Equations 2.38 and 2.39:

$$M = \sum_{T_j \in C_i} \left( I - \frac{T_j T_j^T}{||T_j||^2} \right) \tag{2.38}$$

$$c_i^* = \arg\min_{c} \frac{c^T M c}{||c||^2} \tag{2.39}$$

where $M$ is the covariance matrix for the time series in cluster $C_i$, $C_i$ is the $i$th cluster, $T_j$ is a time series in cluster $C_i$, $I$ is the identity matrix, $c$ is a candidate centroid time series, and $c^T$ is the transpose of the time series $c$.

$k$-SC follows the same algorithm as $k$-means shown in Figure 2.23 but uses the $\hat{d}(a,b)$ distance and the centroid computed under $\hat{d}(a,b)$ described above. Through this process $k$-SC clusters time series based on their shape, even in the presence of scaling and shifting variations.

Figure 2.25 presents the $k$-SC centroids generated for the *Gunpoint* dataset. In contrast to the $k$-means centroids depicted in Figure 2.23, the $k$-SC centroids are noticeably different. One reason for this is that, to facilitate visual inspection and interpretation, the cluster centres have been scaled up. This scaling adjustment does not alter the inherent structure of the centroids but amplifies their magnitude, making key features such as peaks more distinguishable. For example, Cluster 3 may not visually appear to be a very good "average" of the time series in the cluster in terms of scale. However, this issue is due to the visual scaling, as internally the scale is inconsequential. The scale invariance means cluster assignments that may be influenced solely by scale in $k$-means do not occur in $k$-SC.

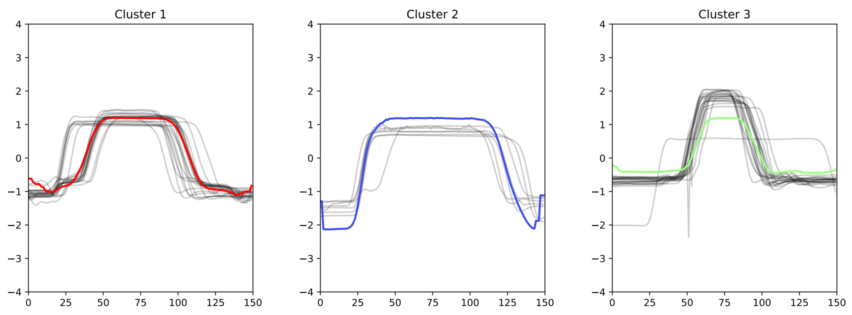


Fig. 2.25 Example of $k$-SC clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

### *k*-shape

*k*-shape [89] is a TSCL specific algorithm that adapts *k*-means by using a novel distance and averaging method that is both scale and shift invariant. Specifically, *k*-shape develops a shape-based distance measure (SBD) that utilises the cross-correlation of two time series.

Cross-correlation distances work by sliding time series $b$ over time series $a$ and computing their inner product for each shift $s$ of $a$. All possible shifts $s$ are considered within the range $s \in [-m, m]$, where $m$ is the length of both time series $a$ and $b$. This is achieved using Equation 2.40 and 2.41.

$$R_k(a,b) = \begin{cases} \sum_{l=1}^{m-k} a_{l+k} \cdot b_l & k \geq 0 \\ R_{-k}(b,a) & k < 0 \end{cases} \tag{2.40}$$

$$CC_w(a,b) = R_{w-m}(a,b) \tag{2.41}$$

where $w \in \{1, 2, \ldots, 2m-1\}$ and $a$ and $b$ are time series of length $m$.

Equation 2.41 produces an array of length $2m-1$. The optimal shift can then be found by identifying the position $w$ at which $CC_w(a,b)$ is maximised. Based on this value of $w$, the optimal shift of $a$ with respect to $b$ is given as $a_{(s)}$, where $s = w - m$ [89]. If the two series are perfectly aligned, then $w = m$.

To make the cross-correlation distance invariant to scale, [89] proposes the use of a coefficient normalisation strategy integrated into the distance measure. The resulting distance measure is called the *Shape-Based Distance* (SBD). The SBD is formally described in Equation 2.42.

$$SBD(a,b) = 1 - \max_{w} \left( \frac{CC_w(a,b)}{\sqrt{R_0(a,a) \cdot R_0(b,b)}} \right) \tag{2.42}$$

where $CC_w(a,b)$ represents the cross-correlation between $a$ and $b$ at lag $w$ shown in Equation 2.41, $R_0(a,a)$ and $R_0(b,b)$ are the auto correlations of a and b at lag 0 shown in Equation 2.40 and $a$ and $b$ are time series of length $m$.

$k$-shape doesn't use the Euclidean distance to compute assignments therefore it is inappropriate to use the arithmetic mean for centroid computation. As such a centroid computation algorithm called shape extraction is proposed. The shape extraction algorithm minimises the sum of squared distances over $SBD(a,b)$.

The shape extraction algorithm aligns each time series $T_j \in C_i$ to the centroid $c_i$ using *SBD*. Once each alignment has been obtained, a new centroid can be obtained by using the maximisation of Rayleigh Quotient [40]. This process extracts the most representative shape from the underlying data [89]. Additionally the centroid produced is scale and shift invariant.

$k$-shape follows the same algorithm as $k$-means shown in Figure 2.23 but uses the *SBD* distance and the shape extraction algorithm to compute new centroids. This process clusters time series based on their shape, even in the presence of scaling and shifting variations.

Figure 2.26 present the $k$-shape centroids generated for the *Gunpoint* dataset.

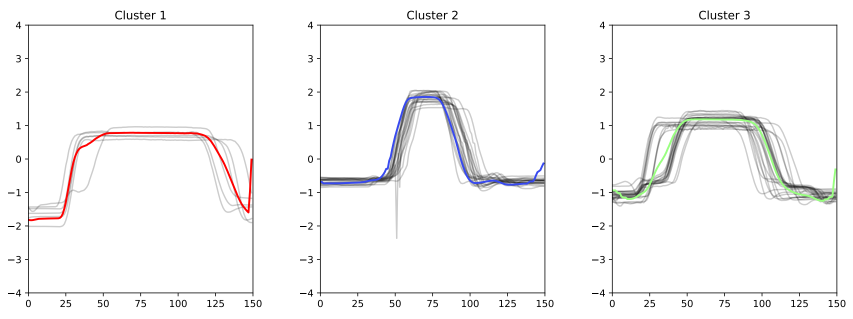


Fig. 2.26 Example of $k$-shape clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

### *k*-means-DBA

*k*-means-DBA [94] is a TSCL-specific algorithm that employs an averaging method known as DTW barycentre averaging (DBA) [94] to compute centroids, and the DTW distance to assign time series to clusters. Since the primary contribution of *k*-means-DBA is its use of DBA as an averaging technique, we now detail how this average is computed.

A barycentre refers to the central sequence that minimises the sum of squared distances between itself and a set of sequences [94]. The most straightforward method for computing this minimum is the arithmetic mean, which identifies the time series that minimises the sum of squared Euclidean distances to all time series in a given collection. The arithmetic mean was previously defined in Equation 2.35.

However, like the Euclidean distance, the arithmetic mean does not attempt to realign time series before computing an average. Consequently, efforts have been made to integrate the optimal DTW alignment path into the averaging process. Specifically, the averaging computation is formulated as an optimisation problem to minimise the DTW Fréchet function [38]. Formally, this is expressed as:

$$F(z) := \frac{1}{n}\sum_{i=1}^{n} DTW(z, x_i)^2 \tag{2.43}$$

where $F$ is the Fréchet function, $z$ is the time series that minimises it, $n$ is the number of time series in the collection $X = (x_1, x_2, \ldots, x_n)$, and $DTW$ is the DTW distance.

However, a polynomial-time algorithm for globally minimising the non-differentiable, non-convex Fréchet function remains unknown [108]. In essence, this means that while an exact average can be computed, the runtime of such algorithms is infeasible for most real-world applications. Therefore, instead of computing the exact minimum of the DTW Fréchet function, approximations are typically employed.

One such approximation, and the most widely used, is DBA [94]. DBA begins with an initial prototype—typically the arithmetic mean of the time series collection—and follows an iterative approach. In each iteration, every series is aligned with the current prototype, and the values mapped to each position are collected. The arithmetic mean is then applied to these collected values while preserving the alignments.

Integrating the DTW distance and DBA averaging technique into *k*-means results in the *k*-means-DBA clusterer. Figure 2.24 illustrates the traditional *k*-means algorithm. To adapt the flow diagram for *k*-means-DBA, the distance measure is set to DTW, and the centroid calculation is replaced with DBA.

Formally, for a given cluster $C_i$, the DBA centroid is computed as follows: Initially, the arithmetic mean is calculated over time series $T_j \in C_i$ to produce an initial centroid $c_i^{'}$. This centroid is iteratively refined by computing the DTW optimal alignment path for each time series $T_j \in C_i$ to $c_i^{'}$ (see Algorithm 2). After computing the optimal alignment path for each $T_j \in C_i$, each point is summed according to its optimal alignment. The current iterations centroid $c_i^{'}$ is then obtained by dividing each time point in the new summed time series by the number of times it was aligned (effectively taking the arithmetic mean of the aligned time series). The newly generated centroid $c_i^{'}$ is then used in the next iteration to be further refined. The centroid will continue to be refined either for a specified number of iterations $max\_iters$ or until the sum of squared DTW distance to centroid $c_i^{'}$ does not change between iterations.

An overview of the iterative DBA algorithm is shown in Figure 2.27.

Figure 2.28 presents the *k*-means-DBA centroids generated for the *Gunpoint* dataset. The centroids produced, when compared to other *k*-means based centroids (Figure 2.23, Figure 2.25 and Figure 2.26) are noticeably different. Particularly looking at the peaks of each centroid, these are not present in other cluster centres.

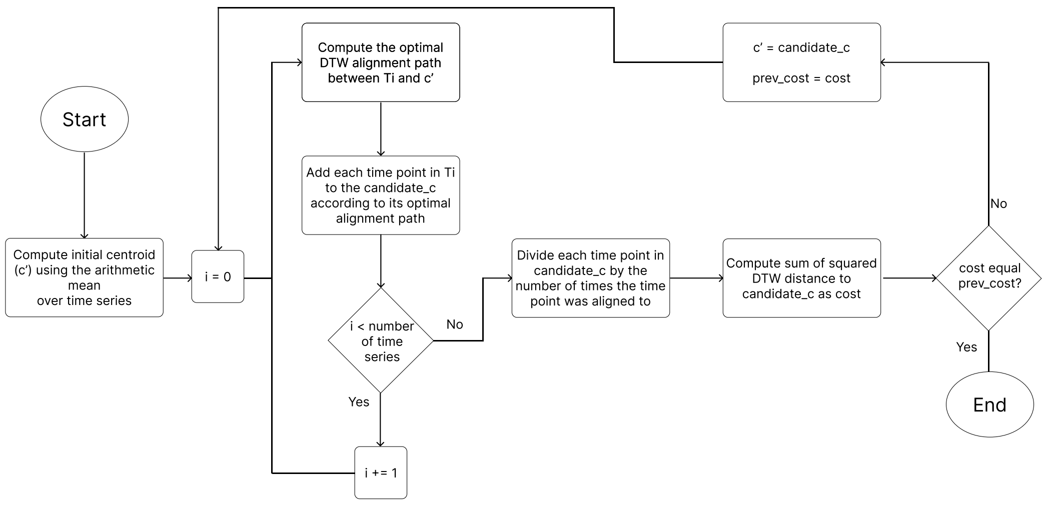

Start
Compute initial centroid (c') using the arithmetic mean over time series
i = 0
Compute the optimal DTW alignment path between Ti and c'
Add each time point in Ti to the candidate_c according to its optimal alignment path
i < number of time series
Yes
i += 1
No
Divide each time point in candidate_c by the number of times the time point was aligned to
Compute sum of squared DTW distance to candidate_c as cost
cost equal prev_cost?
No
c' = candidate_c
prev_cost = cost
Yes
End

Fig. 2.27 Flow diagram of the DBA algorithm

Additionally the centroids seems to have many more local shapes instead of being smooth.

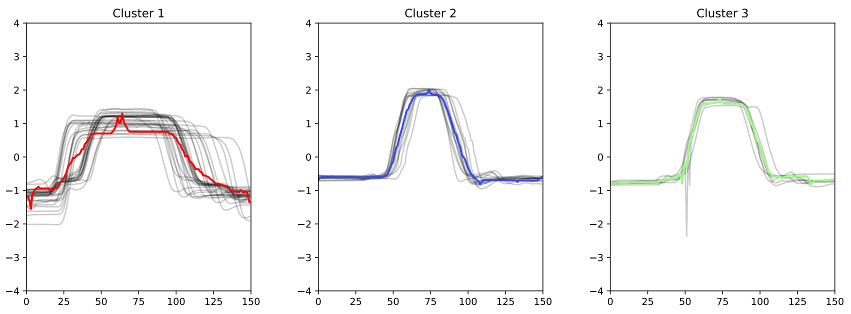


Fig. 2.28 Example of *k*-means-DBA clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

One of the main drawbacks of using DBA is its computational complexity of $O(I \cdot N \cdot m^2)$ where $I$ is the fixed number of iterations, $N$ is the number of time series to average over, and $m$ is the length of the time series. To address this issue, research has focused on improving the computational efficiency of DBA.

[22] proposes a divide-and-conquer strategy with an additional early abandonment condition for DBA, which yields similar performance. Other extensions include implementing a subgradient descent approach to DBA, which finds better

solutions in shorter time [108]. While, in principle, these approaches can replace DBA in *k*-means-DBA, as Chapter 8 discusses, this is a non-trivial task. Further details of these algorithms will be explored in Chapter 8.

#### *k*-soft-DBA

*k*-soft-DBA [21] is a TSCL-specific algorithm that utilises an averaging method known as soft-DTW barycentre averaging (soft-DBA) alongside the soft-DTW distance measure. Its key contribution lies in the application of soft-DBA in the averaging stage of Lloyd's algorithm. We now detail the computation of soft-DBA.

Similar to other averaging methods discussed previously, the objective of soft-DBA is to minimise the Fréchet function presented in Equation 2.43, but with respect to the soft-DTW distance. Although its goal is analogous to that of previously outlined averaging methods, the approach employed by soft-DBA is fundamentally different. While the previously described methods are heuristics that rely on subgradient optimisation (which we define and explore further in Section 8.2), the differentiability of soft-DTW allows for the computation of exact gradients, enabling the application of gradient descent optimisation techniques.

To understand this process, we first examine how a gradient matrix computed between two time series (defined in Section 2.4.7) can be used to help compute a new time series. Once this foundation is established, we explore how gradient descent algorithms can approximate a minimum of the Fréchet function more effectively than previously outlined methods.

Consider a gradient matrix $E$ computed between two time series using Algorithm 13. This gradient matrix effectively captures information about how well these time series align under all possible warping paths, weighted by their probabilities. However, there is a crucial distinction: the gradient matrix $E$ indicates how changes in pairwise distances between points would affect the soft-DTW distance, but it does not directly specify how to modify the time series points to achieve these changes. This is because $E$ represents the derivative of the soft-DTW cost matrix with respect to the distance matrix (i.e., squared Euclidean distances), rather than with respect to the actual time series coordinates.

To address this, [21] employs a Jacobian transformation that maps the gradient matrix of partial derivatives to a matrix of the same shape as the original input time series. This transformation provides specific directions and magnitudes for updating each point in the first time series to make it more similar to the second. When working with a collection of time series $X$, we extend this concept by computing the Jacobian transformation between the current average and each time series in the collection. By summing these transformations at corresponding indices, we obtain a single matrix that represents the aggregate update directions and magnitudes for each point in the average time series. This combined matrix effectively synthesises information from all time series in the collection, indicating how to adjust each point in the current average to better represent the entire set.

Formally, the Jacobian transformation is derived by applying the chain rule, as shown in Equation 2.44. This transformation maps the gradient matrix $E$, computed using Algorithm 13, into updates for each point in the time series. Algorithm 21 provides the formal implementation of this transformation.

$$\begin{aligned}\frac{\partial \text{soft\_DTW}(a,b)}{\partial a_i} &= \sum_{j=1}^{m_2} \frac{\partial \text{soft\_DTW}}{\partial \Delta_{i,j}} \cdot \frac{\partial \Delta_{i,j}}{\partial a_i} \\ &= \sum_{j=1}^{m_2} E_{i,j} \cdot 2(a_i - b_j)\end{aligned} \tag{2.44}$$

where $\Delta$ is a pairwise squared Euclidean distance matrix.

However, directly applying these aggregated updates would be problematic, as different time series often suggest competing movements for the same points. For instance, one time series might suggest moving a point upward, while another suggests moving it downward. Simply summing and applying these contradictory

```
Algorithm 21: Jacobian_Transform ($\mathbf{a}, \mathbf{b}, \mathbf{E}$)
  Input: a (time series of length $m_1$), b (time series of length $m_2$),
         E (gradient matrix of shape $m_1 \times m_2$)
  Output: Jacobian product matrix of shape $m_1$
1 Let product be a vector of length $m_1$ initialised with zeros
2 for $i \leftarrow 1$ to $m_1$ do
3     for $j \leftarrow 1$ to $m_2$ do
4         $product_i \leftarrow product_i + E_{i,j} \cdot 2(a_i - b_j)$
5 return product
```

suggestions could lead to suboptimal results or oscillatory behaviour. This is where another advantage of soft-DTW comes into play: because it is differentiable, we can apply a gradient descent algorithm.

Specifically, [21] proposes using the L-BFGS-B gradient descent optimisation algorithm. While a full exposition of L-BFGS-B is beyond the scope of this thesis, its core mechanism can be summarised as follows: the algorithm begins with an initial average time series and computes the Jacobian transformation matrix across the entire dataset. Using this information, it determines the optimal step sizes and directions that minimise the overall soft-DTW distance between the average and all time series simultaneously. This process iteratively refines the average time series, with each iteration using the previously updated series as its new starting point. The algorithm continues until convergence is reached, typically when the change in the total soft-DTW distance between consecutive iterations falls below a specified tolerance value. At this point, the algorithm returns the final averaged time series.

Integrating the soft-DTW and soft-DBA averaging techniques into $k$-means results in the $k$-soft-DBA clusterer. Figure 2.24 illustrates the traditional $k$-means algorithm. In adapting the flow diagram for $k$-soft-DBA, the distance measure is set to soft-DTW, and the centroid calculation is replaced with soft-DBA.

Figure 2.29 presents the $k$-soft-DBA centroids generated for the *Gunpoint* dataset. When compared to the DBA centroids in Figure 2.29, the centroids in

Figure 2.29 are noticeably smoother. This is because the smoothing parameter $\gamma$ was set to 1. This means that the gradient is smoothed significantly compared to DTW (which has a $\gamma$ value of 0).

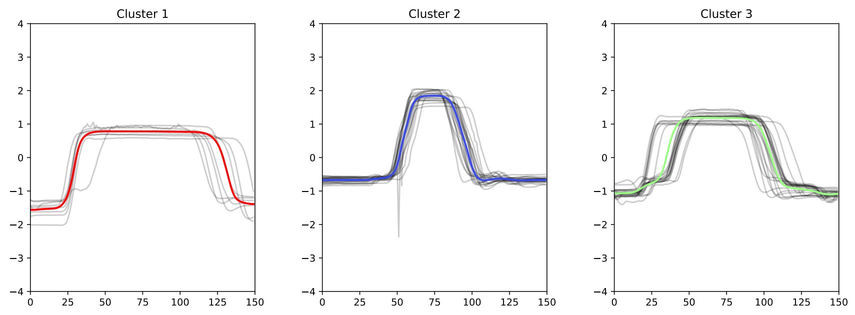


Fig. 2.29 Example of $k$-soft-DBA clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

### *k*-medoids

$k$-medoids is an optimisation problem that, like $k$-means, aims to divide a dataset into $k$ clusters. However, instead of using an average as a centroid, $k$-medoids uses a value within the dataset to serve as a centroid (a medoid). This approach minimises the sum of dissimilarities of each time series $T_j^{(i)} \in C_i$, making it particularly effective in scenarios involving noise or outliers.

Specifically, $k$-medoids aims to minimise the objective function total deviation (TD). Formally, this is defined in Equation 2.45.

$$TD = \sum_{i=1}^{k} \sum_{j=1}^{n_i} ||T_j^{(i)} - m_i|| \tag{2.45}$$

where $k$ is the number of clusters, $n_i$ is the number of time series assigned to cluster $C_i$, $T_j^{(i)}$ is the $j$th time series assigned to cluster $C_i$, $m_i$ is the medoids (centroid) of cluster $C_i$, and $||T_j^{(i)} - m_i||$ is a distance measure between $T_j^{(i)}$ and $m_i$.

If we use the squared Euclidean distance as the distance function in Equation 2.45 (i.e. $||T_j^{(i)} - m_i|| = L_2(T_j^{(i)}, m_i)^2$), we almost obtain the $k$-means objective function, the sum-of-squared errors (Equation 2.32). The difference is that $k$-means is free to choose any $c_i \in \mathbb{R}$ whereas $k$-medoids must choose $m_i \in C_i$ [106].

To solve the above $k$-medoids problem, numerous $k$-medoids algorithms have been proposed. The next four clusterers discussed will be algorithms that find a local minimum for the $k$-medoids problem.

### Alternate $k$-medoids

Alternate $k$-medoids attempts to solve the $k$-medoids optimisation problem by employing Lloyd's [75] algorithm. Traditionally Lloyd's algorithm specifies a mean average of each cluster should be extracted to be centroids. However, alternate $k$-medoids extracts a medoids of each cluster instead. Formally the medoids of cluster $C_i$ is defined in Equation 2.46.

$$m_i = arg \min_{T_j^{(i)} \in C_i} \sum_{T_l^{(i)} \in C_i} ||T_j^{(i)} - T_l^{(i)}|| \tag{2.46}$$

where $C_i$ is the $ith$ cluster, $T_j^{(i)}$ is a candidate medoids from cluster $C_i$, $T_l^{(i)}$ is a time series from cluster $C_i$ to compute the distance between it and $T_j^{(i)}$, and $||T_j^{(i)} - T_l^{(i)}||$ is a distance measure between $T_j^{(i)}$ and $T_l^{(i)}$.

Figure 2.24 shows a flow diagram for the $k$-means algorithm. Alternate $k$-medoids follows the same flow diagram but, instead of computing the mean of each cluster, the medoid of each cluster is computed using Equation 2.46.

One of the main advantages of alternate $k$-medoids (and other $k$-medoids variants) over similar partition-base TSCL approaches is time series specific distance measures can be used in both the assignment step (Equation 2.45) and centroid computation step (Equation 2.46) with no additional logic needed to be added.

Figure 2.30 presents the alternate $k$-medoids centroids generated for the *Gunpoint* dataset. The centroids produced are time series within the given cluster (medoids).

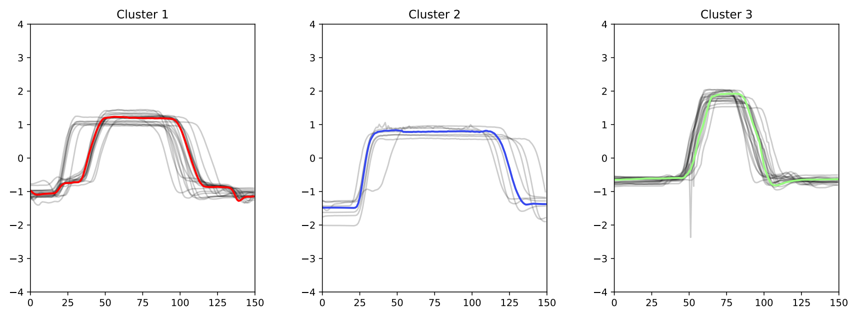


Fig. 2.30 Example of alternate $k$-medoids clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

## Partition Around Medoids (PAM)

Partition Around Medoids (PAM) [69] is another $k$-medoids clusterer. In traditional clustering, PAM is considered better than alternate $k$-medoids [120] and is normally the assumed implementation of $k$-medoids. PAM consists of two algorithms, BUILD to choose initial centroids and SWAP to further improve the clustering towards, a local optimum.

The BUILD algorithm selects $k$ initial centroids to cluster. To begin, BUILD, chooses the time series with the smallest distance to all other time series to be the first medoids. Then it selects the time series that reduces $TD$ by the most to be the next medoids. This process repeats until $k$ initial centroids have been selected. For reasons outlined in Chapters 4 and 6, this thesis will mostly ignore the BUILD initialisation component of PAM and instead will use a random initialisation strategy.

The second algorithm PAM uses is the SWAP algorithm. The SWAP algorithm refines the initial centroids selected by considering all possible changes to the set of

$k$ medoids. This means every non-medoids will be swapped for a medoids to see if a given swap reduces $TD$ [106] (Equation 2.45). This is a greedy steepest-descent method, and the process repeats until no further improvements are found.

As discussed, PAM has been empirically shown to produce better results than other popular k-medoids clustering algorithms [120]. However, this comes at the cost of computational complexity. Specially PAM requires a pairwise distance matrix between every value in the dataset. This has a computational cost of $O(n^2)$. Additionally, when considered in the context of TSCL, many time series distance measures have high computational complexity. For example the most popular time series distance measure, DTW, has computational complexity of $O(m^2)$. As a result this high computational complexity means PAM is not always feesible.

Figure 2.31 presents the PAM centroids generated for the *Gunpoint* dataset. The centroids produces are also time series within each cluster (medoids). The clusters produced are very similar to that of alternate $k$-medoids (Figure 2.30) which is unsurprising as both algorithms attempt to minimise the same objective function given in Equation 2.45.

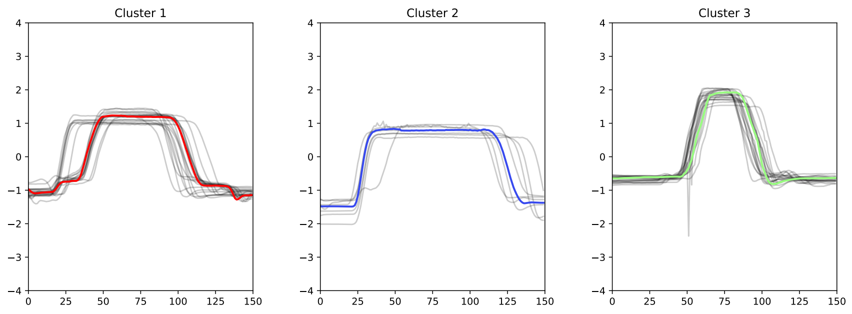


Fig. 2.31 Example of PAM clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

PAM has been extensively used in TSCL. Particularly, before the advent of $k$-means-DBA (Section 2.5.1), PAM was the preferred TSCL approach with DTW [94]. The reason for this is PAM can make better use of time series specific distance measure than similar algorithms such as $k$-means. DTW was used as the distance measure of choice with PAM [43].

### Clustering LARge Applications (CLARA)

Clustering LARge Applications [68] is an extension of PAM which aims to improve the computational complexity of PAM.

CLARA repeatedly applies PAM on a random subset of cases from the dataset. The recommended number of samples to use for each run is $s = 40 + 2k$. Once a random subset of samples has been taken, PAM is performed on this subset and medoids are obtained. After, the remaining cases (not in the subset) are assigned to one of the produced medoids. The $TD$ (Equation 2.45) is then computed for these medoids (considering all the data). This process repeats for multiple iterations and the iteration that has the lowest $TD$ is returned. Due to this optimisation the time complexity of CLARA is reduced to $O(k^3 + s)$.

Figure 2.32 presents the CLARA centroids generated for the *Gunpoint* dataset. The centroids produced, and as a result the time series assignments, are different from other $k$-medoids algorithms (Figures 2.31 and 2.30). These medoids are not as high quality but, they were obtained much faster.

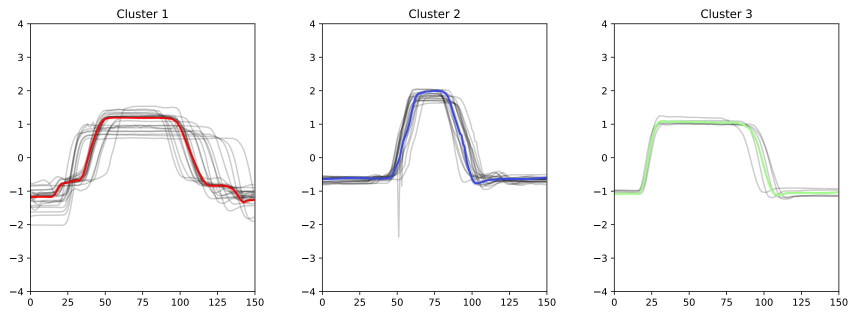


Fig. 2.32 Example of CLARA clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

For TSCL, CLARA has been used as a faster alternative to PAM. One example of CLARA being used in TSCL is for clustering energy consumption time series data using various distance measures [101]. It was selected because the large amount of energy data produced making PAM unviable.

### CLARA based on raNdomised Search (CLARANS)

CLARA based on raNdomised Search (CLARANS) [86] is another extension of PAM which aims to improve the computational complexity of PAM.

Specifically CLARANS updates the SWAP algorithm of PAM to use a greedy optimisation approach. This is done by randomly swapping a non-medoids for a randomly selected medoids. If the swap reduces $TD$ (Equation 2.45) then the swap is performed straight away. If it does not reduce $TD$ a counter of $attempts$ is incremented. If $attempts$ reaches a certain number, defined by a parameter *max_neighbors*, then the search terminates and the current medoids are returned. Additionally this process is repeated with different initial centroids *n_init* times. This random selection gives CLARANS an advantage when handling large datasets by avoiding local minima.

Figure 2.33 presents the CLARANS centroids generated for the *Gunpoint* dataset.

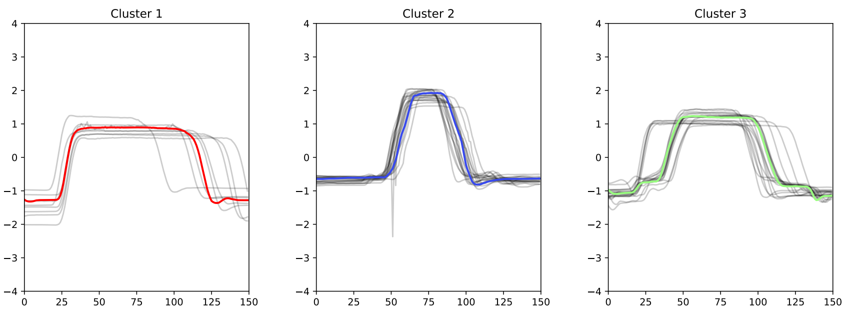


Fig. 2.33 Example of CLARANS clustering for time series *Gunpoint* dataset. The black time series are time series that belong to a given cluster (Clusters 1, 2 and 3). The colored lines through each cluster represent the *centroid* for each cluster.

In the context of TSCL CLARANS has been suggested for use theoretically however, it is yet to be formally researched specifically for time series data.

# Chapter 3

# Experimental Methodology

This chapter outlines the experimental methodology employed throughout this thesis. Evaluating clusterings is inherently challenging, as "clusters are, in large part, in the eye of the beholder" [31]. This subjectivity means there is no single "correct" way to cluster a dataset, making the evaluation of clustering methods particularly complex.

To address these challenges, this chapter will outline a robust experimental methodology for evaluating clusterings. We will describe how we will perform our experimental evaluation, identify key challenges within clustering evaluation, and describe mitigation strategies to allow us to draw meaningful conclusions. Additionally, we will include the software tools used for our evaluation and provide details on how to reproduce our results.

## 3.1 The challenge of cluster evaluation

Cluster evaluation is inherently subjective. Different observers may have varying opinions on what constitutes a "good" cluster, influenced by their domain knowledge, experience, and the specific context of the data. This subjectivity can lead to multiple valid clusterings of the same dataset, each emphasising different aspects of the data.

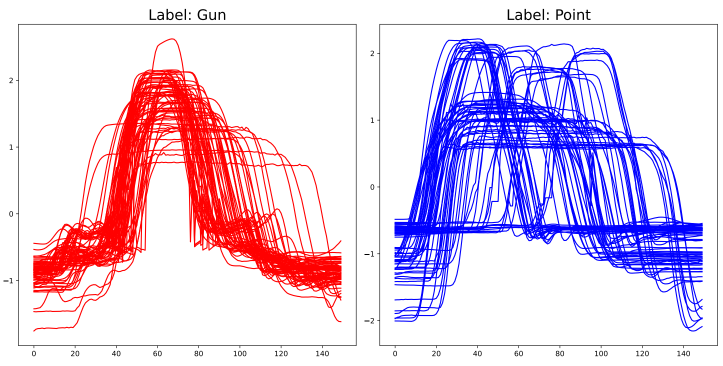
To illustrate this point, consider the GunPoint time series dataset from the UCR archive (the UCR archive will be outlined in Section 3.3). The GunPoint dataset was originally created to capture the motion of actors performing two distinct actions: pointing a gun or pointing with their fingers. This dataset was generated by recording two actors (one male and one female) executing these actions over a five-second period, resulting in a sequence of 150 frames per action. The $x$-axis coordinate of the hand's centroid was extracted from each frame to form the time series data.

In its original form, the dataset was labeled with two class labels: "gun" and "point," reflecting the actor's action. In a TSC task, the objective is to separate the time series into these two categories. However, in the context of clustering, where the labels are unknown, this dataset can be interpreted in multiple valid ways.

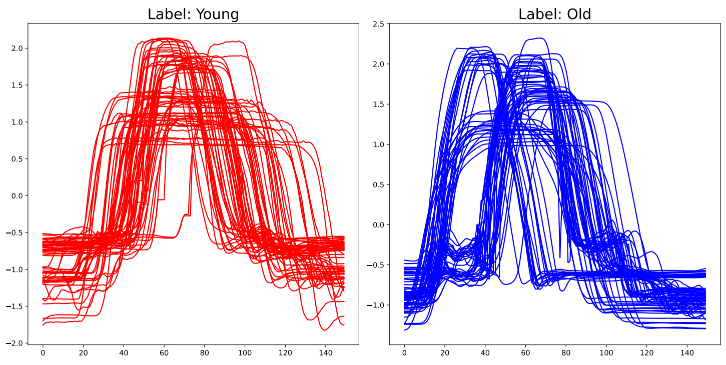
Figure 3.1 illustrates three different interpretations of the GunPoint dataset. Figure 3.1a shows the intended configuration, where the time series are separated based on whether a gun is pointed or a finger is pointed. Figure 3.1b presents an interpretation where the data is clustered by the actor's age, resulting in "Young" and "Old" categories. Finally, Figure 3.1c illustrates an interpretation where the clusters are based on the actor's sex, classifying them as either male or female.

All three interpretations of the GunPoint dataset shown in Figure 3.1 represent valid and interesting cluster configurations. This raises an important question: If our expectation in an experiment is for the clustering algorithm to produce clusters that align with the class labels (e.g., "gun" and "point"), but the algorithm instead separates the data into a different configuration (e.g., "male" and "female"), does this mean the clustering algorithm is incorrect?

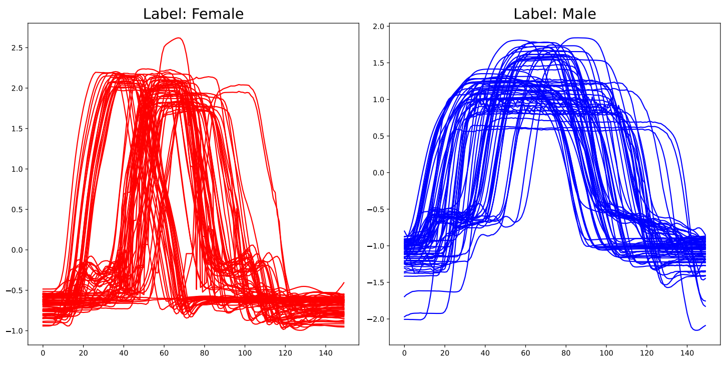
This is one example of the complexity of cluster evaluation. To address this issue and other challenges, we will now present our TSCL methodology.



(a) Interpretation of the GunPoint dataset based on the action performed. The left plot represents the "Gun" action (red time series), while the right plot represents the "Point" action (blue time series).



(b) Interpretation of the GunPoint dataset based on the actor's age. The left plot represents actions performed by the younger actor (red time series), and the right plot represents actions performed by the older actor (blue time series).



(c) Interpretation of the GunPoint dataset based on the actor's gender. The left plot shows actions performed by the female actor (red time series), while the right plot shows actions performed by the male actor (blue time series).

Fig. 3.1 Examples of different interpretations of the GunPoint dataset. Each subfigure represents a distinct clustering criterion: (a) action type (Gun vs. Point), (b) actor's age (Young vs. Old), and (c) actor's gender (Male vs. Female).

## 3.2 TSCL Experimental Methodology

The goal of our TSCL methodology is to evaluate TSCL algorithms across a range of time series problems, utilising various types of clustering algorithms and different statistical evaluation metrics. We aim for our methodology to provide insights into which algorithms and approaches may be most effective for specific clustering tasks, offering guidance to those seeking to cluster their time series data.

We will now discuss the various components of our methodology. Each component has a distinct function individually, but when combined, the aim is to reduce subjectivity of cluster evaluation and provide meaningful insights into the clustering performance of various models.

### 3.2.1 Statistics for Evaluating Performance

As outlined in Section 3.1, determining what constitutes a "good" cluster is challenging. However, numerous statistical evaluation techniques have been proposed to help identify various qualities in clusters. Since no single technique can definitively define "good" and "bad" clustering, this thesis employs a range of measurements designed to highlight different aspects of cluster quality.

During TSCL evaluation, a dataset is defined as $D = \{(T_1, y_1), (T_2, y_2), \dots, (T_n, y_n)\}$, comprising $n$ cases, where $T_n$ is a time series $T_n = \{t_1, t_2, \dots, t_m\}$ of length $m$, with $t_m$ representing the $mth$ time point in $T_n$, and $y_n$ is a class label drawn from a set of $l$ possible class labels, $y \in \{1, 2, \dots, l\}$.

To clarify, while class labels ($y$) are present during evaluation, during training and prediction, a clusterer does not have access to any class labels. From the perspective of a clusterer, a dataset is represented as $D = \{T_1, T_2, \dots, T_n\}$.

The objective of a clustering model is to assign each time series in dataset $D$ to a cluster. The number of clusters does not necessarily correspond to the number of unique class labels. The number of clusters is defined by the value $k$. Additionally, for some clusterers, $k$ is not always predefined, and not every time series must be placed into a cluster (e.g., excluding instances deemed as noise).

The output of a clustering model for a single time series is a probability distribution over $k$ clusters, $\hat{p} = \{p(C_1), p(C_2), \ldots, p(C_k)\}$, where $C_k$ is a cluster label between 1 and $k$. If a time series is considered "noise," it will not have an assignment. Using these probabilities, the predicted cluster is the cluster assignment with the highest probability.

$$\hat{y} = arg \max_{i=1\ldots k} \hat{p}(i) \tag{3.1}$$

If a time series is not assigned to any cluster (i.e. considered noise), then a cluster assignment of $-1$ is given to denote that the value is not assigned to any cluster. Additionally, if a clustering model is unable to output a probability estimate for each cluster, the predicted cluster's probability is set to 1, and all other clusters' probabilities are set to 0.

The various cluster evaluation techniques used will now be outline.

**Mutual Information (MI)**

Mutual Information (MI) is a function that measures the agreement between two sets of labels, disregarding permutations. This means that the specific numeric values assigned to labels or clusters are not important; only the correspondence or matching between them is considered. In the context of cluster evaluation, MI quantifies the agreement between the ground truth label $y_i$ and the predicted cluster assignment $\hat{y}_i$. Formally, MI for a set of cluster predictions is defined in Equation 3.2.

$$MI(Y, \hat{Y}) = \sum_{y \in Y} \sum_{\hat{y} \in \hat{Y}} P(y, \hat{y}) \log \left( \frac{P(y, \hat{y})}{P(y)P(\hat{y})} \right) \tag{3.2}$$

where $Y$ is the vector of ground truth labels for the instances in $D$, $\hat{Y}$ is the vector of predicted cluster labels for the same instances in $D$, $y \in Y$ represents a specific ground truth label, $\hat{y} \in \hat{Y}$ represents a specific predicted cluster label, $P(y, \hat{y})$ is the joint probability distribution of $y$ and $\hat{y}$, $P(y)$ is the marginal probability of $y$, and $P(\hat{y})$ is the marginal probability of $\hat{y}$.

The joint probability distribution $P(y, \hat{y})$ is the probability that an instance has both the ground truth label $y$ and the predicted cluster label $\hat{y}$. It is computed from the co-occurrences of labels in the dataset (i.e., how many instances with the same ground truth label appear in the same cluster). This makes MI permutation invariant.

The marginal probabilities $P(y)$ and $P(\hat{y})$ represent the probability of an instance having the ground truth label $y$ or the predicted cluster label $\hat{y}$, respectively.

While Mutual Information (MI) is a valuable measure for assessing the agreement between clustering results and ground truth labels, it has certain limitations that make it less effective in some scenarios. MI is not inherently normalised, meaning its values can vary significantly depending on the size of the dataset and the number of clusters, which can make it difficult to compare results across different clustering experiments. Additionally, MI does not account for the agreement that could occur by chance, particularly when the number of clusters is large.

As such, for cluster evaluations across multiple datasets and varying numbers of clusters, MI is not a suitable evaluation metric. Therefore, this thesis will not use the regular form of MI. Instead, two variants — Adjusted Mutual Information (AMI) and Normalised Mutual Information (NMI), will be used. As will be outlined, these two variants compensate for MI's weaknesses.

### Normalised Mutual Information (NMI)

MI can produce misleading interpretations when used to compare clusterers with different numbers of clusters or across different datasets. To address this, a normalised version of MI — Normalised Mutual Information (NMI) — was proposed to scale the value of MI between 0 and 1. An NMI value of 0 indicates no mutual information between the ground truth and cluster labels, while an NMI value of 1 indicates perfect mutual information between them. Formally, NMI is defined in Equation 3.3.

$$NMI(Y,\hat{Y}) = \frac{MI(Y,\hat{Y})}{\frac{1}{2}(H(Y)+H(\hat{Y}))} \tag{3.3}$$

where $D$, $MI(Y,\hat{Y})$ is the Mutual Information as defined in Equation 3.2, and $H(Y)$ and $H(\hat{Y})$ are the entropies of the ground truth labels and predicted labels, respectively, as defined in Equation 3.4.

$$H(X) = -\sum_{x\in X} P(x)\log P(x) \tag{3.4}$$

where $H(X)$ represents the entropy of a set of labels $X$, and $P(x)$ is the marginal probability of label $x$ in the set $X$.

By dividing by the mean of the clustering entropies, NMI ensures that its values are independent of the absolute number of clusters or labels, regardless of the dataset size. This provides a more standardised way to compare clustering results across a range of datasets and numbers of clusters.

In its normalised form, Mutual Information offers valuable insight for comparing a variety of clustering models across a wide range of datasets. NMI quantifies how much information is shared between the ground truth labels and the clustering labels. Additionally, it considers the purity of the clusters (i.e., the proportion of instances with the same ground truth labels within the same cluster), providing insights beyond what simple accuracy measures can offer.

**Adjusted Mutual Information (AMI)**

Adjusted Mutual Information (AMI) [125] is another extension of MI that accounts for the possibility that clustering agreement could occur purely by chance. MI scores tend to be higher as the number of clusters increases because there are more opportunities for random alignments between the ground truth labels and the predicted cluster labels to occur. In the context of MI, even if a cluster is meaningless and formed by random chance, a higher score may be returned if some level of agreement exists. AMI attempts to correct for this by adjusting the MI score to reflect only the agreements that do not occur by chance.

Formally, AMI is defined in Equation 3.5.

$$AMI(Y,\hat{Y}) = \frac{MI(Y,\hat{Y}) - \mathbb{E}[MI(Y,\hat{Y})]}{\max(H(Y),H(\hat{Y})) - \mathbb{E}[MI(Y,\hat{Y})]} \tag{3.5}$$

where $\mathbb{E}[MI(Y,\hat{Y})]$ is the expected Mutual Information under a random model, and $H(Y)$ and $H(\hat{Y})$ are the entropies of the ground truth labels and predicted labels, respectively, as defined in Equation 3.4.

The Expected Mutual Information (EMI) is computed by finding the average MI that would be expected if cluster assignments were made randomly. By subtracting this value from the computed MI, the impact of random chance is greatly reduced. Additionally, as shown in Equation 3.5, the normalisation of MI occurs similarly to what is proposed in NMI (Equation 3.3), allowing the metric to be used across datasets and models.

AMI offers similar benefits to NMI while also correcting for random chance, making it potentially more accurate in some scenarios. However, this thesis will evaluate over both NMI and AMI, even though the measurements are similar. The primary reason is NMI is one of the most popular measures in the literature. This ensures that results can be more easily compared to work not directly considered in this thesis. The second reason NMI is used alongside AMI is that by comparing the difference between them, provides some insight into the impact of random chance on specific models.

### Rand Index (RI)

The Rand Index [97] is one of the most popular clustering evaluation metrics. It measures the similarity between the ground truth labels and the predicted labels by considering all pairs of samples and counting how many pairs are assigned to the same or different clusters in both the ground truth and the predicted labels. The RI score is from 0 to 1. Where 0 means there is no agreement between the ground truth and cluster predictions and 1 indicates there is perfect agreement between the ground truth and cluster predictions. Formally, this is defined in Equation 3.6.

$$RI(Y,\hat{Y}) = \frac{TP(Y,\hat{Y}) + TN(Y,\hat{Y})}{TP(Y,\hat{Y}) + TN(Y,\hat{Y}) + FP(Y,\hat{Y}) + FN(Y,\hat{Y})} \tag{3.6}$$

where $TP(Y,\hat{Y})$ is the number of pairs of points that are in the same cluster in both $\hat{Y}$ and $Y$, $TN(Y,\hat{Y})$ is the number of pairs of points that are in different clusters in both $\hat{Y}$ and $Y$, $FP(Y,\hat{Y})$ is the number of pairs of points that are in the same cluster in $\hat{Y}$ but in different clusters in $Y$, and $FN(Y,\hat{Y})$ is the number of pairs of points that are in different clusters in $\hat{Y}$ but in the same cluster in $Y$.cluster in $Y$.

One of the main drawbacks of the RI is that scores are typically closer to 1, even for random clustering, due to the inherent agreement between most element pairs. This occurs because the majority of element pairs are often assigned to different clusters in both the predicted and ground truth clusterings, resulting in a high proportion of agreeing pairs. As a result, the RI can yield high scores even when the clustering does not accurately reflect the true structure of the data.

However, the RI will be used in this thesis due to its widespread usage in the literature and its ease of interpretation, being such a simple measure.

### Adjusted Rand Index (ARI)

The Adjusted Rand Index (ARI) is an extension of RI that accounts for the chance that clustering agreement could occur purely by chance. Formally ARI is defined in Equation 3.7.

$$ARI(Y,\hat{Y}) = \frac{RI(Y,\hat{Y}) - \mathbb{E}[RI(Y,\hat{Y})]}{1 - \mathbb{E}[RI(Y,\hat{Y})]} \tag{3.7}$$

where $RI(Y,\hat{Y})$ is the Rand Index between $Y$ and $\hat{Y}$, and $\mathbb{E}[RI(Y,\hat{Y})]$ is the Expected Rand Index under a random labeling model.

The Expected Rand Index (ERI) is computed by finding the average Rand Index that would be expected if cluster assignments were made randomly. By subtracting this value from the computed Rand Index, the impact of random chance is significantly reduced.

Additionally, by accounting for the random chance of cluster assignments, the ARI score is significantly deflated compared to the RI. The ARI score ranges between $-1$ and 1. A score of $-1$ indicates perfect disagreement, meaning the ground truth labels and predicted labels are exact opposites. A score of 0 indicates that the clustering similarity is no better than random assignment. A score of 1 indicates perfect agreement, where the ground truth labels and predicted labels are identical.

ARI scores tend to be closer to 0, making it easier to distinguish the performance of different models across multiple datasets, and allowing for more meaningful comparisons. Therefore, ARI provides valuable and unique insights into cluster quality, making it an essential tool for overall cluster evaluation in this thesis.

**Clustering Accuracy (CL-ACC)**

Clustering Accuracy (CL-ACC) is a supervised evaluation metric that measures the proportion of correctly predicted instances relative to the total number of instances. To determine whether a cluster prediction is considered “correct”, each cluster is assigned to a corresponding class label. The assignment is done by selecting the permutation of cluster and class label assignments $S_k$ that maximises accuracy.

Determining the optimal assignment of class labels that maximises accuracy is computationally expensive. To address this, combinatorial optimisation techniques, the Hungarian algorithm [62] is employed. This approach involves constructing a contingency matrix of cluster assignments and class labels, which is then transformed into a cost matrix. The Hungarian algorithm is used to find the optimal assignment, and the accuracy is subsequently calculated by comparing the predicted cluster labels with the optimally assigned ground truth labels.

Formally, CL-ACC is defined in Equation 3.8:

$$\text{CL-Acc}(y,\hat{y}) = \max_{s\in S_k} \frac{1}{|y|} \sum_{i=1}^{|y|} \begin{cases} 1, & y_i = s(\hat{y}_i) \\ 0, & \text{otherwise} \end{cases} \tag{3.8}$$

where $S_k$ is the set of all possible permutations of cluster assignments, $|y|$ is the number of ground truth labels, and $s(\hat{y}_i)$ represents the predicted cluster label for instance $i$ after applying the permutation $s$.

CL-ACC provides an interesting interpretation of cluster performance that is different from other metrics considered in this thesis. While the primary objective of clustering is not classification, CL-ACC assumes that the ground truth labels represent one “correct” cluster configuration.

**Davies-Bouldin Index (DBI)**

The Davies-Bouldin Index (DBI) [25] is an unsupervised measures that evaluates the average similarity ratio of each cluster with its most similar cluster. An unsupervised measure is one that does not require ground truth labels to produce a score.

DBI evaluates both intra-cluster relationships (similarity among values within the same cluster) and inter-cluster relationships (similarity between different clusters). A DBI value closer to 0 indicates better clustering performance, where 0 signifies that the clusters are both compact (intra-cluster similarity) and well-separated (inter-cluster dissimilarity). The formal definition of DBI is provided in Equation 3.9.

$$DBI(\hat{y}) = \frac{1}{k}\sum_{i=1}^{k}\max_{j\neq i}\left(\frac{s_i + s_j}{L_2(\bar{C_i},\bar{C_j})}\right) \tag{3.9}$$

where $s_i$ is the average distance between each data point in cluster $i$ and the average time series of cluster $C_i$, $s_j$ is the average distance between each data point in cluster $j$ and the average time series of cluster $C_j$, $\bar{C_i}$ is the mean average of cluster $i$ ($C_i$), $\bar{C_j}$ is the mean average of cluster $j$ ($C_j$), and $L_2(\bar{C_i},\bar{C_j})$ is the Euclidean distance between the centroids $\bar{C_i}$ and $\bar{C_j}$.

DBI is an interesting measure to consider, as it is unsupervised. However, as this thesis will show, a limiting factor in using DBI (and similar unsupervised metrics) for TSCL evaluation lies in the distance and averaging computations. This thesis will demonstrate that elastic distances outperform traditional ones, such as the squared Euclidean distance, for clustering tasks. Moreover, integrating elastic distances into the averaging computations produces significantly better averages for time series data. Therefore, in the context of TSCL, DBI provides limited insight unless paired with an elastic distance. An elastic version of DBI has yet to be formally defined in the literature. In Chapter 9, we will introduce this elastic

DBI and demonstrate that, when both an elastic distance and an elastic averaging technique are used, DBI's utility for TSCL is significantly enhanced.

**Calinski-Harabasz Index (CHI)**

The Calinski-Harabasz Index (CHI) [15], also known as the Variance Ratio Criterion [92], is an unsupervised metric that evaluates clustering by comparing between-cluster dispersion with within-cluster dispersion.

The between-cluster dispersion is quantified by the Between-Cluster Sum of Squares (BCSS), defined in Equation 3.10.

$$BCSS = \sum_{i=1}^{k} n_i L_2(\bar{C}_i, \bar{C})^2 \tag{3.10}$$

where $n_i$ is the number of points in cluster $C_i$, and $\bar{C}$ is the overall centroid of the dataset.

The within-cluster dispersion is measured by the Within-Cluster Sum of Squares (WCSS), shown in Equation 3.11.

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} L_2(x, \bar{C}_i)^2 \tag{3.11}$$

where $x$ is a point in the $i$-th cluster.

The CHI is then computed as the ratio of the between-cluster dispersion (BCSS) to the within-cluster dispersion (WCSS), normalised by their degrees of freedom. Formally, the CHI is expressed as:

$$CHI = \frac{BCSS/(k-1)}{WCSS/(n-k)} \tag{3.12}$$

where $k$ is the number of clusters, and $n$ is the total number of data points.

A higher CHI indicates better-defined clusters, as it reflects greater between-cluster separation relative to within-cluster cohesion. Like DBI, CHI offers a distinct perspective on clustering quality. However, it also relies on the squared Euclidean distance, which, as we will show, is ineffective for TSCL. Additionally, the calculation of cluster centroids ($\bar{C_i}$) and the overall dataset centroid ($\bar{C}$) uses the arithmetic mean, which we demonstrate does not perform well for time series clustering tasks. As a result, in its original form, CHI provides limited insight into time series clustering. In Chapter 9, we propose an elastic variant of CHI, which we show offers a more accurate evaluation of time series clusterings compared to the traditional CHI.

#### FitTime

An important consideration when evaluating clustering performance is the computation time required to train the model. FitTime refers to the duration taken to fit the model to the training data and generate initial predictions. Although FitTime is not the primary metric for assessing clustering performance, it plays a crucial role in practical applications. In real-world scenarios, where computational resources and time are often limited, a model's efficiency can be a very important consideration. Therefore, FitTime is a valuable metric to consider when making recommendations for the deployment of clustering algorithms in practical settings.

### 3.2.2 Comparison of Clustering Algorithms

The next component of our methodology is to define how we will use these evaluation measures to compare clustering algorithms. The comparison of clustering algorithms is a key aspect of our methodology. To conduct these comparisons effectively, several factors are taken into account. This section will detail how different variables are considered within out methodology to compare different clustering algorithms.

### Over a Single Dataset

To begin, we will outline the simplest comparison variable: comparing multiple models over a single dataset. A dataset, is assumed to have ground truth labels present. These ground truth labels will not be used at any point during a model's training or prediction stage but will be used during evaluation. While this type of evaluation wouldn't be possible in real-world clustering (due to the lack of labels), for an experimental methodology, supervised evaluation metrics offer the best insight into quantifying "good" clustering. The following statistics will be extracted from a model's clustering: AMI, ARI, CLACC, NMI, RI, and FitTime. We do not use any unsupervised metrics as we believe they do not provide good insights into time series clusterings for reasons previously outlined. The reason we choose to use multiple statistics is that each gives unique insight into specific aspects of clustering quality.

Assuming each statistic has been extracted from the results of multiple TSCL models, each model can be compared to another model for a specific statistic. Models that perform better in each statistic can then be identified. Conclusions as to why one model performs better, in the context of the dataset, can then be drawn.

### Over Multiple Datasets

Evaluating a model's performance over a single dataset only allows conclusions to be drawn about the model's performance for that specific dataset. However, this thesis aims to draw more general conclusions about which TSCL models generally perform best over a large range of different datasets. This will provide better insight into which TSCL models would likely perform best for new time series data not yet considered. To achieve this, we consider each models performance over 112 datasets (outlined in Section 3.3) so that broader conclusions can be drawn.

However, evaluating so many different and diverse datasets presents significant challenges for evaluation.

The first consideration is how each statistic is impacted by differences in size, semantics, and the number of clusters a dataset possesses. Some clustering evaluation metrics can be inflated for datasets with a large number of instances or clusters. This is because having more clusters or more data increases the likelihood that cluster "correctness" occurs purely by chance.

For example, in the case of MI, the probability of alignment between ground truth and predicted labels increases with the number of clusters. Similarly, for the RI evaluation measure, as the amount of data or the number of clusters increases, so does the random chance of having agreeing or disagreeing pairs. In the UCR dataset collection (outlined in Section 3.3), there are datasets with significantly different numbers of class labels (clusters), such as the "ShapeAll" dataset with 60 unique class labels compared to "GunPoint," which has only 2 unique class labels. Furthermore, datasets like "ElectricDevices" have $16,637$ unique time series instances, compared to "BeetleFly," which only has 40 unique time series instances. A summary of various differences in the datasets we consider can be found in Tables 3.3, 3.4 and 3.5.

Due to the diverse range of semantics, lengths and number of clusters we will be considering, when drawing conclusions about model performance, specific emphasis is placed on the AMI and ARI evaluation metrics. This is because they are adjusted for random chance. The other cluster quality metrics (RI, NMI, and CLACC), while still considered due to their popularity, will have less impact on the general conclusions drawn in this thesis. Once statistics for each model, for each dataset, have been obtained, the results will be presented and analysed in four different ways.

The first, and most common way this thesis will analyse results, is by using an adapted version of critical difference diagrams [26]. Specifically, we compare clusterers using a pairwise Wilcoxon signed-rank test and form cliques using the Holm correction rather than the post-hoc Nemenyi test for each metric. This change follows the work of [39] and [11]. While originally intended for classification models, we believe critical difference diagrams offer similar insights for clustering models.

An example of a critical difference diagram is shown in Figure 3.2. Figure 3.2 shows five dummy clusterers compared over some evaluation metric across a number of datasets. The number line at the top shows the potential average rank of a clusterer. In Figure 3.2, since there are five clusterers being compared, a clusterer could have a rank between 1 and 5. A rank of 1 means the clusterer outperforms all other considered clusterers on every dataset. A rank of 5 means the clusterer performs worse than all other clusterers on every dataset. Therefore, a lower rank indicates better performance in relation to the evaluation metric. Clusterers intersect the average rank number line at the point where their average rank is positioned. Each clusterer's average rank is labelled on their line next to their name. Clusterers are displayed in descending rank order.

In Figure 3.2, clusterer 2 performs best for the given evaluation metric over some number of datasets with an average rank of 2.3243. This means over each dataset for a given evaluation metric clusterer 1 was on average ranked 2.5586. Clusterer 4 performs the worst, with an average rank of 3.6261.

In Figure 3.2, some clusterers' lines are joined by thick black lines, indicating that they belong to the same clique, meaning they are not significantly different from each other (as described previously). Cliques are important for evaluation. For example, clusterer 1 is in the same clique as clusterer 2, which means that although clusterer 2 has a higher average rank, clusterers 1 and 2 are not statistically different. Additionally, clusterer 4 and clusterer 5 form another clique at the lower rankings, indicating that they, too, are not statistically different. Clusterer 3, however, is not part of any clique, which signifies that it is statistically different from all other

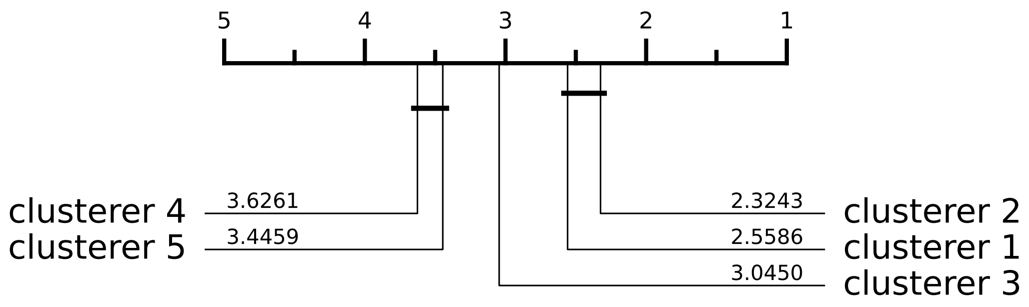


Fig. 3.2 An example critical difference diagram using dummy clusterers over some metric, over some amount of datasets.

clusterers. When interpreting this diagram, we would conclude that clusterers 1 and 2 are the best-performing clusterers, followed by clusterer 3, with clusterers 4 and 5 performing statistically the worst.

While critical difference diagrams are useful, they can be deceptive when used in isolation. This is because they do not display the magnitude of the differences, and the linear nature of clique finding can mask relationships between results [84]. For example, small differences in single cases across thousands of results can lead to a higher rank despite the difference being minimal. This is a particularly important consideration for this thesis due to the high number of datasets we evaluate and the types of clusterers we consider. Additionally, this issue is likely to occur when comparing similar clusterers or the same clusterer with different parameters, which is central to this thesis.

As such, the second evaluation tool this thesis will use is tables showing averaged results for each chosen metric. An example of one of these tables it shown in Figure 3.1 and 3.2. Figure 3.1 shows the average scores across all datasets. Figure 3.2 shows the average rank by domain. The domains are a subset of the 112 which will be defined in Section 3.3. While in isolation, tables such as 3.1 can be misleading, when combined with our other techniques, they provide useful insights into clustering performance.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| clusterer 1 | **0.204** | **0.257** | **0.543** | **0.281** | **0.682** |
| clusterer 2 | 0.183 | 0.234 | 0.529 | 0.260 | 0.676 |
| clusterer 3 | 0.169 | 0.207 | 0.511 | 0.234 | 0.658 |
| clusterer 4 | 0.165 | 0.223 | 0.513 | 0.248 | 0.656 |
| clusterer 5 | 0.159 | 0.212 | 0.507 | 0.236 | 0.632 |

Table 3.1 Summary of average performance of dummy clusterers across multiple evaluation metrics

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| clusterer 1 | **0.341** | 0.158 | 0.227 | **0.358** | 0.085 | **0.207** | **0.376** |
| clusterer 2 | 0.306 | 0.156 | 0.207 | 0.308 | 0.081 | 0.194 | 0.348 |
| clusterer 3 | 0.269 | 0.132 | 0.205 | 0.253 | **0.142** | 0.106 | 0.352 |
| clusterer 4 | 0.273 | **0.162** | 0.214 | 0.353 | 0.076 | 0.167 | 0.311 |
| clusterer 5 | 0.273 | 0.156 | **0.242** | 0.236 | 0.071 | 0.120 | 0.298 |

Table 3.2 Average AMI score of dummy clusterers on problems split by problem domain

In addition to tables and critical difference diagrams, we can use two additional evaluation strategies to further explore the results: violin plots and clusterer scatter diagrams.

A violin plot is similar to a box plot but provides additional information by showing the distribution and probability density of the data. Figure 3.3 shows an example of a violin plot. In Figure 3.3, the $x$-axis represents the different clusterers, and the $y$-axis shows the CLACC scores. The plot's vertical range indicates the minimum and maximum scores, while the shape and width of the violin at different y-values reveal where the data points are concentrated. Wider sections of the plot indicate a higher density of scores, meaning more data points fall within that range. Additionally, within each violin, there is a box plot that shows the interquartile range (the line inside each violin) and the median score (the gap in the line inside each violin). By assessing violin plots, we can better understand the distribution of a clusterer's performance.

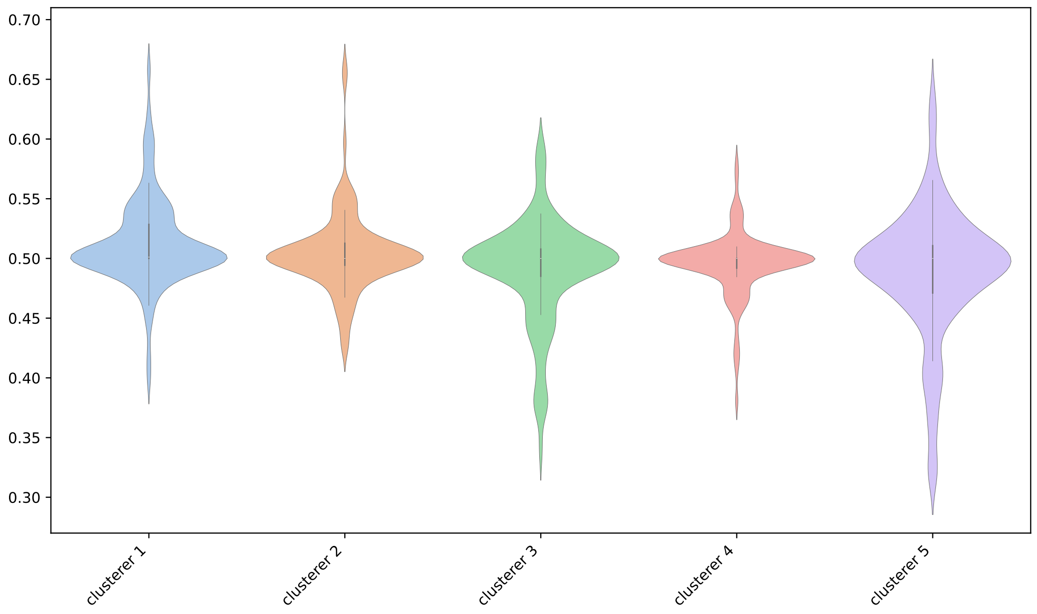


Fig. 3.3 Violin plot for CLACC of dummy clusterers over multiple datasets

Additionally, for statistics such as FitTime, when using a Violin plot we apply a relative scaling to address the considerable variability in execution times across datasets. The relative FitTime is calculated by dividing each original value by the sum of that value and the dataset's median (i.e., $\frac{\text{value}}{\text{value}+\text{median}}$). This transformation scales all values into a range between 0 and 1, with lower values signifying faster performance relative to the median. Such scaling facilitates more meaningful comparisons, especially when execution times of some models differ by orders of magnitude.

The final analysis tool used in this thesis is a scatter diagram that compares the distribution of results between two clusterers. Figure 3.4 shows an example scatter plot comparing clusterer 1 and clusterer 2. The $x$-axis represents the metric score for one clusterer, while the $y$-axis represents the metric score for the other. Each point on the plot corresponds to the best result for a specific dataset. The colour of the point indicates which clusterer achieved the higher score for that dataset: green if the clusterer on the $y$-axis performed better, blue if the clusterer on the $x$-axis performed better, and orange if both clusterers achieved the same score.

The gray diagonal line down the center of the plot represents the threshold where both clusterers perform equally. The further a point is from this line, the greater the difference in performance between the two clusterers for a given dataset. The keys display the specific numbers of wins, ties, and losses for each clusterer. Finally, the dashed lines represent the median scores of each clusterer.

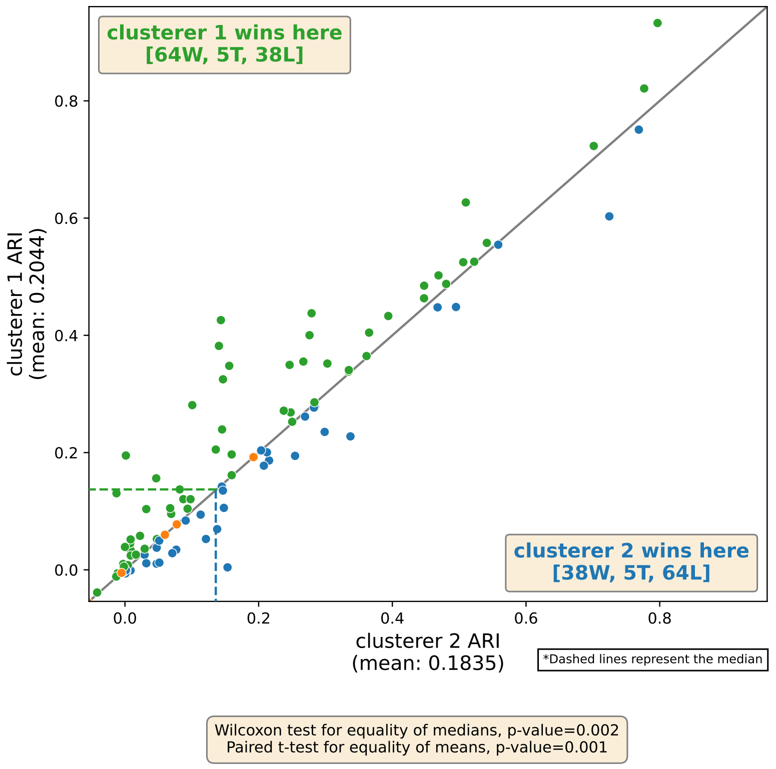


Fig. 3.4 Example scatter plot between clusterer 1 and clusterer 2 for ARI over 107 datasets

Overall, using these four evaluation techniques, multiple clusterers will be evaluated over multiple datasets of different sizes, number of clusters, semantics and domains.

### Over Combined Test-Train Data Split

In supervised machine learning fields, such as TSC, datasets are typically divided into a training split and a test split. The training split is used to “train” the model, while the test split (which remains unseen by the model during training) is used to evaluate the model's performance. This approach helps prevent overfitting, where the model memorises label assignments rather than learning the underlying patterns. Additionally, evaluating on unseen data provides valuable insight into the model's ability to generalise from the data.

However, in unsupervised tasks such as TSCL, labels do not exist. This means that overfitting is not a concern, and any patterns or structures learned by the model must come from the model's inherent understanding of the data itself, rather than from predefined labels [51]. Consequently, a common approach in clustering is to provide the model with all available data during training, since a separate prediction step is not required. Additionally some TSCL models only work with the combined test-train split and do not work with the test-train split [130, 2, 63].

Therefore, to evaluate clusterers for general clustering tasks, this thesis will conduct evaluations over the combined test-train split. This means that models will be trained on all available data which means they will be trained on the same data they are predicting over.

#### Over Test-Train Data Split

Evaluating over the combined test-train split is the most common approach for assessing clusterers. However, some TSCL clusterers feature distinct "fit" and "predict" stages, allowing new time series to be added to existing clusters without the need to recompute the entire model [89, 21, 94, 64, 128]. Models with this capability (such as *k*-means) are particularly useful in scenarios where clustering is a step in a larger pipeline. Examples include pipelines that require real-time processing, such as streaming data, real-time anomaly detection, and other dynamic applications [48].

Furthermore, evaluating models on unseen data provides a good indication of how well models can extrapolate meaningful general patterns from the data. A model that performs well on unseen data has likely learned more general qualities about the data, potentially making its clusterings more valuable.

Therefore, in addition to evaluating over the combined test-train split, this thesis will also consider using the default UCR test-train split to identify models that generate more generalised clusterings.

**Over a Set Number of Clusters**

One of the key parameters for all clustering models is the number of clusters. Some clustering algorithms determine this parameter automatically, while others require it to be set manually. Finding the optimal number of clusters for a dataset, given a specific model, is a challenging task. For example, the performance of the *k*-means algorithm on any given dataset is undeniably dependent on the number of clusters specified [48]. Consequently, entire research fields have emerged dedicated to identifying the optimal number of clusters for a given model.

The objectives of this thesis, however, do not focus on determining the optimal number of clusters. Instead, we adopt a methodology commonly used in similar literature [89, 91, 64, 46, 47]. Specifically, the number of clusters is set equal to the number of unique class labels in the dataset. While this may not represent the optimal number of clusters for a given model, one reasonable assumption is that a "correct" clustering configuration could mirror the clustering of the ground truth labels. Thus, it is expected, at least to some extent, that instances sharing the same ground truth labels would exhibit some common, distinguishable characteristics. Therefore, while the goal of clustering is not to perfectly replicate the predefined groupings of ground truth labels, the ability to identify and organise data around these distinguishable characteristics demonstrates desirable clustering qualities.

In addition, setting the same number of clusters for each dataset facilitates more straightforward comparisons between different models. This is particularly important because some evaluation metrics are sensitive to the number of clusters chosen (as discussed in Section 3.2.1). By standardising the number of clusters across models, we introduce an additional control variable, which helps to isolate the effects of other factors that this thesis is concerned with.

Consequently, our methodology involves setting the number of clusters for models that require this parameter to the number of unique ground truth labels in the dataset. This approach allows for a consistent comparison of clusterers across multiple datasets, ensuring that differences in performance are more likely attributable to the models themselves rather than variations in cluster quantity.

#### Over the Same Model Type

When evaluating clustering models of the same type (e.g., density-based, partition-based, hierarchical-based), the models generally aim to achieve similar objectives, making their clustering results directly comparable. Therefore, our methodology will organise comparisons by focusing on performance within each defined type of clustering model. Our experimentation will focus on partition-based clustering as this will allow us to best compare different models.

### 3.2.3 Tuning of Parameters

In supervised learning tasks, a common practice in model evaluation is to tune the model to maximise its performance based on a specific supervised metric like classification accuracy. However, in TSCL, where labels are absent, this approach is not applicable.

An alternative method for tuning clustering models is to use an unsupervised metric, such as CHI or DBI. However, both of these metrics typically rely on the Euclidean distance. As we will demonstrate in Chapter 9, using unsupervised metrics with the Euclidean distance and arithmetic mean is not suitable for tuning

TSCL models. Theoretically (and as we explore in Chapter 9), substituting the Euclidean distance and arithmetic mean with a time series-specific distance measure and averaging technique could improve the effectiveness of unsupervised metrics. However, there have been no empirical studies in the literature that specify how, or which, time series distances should be used. This leaves many unknowns regarding the impact that adapted time series-based unsupervised metrics may have on results.

Given the uncertainties surrounding unsupervised tuning in the context of TSCL, we will not adopt this approach. Instead, we will focus on providing generally accepted "good" default parameter values sourced from the literature. If a distance or clusterer significantly under performs relative to our expectations, we may update these values. In such cases, we will conduct limited tuning using supervised metrics to improve the results, though this process will not be exhaustive. Additionally, any tuning will be applied across all datasets rather than on a per-dataset basis, with the goal of identifying generally effective parameter values, rather than maximising clustering performance for specific datasets.

## 3.3 Datasets

We conduct our experiments using the time series data from the University of California, Riverside (UCR) archive [23]. Our focus is on univariate time series, and for all experiments, we utilise 112 of the 128 datasets available in the UCR archive. We exclude datasets that contain series of unequal length or missing values.

The univariate UCR archive is a highly diverse collection, enabling us to perform experiments across a wide range of distinct time series. Tables 3.3, 3.4 and 3.5 provides summary distribution information about the 112 of the UCR univariate datasets we consider.

A key aspect of our evaluation methodology is the significant variation in dataset sizes, both in terms of the test-train split and the combined test-train split dataset size. This variation allows us to explore how different clustering algorithms perform with varying amounts of data and across time series of different lengths. The use of the test-train split specifically demonstrates how the models generalise to unseen data, while the combined test-train split dataset size reflects performance in traditional clustering scenarios.

Moreover, the wide range of unique class labels provides insights into how clustering algorithms perform with different numbers of clusters, as the unique labels will determine the number of clusters used in our experiments. Finally, the diverse time series domains in the UCR archive offer valuable insights into how models perform across different domains, revealing their strengths and weaknesses in various contexts.

| Number of training instances | | Number of test instances | |
|---|---|---|---|
| 1-100 | 41 (36.61%) | 1-100 | 18 (16.07%) |
| 101-250 | 20 (17.86%) | 101-250 | 27 (24.11%) |
| 251-500 | 29 (25.89%) | 251-500 | 26 (23.21%) |
| 501-1000 | 15 (13.39%) | 501-1000 | 13 (11.61%) |
| 1001-2000 | 3 (2.68%) | 1001-2000 | 10 (8.93%) |
| 2001-5000 | 2 (1.79%) | 2001-5000 | 14 (12.50%) |
| 5001+ | 2 (1.79%) | 5001+ | 4 (2.68%) |
| Minimum | 16 | Minimum | 20 |
| Maximum | 8926 | Maximum | 16800 |

Table 3.3 Summary of number of training and test instances per dataset distribution for 112 of the univariate UCR archive.

| Combined test-train split instances | |
|---|---|
| 1-200 | 22 (19.64%) |
| 201-500 | 23 (20.54%) |
| 501-1000 | 31 (27.68%) |
| 1001-2000 | 10 (8.93%) |
| 2001-5000 | 22 (19.64%) |
| 5001-10000 | 2 (1.79%) |
| 10001+ | 2 (1.79%) |
| Minimum | 40 |
| Maximum | 24000 |

| Number of unique class labels | |
|---|---|
| 1-2 | 40 (35.71%) |
| 3-8 | 50 (44.64%) |
| 9-20 | 11 (9.82%) |
| 21-30 | 2 (1.79%) |
| 31-40 | 2 (1.79%) |
| 41-50 | 3 (2.68%) |
| 51+ | 4 (3.57%) |
| Minimum | 2 |
| Maximum | 60 |

Table 3.4 Summary of number of combined test and train instances and number of unique class labels per dataset distribution for 112 of the univariate UCR archive.

| Time series length | |
|---|---|
| 1-200 | 40 (35.71%) |
| 201-500 | 31 (27.68%) |
| 501-1000 | 20 (17.86%) |
| 1001-2000 | 19 (16.96%) |
| 2001+ | 2 (1.79%) |
| Minimum | 15 |
| Maximum | 3000 |

| Dataset domain | |
|---|---|
| Device | 9 |
| ECG | 6 |
| Image | 32 |
| Motion | 17 |
| Sensor | 28 |
| Simulated | 8 |
| Spectro | 12 |

Table 3.5 Summary of number of time series lengths and dataset domain per dataset distribution

## 3.4 Normalisation

One of the key decisions we made was to apply z-normalisation to all our data. While we acknowledge that some argue "*any improvement resulting from pre-processing (normalisation) should not be attributed to the clustering method itself*" [51], others contend that "*in order to make meaningful comparisons between two time series, both must be normalised*" [96].

Therefore, the decision to normalise or not remains an ongoing research question. Ideally, with unlimited time and computational resources, we would run both normalised and un-normalised experiments to compare the results. However, given the extensiveness of our experimentation, this is not feasible, and we are forced to make a choice. Following the recommendation of [59] and [96] we choose to normalise our data.

## 3.5 Software and Research Reproducibility

One of the priorities of this thesis is to open source all code and results to make them available and reproducible by other researchers. During the course of this research, it became evident that one of the most glaring differences between the fields of TSCL and TSC is the availability of code for models and the consistency of results across datasets and methodologies.

We implemented all of our models and evaluation code in Python [1]. To ensure reproducibility, we open sourced all of our clusterers in the *aeon* [2] open source repository [82]. We also open sourced all of our evaluation code in the *tsml-eval* [3] open source project. Within the aeon toolkit, we also use the *tslearn* [4] open source repository [118] and the *scikit-learn* [5] open source repository [92].

All of our experiments were run on a single thread of an Ice Lake Intel Xeon Platinum 8358 2.6GHz processor on the University of East Anglia's high-performance computer (HPC) cluster with a 7 day computational runtime limit.

### 3.5.1 Time Series Clustering in Python: aeon

*aeon* is an open-source toolkit for time series machine learning. It is compatible with scikit-learn and provides access to the latest algorithms for time series machine learning, in addition to a range of classical techniques for tasks such as forecasting, clustering, and classification [82].

[1]https://www.python.org/
[2]https://github.com/aeon-toolkit/aeon
[3]https://github.com/time-series-machine-learning/tsml-eval
[4]https://github.com/tslearn-team/tslearn
[5]https://github.com/scikit-learn/scikit-learn

Processing time series data in *aeon* can be approached in multiple ways. An example of loading and clustering data using *aeon* is provided below. In the example, we create a time series dataset *X_train* consisting of six time series instances, each with 3 time points. *aeon* also offers utilities to load all UCR datasets referenced throughout this thesis. These datasets can be downloaded at timeseriesclassification.com. The example below was generated using version 0.11.0 of the *aeon* toolkit.

```python
import numpy as np
from aeon.clustering import TimeSeriesKMeans

X_train = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [13, 14,
    15], [16, 17, 18], [19, 20, 21]])

combined_X = np.concatenate([X_train, X_test])

clst = TimeSeriesKMeans(n_clusters=3, distance="dtw",
    random_state=0)

labels = clst.fit_predict(combined_X)
```

# Chapter 4

# Lloyd's-based TSCL

## Contributing Publications

- Holder, C., Bagnall, A., Lines, J. (2024). On time series clustering with $k$-means. *arXiv preprint arXiv:2410.14269*. Available at: https://arxiv.org/abs/2410.14269.

Before exploring TSCL with elastic distances, we first establish a baseline to compare our results against. The most common way to cluster time series is to use $k$-means combined with a time series-specific distance and averaging technique [89, 46, 47, 128, 21, 94] .One of the most common implementations of $k$-means is using Lloyd's algorithm. However, since Lloyd's original proposal [75], numerous modifications have been introduced. In traditional clustering, many of these modifications are considered essential for achieving meaningful results and as such the "baseline" $k$-means is assumed to adopt many of these strategies [92].

However, in TSCL there does not seem to be a well defined version of Lloyd's algorithm that is consistently adopted. As will be shown, some researchers incorporate specific modifications suggested in traditional clustering literature, while others use the unmodified, original Lloyd's algorithm. Furthermore, we observe instances where different configurations of Lloyd's-based algorithms are used within the same paper. We argue that meaningful comparisons between Lloyd's-based algorithms cannot be performed if the Lloyd's-based algorithms are poorly configured, lack essential modifications, or are configured inconsistently.

We dedicate an entire chapter to this topic because Lloyd's-based algorithms and their comparison are a central focus of this thesis. Much of our contribution lies in the evaluation and comparison of various Lloyd's-based algorithms using different elastic distances and averaging techniques. Additionally, we aim to address a gap in the research by providing a well-defined configuration for Lloyd's-based algorithms, which is currently lacking in the literature.

Therefore, in this chapter, we aim to establish a robust baseline for Lloyd's algorithm in the context of TSCL. We begin by reviewing the existing literature and model configurations to highlight the inconsistencies in prior work. Next, we propose a modified version of Lloyd's algorithm, clearly explaining our design choices and providing pseudocode for clarity. After defining our version of Lloyd's algorithm, we detail our experimental setup and recommend default parameters for Lloyd's algorithms. Our goal is to keep as many variables constant across different Lloyd's variants, adjusting only the specific parameters of each variant to isolate their impact on results.

Finally, with a consistent Lloyd's-based configuration and well-defined default parameters, we perform a baseline experiment of the existing TSCL literature using our Lloyd's algorithm and experimental methodology. While our results are generally consistent with the existing TSCL literature, we uncover some notable findings that we believe are only revealed due to our robust configuration and experimental approach.

## 4.1 Introduction

*k*-means is one of the simplest and most well-researched approaches for TSCL. Several TSCL-specific variants of *k*-means have been proposed, such as *k*-shapes [89], *k*-SC [128], and *k*-DBA [94]. All of these algorithms use Lloyd's algorithm [75] (outlined in Figure 2.24) but modify either the distance measure used and/or the averaging method employed.

In traditional clustering, *k*-means has an extensive body of literature dedicated to improving every aspect of the algorithm, including initialisation of centroids, selecting the optimal number of clusters, early convergence criteria, maximum number of iterations, distance measures, and averaging methods, among others. The list of improvements to the base *k*-means is so extensive that entire papers are dedicated to reviewing the literature and summarising recent research on *k*-means [48].

However, within the context of TSCL, there does not appear to be an agreed-upon "default" version of Lloyd's algorithm. Some research employs an unmodified version of Lloyd's algorithm [89], while other studies incorporate select optimisations [46, 47]. Consequently, when comparing Lloyd's-based algorithms, it is often unclear whether the observed differences in results stem from the proposed modifications or from variations in the configurations of Lloyd's algorithm.

To address this issue, we will first identify the various versions of Lloyd's algorithm employed in the TSCL literature. Through this review, we will demonstrate that there is a lack of consistency in the configuration of Lloyd's algorithm, making it difficult to compare results across different studies. This inconsistency suggests that previous claims regarding the success of certain models may be influenced more by the specific configuration of Lloyd's algorithm rather than the inherent effectiveness of the proposed methods.

Following this, we will explicitly define our configuration of Lloyd's algorithm, providing justification for each configuration choice. Subsequently, we will run each of the aforementioned Lloyd's-based algorithms using our defined configuration, applying our methodology to reassess previous work. These results will serve as a baseline for our subsequent experiments.

## 4.2 Lloyd’s algorithm in TSCL

Lloyd’s-based methods have been the most popular approach for TSCL. Numerous papers have proposed variants of Lloyd’s algorithm. Additionally, many of these Lloyd’s-based methods, such as *k*-shapes, *k*-DBA, and *k*-SC, have been considered by many as baselines for comparison. Table 4.1 provides an overview of some of the literature that employs Lloyd’s algorithm-based techniques. The table highlights a diverse range of TSCL literature, from general reviews of TSCL, such as [51], to the proposal of new models that are compared against Lloyd’s techniques, such as [89], and real-world use cases that utilise Lloyd’s methods, such as [93].

Table 4.1 also highlights the specific configuration decisions made for each Lloyd’s technique. Upon reviewing Table 4.1, a keen observer may notice that no two experiments use the same configuration. We conducted an extensive search and found it is very rare to find two papers with the same Lloyds configurations. We believe a significant factor contributing to this is the frequent reference to a “default” or “traditional” version of *k*-means (Lloyd’s algorithm), without any clear authority defining this “default” version.

As we are unable to find a clear definition of a “default” variant of Lloyd’s algorithm for TSCL, we take it upon ourselves to first define an explicit “default” implementation of Lloyd’s algorithm to be used consistently across all our experiments. By doing so, we will eliminate differences in the performance of our Lloyd’s algorithm due to varying configurations. To achieve this, we will go through each stage of Lloyd’s algorithm and clearly define our choices for different optimisations. We will then perform basic experiments with *k*-means to provide further validation for our choices. Our goal is not to create the most well-optimised version of Lloyd’s algorithm, but rather to create a version that assumes the average case for most use cases and will allow meaningful experimentation to be performed using the models.

| Reference | Num. Cites | Init | Distance | Averaging | Max iters | Early stopping |
|---|---|---|---|---|---|---|
| [64] | 50 | *k*-means++ | Euclidean, SBD, DTW | Mean, Shape extraction, DBA | 200 | Inertia change |
| [89] | 504 | Random | Euclidean, SBD, DTW, KSC dist | Mean, Shape extraction, DBA, KSC average | 100 | Membership doesn't change between iterations |
| [94] | 1248 | Forgy | DTW | DBA | 10 | - |
| [51] | 148 | Forgy | DTW, Euclidean, SBD | DBA, Mean, Shape extraction | 15 | Stops if number of clusters reduces |
| [61] | 2 | Forgy | DTW, Euclidean, soft-DTW | DBA, Mean, soft-DBA | - | - |
| [130] | 95 | - | Euclidean, SBD, DTW, KSC dist | Mean, Shape extraction, DBA, KSC average | - | - |
| [93] | 271 | - | DTW, Euclidean | DBA, Mean | - | - |
| [90] | 144 | Random | Euclidean, SBD, DTW, KSC dist | Mean, Shape extraction, DBA, KSC average | 100 | Membership doesn't change between iterations |
| [77] | 213 | Random with 5 restarts | Euclidean, SBD, DTW, KSC dist | Mean, Shape extraction, DBA, KSC average | - | - |
| [3] | 50 | Forgy | Euclidean, SBD, DTW, KSC dist | Mean, Shape extraction, DBA, KSC average | - | - |
| [88] | 263 | Random | Euclidean, DTW | Mean | - | Membership doesn't change between iterations |
| [91] | 1 | Random | Euclidean, SBD, DTW, KSC dist | Mean, Shape extraction, DBA, KSC average | 300 | Membership doesn't change between iterations |

Table 4.1 A sample of TSCL literature using Lloyd-based methods. The Reference column lists relevant papers, with Num. Cites showing citation counts (as of August 2024). Init indicates the initialisation strategy, Distance specifies the distance measure, and Averaging lists the corresponding averaging methods. Max Iters defines the iteration limit, while Early Stopping outlines convergence criteria. A "—" denotes unspecified or missing values.

## 4.3 A TSCL configuration for Lloyd's-based algorithms

We will now outline our baseline configuration for Lloyd's algorithm in the context of TSCL. Our configuration choices are motivated by traditional clustering literature, as many of these decisions are data-independent. However, to validate these hypotheses for time series data, we also conduct basic experiments to demonstrate that our reasoning holds true for TSCL.

The most basic version of Lloyd's algorithm is outlined as a flow diagram in Figure 2.24. In addition Algorithm 22 shows the most basic implementation of Lloyd's algorithms.

**Algorithm 22:** Lloyd's Algorithm for k-means (**X**, **k**)

```
Input: X (Dataset of time series of length $n$), k (Number of clusters)
Output: Assignment of each time series to a cluster
1  Let centres be an array of $k$ randomly chosen time series from dataset X
2  Let assignments be an empty array of length $n$
3  repeat
4      for each time series $x_i$ in X do
5          Compute the distance between $x_i$ and each of the $k$ centres
6          Assign $x_i$ to the nearest centre
7      for each centre $c_j$ in centres do
8          Update $c_j$ to be the mean of all time series assigned to it
9  until assignments does not change between iterations;
10 return assignments
```

In Algorithm 22 line 1 uses a random initialisation strategy to define initial centroids. Lines 3 and 9 outline the stopping condition: continue until assignments don't change between iterations. Lines 4 to 6 show the assignment phase of Lloyd's algorithm. Lines 7 to 8 demonstrate the centroid update stage. Finally, line 10 shows the returning of the assignments for each time series in $X$.

### 4.3.1 Initialisation Strategy

The first explicit definition we will provide is the choice of cluster initialisation. Aside from selecting an appropriate number of clusters, the initialisation strategy is one of the most critical decisions in Lloyd's algorithm. However, there is no universally "best" solution to the initialisation problem, and it remains an active area of research [48]. In traditional clustering, default approaches are often recommended for initialisation, but these have not been commonly adopted in TSCL.

Numerous initialisation strategies have been proposed. The most comprehensive review of these strategies was conducted by [17], who compared eight of the most commonly used methods. In Table 4.1, three different initialisation strategies are outlined: Forgy, Random and $k$-means++.

- *Forgy* [35] initialisation chooses $k$ random time series from the dataset to be the initial centroids.
- *k-means++* [6] initialisation starts by choosing the first centroid randomly. Each subsequent centroid is then chosen based on a probability proportional to the squared Euclidean distance from the nearest existing centroid. This strategy ensures that centroids are spread out more evenly.
- *Random* initialisation selects $k$ initial centroids by randomly assigning each value in the dataset to a cluster. Once all values have been assigned, the average of each cluster is computed, and these averages are used as the initial centroids.
- *Greedy k-means++* [6] is a variant of $k$-means++ that, instead of using a probabilistic selection process for each new centroid, deterministically finds the time series in the dataset that maximises the minimum distance from the

already chosen centroids. This ensures that the centroids are as spread out as possible from the initial selection.

While [17] did not provide a definitive answer to the best initialisation strategy, some general conclusions were drawn. First, initialisation strategies such as Forgy and random initialisation should be avoided due to their unreliability. Instead, [17] recommended approaches such as greedy *k*-means++. Others have found similar results, such as [1], who found that randomly generated centroids, without considering the position of such centroids in the datasets, lead to unexpected convergence. Overall, randomly selected centroids cause the clustering operation to get stuck in a low local minimum [37].

Greedy *k*-means++ traditionally employs a distance measure, typically the squared Euclidean distance, to compute similarity. However, as many others [74, 111], and this thesis, have found, the Euclidean distance is not a suitable measure for assessing time series similarity. Thus, while it is desirable to use greedy *k*-means++ due to its success in traditional clustering, it is unknown whether it will have the same positive impact for TSCL (due to the use of the Euclidean distance). While the logical step would be to change the distance measure *k*-means++ uses to match that of Lloyd's algorithm in use (e.g., use DTW), this may adversely impact some distances more than others, thus making the overall impact difficult to measure. Therefore, before *k*-means++ can be recommended, this gap in the research needs to be addressed. We seek to investigate this problem in Chapter 8.

As there are many unknown variables when using *k*-means++ for TSCL, we will not be using it for our experimentation. Additionally, we will not use single run traditional methods such as Forgy and random initialisation, as random initialisation strategies have been shown to lead to unexpected clustering convergence (local minima). However, one proposed method to escape local minima is to rerun Lloyd's algorithm with *r* different starting centroids. Specifically, one can perform Forgy or random initialisation to initialise $k$-means $r$ times and then select the final clustering with the minimum SSE (Equation 2.32) [116]. The main problem with this method is that it does not guarantee obtaining the optimal solution unless $r$ is very large (thus increasing the time complexity) [116]. Therefore, this approach is likely unviable in real-world TSCL use cases, but it is useful for experimental methodologies (until other initialisation strategies are understood for TSCL), as each experiment has $r$ opportunities to form optimal clusterings.

Therefore, we will adopt an initialisation strategy using Forgy initialisation and perform 10 restarts, selecting the restart that yields the smallest SSE for the clustering configuration. We chose the value of 10 as it is the default recommendation by [92] when using a random initialisation strategy. For larger datasets, this value may be insufficient, so we have implemented additional strategies to reduce the likelihood of converging to local optima, including contingencies for empty cluster formation, which will be discussed further below. We chose Forgy over random initialisation because random initialisation involves an averaging stage. As has been observed by others, and in this thesis, the traditional arithmetic mean is not suitable for averaging time series. Therefore, we will adopt the simplest approach with the fewest potential variables in the context of TSCL: Forgy initialisation with 10 restarts.

Algorithm 23 demonstrates the process of enhancing Lloyd's algorithm with restarts. Specifically, Algorithm 23 introduces a new parameter called $n\_init$. For our experimentation, we set $n\_init$ to 10 to perform 10 restarts with different initial centroids. To keep track of the best clustering, lines 1 to 3 introduce three variables to track the best solution found. The variable $best\_inertia$ is initialised to infinity and will store the inertia of the best restart. Line 4 begins the loop that performs $n\_init$ runs of Lloyd's algorithm with different initial centroids. After the standard Lloyd's algorithm completes on line 13, line 14 calculates the inertia ($curr\_inertia$) for the current iteration, and line 15 compares this value to the previous *best_inertia*. If the *curr_inertia* is less than *best_inertia*, the variables tracking the best run are updated accordingly. Finally, after all restarts have been executed, the best clustering is returned on line 19.

**Algorithm 23:** Lloyd's Algorithm with restart initialisation strategy (**X**, **k**, **n_init**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_init (Number of restarts with different initial centroids)
Output: Assignment of each time series to a cluster
1  best_inertia ← $\infty$
2  Let best_assignments be an empty array of length $n$
3  Let best_centres be an empty array of length $n$
4  for $i \leftarrow 1$ to n_init do
5      Let centres be an array of $k$ randomly chosen time series from dataset X
6      Let assignments be an empty array of length $n$
7      repeat
8          for each time series $x_i$ in X do
9              Compute the distance between $x_i$ and each of the $k$ centres
10             Assign $x_i$ to the nearest centre
11         for each centre $c_j$ in centres do
12             Update $c_j$ to be the mean of all time series assigned to it
13     until assignments does not change between iterations;
14     Let curr_inertia be the SSE of the current clustering
15     if curr_inertia < best_inertia then
16         best_inertia ← curr_inertia
17         best_assignments ← assignments
18         best_centres ← centres
19 return best_assignments
```

To demonstrate our hypothesis that Forgy initialisation with 10 restarts across the UCR dataset is a sensible baseline, we conducted a series of simple experiments using the traditional $k$-means algorithm with different initialisation techniques.

The CD diagrams in Figure 4.5 show that for $k$-means, all the initialisation strategies fall into the same clique and therefore are not critically different. Table 4.2 presents the specific average scores across the UCR archive. While the values in Table 4.2 appear fairly similar across all metrics, assessing the variation highlights the rationale behind our choice.

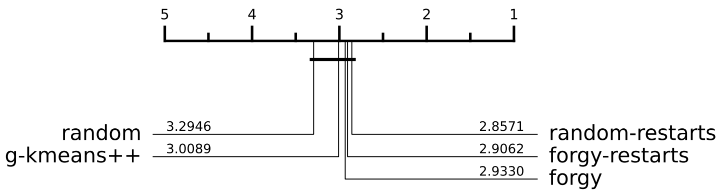


Fig. 4.1 AMI

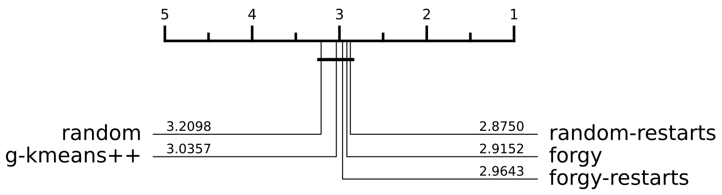


Fig. 4.2 ARI

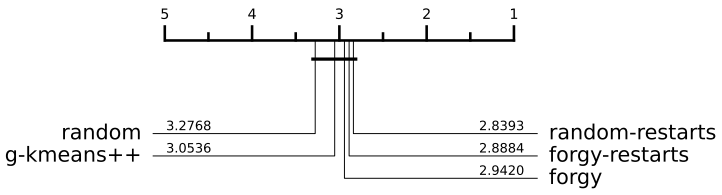


Fig. 4.3 NMI

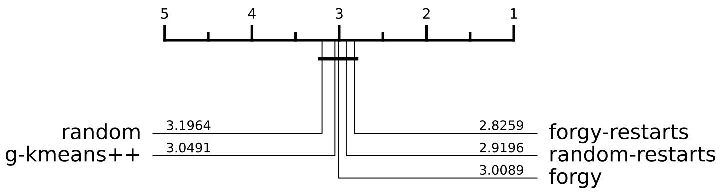


Fig. 4.4 CLACC

Fig. 4.5 CD diagram of different initialisation strategies for *k*-means over 112 datasets from the UCR archive using the combined test-train split. "random" refers to random initialisation, "random-restarts" refers to random initialisation with 10 restarts, where the restart with the lowest inertia is selected. "forgy" denotes Forgy initialisation, "forgy-restarts" represents Forgy initialisation with 10 restarts, where the restart with the lowest inertia is selected, and "g-kmeans++" denotes greedy *k*-means++.

Figure 4.6 shows a violin plot of the different initialisation strategies across the UCR archive for CLACC. In Figure 4.6, Forgy, random, and greedy *k*-means++ display large variability in their performance, as demonstrated by the distribution of their plots. However, Forgy with 10 restarts and random with 10 restarts produce significantly more consistent results with very little variability across the UCR archive. This is the type of performance we seek for our experimental methodology.

To clarify, based on Figure 4.6, Table 4.2, and Figure 4.5, it is clear that we could choose either random or Forgy initialisation with 10 restarts. However, as stated, we elect to use Forgy as it is the simplest option and introduces the fewest variables that could impact our results.

One of the disadvantages of using an initialisation with restarts is the greatly increased computational time required to run the model. In our case, by using 10

| | ARI | AMI | CLACC | NMI | RI |
|---|---|---|---|---|---|
| random-restarts | **0.209** | **0.258** | 0.525 | **0.283** | **0.700** |
| forgy-restarts | 0.208 | 0.257 | **0.526** | 0.283 | 0.699 |
| random | 0.201 | 0.251 | 0.518 | 0.276 | 0.697 |
| forgy | 0.203 | 0.255 | 0.521 | 0.281 | 0.696 |
| g-*k*-means++ | 0.203 | 0.250 | 0.518 | 0.276 | 0.693 |

Table 4.2 Summary of initialisation strategies' average scores across multiple evaluation metrics over 112 datasets from the UCR archive using the combined test-train split.

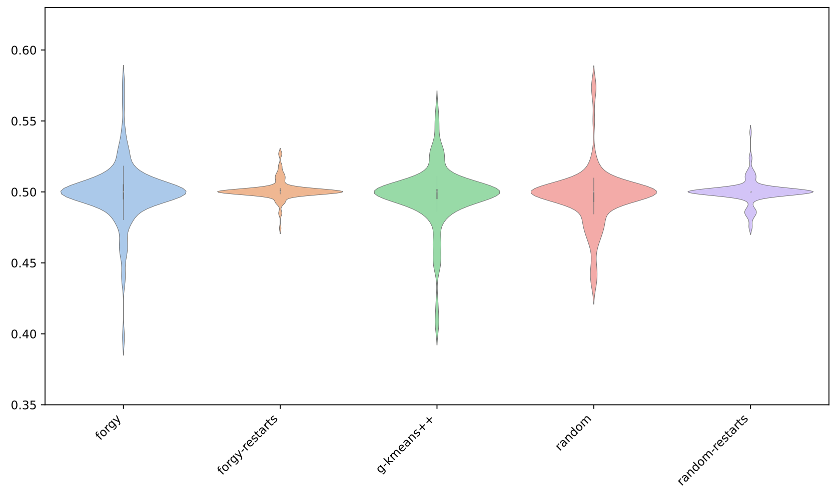


Fig. 4.6 CLACC violin plot for different initialisation strategies over the 112 of the UCR archive using the combined test-train split.

restarts, we increase our run time by 10 times. This is illustrated in Figure 4.7, where the distribution of run times for initialisation techniques that use restarts is significantly higher than for those that only run once.

## 4.3.2 Early Stopping Conditions

Lloyd's algorithm, in its original form, is proven to always converge in a finite number of iterations [75]. While this convergence may not necessarily lead to the global optimum, it will always converge to some local optimum. Lloyd's algorithm considers convergence to be achieved when the SSE does not change between iterations. When the SSE remains the same across iterations, it indicates that the

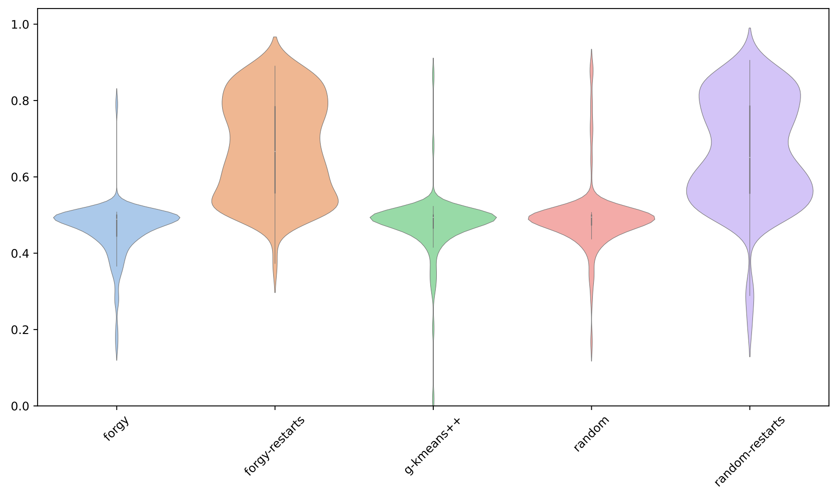


Fig. 4.7 Relative FitTime violin plot for different initialisation strategies over the 112 of the UCR archive using the combined test-train split.

cluster assignments and, consequently, the centroids do not change, signifying that the algorithm has converged. This is the default stopping condition for Lloyd's algorithm.

However, in addition to this stoppage condition, numerous early stopping conditions have been proposed for Lloyd's algorithm. The purpose of these early stopping conditions is to terminate the algorithm before full convergence is reached, thereby saving computation time. Early stopping is often utilised because Lloyd's algorithm exhibits diminishing returns with each iteration. Theoretically, as Lloyd's algorithm progresses towards convergence, the number of changes to cluster assignments should decrease with each iteration. This suggests that while a substantial number of changes might occur in the initial iterations, these changes should significantly diminish as the algorithm continues.

As a result, Lloyd's algorithm may reach a point where it updates very little between iterations, yet it takes a considerable amount of time to reach a final converged solution. This final solution might not be significantly better than the one obtained if the algorithm were terminated early. Therefore, early stopping conditions aim to strike a balance between obtaining good, near-converged results while terminating at an appropriate time to save computational resources.

### 4.3.3 Early Stopping Conditions: Maximum iterations

The first early stopping condition commonly used is setting a maximum number of iterations before the algorithm must terminate. This stopping condition acts as a safety net to prevent the algorithm from running for an infeasibly long time.

The maximum number of iterations is set by a parameter called *max_iters*, which represents the maximum number of iterations before the algorithm is terminated. *max_iters* should be set so that, in most cases, the maximum number of iterations is never reached. However, in the rare case that this condition is met, the maximum number of iterations should be sufficiently high to ensure that cluster assignments do not change significantly between iterations.

The value for the maximum number of iterations is difficult to estimate because the number of iterations Lloyd's algorithm may take to converge depends on several factors: the number of clusters, the size of the dataset, and the initial centres selected. Additionally, some algorithms in the context of TSCL require more iterations than traditional Lloyd's due to the use of approximation strategies. For example, the $k$-means-DBA averaging stage employs DBA, which is an approximation of the average under DTW. The use of approximations can result in slower convergence.

With these factors considered, we aim to set our maximum iterations as high as possible while being conscious of our computational resources. Table 4.3 shows the average, minimum, and maximum iterations for computing squared Euclidean $k$-means over 112 datasets from the UCR archive using the combined test-train split. On average, a dataset in the UCR archive will converge in under 20 iterations. However, there are 9 datasets that take, on average, more than 40 iterations in their "best iteration" to converge. These datasets are shown in Table 4.4. Of the 9 datasets, only 2 exceed 50 iterations on an average iteration.

| | Average Iterations | Best Iteration |
|---|---:|---:|
| Mean | 17.42 | 18.44 |
| Min | 3.80 | 3.00 |
| Max | 74.90 | 140.00 |

Table 4.3 Number of iterations required for the squared Euclidean distance clustering algorithm to converge without an early stopping condition on 112 datasets from the UCR archive using the combined test-train split. The column labelled "Average Iterations" indicates the average number of iterations across 10 restarts. The "Best Iteration" column represents the number of iterations taken by the restart that achieved the lowest inertia.

| Dataset | Best Iteration | Average Iterations |
|---|---:|---:|
| ElectricDevices | **140** | **74.9** |
| Crop | 59 | 66.2 |
| FaceAll | 52 | 37.0 |
| UWaveGestureLibraryAll | 51 | 26.6 |
| UWaveGestureLibraryZ | 51 | 46.0 |
| FordA | 47 | 48.0 |
| SemgHandSubjectCh2 | 47 | 31.8 |
| UWaveGestureLibraryY | 46 | 43.3 |
| FacesUCR | 42 | 37.4 |

Table 4.4 The 9 datasets that averaged over 40 iterations in their "Best Iterations" for the squared Euclidean *k*-means, out of 112 datasets from the UCR archive using the combined test-train split.

With these baseline statistics on the number of iterations it takes for the squared Euclidean distance to converge over the UCR archive, we elect to set our maximum number of iterations to 50. While this number means that 5 datasets would not be able to find their optimal convergence (as shown in Table 4.4), we are limited by computational resources. If we had unlimited computational resources, we would set the maximum number of iterations to 300 or more to ensure convergence in every scenario. However, when considering the computational cost of experiments with such high iterations, which run with 10 restarts, this value is not viable.

Furthermore, for TSCL specifically, some of the variants of Lloyd's algorithm that this thesis will review use extremely computationally expensive distance and averaging methods. For example, $k$-means-DBA uses the DTW distance for the assignment stage, which is prohibitively computationally expensive. The averaging technique used, DBA, also uses DTW, which further makes this algorithm computationally impractical.

As such, while for some datasets optimal convergence will likely not be reached, this is expected to impact the results of only 2 datasets (ElectricDevices and Crop) on an average iteration. Additionally, as every variant of Lloyd's algorithm is constrained by the same maximum iteration parameter, they are all subject to the same experimental conditions, which ensures that our experimental results will remain valid.

Finally, a consideration is that in the context of the squared Euclidean $k$-means clusterer, which uses the Forgy initialisation strategy, we would expect the number of iterations taken to converge to be higher. As mentioned, the number of iterations required to converge is directly linked to the quality of the initial centroids.

The second consideration as to why the squared Euclidean $k$-means clusterer may have a higher number of iterations than expected is that the squared Euclidean distance and the arithmetic mean are not ideal similarity or averaging techniques for time series data. Therefore, theoretically, if we improve the distance and averaging techniques for time series data, we would expect the algorithm to converge faster, as more informed decisions are made by the model. This suggests that, as we explore TSCL-specific variants of Lloyd's algorithm, the number of iterations required to converge should decrease, further reducing the likelihood that the maximum iteration early stopping condition will be reached.

Algorithm 24 shows an updated version of Lloyd's algorithm that incorporates a maximum number of iterations. Specifically, Algorithm 24 introduces an additional parameter called $max\_iters$. This parameter defines the maximum number of iterations before the algorithm is forcibly terminated. Line 7 shows the loop from 1 to *max_iters*, which replaces the repeat-until loop in the previous version of Lloyd's algorithm. The original early stopping condition of Lloyd's algorithm is then moved to the end of the iteration on line 13. This condition checks whether the assignments have changed between iterations, and if they have not, convergence has been reached and the loop is therefore terminated.

**Algorithm 24:** Lloyd's Algorithm with a maximum number of iterations (**X**, **k**, **n_init**, **max_iters**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_init (Number of restarts with different initial centroids),
       max_iters (Maximum number of iterations before forced
       termination)
Output: Assignment of each time series to a cluster
1  best_inertia ← $\infty$
2  Let best_assignments be an empty array of length $n$
3  Let best_centres be an empty array of length $n$
4  for $i$ ← 1 to n_init do
5      Let centres be an array of $k$ randomly chosen time series from dataset X
6      Let assignments be an empty array of length $n$
7      for $j$ ← 1 to max_iters do
8          for each time series $x_i$ in X do
9              Compute the distance between $x_i$ and each of the $k$ centres
10             Assign $x_i$ to the nearest centre
11         for each centre $c_j$ in centres do
12             Update $c_j$ to be the mean of all time series assigned to it
13         if assignments does not change between iterations then
14             break
15     Let curr_inertia be the SSE of the current clustering
16     if curr_inertia < best_inertia then
17         best_inertia ← curr_inertia
18         best_assignments ← assignments
19         best_centres ← centres
20 return best_assignments
```

### 4.3.4 Early Stopping Conditions: Inertia Change

The second early stopping condition we adopt is measuring the inertia change (or change in sum of squared distances) of assignments between iterations. Instead of checking whether the exact assignment of time series change between iterations, we measure the change in inertia between iterations. This functionally achieves the same result as checking exact assignment changes, but allows practitioners to set a parameterised threshold ($tol$), enabling the detection of sufficient convergence, so that small changes in assignments between iterations are still considered convergence.

The computation of inertia is the sum of the squared distances of each time series to its closest centroid (i.e., the distance to its cluster assignment). By measuring the inertia between iterations, if cluster assignments do not change, the inertia change between iterations will be 0. Assuming $tol \geq 0$, this means convergence will be reached. This scenario is equivalent to the traditional Lloyd's stopping condition of assignments not changing between iterations.

By having a tolerance threshold, practitioners can control the definition of convergence for the algorithm. Additionally, as outlined, Lloyd's iterations exhibit diminishing returns. This means that, as Lloyd's algorithm moves toward convergence, smaller and smaller changes to assignments will be made. Sometimes, this also means Lloyd's can get stuck in local optima, leading to minor fluctuations in cluster assignments that yield very little improvement in overall cluster performance. Under traditional Lloyd's, the algorithm could remain stuck in a local optima for a long time, until it eventually converges, which may yield minimal improvements. By using inertia between iterations as the stopping criterion, these slight fluctuations are detected, allowing the algorithm to converge without wasting additional computational resources on small improvements.

The value of $tol$ is difficult to set, as it depends on what a practitioner may want to define as convergence. However, for our experiments, while computational runtime is a consideration, our focus is more on getting as close to "true" Lloyd's convergence as possible without wasting iterations on very minimal improvements. As such, we set our value of $tol$ to be $1 \times 10^{-6}$. We chose this value as it is the default for the `tslearn` package. Additionally, we considered the `scikit-learn` package, which uses a value of $1 \times 10^{-4}$ for $tol$. We opted for the smaller value to be conservative, so that we do not change the convergence criteria of Lloyd's algorithm too much; rather, we aim to save computational resources where very small fluctuations are occurring. This value of $tol$ may not be optimal for reaping the most runtime benefit; however, for the sake of our experimentation, we prefer a conservative estimate to ensure consistent convergence.

Algorithm 25 shows the updated version of Lloyd's algorithm to use an inertia tolerance threshold. Specifically, Algorithm 25 introduces an additional parameter $tol$, which is the inertia variation threshold. This means that if the inertia changes by less than $tol$ between iterations, the algorithm will converge. To track the inertia change, line 7 defines a new variable $prev\_inertia$, which tracks the previous iteration's inertia. Next, for each iteration, $curr\_inner\_inertia$ is computed on line 14. This value is then compared to $prev\_inertia$ on line 15. If this difference is less than the tolerance threshold, the algorithm converges. However, if it is greater than the tolerance threshold, $prev\_inertia$ is set to the current iteration's inertia, and the refinement continues.

**Algorithm 25:** Lloyd's Algorithm with an inertia tolerance threshold (**X**, **k**, **n_init**, **max_iters**, **tol**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_init (Number of restarts with different initial centroids),
       max_iters (Maximum number of iterations before forced
       termination), tol (Inertia variation threshold)
Output: Assignment of each time series to a cluster
1  $best\_inertia \leftarrow \infty$
2  Let $best\_assignments$ be an empty array of length $n$
3  Let $best\_centres$ be an empty array of length $n$
4  for $i \leftarrow 1$ to $n\_init$ do
5      Let $centres$ be an array of $k$ randomly chosen time series from dataset $\mathbf{X}$
6      Let $assignments$ be an empty array of length $n$
7      Let $prev\_inertia \leftarrow \infty$
8      for $j \leftarrow 1$ to $max\_iters$ do
9          for each time series $x_i$ in $\mathbf{X}$ do
10             Compute the distance between $x_i$ and each of the $k$ centres
11             Assign $x_i$ to the nearest centre
12         for each centre $c_j$ in $centres$ do
13             Update $c_j$ to be the mean of all time series assigned to it
14         Let $curr\_inner\_inertia$ be the SSE of the current clustering
15         if $|curr\_inner\_inertia - prev\_inertia| < tol$ then
16             break
17         $prev\_inertia \leftarrow curr\_inner\_inertia$
18     Let $curr\_inertia$ be the SSE of the current clustering
19     if $curr\_inertia < best\_inertia$ then
20         $best\_inertia \leftarrow curr\_inertia$
21         $best\_assignments \leftarrow assignments$
22         $best\_centres \leftarrow centres$
23 return $best\_assignments$
```

### 4.3.5 Empty Clusters

As outlined, the performance of Lloyd's algorithm on any given dataset is undeniably dependent on the number of clusters specified [48]. Choosing the wrong number of clusters can lead to strange convergence. Additionally, how clusters are initialised is critically important as well. Setting an inappropriate number of clusters or having poor initial centroids can result in empty clusters being formed. An empty cluster is a cluster that has no values assigned to it, which is problematic because it means this cluster is "stuck" and cannot be updated further. Fundamentally, this results in one fewer cluster than specified being formed.

The traditional Lloyd's algorithm does not provision for the formation of empty clusters. Additionally, in many implementations, there is no detection of an instance where an empty cluster is formed. In our initial implementation, we did not have any explicit detection of empty cluster formation, and our only indication empty clusters were being formed was due to errors being thrown when we tried to compute the average of an empty assignment arrays.

On the one hand, if Lloyd's algorithm forms an empty cluster, one could consider this a form of early convergence. In [51], their *k*-means clusterer was set up in this way. However, in our experiments, the formation of an empty cluster is likely a weakness of our underlying model configuration rather than a reflection of the model's incapacity to properly cluster. Specifically, setting an arbitrary number of clusters makes the possibility of empty clusters more likely.

Therefore, we looked for a potential solution in the TSCL literature. However, we were unable to find any examples where the formation of empty clusters was explicitly addressed for TSCL. The only acknowledgement of empty clusters forming in TSCL that we could find was in [51]. [51] did not explicitly discuss empty clusters in their paper, but in the source code provided, we found that they used the formation of an empty cluster as a form of early stopping condition.

While we could follow [51] and use the formation of an empty cluster as an early stopping condition, we do not believe this is a good criterion to express early convergence. As such, we look for a more robust solution in the traditional clustering literature.

In traditional clustering, there is limited literature on defining protocols for handling empty clusters. The general advice to practitioners to reduce the number of clusters [92]. Due to the limitations described in this chapter and in Chapter 3, we want to maintain a consistent number of clusters matching the number of ground truth labels.

As such, we adopt a strategy where, when an empty cluster is formed, we choose a time series from the dataset to become a new centroid. The choice of which time series to select is an important consideration. There are two methods for selecting this new centroid: randomly choosing a value from the dataset or choosing a value that reduces inertia by the largest amount.

Randomly selecting a time series from the dataset to be a new centroid is a simple solution but has certain implications. Firstly, the selected value could be located in a similar position to the previous empty cluster, potentially leading to another empty cluster forming shortly thereafter, necessitating yet another random centroid selection. Secondly, at the point where the empty cluster forms, the algorithm is likely already partially converged towards a local optimum. Selecting a new random centroid could entirely change the optimum the algorithm was converging towards. To some extent, this could be considered equivalent to a complete restart of the algorithm with new initial centroids, leading to convergence towards a different optimum. As such, random selection may unintentionally bias clustering results. Therefore, we choose not to use this strategy.

Choosing a time series that reduces inertia by the largest amount as the new centroid is generally a more effective strategy than random selection because it directly targets the objective of the clustering algorithm. This strategy is the default solution that scikit-learn [92] employs to handle empty clusters in their *k*-means implementation.

By selecting a time series that minimises inertia, the algorithm ensures that the new centroid contributes to a more optimal clustering configuration. This approach helps the algorithm maintain its progress toward convergence rather than potentially disrupting it with a random selection that could lead to a less efficient clustering outcome. Furthermore, it reduces the likelihood of forming another empty cluster, as the selected time series is likely to be situated in a region of the dataset where its inclusion will meaningfully improve cluster quality.

To identify the time series that would reduce the inertia by the largest amount, the time series that is furthest from its assigned cluster centroid should be chosen. This approach ensures that the new centroid is positioned in a way that maximally improves the overall clustering by reducing the distance of an outlier data point, thereby contributing the most to the reduction of inertia.

Algorithm 26 shows our final version of Lloyd's algorithm, which can handle empty clusters. Specifically, lines 12 to 19 outline how this is managed. The empty cluster algorithm will continue to loop until there are no more empty clusters.

### 4.3.6 Distance Measure and Averaging Technique

The objective of *k*-means is to minimise the sum of squared errors (SSE) as given in Equation 2.32. Lloyd's algorithm achieves this by iteratively performing two key steps: assignment and centroid computation. The assignment phase traditionally employs the squared Euclidean distance, while the computation of centroids is based on the arithmetic mean. The squared Euclidean distance and arithmetic mean are fundamentally linked because the arithmetic mean minimises the sum of squared Euclidean distances between a set of data points and their centroid.

This implies that when considering a distance measure for the *k*-means algorithm, the corresponding averaging technique must minimise the sum of squared distances to ensure the methods validity. Although there are numerous distance measures for time series data, only a limited number of time series averaging techniques satisfy this criterion, which restricts their applicability within Lloyd's framework.

In the TSCL literature, five primary variants of Lloyd's algorithm have been defined, each specifying a distance measure and an associated averaging technique. These have been outlined in Section 2.5.1 of our literature review, however, in summary:

- *k-means* [75] is the traditional *k*-means algorithm that uses the squared Euclidean distance and the arithmetic mean, which minimises the sum of squared Euclidean distances.
- *k-means-DBA* [94] is a variant of *k*-means that uses the DTW distance and minimises over the DTW distance using DBA.
- *k-SC* [128] is another variant of *k*-means that utilises the *k*-SC distance (which does not have a specific name) and an averaging technique that minimises the *k*-SC distance.
- *k-shape* [89] is a variant that uses the SBD and employs a shape extraction algorithm to derive an average that minimises SBD.
- *k-means-soft-DBA* [21] is a variant of *k*-means that use the soft-DTW distance and minimises over the soft-DTW distance using soft-DBA.

## 4.4 Lloyd's Baseline

Before we begin exploring elastic distances for TSCL, we must first define a baseline for comparison. While similar baseline comparisons already exist in the literature, none of these results have been generated following our extensive TSCL methodology outlined in Chapter 3. Additionally, none of the existing results configure their Lloyd's-based models consistently (as shown in Table 4.1).

Therefore, we recreate this baseline under our methodology and model configuration. We believe that our results isolate and showcase each model for its merits rather than secondary factors, such as the influence of the initialisation strategy or convergence criteria on the results. Under our methodology, we find that the ability of some models have been overestimated and that the performance of models considered state-of-the-art are not significantly better than traditional Euclidean *k*-means.

## 4.5 Experiment Setup

As previously outlined, one of the primary goals of this thesis is to ensure clear and reproducible experimentation for TSCL. To achieve this, we will begin by detailing the methodology for our baseline experiments.

First, we will explicitly define the configuration of our Lloyd's algorithm, presented in clear pseudocode that incorporates all the configuration options previously discussed. Following this, we will establish a baseline using current models from the literature. This will involve providing detailed, model-specific parameterisation, along with justification for each choice. Once this baseline has been established, we will proceed with our experiments involving *k*-means clustering using elastic distances.

## 4.6 Configuration

Algorithm 26 defines our Lloyd's algorithm, which utilises Forgy initialisation with restarts, a maximum iteration stopping condition, an inertia-based early convergence criterion, and a mechanism for handling empty clusters.

Using our Lloyd's configuration, we implemented five of the most commonly used Lloyd's-based algorithms in the literature: $k$-shapes, $k$-SC, $k$-means, $k$-means-ba-DTW, and $k$-means-soft-DBA. Implementations for each of these models are available in the *aeon* repository.

We refer to $k$-means-DBA as $k$-means-ba-DTW for clarity, as in Chapter 7, we propose new barycentre averaging techniques under different distance measures. The $k$-means-soft-DBA retains the DBA in its name because we do not experiment further with this type of barycentre averaging, which uses a gradient descent method with a differentiable distance measure.

Each model shares Lloyd's-specific parameters. These parameters are detailed in Table 4.5.

| | $max\_iters$ | $tol$ | $n\_init$ | $init\_algo$ | $distance$ | $averaging$ |
|---|---|---|---|---|---|---|
| $k$-means-Euclidean | 50 | $1 \times 10^{-6}$ | 10 | Forgy | Euclidean | Arithmetic mean |
| $k$-shapes | 50 | $1 \times 10^{-6}$ | 10 | Forgy | SBD | Shape extraction |
| $k$-means-ba-DTW | 50 | $1 \times 10^{-6}$ | 10 | Forgy | DTW | DBA |
| $k$-SC | 50 | $1 \times 10^{-6}$ | 10 | Forgy | $k$-SC distance | $k$-SC average |
| $k$-means-soft-DBA | 50 | $1 \times 10^{-6}$ | 10 | Forgy | soft-DTW | soft-DBA |

Table 4.5 Baseline Lloyd's-based models parameters

In Table 4.5, five of Lloyd's-specific parameters are kept constant (control variables). The justification for each parameter choice has been outlined previously in this chapter. The independent variable for this baseline experiment is, therefore, the distance and averaging technique used.

In addition to Lloyd's-specific parameters, some models require additional parameters for their distance functions. These distance-specific parameters are summarised in Table 4.6.

The $k$-SC distance measure includes a parameter, $max\_shift$, which is an integer ranging from 0 to $m$, where $m$ is the length of the time series. This parameter controls the shifts that $k$-SC can perform to find the best position to "align" two time series. We set $max\_shift$ to $m$ to allow $k$-SC to find the optimal alignment across all possible shifts for each time series considered.

| Acronym | Metric | Parameters | Default |
|---|---|---|---|
| SBD (Equation 2.42) | No | - | - |
| $k$-SC distance (Equation 2.37) | Yes | $max\_shift \in [0,\dots,m]$ | $max\_shift = m$ |
| DTW (Equation 2.4) | No | $w \in [0,\dots,1]$ | $w = 1.0$ |
| Euclidean distance (Equation 2.2) | Yes | - | - |
| soft-DTW (Equation 2.17) | No | $\gamma \in [0,\dots,\infty]$ | $\gamma = 1.0$ |

Table 4.6 Baseline Lloyd's-based models distance parameters.

soft-DTW takes a parameter $\gamma$ which controls the smoothness of the gradient. $\gamma$ is challenging to set because small changes significantly impact results. [21] experimented with four values of $\gamma$: $\{1.0, 0.1, 0.01, 0.001\}$. They found that smaller values of $\gamma$ often lead to barycentres getting stuck in bad local minima. However, their results also demonstrated that for some datasets, better results could be obtained with lower values of $\gamma$ (i.e., 0.01 and 0.001). Ultimately, they concluded, however, that in the average case, it was better to use a higher value of 1.0, as it consistently converged to "reasonable" solutions. We, therefore, opt to use a value of 1.0 to optimise for the average case.

DTW can use a window parameter $w$. For now, we set $w = 1.0$, meaning a full window will be used. This is done because there has been no experimentation with the window parameter in the context of TSCL.

Finally, the defined averaging techniques can also take additional parameters. For example, $k$-means-DBA uses DTW, which can be parameterised with a window. Similarly, $k$-SC averaging takes a $max\_shift$ parameter and $k$-means-soft-dba averaging takes a $\gamma$ parameter. Throughout all of our experiments in this thesis, unless explicitly stated otherwise, the same parameters specified for the distance computation are also applied in the averaging computations.

**Algorithm 26:** Lloyd's Algorithm with all of our configurations (**X**, **k**, **max_iters**, **tol**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       max_iters (Maximum number of iterations before forced
       termination), tol (Inertia variation threshold)
Output: Assignment of each time series to a cluster
1  $best\_inertia \leftarrow \infty$
2  Let $best\_assignments$ be an empty array of length $n$
3  Let $best\_centres$ be an empty array of length $n$
4  for $i \leftarrow 1$ to 10 do
5      Let $centres$ be an array of $k$ randomly chosen time series from dataset X
6      Let $assignments$ be an empty array of length $n$
7      Let $prev\_inertia \leftarrow \infty$
8      for $j \leftarrow 1$ to $max\_iters$ do
9          for each time series $x_i$ in X do
10             Compute the distance between $x_i$ and each of the $k$ centres
11             Assign $x_i$ to the nearest centre
12         if any cluster has no assignments then
13             repeat
14                 for each cluster $c_j$ in $centres$ do
15                     if cluster $c_j$ has no assignments then
16                         Set $best\_candidate$ to the time series that reduces
                             inertia the most and is not currently a centroid
17                         $c_j \leftarrow best\_candidate$
18                         Recompute cluster assignments
19             until every cluster has at least one assignment;
20         for each centre $c_j$ in $centres$ do
21             Update $c_j$ to be the mean of all time series assigned to it
22         Let $curr\_inner\_inertia$ be the SSE of the current clustering
23         if $|curr\_inner\_inertia - prev\_inertia| < tol$ then
24             break
25         $prev\_inertia \leftarrow curr\_inner\_inertia$
26     Let $curr\_inertia$ be the SSE of the current clustering
27     if $curr\_inertia < best\_inertia$ then
28         $best\_inertia \leftarrow curr\_inertia$
29         $best\_assignments \leftarrow assignments$
30         $best\_centres \leftarrow centres$
31 return $best\_assignments$
```

## 4.7 Result

With the configuration and model definitions for our baseline experiment now established, we proceed with the experimentation and analysis of results. Results have been divided into two categories: combined test-train split and test-train split. Our results will now be presented.

### 4.7.1 Combined test-train split results

Figure 4.12 presents the critical difference diagrams for our baseline models across four clustering metrics for the combined test-train split. We observe that *k*-means-soft-DBA significantly outperforms all other baseline Lloyd's clusterers across all evaluation metrics. However, due to the extended runtime of *k*-means-soft-DBA, we were only able to obtain results from 75 datasets, as the runtime for 27 datasets exceeded our maximum seven-day limit for *k*-means-soft-DBA. For the datasets that did complete, the FitTime comparison is shown in Figure 4.13. The figure demonstrates that *k*-means-soft-DBA requires significantly more computational time than the other clusterers.

Given the substantial number of missing results for *k*-means-soft-DBA, our analysis is constrained. As a result, we have decided to exclude *k*-means-soft-DBA from the general Lloyd's baseline analysis and only reintroduce it in the final analysis to contextualise our findings. We will start by specifically analysing *k*-means-soft-DBA across the 75 datasets for which results are available. Following this, we will exclude *k*-means-soft-DBA from our baseline and evaluate the remaining models across the entire UCR archive.

In Table 4.7, *k*-means-soft-DBA, on average, outperforms all other clusterers across all metrics considered, and it does so by a significant margin. However, when we divide the 75 datasets by problem domain, we find that, it is not the best clusterer in every domain. Table 4.8 shows the ARI by problem domain. Interestingly, *k*-

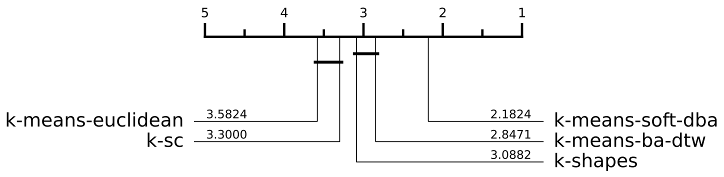


Fig. 4.8 AMI

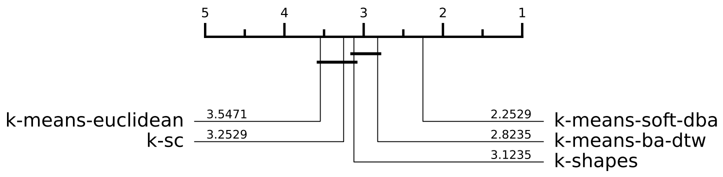


Fig. 4.9 ARI

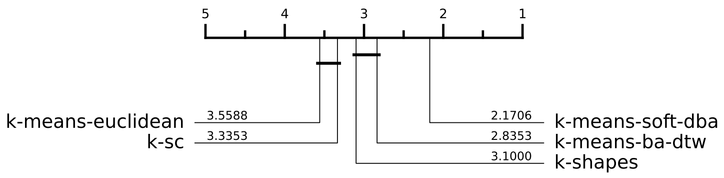


Fig. 4.10 NMI

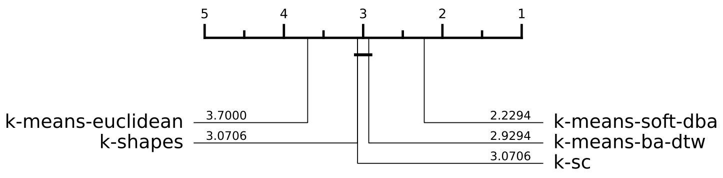


Fig. 4.11 CLACC

Fig. 4.12 CD diagrams of Lloyd's-based algorithm over 75 datasets from the UCR archive using the combined test-train split. The excluded datasets are detailed in Appendix A, Table A.2. The reason for the exclusion is due to *k*-means-soft-DBA being unable to finish within our seven day runtime limit.

means-soft-DBA is the best in only four of the seven categories, notably struggling with ECG and Spectro data.

While previous studies have observed that soft-DBA can outperform methods such as *k*-shapes or *k*-means-ba-DTW, we have not found any extensive analysis in the literature highlighting the significant disparity between *k*-means-soft-DBA and other traditionally considered state-of-the-art approaches. Overall, we find that although computationally expensive, *k*-means-soft-DBA is by far the most effective TSCL approach over the combined test-train split using 75 datasets.

We now exclude *k*-means-soft-DBA from our baseline experiment to provide a more accurate baseline representation across the UCR archive. Figure 4.18 presents the critical difference diagrams for our Lloyd's baseline experiment. We include results from 106 of the 112 datasets. 5 datasets are excluded due to *k*-means-ba-DTW's computational complexity preventing it from completing within our seven-day runtime limit (detailed in Table A.4). Figure 4.19 shows the significant

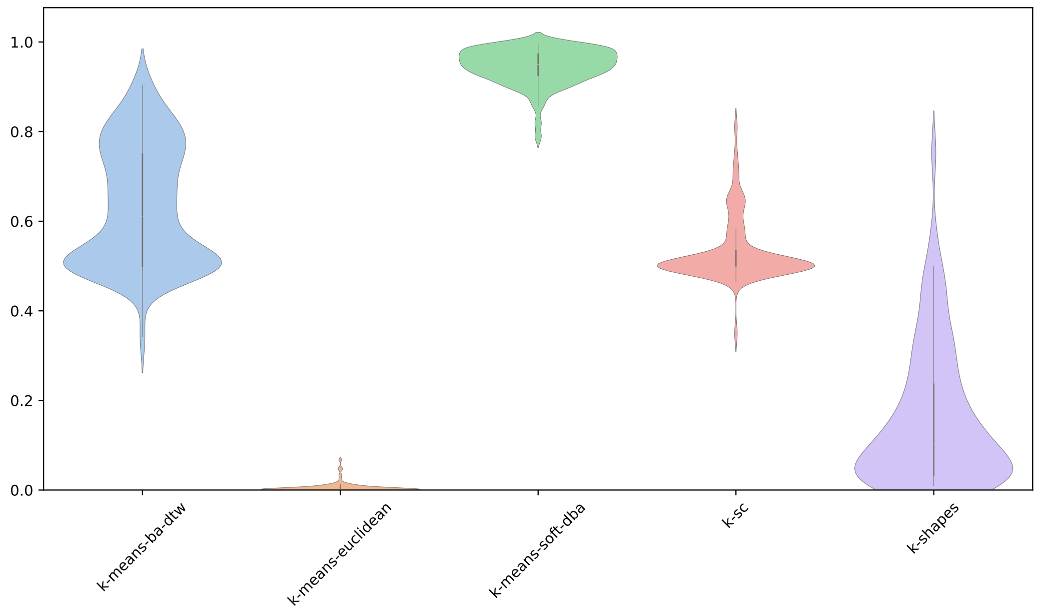


Fig. 4.13 Relative FitTime violin plot of baseline Lloyd's clusterers over 75 datasets from the UCR archive using the combined test-train split.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-DTW | 0.265 | 0.310 | 0.593 | 0.323 | 0.698 |
| k-means-euclidean | 0.204 | 0.251 | 0.538 | 0.265 | 0.675 |
| k-means-soft-dba | **0.305** | **0.346** | **0.623** | **0.357** | **0.714** |
| k-sc | 0.217 | 0.252 | 0.557 | 0.266 | 0.638 |
| k-shapes | 0.239 | 0.292 | 0.575 | 0.304 | 0.688 |

Table 4.7 Lloyd's baseline experiment with *k*-means-soft-DBA average scores across multiple evaluation metrics over 75 datasets from the UCR archive using the combined test-train split.

computational requirement of *k*-means-ba-DTW compared to our other baseline models.

Therefore, while our baseline combined test-train split analysis is missing 5 datasets, the inclusion of 106 datasets is sufficient to draw meaningful conclusions. Additionally, in our separate test-train split analysis, all 112 datasets are included, which helps to address the gap in dataset coverage.

Across all the clustering evaluation metrics considered, *k*-means-ba-DTW outperforms the other Lloyd's-based clusterers, with the exception of AMI, where

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-DTW | 0.380 | 0.289 | 0.246 | **0.557** | 0.245 | 0.207 | 0.140 |
| k-means-euclidean | 0.321 | 0.323 | 0.252 | 0.261 | 0.106 | 0.140 | 0.159 |
| k-means-soft-dba | **0.419** | 0.311 | **0.327** | 0.524 | **0.292** | **0.212** | 0.181 |
| k-sc | 0.299 | **0.339** | 0.318 | 0.098 | 0.073 | 0.137 | 0.355 |
| k-shapes | 0.339 | 0.289 | 0.250 | 0.464 | 0.159 | 0.195 | **0.364** |

Table 4.8 Average ARI score on problems split by problem domain over 75 datasets from the UCR archive using the combined test-train split.

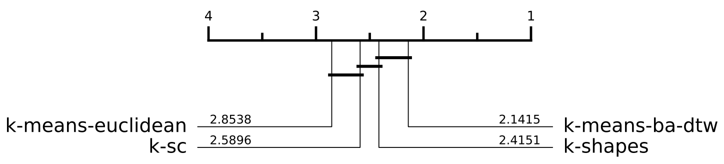


Fig. 4.14 AMI

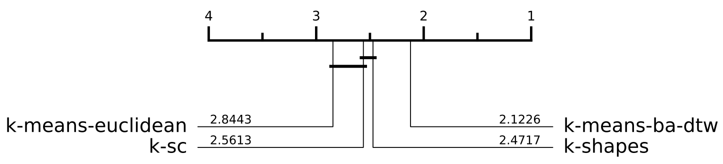


Fig. 4.15 ARI

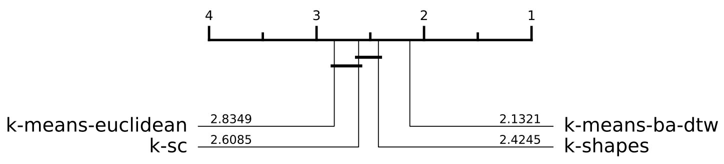


Fig. 4.16 NMI

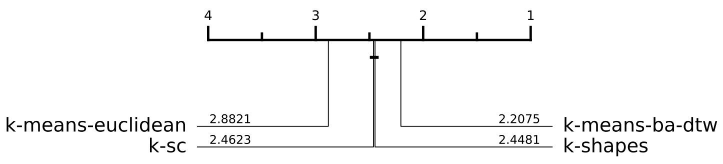


Fig. 4.17 CLACC

Fig. 4.18 CD diagrams of Lloyd's-based algorithm over 106 datasets from the UCR archive using the combined test-train split. The excluded datasets are detailed in Appendix A, Table A.4. The reason for the exclusion is due to *k*-means-ba-DTW being unable to finish within our seven day runtime limit.

*k*-shapes is not significantly different from *k*-means-ba-DTW (shown in Figure 4.18). The specific average scores for each metric are provided in Table 4.9. While *k*-means-ba-DTW is, on average, the best-performing clusterer in our baseline, an examination of the results by problem domain for ARI and AMI reveals that *k*-sc outperforms *k*-means-ba-DTW in three of the seven categories as shown in Tables 4.10 and 4.11. Specifically, *k*-sc significantly outperforms *k*-means-ba-DTW in the ECG domain and surpasses all other clusterers in the Spectro domain. However,

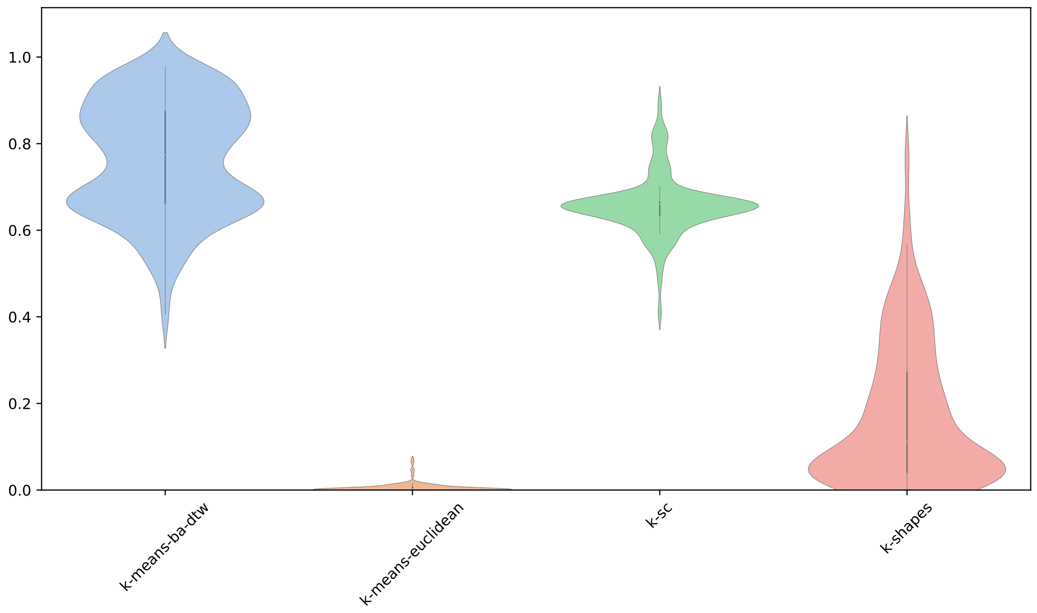


Fig. 4.19 Relative FitTime violin plot of baseline Lloyd's clusterers over 106 datasets from the UCR archive using the combined test-train split.

*k*-sc's overall average performance is diminished by its particularly poor results in the Device domain, where it achieved only slightly better than random clustering.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-DTW | **0.254** | **0.302** | **0.569** | **0.326** | **0.711** |
| k-means-euclidean | 0.200 | 0.250 | 0.521 | 0.276 | 0.692 |
| k-sc | 0.213 | 0.257 | 0.538 | 0.278 | 0.654 |
| k-shapes | 0.230 | 0.289 | 0.552 | 0.311 | 0.702 |

Table 4.9 Lloyd's baseline experiment average scores across multiple evaluation metrics over 106 datasets from the UCR archive using the combined test-train split.

On average, *k*-shapes outperforms *k*-sc; however, it does not achieve the best performance in any specific problem domain. Despite this, *k*-shapes performs consistently across all problem domains, ranking second-best in six out of the seven domains. This consistent performance is illustrated in Figure 4.20. While *k*-shapes does not attain the highest ARI scores, it has the highest median score and as result a much more consistent distribution than the other clusterers. Additionally, as shown in Figure 4.19, *k*-shapes is significantly faster in runtime than *k*-SC and *k*-means-ba-dtw.

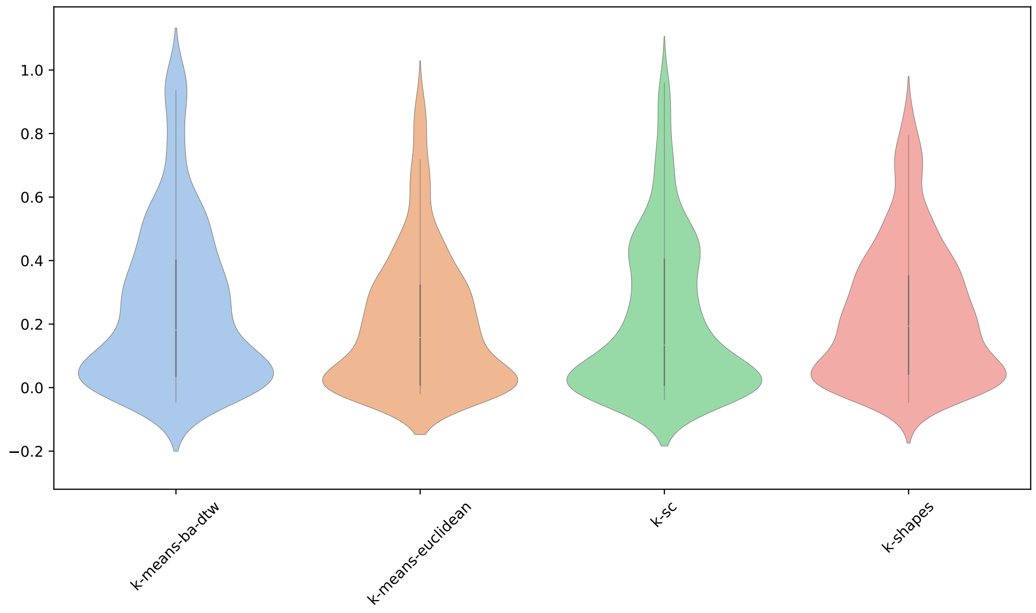


Fig. 4.20 ARI of baseline Lloyd's clusterers over 106 datasets from the UCR archive using the combined test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-DTW | **0.307** | 0.209 | 0.195 | **0.586** | **0.173** | **0.164** | 0.246 |
| k-means-euclidean | 0.250 | 0.219 | 0.191 | 0.306 | 0.052 | 0.104 | 0.274 |
| k-sc | 0.241 | **0.229** | **0.268** | 0.187 | 0.044 | 0.080 | **0.422** |
| k-shapes | 0.267 | 0.183 | 0.198 | 0.429 | 0.102 | 0.152 | 0.405 |

Table 4.10 Lloyd's baseline experiment average ARI score on problems split by problem domain over 106 datasets from the UCR archive using the combined test-train split.

While our three TSCL-specific Lloyd's algorithms outperform the traditional *k*-means-Euclidean clusterer on average, the raw value improvements are relatively small. Table 4.12 shows the average raw value difference between each baseline clusterer and *k*-means-Euclidean. For ARI, *k*-means-ba-DTW is, on average, 5.4% better, *k*-sc is 1.3% better, and *k*-shapes is 3% better than *k*-means-Euclidean. Similar improvements are observed across the other considered metrics. Notably,

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-DTW | **0.384** | 0.238 | 0.221 | **0.606** | **0.193** | **0.220** | 0.337 |
| k-means-euclidean | 0.327 | 0.263 | 0.221 | 0.344 | 0.076 | 0.153 | 0.348 |
| k-sc | 0.304 | **0.272** | **0.301** | 0.202 | 0.066 | 0.127 | **0.482** |
| k-shapes | 0.347 | 0.236 | 0.239 | 0.514 | 0.135 | 0.203 | 0.471 |

Table 4.11 Lloyd's baseline experiment average AMI score on problems split by problem domain over 106 datasets from the UCR archive using the combined test-train split.

*k*-shapes performs 3.9% better for AMI and 3.5% better for NMI. The RI metric shows significantly smaller improvements, which, as discussed in Section 3.2.1, is due to RI not adequately accounting for random chance.

Overall, this baseline highlights the challenges of TSCL and helps set realistic expectations. In contrast, the current state-of-the-art model in TSC, HIVE-COTE 2.0 [83], achieves an average accuracy of 89.14% [84] across 112 datasets from the UCR archive, compared to the 1*NN*-Euclidean baseline, which achieves 68.62% accuracy [23]. This results in a classification accuracy difference of 20.52%. From our baseline experiment, it is evident that TSCL has yet to achieve this level of improvement over traditional approaches. Therefore, it is crucial to contextualise TSCL results in comparison to related fields such as TSC.

Our combined test-train split will serve as our primary evaluation baseline; however, we will also consider the test-train split. The baseline results for the test-train split will be outlined next.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-DTW | **0.054** | **0.052** | **0.048** | **0.050** | **0.019** |
| k-sc | 0.013 | 0.007 | 0.017 | 0.002 | -0.038 |
| k-shapes | 0.030 | 0.039 | 0.031 | 0.035 | 0.010 |

Table 4.12 Difference in performance between each clusterer and *k*-means-Euclidean across multiple evaluation metrics over 106 datasets from the UCR archive using the combined test-train split.

### 4.7.2 Test-train split results

Similar to our combined test-train results, we will begin our evaluation by including *k*-means-soft-DBA. Although more datasets completed within the seven-day runtime limit, for consistency, we will also exclude *k*-means-soft-DBA from our test-train baseline experiment after the initial evaluation, reintroducing it only for contextualisation.

Figure 4.25 shows the critical difference diagrams for four clustering metrics across 104 UCR datasets. As with the combined test-train split critical diagrams in Figure 4.12, *k*-means-soft-DBA significantly outperforms the other clusterers. However, similar to the combined test-train split results, we observe that while *k*-means-soft-DBA outperforms each clusterer on average, it is not the best in every domain. Table 4.13 shows the ARI performance by problem domain, where *k*-means-soft-DBA performs best in four of the seven categories. *k*-means-soft-DBA particularly struggle with the Spectro and ECG domain. In the Spectro domain *k*-means-soft-DBA is outperformed by *k*-means-Euclidean and *k*-means-ba-DTW on average for ARI. This was also the case in the combined test-train split further highlighting a weakness in of *k*-means-soft-ba.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-DTW | 0.355 | 0.235 | 0.231 | 0.515 | 0.192 | 0.212 | 0.128 |
| k-means-euclidean | 0.301 | **0.242** | 0.210 | 0.332 | 0.058 | 0.163 | 0.146 |
| k-means-soft-dba | **0.400** | 0.227 | 0.232 | **0.561** | **0.232** | **0.238** | 0.162 |
| k-sc | 0.308 | 0.222 | **0.238** | 0.298 | 0.064 | 0.144 | **0.335** |
| k-shapes | 0.229 | 0.212 | 0.191 | 0.176 | 0.109 | 0.146 | 0.056 |

Table 4.13 Average ARI score on problems split by problem domain over 104 datasets from the UCR archive using the test-train split.

To maintain consistency with our combined test-train split evaluation, we now exclude *k*-means-soft-DBA from our baseline analysis. Figure 4.30 presents the critical difference diagrams for our baseline experiments across 112 UCR archive

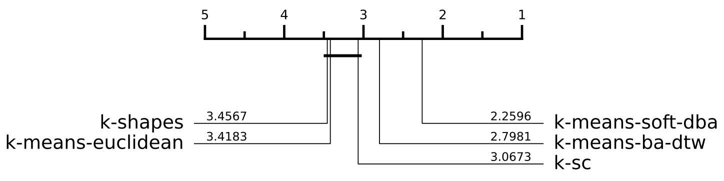


Fig. 4.21 AMI

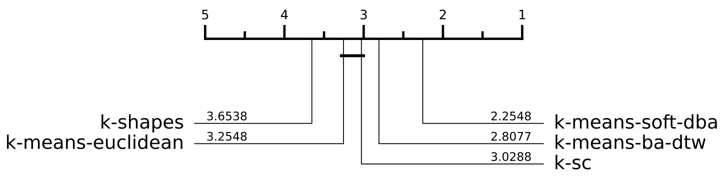


Fig. 4.22 ARI

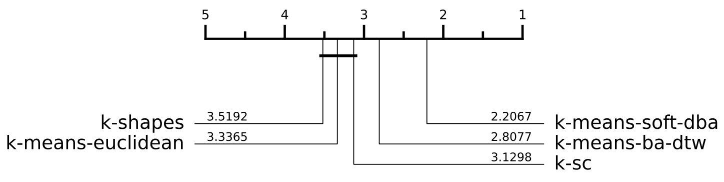


Fig. 4.23 NMI

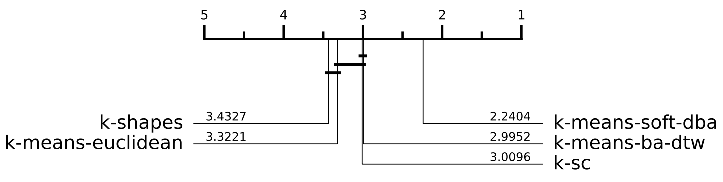


Fig. 4.24 CLACC

Fig. 4.25 CD diagrams of Lloyd's-based algorithm over 104 datasets from the UCR archive using the test-train split. The excluded datasets are detailed in Appendix A, Table A.3. The reason for the exclusion is due to *k*-means-ba-DTW being unable to finish within our seven day runtime limit.

datasets using the test-train split. The figure shows that *k*-means-ba-DTW remains the best-performing clusterer, though by a smaller margin. However, for CLACC, *k*-means-ba-DTW is not significantly different from *k*-sc and *k*-means-euclidean.

Another notable observation from Figure 4.30 is that *k*-shapes performs particularly poorly, with the worst average rank across all four clustering metrics. This is surprising, as *k*-shapes performed well on average in the combined test-train split experiments. This discrepancy potentially highlights a weakness of *k*-shapes: it may struggle to learn robust general representations of the data, leading to poorer performance on new, unseen data.

The clustering scores for each clusterer are presented in Table 4.16. Compared to *k*-means-euclidean, the performance increase is lower than what was observed in the combined test-train split. This is highlighted in Table 4.15. For the test-train split, *k*-means-ba-DTW shows an average improvement of approximately 4% over *k*-means-euclidean (excluding RI). The performance of *k*-sc remains consistent with

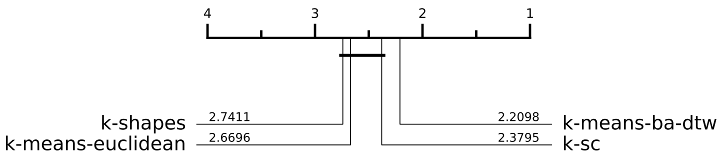


Fig. 4.26 AMI

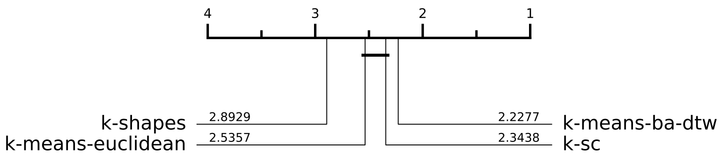


Fig. 4.27 ARI

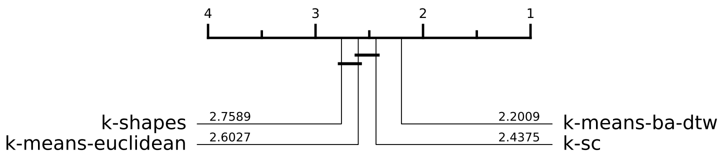


Fig. 4.28 NMI

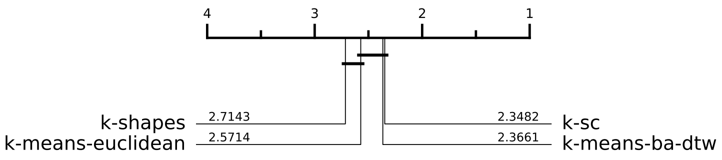


Fig. 4.29 CLACC

Fig. 4.30 CD diagrams of Lloyd's-based algorithm over 112 datasets from the UCR archive using the combined test-train split.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-DTW | **0.226** | **0.281** | **0.553** | **0.315** | **0.701** |
| k-means-euclidean | 0.185 | 0.235 | 0.521 | 0.271 | 0.686 |
| k-sc | 0.194 | 0.244 | 0.534 | 0.277 | 0.672 |
| k-shapes | 0.121 | 0.191 | 0.485 | 0.223 | 0.608 |

Table 4.14 Lloyd's baseline experiment average ARI score on problems split by problem domain over 112 datasets from the UCR archive using the test-train split.

the results from the combined test-train split, while *k*-shapes, as noted, performs significantly worse than in the combined test-train baseline experiment.

When assessing the results by problem domain, *k*-means-ba-DTW performs best in the same four domains. Additionally, it ties with *k*-sc for the best performance in the Sensor domain. Comparing *k*-means-ba-DTW's values across domains with those in Table 4.12, we find that for the Spectro, Sensor, Device, Motion, and ECG domains, *k*-means-ba-DTW achieves similar ARI scores to those in the combined test-train results. However, *k*-means-ba-DTW performs significantly worse in the Simulated and Image domains. Notably, in the Simulated domain, *k*-means-ba-DTW's ARI is reduced by over 1%.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-DTW | **0.041** | **0.046** | **0.032** | **0.044** | **0.015** |
| k-sc | 0.009 | 0.009 | 0.013 | 0.006 | -0.014 |
| k-shapes | -0.064 | -0.044 | -0.036 | -0.048 | -0.078 |

Table 4.15 Performance difference against *k*-means-euclidean across multiple evaluation metrics over 112 datasets from the UCR archive using the test-train split. The raw value increase (or decrease) is presented in each cell.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-DTW | **0.265** | 0.191 | **0.181** | **0.454** | **0.175** | **0.169** | 0.239 |
| k-means-euclidean | 0.222 | **0.207** | 0.164 | 0.274 | 0.040 | 0.140 | 0.260 |
| k-sc | 0.227 | 0.185 | **0.181** | 0.250 | 0.046 | 0.138 | **0.404** |
| k-shapes | 0.134 | 0.167 | 0.122 | 0.104 | 0.073 | 0.093 | 0.125 |

Table 4.16 Lloyd's baseline experiment average ARI score on problems split by problem domain over 112 datasets from the UCR archive using the test-train split.

Overall, our results suggest that *k*-means-ba-DTW learns more robust and meaningful representations of the data than the other baseline methods. In contrast, *k*-shapes struggles to obtain representations that generalise well to unseen data. This is likely due to the use of cross-correlation, which can extract effective representations for known data but does not translate well to new, unseen data.

## 4.8 Conclusion

We have now completed a baseline experiment across five Lloyd's-based clusterers commonly used in the TSCL literature. Our assessment has highlighted the strengths and weaknesses of each clusterer, with a focus on their relative performance improvements compared to *k*-means-euclidean. While our findings align with existing literature, we believe our experiments provide new insights into these clusterers and more accurately presented their relative performance.

Our findings show that *k*-means-soft-DBA significantly outperforms the other baseline clusterers. Although it is known that *k*-means-soft-DBA can perform better than many other clusterers, its superiority has not been thoroughly demonstrated through a robust methodology that clearly establishes its significant advantage over other TSCL approaches in both the combined test-train split and the test-train split. However, this performance comes at the cost of computational time. Due to its high computational demands, we were unable to complete a full benchmark using *k*-means-soft-DBA, which led to its exclusion from our baseline experiments.

Excluding *k*-means-soft-DBA, we find that *k*-means-ba-DTW is the best-performing clusterer, followed by *k*-shapes for the combined test-train split. However, for the test-train split, while *k*-means-ba-DTW remains the top-performing baseline clusterer, *k*-shapes' performance significantly degrades, suggesting that it struggles to learn generalised representations of the data. Overall, our best baseline clusterer shows an average improvement of around 5% over *k*-means-euclidean in the combined test-train split, and an average improvement of around 4% in the test-train split. With this baseline established using a robust methodology and a consistently parameterised Lloyd's algorithm, we will now begin exploring the use of elastic distances for *k*-means.

# Chapter 5

# $k$-means clustering using elastic distances

## Contributing Publications

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

A robust benchmark has been established which we can evaluate our results against. We will now begin experimenting with elastic distances. Our first experiment will consider the popular $k$-means clusterer. We will try $k$-means with 12 different elastic distances and compare the results to each other and our baseline clusterers. We aim to keep our initial experiments as simple as possible only altering the distance measurement in the assignment phase.

As such we have set broad expectations. Our hypothesis is that substituting distance measures without altering the averaging methods will not yield better results than the current state-of-the-art established in our baseline. However, we expect many of the distances to outperform $k$-means-euclidean. We expect that our results and the ordering of elastic distances to be similar to that of the 1-NN classifier in TSC.

## 5.1 Experiment Setup

For our initial experimentation we will swap out the traditional Euclidean distance for an elastic distance in the *k*-means clusterer. We acknowledge that considering a distance measure in isolation as a parameter without regard for the averaging technique means there is no guarantee of convergence under *k*-means, however, we hypothesise that *k*-means will still converge sufficiently for many datasets. Additionally, we expect our experiment to offer an initial overview of which elastic distances perform best for TSCL. Furthermore, when we introduce a new elastic averaging technique in Chapter 7 for use in *k*-means with elastic distances, comparing the results to these initial experiments will showcase the improvements of our new averaging technique.

## 5.2 Configuration

We conduct our experiment using the *k*-means clusterer, employing the same Lloyd's algorithm (Algorithm 26) as used in our baseline experiment. The distance measure is our independent variable, while all other parameters remain constant. Table 5.1 outlines the specific parameters used for each model.

Each elastic distance has its own set of parameters, which are detailed in Section 2.4. Our goal is not to fine-tune these elastic distances to maximise performance across the UCR archive. Instead, we aim to establish sensible default parameters that serve as a starting point for practitioners to achieve effective clustering. Our initial default parameters are outlined in Table 5.2.

| | *max_iters* | *tol* | *n_init* | *init_algo* | *distance* | *averaging* |
|---|---|---|---|---|---|---|
| $k$-means-adtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | ADTW | Arithmetic mean |
| $k$-means-ddtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | DDTW | Arithmetic mean |
| $k$-means-dtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | DTW | Arithmetic mean |
| $k$-means-edr | 50 | $1 \times 10^{-6}$ | 10 | Forgy | EDR | Arithmetic mean |
| $k$-means-erp | 50 | $1 \times 10^{-6}$ | 10 | Forgy | ERP | Arithmetic mean |
| $k$-means-lcss | 50 | $1 \times 10^{-6}$ | 10 | Forgy | LCSS | Arithmetic mean |
| $k$-means-msm | 50 | $1 \times 10^{-6}$ | 10 | Forgy | MSM | Arithmetic mean |
| $k$-means-twe | 50 | $1 \times 10^{-6}$ | 10 | Forgy | TWE | Arithmetic mean |
| $k$-means-wddtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | WDDTW | Arithmetic mean |
| $k$-means-wdtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | WDTW | Arithmetic mean |
| $k$-means-shape-dtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | shape-DTW | Arithmetic mean |
| $k$-means-soft-dtw | 50 | $1 \times 10^{-6}$ | 10 | Forgy | soft-DTW | Arithmetic mean |

Table 5.1 Elastic distance $k$-means model parameters

These default parameters were selected based on recommendations from the TSC literature [111, 76]. In the TSC literature, many elastic distance parameters are suggested to be tuned on a per-dataset basis. For example, [111] recommends selecting the default value for TWE's $\nu$ from an exponentially growing sequence, $\{10^{-5}, 5 \times 10^{-5}, 10^{-4}, 5 \times 10^{-4}, 10^{-3}, 5 \times 10^{-3}, \ldots, 1\}$, resulting in 100 possible parameterisations for each dataset. However, in clustering tasks, evaluating 100 different parameterisations is impractical. As outlined in Chapter 3, tuning in clustering is often infeasible for practitioners due to the lack of labels in real-world scenarios. Therefore, for parameters that require tuning in the TSC literature, we rely on the suggested single-value defaults provided in the original papers.

If the performance of a distance measure differs significantly from our expectations—based on its performance in the TSC literature—we may perform additional tuning. This could either demonstrate that tuning does not significantly impact the distance measure's effectiveness in TSCL or prompt a reassessment of the initial defaults.

| Acronym | Metric | Parameters | Default |
|---|---|---|---|
| ADTW (Equation 2.12) | No | $\omega \in [0,\dots,\infty]$ | $\omega = 1.0$ |
| DTW (Equation 2.4) | No | $w \in [0,\dots,1]$ | $w = 1.0$ |
| DDTW (Equation 2.6) | No | $w \in [0,\dots,1]$ | $w = 1.0$ |
| WDTW (Equation 2.9) | No | $g \in [0,\dots,\infty]$ | $g = 0.05$ |
| WDDTW (Equation 2.10) | No | $g \in [0,\dots,\infty]$ | $g = 0.05$ |
| LCSS (Equation 2.19) | No | $\varepsilon \in [0,\dots,\infty]$ | $\varepsilon = 1.0$ |
| ERP (Equation 2.26) | Yes | $g \in [0,\dots,\infty]$, | $g = \sigma(X)$ |
| EDR (Equation 2.22) | No | $\varepsilon \in [0,\dots,\infty]$ | $\varepsilon = \frac{1}{4}\sigma(X)$ |
| MSM (Equation 2.29) | Yes | $c \in [0,\dots,\infty]$ | $c = 1$ |
| TWE (Equation 2.31) | Yes | $\nu, \lambda \in [0,\dots,\infty]$ | $\nu = 0.001, \lambda = 1$ |
| shape-DTW | No | $reach \in [0,\dots,\infty]$ | $reach = 30$ |
| soft-DTW | No | $\gamma \in [0,\dots,\infty]$ | $\gamma = 1.0$ |

Table 5.2 Summary of elastic distance functions, and our initial parameters

## 5.3 Results

Our analysis is divided into two sections: the combined test-train split results and the test-train results. After analysing the combined test-train and the test-train splits individually, we will investigate unexpected results in our experiments, ultimately concluding with a summary of our findings.

As outlined previously, since we are not updating the averaging method to minimise each distance, there is no guarantee of convergence. This has led to some unexpected outcomes. Specifically, we observed that some clusterers are prone to repeatedly forming empty clusters. Although we apply a mitigation strategy (outlined in Section 4.6), we have set a limit on how many attempts can be made to resolve them (to prevent infinite loops). If empty clusters persist, the algorithm is terminated, and no results are returned for that set of initial centroids. If empty clusters form across all ten reruns with different initial centroids, no results can be obtained for that experiment, leading to missing results. Therefore, for our initial $k$-means experiments we have a number of missing results. A full list of all the missing results for each model is provided in Table A.5.

Repeated empty cluster formation can also occur in traditional *k*-means using Euclidean distance, but it is typically rare and often indicates an incorrect number of clusters is set. We believe that, because our distance measures are not minimised under the arithmetic mean and an arbitrary number of clusters is set, this issue is exacerbated. This likely explains why distances that diverge the most from Euclidean distance (e.g., LCSS and shape-DTW) exhibit more frequent occurrences of empty cluster formation.

However, we believe enough datasets across all models completed to draw meaningful conclusions from. We do find however, for the combined test-train split some distances in particular failed on significantly more datasets than other. As such we will begin by including these distances in the analysis but then exclude the distances with a large number of missing datasets in our later analysis to draw more meaningful conclusion for the clusterers that did obtain a full set of results.

### 5.3.1 Combined test-train split results

Figure 5.5 shows the critical difference diagrams for our *k*-means elastic distance experiments for all of the distances considered. We observe that ADTW, shape-DTW, MSM, soft-DTW, TWE, and WDTW consistently appear in the top clique across all evaluation metrics. Conversely, DDTW, EDR, DTW, and LCSS consistently fall into the bottom clique, and notably perform worse, on average, than Euclidean distance.

In the critical difference diagrams in Figure 5.5, while ADTW consistently achieves the top rank, MSM follows closely, ranking second in three out of the four metrics. However, if we examine the raw metric values across the 78 datasets, shape-DTW performs best in four of the five evaluation metrics, as shown in Table 5.3.

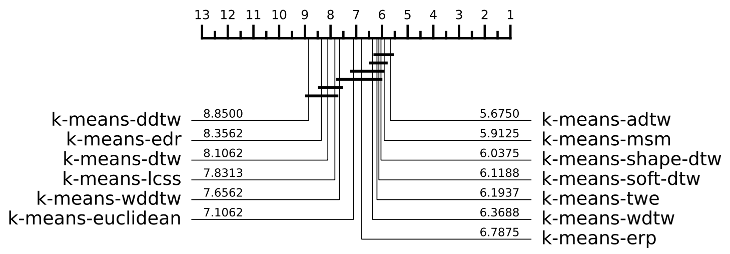


Fig. 5.1 AMI

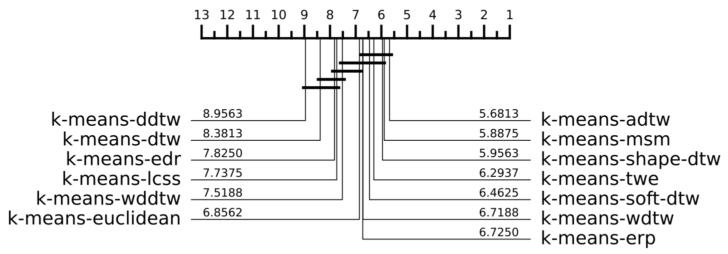


Fig. 5.2 ARI

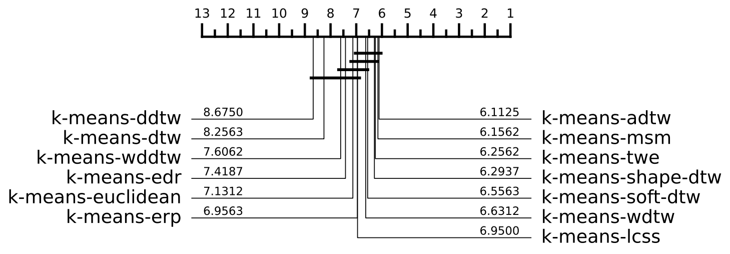


Fig. 5.3 CLACC

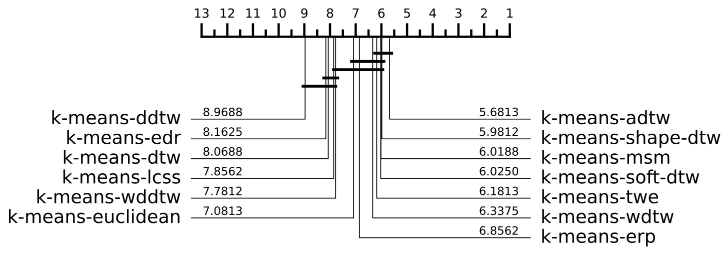


Fig. 5.4 NMI

Fig. 5.5 CD diagrams for k-means with 13 distances over 78 datasets from the UCR archive using the combined test-train split.

To understand why shape-DTW achieves the highest average values but not the top rank across evaluation metrics, we can examine the results by problem domain. Table 5.3 presents the average ARI value by problem domain for each clusterer. In some domains, such as simulated and ECG, shape-DTW significantly outperforms the other distances. For example, in the ECG domain, shape-DTW surpasses the next best clusterer by 13.6% in terms of ARI. However, in other domains like Devices, shape-DTW ranks among the lowest-performing clusterers.

Although we don't directly use missing datasets as part of our measurable analysis, it is notable that shape-DTW failed to produce results for 22 datasets due to repeated empty cluster formation. This could indicate that shape-DTW struggles with certain types of time series data, which may explain the significant variability in the quality of its results.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-adtw | 0.194 | 0.232 | 0.556 | 0.239 | **0.656** |
| k-means-ddtw | 0.135 | 0.164 | 0.509 | 0.172 | 0.586 |
| k-means-dtw | 0.147 | 0.187 | 0.522 | 0.196 | 0.611 |
| k-means-edr | 0.133 | 0.160 | 0.519 | 0.170 | 0.616 |
| k-means-erp | 0.174 | 0.210 | 0.545 | 0.218 | 0.648 |
| k-means-euclidean | 0.169 | 0.205 | 0.534 | 0.213 | 0.646 |
| k-means-lcss | 0.135 | 0.170 | 0.527 | 0.180 | 0.613 |
| k-means-msm | 0.188 | 0.228 | 0.558 | 0.236 | 0.651 |
| k-means-shape-dtw | **0.204** | **0.243** | **0.563** | **0.251** | 0.650 |
| k-means-soft-dtw | 0.183 | 0.225 | 0.553 | 0.233 | 0.635 |
| k-means-twe | 0.184 | 0.220 | 0.555 | 0.228 | 0.646 |
| k-means-wddtw | 0.158 | 0.191 | 0.534 | 0.199 | 0.631 |
| k-means-wdtw | 0.177 | 0.215 | 0.553 | 0.223 | 0.649 |

Table 5.3 Summary of average score across multiple evaluation metrics for k-means with 13 distances over 78 datasets from the UCR archive using the combine test-train split split.

Soft-DTW consistently appears in the top performing clique for all evaluation metrics and performs well across all problem domains. However, while it performs better than other distances on average, it does not achieve the highest maximum results or the lowest minimum results compared to other top distances like ADTW, shape-DTW, MSM, and TWE. Figure 5.6 illustrates this, showing that soft-DTW's minimum result are higher than most other distances, but its maximum results are lower than the other top performing distances. Overall, this demonstrates that soft-DTW is one of the most consistent performers, and it tends to avoid the extreme highs and lows.

LCSS was one of the worst-performing distances on average, consistently outperformed by the Euclidean distance across all evaluation metrics. Its results are generally only marginally better than random clustering. We observed that when the distance between a centroid generated using the arithmetic mean and a time series is measured using LCSS, a large number of values are considered gaps. We suspect this could be a cause of poor performance.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-adtw | **0.254** | 0.091 | 0.231 | 0.239 | 0.058 | **0.143** | 0.266 |
| k-means-ddtw | 0.148 | **0.128** | 0.224 | 0.108 | 0.046 | 0.044 | 0.212 |
| k-means-dtw | 0.163 | 0.065 | 0.207 | 0.260 | 0.027 | 0.095 | 0.142 |
| k-means-edr | 0.163 | 0.057 | 0.181 | 0.150 | 0.053 | 0.067 | 0.232 |
| k-means-erp | 0.217 | 0.067 | 0.224 | 0.216 | 0.079 | 0.119 | 0.184 |
| k-means-euclidean | 0.208 | 0.094 | 0.214 | 0.221 | 0.052 | 0.125 | 0.174 |
| k-means-lcss | 0.104 | 0.081 | 0.188 | 0.188 | 0.062 | 0.123 | 0.214 |
| k-means-msm | 0.252 | 0.063 | 0.228 | 0.225 | **0.094** | 0.122 | 0.227 |
| k-means-shape-dtw | 0.239 | 0.099 | 0.255 | **0.307** | 0.050 | 0.118 | **0.402** |
| k-means-soft-dtw | 0.199 | 0.104 | 0.228 | 0.270 | 0.080 | 0.138 | 0.221 |
| k-means-twe | 0.214 | 0.071 | **0.261** | 0.236 | 0.079 | 0.107 | 0.220 |
| k-means-wddtw | 0.201 | 0.086 | 0.245 | 0.110 | 0.041 | 0.088 | 0.213 |
| k-means-wdtw | 0.195 | 0.070 | 0.238 | 0.275 | 0.064 | 0.139 | 0.139 |

Table 5.4 Average ARI score on problems split by problem domain for k-means with 13 distances over 78 datasets from the UCR archive using the combine test-train split split.

Finally, before excluding shape-DTW, soft-DTW, and LCSS from the analysis to include more datasets from the UCR archive, we evaluate the FitTime for each distance. Figure 5.7 shows the FitTime for each clusterer. The most noticeable observation is that *k*-means with shape-DTW and soft-DTW takes significantly longer to run relative to the other distances.

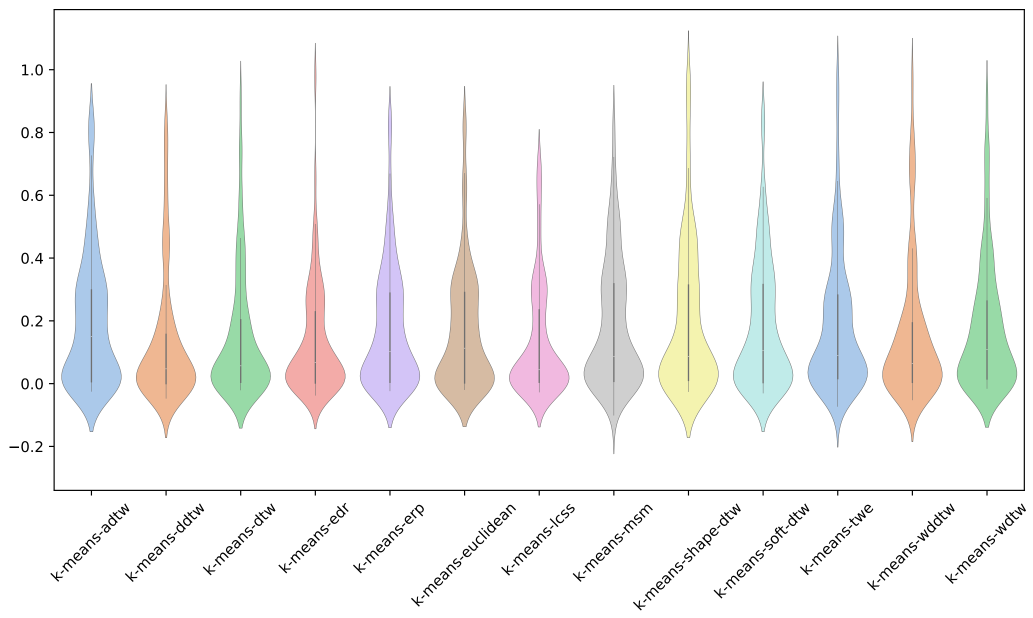


Fig. 5.6 ARI of *k*-means with 13 distances over 80 datasets from the UCR archive using the combined test-train split.

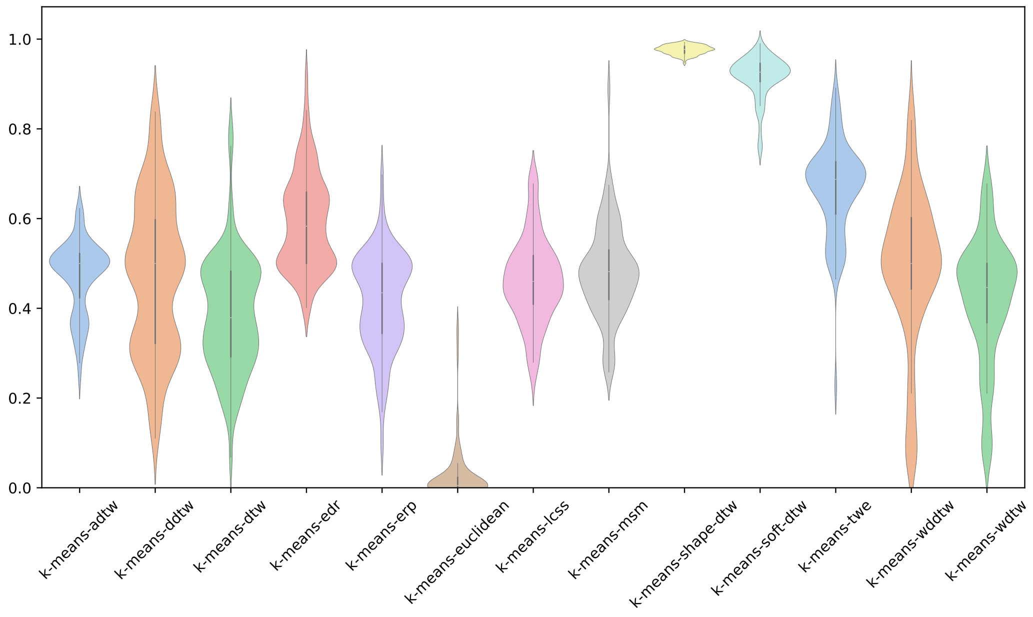


Fig. 5.7 Relative FitTime violin plot of *k*-means with 13 distances over 80 datasets from the UCR archive using the combined test-train split.

We now exclude shape-DTW, soft-DTW, and LCSS from our analysis to include more datasets in the evaluation of the remaining distances. After excluding these three models, all other models have complete results for 103 datasets. The missing datasets are shown in Table A.7, and they are missing due to repeated empty cluster formation across all initial Forgy centroids.

Figure 5.12 shows the critical difference diagram for 103 datasets across 10 different distances. ADTW, MSM, and TWE consistently appear in the top clique. Additionally, for CLAcc, WDTW and ERP also appear in the top clique. DDTW, DTW, and EDR are consistently in the bottom clique. For every evaluation metric, DDTW, DTW, EDR, and WDTW perform worse than Euclidean distance.

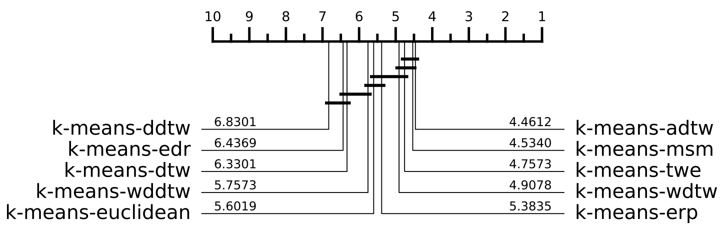


Fig. 5.8 AMI

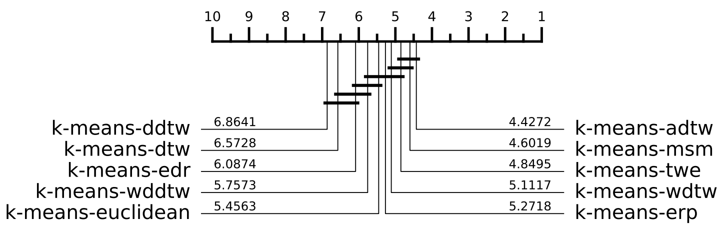


Fig. 5.9 ARI

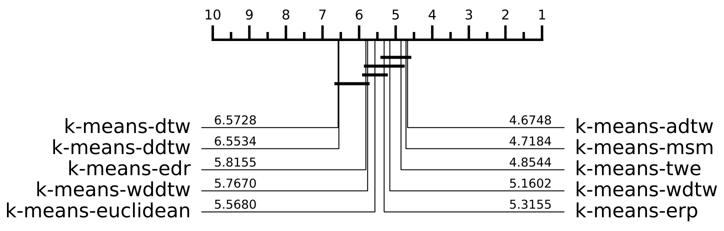


Fig. 5.10 CLACC

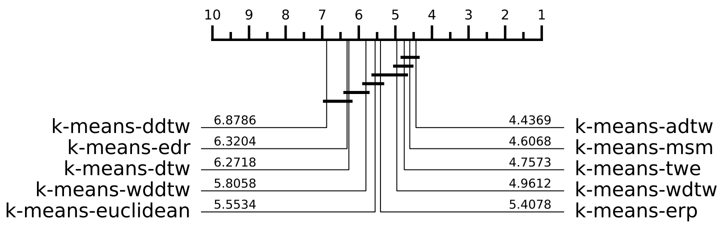


Fig. 5.11 NMI

Fig. 5.12 CD diagrams for k-means with 10 distances over 103 datasets from the UCR archive using the combined test-train split.

The distances that appear in the top clique share a common characteristic: they all explicitly penalise warping off the diagonal with a constant value. ADTW uses the constant $\omega$, TWE uses $\lambda$, and MSM employs a constant cost $c$. While

WDTW also applies a penalty, it is a “soft” penalty that increases gradually as more warping occurs. As such it performs well over our evaluation metrics but is unable to consistently perform as well as ADTW, MSM or TWE. Table 5.5 shows the average score for each clusterer. MSM achieved the highest average score for four evaluation metrics, joint best with ADTW for one, and ADTW was best for one other.

As further evidence supporting our hypothesis, we examine the performance difference between DTW and ADTW. ADTW is the same algorithm as DTW, but with the addition of a constant penalty applied for moving off the diagonal. Despite this relatively simple modification, ADTW ranks among our best-performing clusterers, while DTW is one of the worst. This stark contrast in performance highlights the contribution of a explicit warping penalty.

Across different problem domains, for AMI, as shown in Table 5.6, MSM, TWE, and ADTW are the top performers in five out of seven categories, with WDTW leading in one category. For ARI, as shown in Table 5.7, MSM, TWE, and ADTW dominate in four categories, while WDTW is best in two.

In the Image, Sensor, Simulated, and ECG domains, the average AMI and ARI scores are significantly higher compared other domains. In the Device, Sensor, and Motion domains, the average ARI and AMI scores are much lower. Notably, in the Spectro domain, Euclidean distance achieves the highest average score for both AMI and ARI.

DTW, DDTW, and EDR consistently appear in the bottom clique and perform worse than Euclidean distance. This is somewhat surprising, as we initially expected that DTW would perform well and at least better than $k$-means-euclidean. Additionally, WDDTW also performs worse than Euclidean distance on average.

We hypothesise that the poor performance of these distances is due to their lack of an explicit penalty for warping off the diagonal. DTW, DDTW, and EDR rely on an implicit penalty, where warping further from the diagonal naturally increases the

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-adtw | **0.224** | 0.269 | 0.552 | 0.278 | **0.684** |
| k-means-ddtw | 0.156 | 0.196 | 0.497 | 0.206 | 0.605 |
| k-means-dtw | 0.169 | 0.216 | 0.511 | 0.227 | 0.635 |
| k-means-edr | 0.161 | 0.190 | 0.508 | 0.202 | 0.641 |
| k-means-erp | 0.207 | 0.251 | 0.542 | 0.261 | 0.681 |
| k-means-euclidean | 0.202 | 0.246 | 0.532 | 0.257 | 0.679 |
| k-means-msm | **0.224** | **0.270** | **0.556** | **0.280** | 0.683 |
| k-means-twe | 0.214 | 0.259 | 0.549 | 0.270 | 0.677 |
| k-means-wddtw | 0.189 | 0.229 | 0.526 | 0.240 | 0.658 |
| k-means-wdtw | 0.209 | 0.254 | 0.548 | 0.264 | 0.679 |

Table 5.5 Summary of average score across multiple evaluation metrics for k-means with 10 distances over 103 datasets from the UCR archive using the combine test-train split split.

overall distance, thereby leading to a higher cost. However, without control over the degree of warping, this can result in pathological warping [24], which we suspect occurred here. We will test this hypothesis in Section 5.4 by applying a bounding window to DTW and DDTW, and analysing where and how much warping occurs.

Before exploring the test-train split results, we reintroduce our baseline Lloyd's clusterers to contextualise our findings. Figure 5.17 shows the critical difference diagrams for our *k*-means experiment with elastic distances, alongside the baseline Lloyd's models.

The results are somewhat surprising. Despite the intentional simplification of our experimentation, MSM, ADTW, and TWE consistently appear in the top clique across all evaluation metrics, alongside *k*-means-ba-DTW and *k*-shapes. Furthermore, for three of the evaluation metrics, MSM achieves a better average rank than *k*-shapes, while ADTW outperforms *k*-shapes across all four metrics. The performance gap between ADTW and *k*-means-ba-DTW is more pronounced, which is expected since elastic distances are used in both the assignment and centroid computation stages for *k*-means-ba-DTW.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-adtw | 0.364 | 0.181 | 0.261 | 0.370 | 0.083 | **0.216** | **0.226** |
| k-means-ddtw | 0.266 | 0.165 | 0.233 | 0.229 | 0.077 | 0.091 | 0.156 |
| k-means-dtw | 0.284 | 0.135 | 0.227 | 0.390 | 0.061 | 0.147 | 0.128 |
| k-means-edr | 0.255 | 0.157 | 0.209 | 0.269 | 0.086 | 0.084 | 0.187 |
| k-means-erp | 0.329 | 0.167 | 0.263 | 0.341 | 0.107 | 0.194 | 0.168 |
| k-means-euclidean | 0.322 | **0.187** | 0.254 | 0.344 | 0.076 | 0.199 | 0.159 |
| k-means-msm | **0.374** | 0.164 | 0.272 | 0.350 | **0.120** | 0.199 | 0.198 |
| k-means-twe | 0.339 | 0.179 | **0.283** | 0.371 | 0.108 | 0.171 | 0.194 |
| k-means-wddtw | 0.329 | 0.155 | 0.264 | 0.227 | 0.067 | 0.146 | 0.159 |
| k-means-wdtw | 0.325 | 0.157 | 0.268 | **0.393** | 0.085 | 0.211 | 0.137 |

Table 5.6 Average AMI score on problems split by problem domain for k-means with 10 distances over 103 datasets from the UCR archive using the combine test-train split split.

These findings highlight the significant potential of elastic distances. Even without a fully developed model that integrates elastic distances into the centroid computation, several of these distances already achieve state-of-the-art performance. Observing the improvement from *k*-means-DTW to *k*-means-ba-DTW, we hypothesise that developing bespoke averaging techniques for MSM, ADTW, and TWE could lead to substantial advancements in clustering performance, as they start from a higher baseline accuracy than DTW.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-adtw | 0.294 | 0.142 | 0.234 | 0.328 | 0.058 | 0.160 | **0.266** |
| k-means-ddtw | 0.180 | 0.145 | 0.227 | 0.185 | 0.046 | 0.050 | 0.212 |
| k-means-dtw | 0.205 | 0.098 | 0.212 | 0.329 | 0.027 | 0.094 | 0.142 |
| k-means-edr | 0.212 | 0.109 | 0.188 | 0.243 | 0.053 | 0.062 | 0.232 |
| k-means-erp | 0.263 | 0.135 | 0.232 | 0.306 | 0.079 | 0.139 | 0.184 |
| k-means-euclidean | 0.255 | **0.147** | 0.223 | 0.306 | 0.052 | 0.143 | 0.174 |
| k-means-msm | **0.300** | 0.135 | 0.233 | 0.315 | **0.094** | 0.149 | 0.227 |
| k-means-twe | 0.260 | 0.143 | **0.259** | 0.326 | 0.079 | 0.123 | 0.220 |
| k-means-wddtw | 0.263 | 0.123 | 0.247 | 0.178 | 0.041 | 0.087 | 0.213 |
| k-means-wdtw | 0.256 | 0.118 | 0.243 | **0.341** | 0.064 | **0.163** | 0.139 |

Table 5.7 Average ARI score on problems split by problem domain for k-means with 10 distances over 103 datasets from the UCR archive using the combine test-train split split.

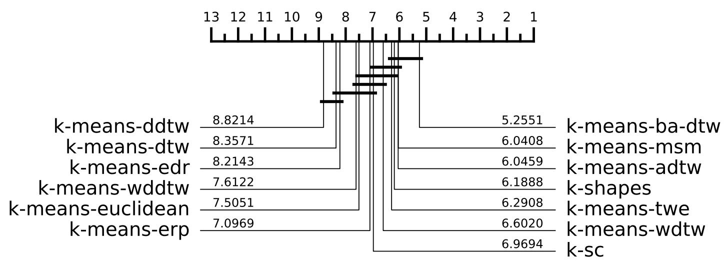


Fig. 5.13 AMI

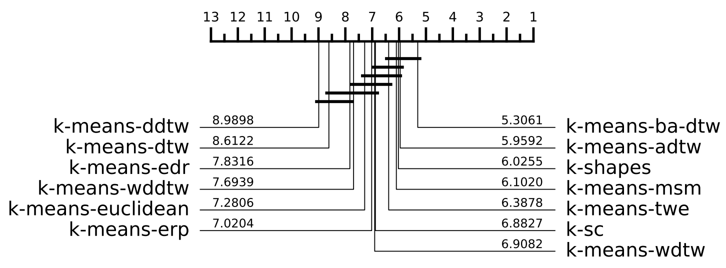


Fig. 5.14 ARI

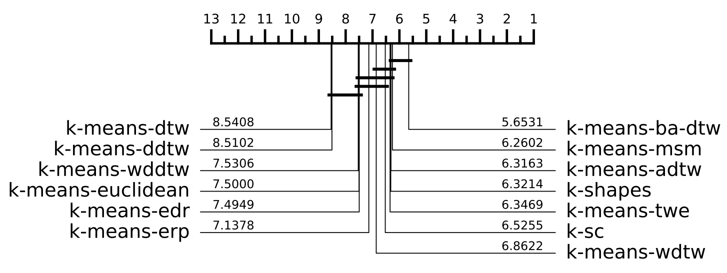


Fig. 5.15 CLACC

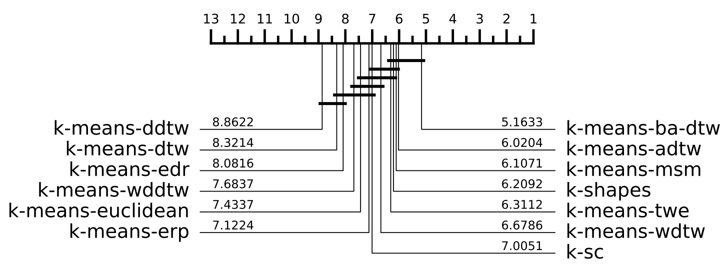


Fig. 5.16 NMI

Fig. 5.17 CD diagrams for *k*-means using 10 distances with the baseline Lloyd's models over 98 datasets from the UCR archive using the combine test train split.

### 5.3.2 Test-train split results

We now consider the test-train split. Across all distances, 103 datasets were successfully completed by all models. The datasets that were excluded were due to repeated empty clusters forming for certain distances. A complete list of excluded datasets is provided in Table A.7.

Figure 5.22 shows the critical difference diagram for the test-train split results, revealing a similar ranking of distances as observed in the combined test-train split. MSM, ADTW, soft-DTW, and TWE consistently appear in the top clique, while shape-DTW and WDTW rank in the top clique for ARI but fall into the second-best clique for other evaluation metrics. DDTW, DTW, WDDTW, LCSS, and EDR consistently appear in the bottom clique and perform worse than *k*-means-euclidean.

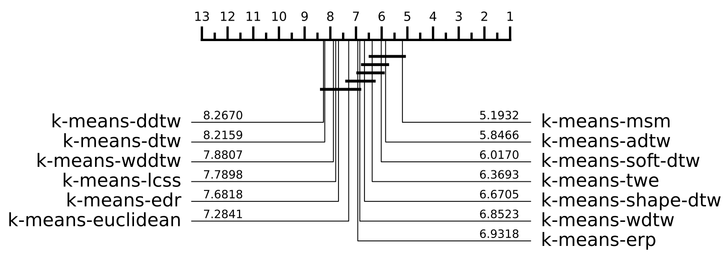


Fig. 5.18 AMI

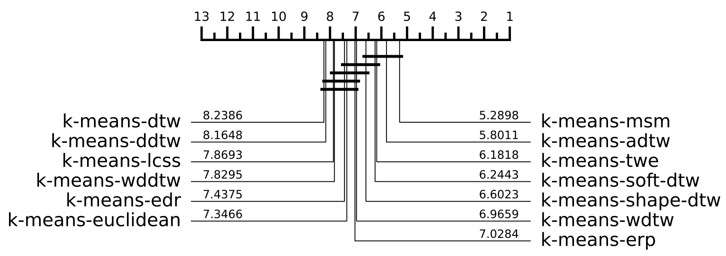


Fig. 5.19 ARI

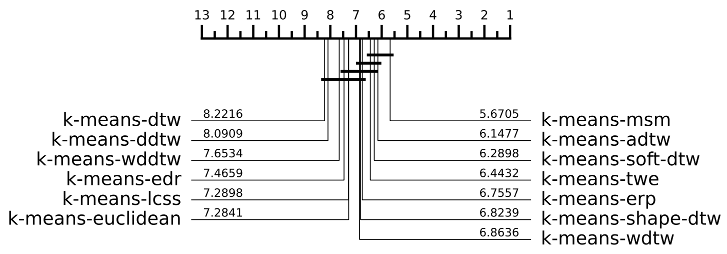


Fig. 5.20 CLACC

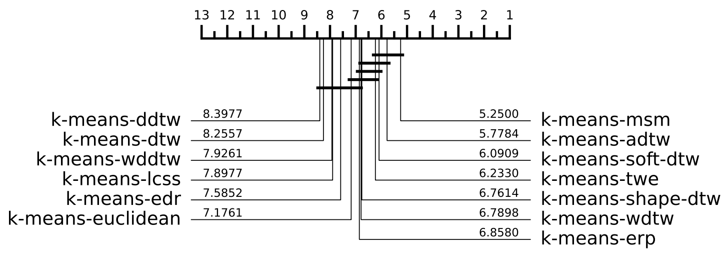


Fig. 5.21 NMI

Fig. 5.22 CD diagrams for k-means using 13 distances over 103 datasets from the UCR archive using the test-train split.

MSM achieves the highest average score across all evaluation metrics, as shown in Table 5.8. This contrasts with our findings from the combined test-train split, where shape-DTW had the highest average score across all metrics except RI. This suggests that shape-DTW does not generalise as well to unseen data, indicating it may struggle to learn robust representations. However, even in the combined test-train split, MSM performed consistently well across all metrics.

To further contextualise our results we also include the baseline Lloyd's models in our results. Figure 5.27 shows the critical difference diagrams for all 12 elastic distances with our baseline Lloyd's models incorporated. Figure 5.27 shows that MSM and ADTW are the two best performing distances notably outperform *k*-means-ba-DTW on average. This suggests that the general representations learned by MSM and ADTW using the arithmetic mean are better than those learned using DBA. We hypothesise that this is likely not due to DBA producing poor averages, but rather that the assignments determined by the DTW distance are poor and thus the average produced in also poor. Similar to the combined test-train results, ADTW, MSM, and TWE consistently appear in the top clique, along with *k*-means-ba-DTW.

Our findings are consistent with the combined test-train split and support our hypothesis that distances which explicitly penalise warping perform well, while those with implicit penalties perform poorly. While the general rank order of the distances remains consistent, shape-DTW performs notably worse than in the combined test-train split. Additionally, when contextualising our results with the state-of-the-art methods, we found that ADTW and MSM surprisingly outperform all other approaches.

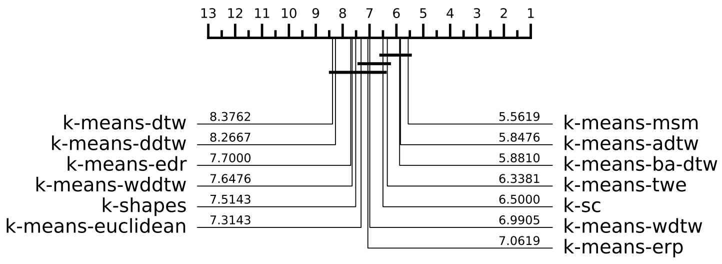


Fig. 5.23 AMI

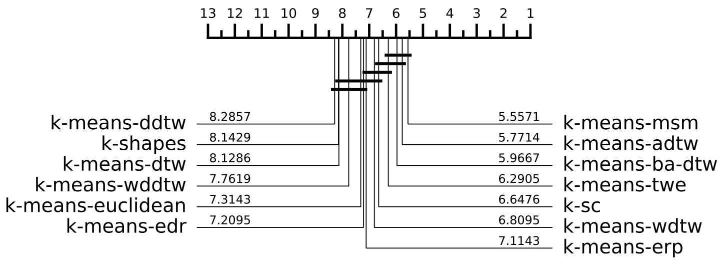


Fig. 5.24 ARI

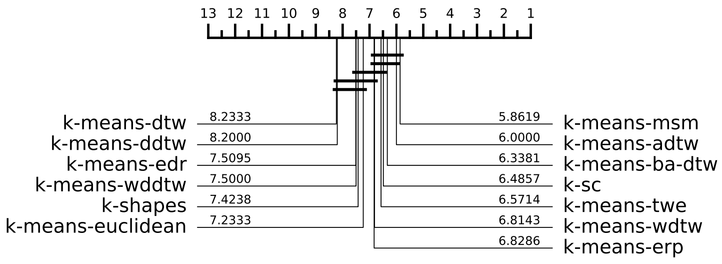


Fig. 5.25 CLACC

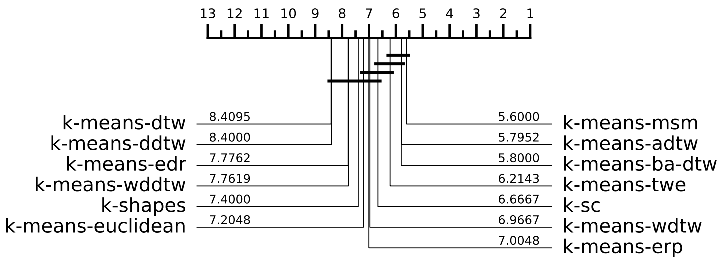


Fig. 5.26 NMI

Fig. 5.27 CD diagrams for k-means using 13 distances with the baseline Lloyd's models over 103 datasets from the UCR archive using the test train split

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-adtw | 0.178 | 0.221 | 0.547 | 0.236 | 0.650 |
| k-means-ddtw | 0.137 | 0.179 | 0.512 | 0.194 | 0.591 |
| k-means-dtw | 0.144 | 0.189 | 0.526 | 0.204 | 0.614 |
| k-means-edr | 0.146 | 0.175 | 0.520 | 0.191 | 0.628 |
| k-means-erp | 0.158 | 0.200 | 0.536 | 0.216 | 0.645 |
| k-means-euclidean | 0.154 | 0.195 | 0.528 | 0.211 | 0.642 |
| k-means-lcss | 0.145 | 0.185 | 0.527 | 0.201 | 0.620 |
| k-means-msm | **0.188** | **0.232** | **0.555** | **0.247** | **0.653** |
| k-means-shape-dtw | 0.169 | 0.213 | 0.543 | 0.227 | 0.639 |
| k-means-soft-dtw | 0.181 | 0.223 | 0.552 | 0.237 | 0.648 |
| k-means-twe | 0.176 | 0.215 | 0.549 | 0.230 | 0.646 |
| k-means-wddtw | 0.153 | 0.194 | 0.528 | 0.210 | 0.629 |
| k-means-wdtw | 0.158 | 0.208 | 0.537 | 0.223 | 0.644 |

Table 5.8 Summary of average score across multiple evaluation metrics over 103 datasets from the UCR archive using the test-train split.

## 5.4 Tuning

Generally, most of the distances performed as expected, and the overall rank order aligns with TSC results [111]. However, DTW was an exception, as it did not perform as well as anticipated.

Throughout our experiments, we observed that DTW was one of the worst-performing distances and was notably always worse than *k*-means-Euclidean for both the combined test-train split and the test-train split. In the TSC literature, DTW is traditionally used with a Sakoe-Chiba band set between 5% and 20% because it is prone to pathological warping [24]. However, in our initial experiments, we opted not to use a bounding window, instead keeping a full window. The reason for this is that DTW is the only distance that has been extensively used in the TSCL literature, and we have not found examples where a warping window was used.

We suspect that DTW's poor performance is due to pathological warping. To investigate this, we conducted a tuning experiment using a warping window for DTW. Additionally, we examined DDTW, as it also uses DTW after applying the first derivative. While we did not expect DDTW to be among the best-performing distances, it may still suffer from pathological warping, warranting further investigation.

To test this hypothesis, we reran *k*-means-DTW, *k*-means-DDTW, and *k*-means-ba-DTW with a 20% bounding window for both the test-train split and the combined test-train split. Similar to our previous experiments, due to empty clusters continuing to form, we were unable to obtain a complete set of results. Therefore, the combined test-train split evaluation is conducted over 101 datasets (missing datasets are listed in Table A.10), and the test-train split evaluation is conducted over 108 datasets (missing datasets are listed in Table A.11).

The critical difference diagrams for the combined test-train split are shown in Figure 5.32, and the critical difference diagram for the test-train split is shown in

Figure 5.37. Clusterers labelled with “20-window” indicate that a 20% bounding window was used in the computation.

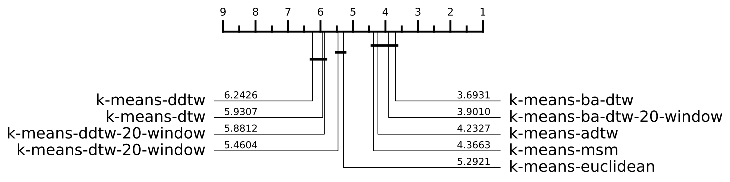


Fig. 5.28 AMI

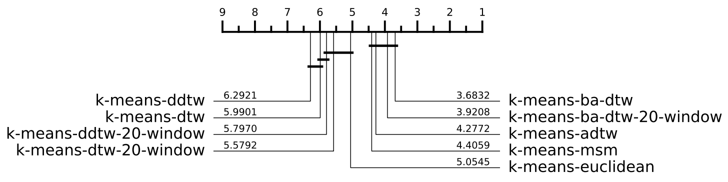


Fig. 5.29 ARI

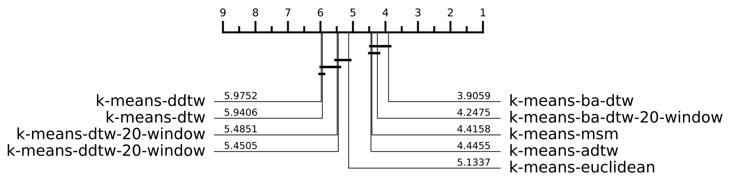


Fig. 5.30 CLACC

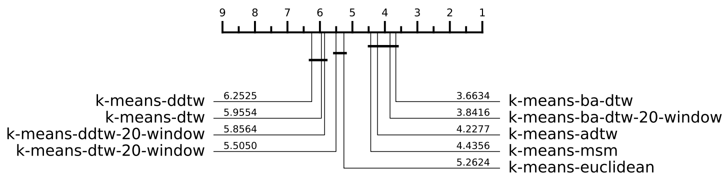


Fig. 5.31 NMI

Fig. 5.32 CD diagrams for our tuned DTW *k*-means clusterers over 101 datasets from the UCR archive using the combine test train split.

In both the test-train split and the combined test-train split, applying a 20% window improves the performance of DTW and DDTW, but not by a significant margin. Despite this improvement, *k*-means-dtw-20-window still performs worse than *k*-means-Euclidean. The scatter plots in Figure 5.38 provide more detailed insight into this. For the ARI evaluation metric, even with a 20% window, DTW is significantly outperformed by Euclidean distance, as highlighted by the difference in medians shown in Figure 5.38a. When comparing full-window DTW to the 20% window DTW in Figure 5.38b, the 20% window does result in significantly more wins for the ARI metric. However, the large number of ties suggests that the warping window was ineffective for many datasets.

Similarly, for *k*-means-ba-DTW, there is no significant difference between *k*-means-ba-DTW and *k*-means-ba-DTW-window-20. Figure 5.39 provides a direct comparison of *k*-means-ba-DTW and *k*-means-ba-DTW-window-20 in terms of

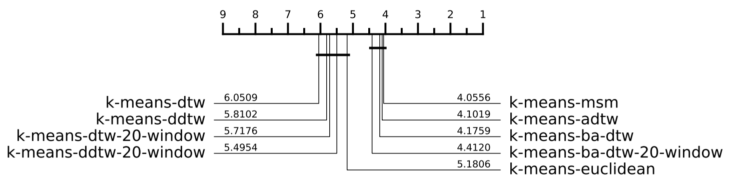


Fig. 5.33 AMI

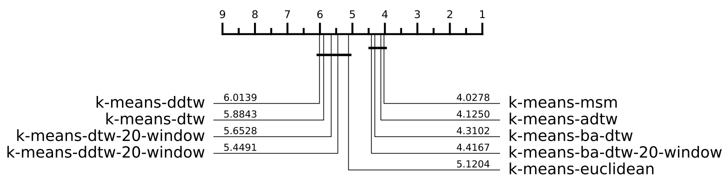


Fig. 5.34 ARI

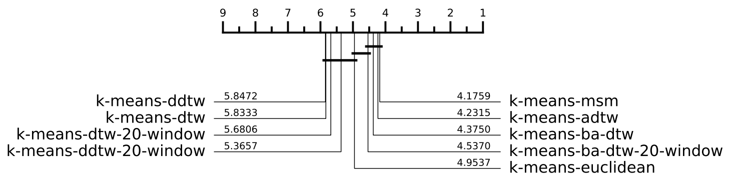


Fig. 5.35 CLACC

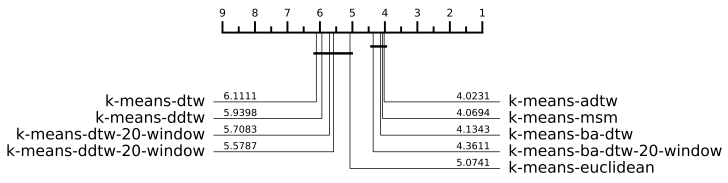
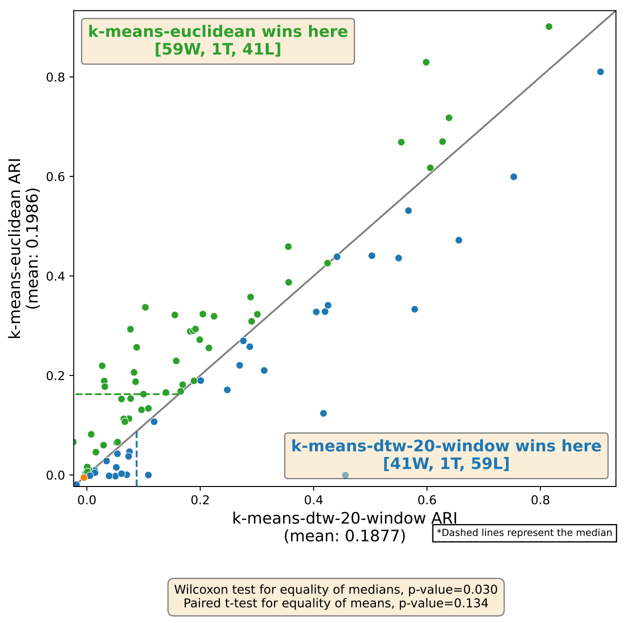
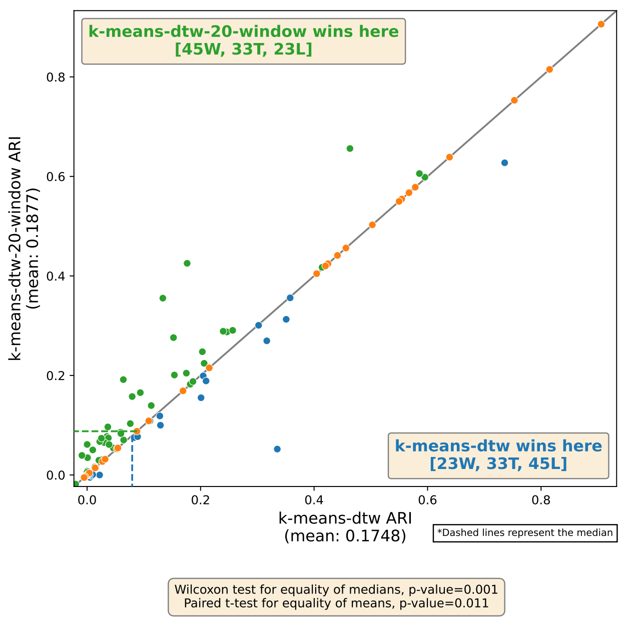


Fig. 5.36 NMI

Fig. 5.37 CD diagrams for our tuned DTW *k*-means clusterers over 108 datasets from the UCR archive using the test train split.



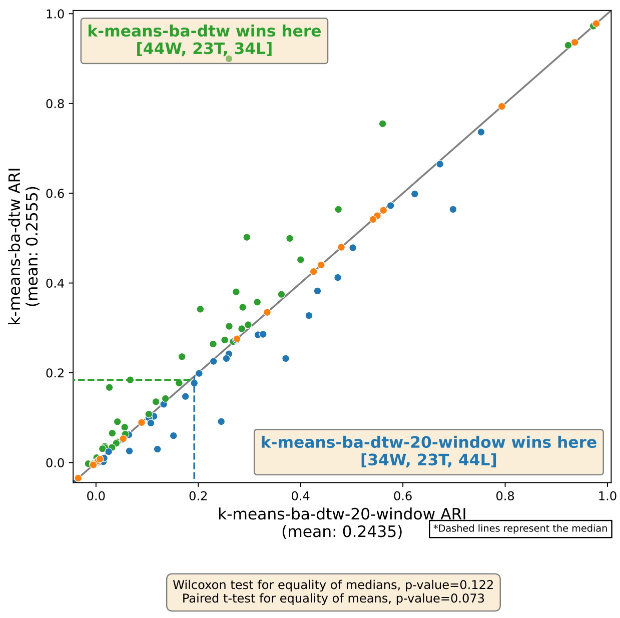
(a) Scatter comparison of *k*-means-Euclidean and *k*-means-DTW-20-window over 101 of the combined test-train UCR archive.



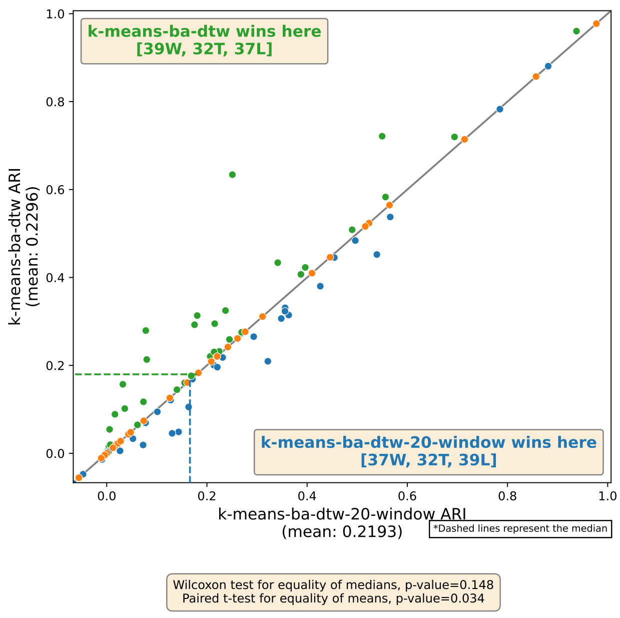
(b) Scatter comparison of *k*-means-DTW and *k*-means-DTW-20-window over 101 of the combined test-train UCR archive.

Fig. 5.38 Scatter plot comparison of *k*-means-DTW-20-window

ARI over both the combined test-train and test-train splits. Overall, there are many ties in results for both splits, suggesting that the performance is largely independent of the warping window. However, in terms of absolute ranking *k*-means-ba-DTW with no bounding does outperform *k*-means-ba-window-20.



(a) Scatter comparison of *k*-means-Euclidean and *k*-means-DTW-20-window over 101 combined test-train UCR datasets.



(b) Scatter comparison of *k*-means-ba-DTW and *k*-means-DTW-20-window over 108 test-train UCR datasets.

Fig. 5.39 Scatter plot comparison of *k*-means-ba-DTW-20-window compared to *k*-means-ba-DTW over the combined test-train split and the test-train split.

Overall, from our DTW tuning experiment, we found that while a bounding window can improve the performance of DTW and DDTW, the improvements were not significant. We initially expected a larger performance gain. To investigate why the clustering results remained poor even with a 20% bounding window, we conducted a further experiment to evaluate how and where different distances warp, aiming to reveal the specific warping patterns that lead to better performance.

To perform this analysis, we took the final centroids produced by *k*-means-DTW-20-window, *k*-means-DTW-5-window, *k*-means-TWE, and *k*-means-MSM on the training split. We added a new clusterer, *k*-means-DTW-5-window, which uses a 5% warping window with DTW to further investigate pathological warping. For each clusterer, we then counted the number of diagonal, horizontal, and vertical moves required in the warping path when assigning each test instance to its closest centroid for each distance. This was done for all datasets in the UCR archive. The number of horizontal moves is always equal to the number of vertical moves, as an optimal warping path must end at $(0,0)$. Our findings are presented in Table 5.9.

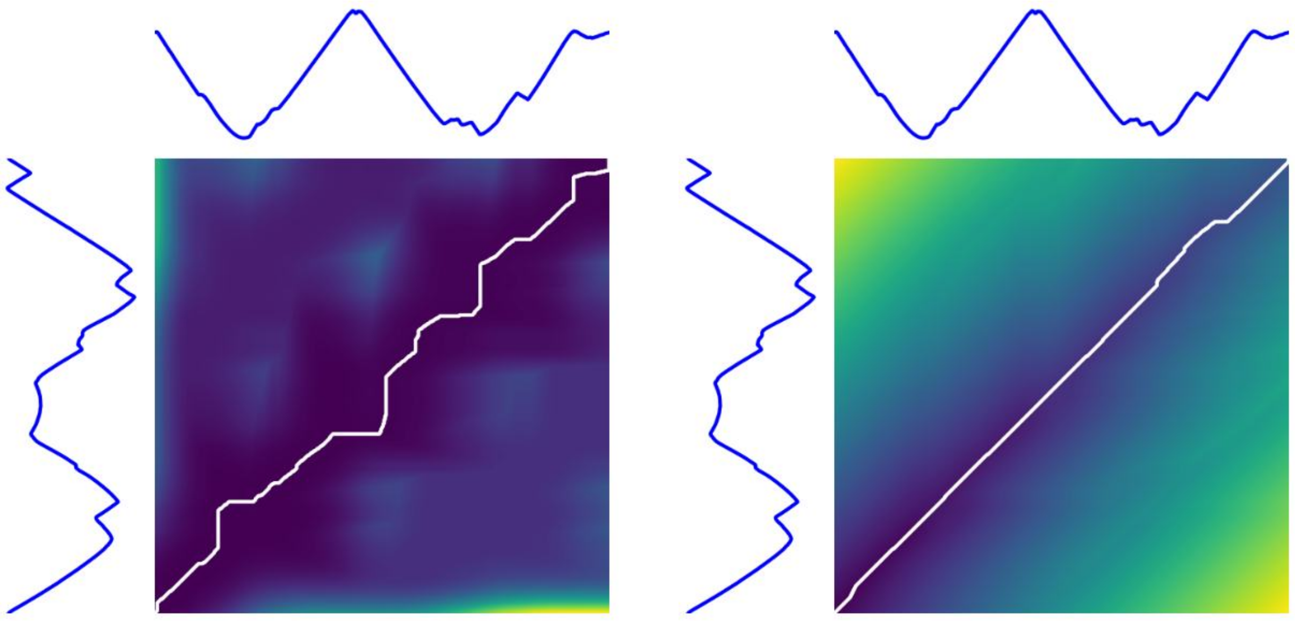
| **Distance** | **Average Optimal Path Length** | **Average Number of Diagonal Moves** | **Average Number of Horizontal/Vertical Moves** | **Ratio of Diagonal Moves to Horizontal/Vertical Moves** |
|---|---|---|---|---|
| DTW (20% window) | 863 | 230 | 633 | 2.75 |
| DTW (5% window) | 690 | 326 | 363 | 1.11 |
| MSM | 576 | 499 | 77 | 0.15 |
| TWE | 591 | 484 | 107 | 0.22 |

Table 5.9 Average length of the optimal warping path for each time series in the test split assigned to its closest centroid. The UCR 112 univariate archive has an average time series length of 551.

In Table 5.9, we observe that for MSM, for every one diagonal move, there are 0.15 horizontal/vertical moves. Similarly, for TWE, for every one diagonal move made, there are 0.22 horizontal/vertical moves. This indicates that the warping paths for MSM and TWE are very constrained, generally consisting of long stretches of diagonal moves with minimal horizontal or vertical adjustments. Consequently, the total average warping path lengths for MSM and TWE are 576 and 591, respectively.

In contrast, even with a 20% or 5% warping window, DTW exhibits significantly more horizontal and vertical movement compared to MSM and TWE. For every one diagonal move, DTW with a 20% window makes 2.75 horizontal/vertical moves, while DTW with a 5% window makes 1.11 such moves. This means that DTW with a 5% warping window makes over seven times more horizontal/vertical moves than MSM, and with a 20% window, DTW makes over 18 times more horizontal/vertical moves. Essentially, even when a bounding window is applied, DTW still pushes its movements to the window's limit, moving more horizontally and vertically than diagonally. The average warping path length for DTW with a 20% window is 863, and for DTW with a 5% window, it is 690. Both of these values are significantly higher than the warping path lengths for MSM and TWE.

A visualisation of this difference in warping paths is shown in Figure 5.40, which compares the warping paths of two time series from the Fish dataset using DTW and MSM distances. The MSM warping path is much more constrained, beginning with a slight deviation off the diagonal but maintaining a relatively constant trajectory thereafter. In contrast, the DTW path fluctuates significantly, with more pronounced vertical and horizontal movements compared to MSM. Overall, these additional movements result in the DTW alignment path being significantly longer than that of MSM.



(a) DTW warping path (b) MSM warping path

Fig. 5.40 Comparison of DTW and MSM warping paths for two time series in the Fish UCR dataset. The warping path is the white line on top of a heat map representation of each distance's cost matrix.

Overall, our tuning results further support our hypothesis that using an explicit penalty for warping off the diagonal, rather than relying on an implicit one, leads to better TSCL results. Additionally, we found that applying a warping window does not significantly improve the performance of DTW.

## 5.5 Conclusion

We have conducted an extensive review of the *k*-means clusterer using elastic distances. We have found that distances such as MSM, TWE, and ADTW consistently outperformed other elastic distances. Conversely, DTW, DDTW, EDR, and WDDTW performed significantly worse, often being outperformed by *k*-means-Euclidean. Our results suggest that for *k*-means clustering, distances that explicitly penalise warping off the diagonal consistently deliver better results than those using an implicit penalty.

To contextualise our findings, we included commonly used time series clusterers from the literature, two of which are considered state-of-the-art. We found that ADTW, MSM, and TWE were not significantly different from *k*-means-ba-DTW and *k*-shapes across all evaluation metrics, though they did not outperform *k*-means-ba-DTW in terms of absolute ranking on the combined test-train split.

We also extended our experimentation to the test-train split, where we observed consistent results with the combined test-train split. However, for the test-train split we observed that MSM and ADTW outperformed all considered clusterers including the state-of-the-art baseline models.

In addition we performed an in-depth analysis to understand why DTW performed poorly. Although we experimented with different window sizes for tuning DTW, we found that this did not lead to significant improvements. Our analysis revealed that the pathological warping in DTW, compared to more successful distances like TWE and MSM, is likely responsible for its subpar performance. These findings further support our hypothesis that explicitly penalising warping off the diagonal is a key factor in the success of clustering distances.

In conclusion, although our experiments were intentionally simplified, the results exceeded our expectations and demonstrate the potential of elastic distances—particularly those with explicit warping penalties—for TSCL. By comparing the performance difference between *k*-means-DTW with *k*-means-ba-DTW, we hypothesise that leveraging better elastic distances in centroid computation could lead to significantly improved clustering results. With this in mind, we will now explore *k*-medoids clusterers, which can utilise elastic distances not only in the assignment stage but also in centroid computation without requiring adjustments to the baseline algorithm.

# Chapter 6

# *k*-medoids clustering using elastic distances

## Contributing Publications

- Holder, C., Guijo-Rubio, D., Bagnall, A. (2023). Clustering Time Series with k-Medoids Based Algorithms. In: Ifrim, G., et al. Advanced Analytics and Learning on Temporal Data. AALTD 2023. Lecture Notes in Computer Science(), vol 14343. Springer, Cham. https://doi.org/10.1007/978-3-031-49896-1_4

A limitation of the *k*-means elastic distance methods discussed in Chapter 5 lies in the computation of centroids. By relying on the arithmetic mean, these methods fail to account for the alignment of time series during centroid calculation, which considerably hampers the performance of elastic distances in *k*-means clustering.

In contrast, *k*-medoids algorithms, although similar in structure to *k*-means, differ significantly in their approach to centroid computation. In *k*-medoids, centroids (or medoids) are selected based on the distance that minimises the total deviation (TD), thereby incorporating elastic distances into the centroid calculation. Given this advantage, we hypothesise that *k*-medoids-based algorithms will outperform *k*-means-based methods when elastic distances are employed.

## 6.1 Introduction

Before the introduction of DBA in 2011 [94], PAM-DTW [69] (PAM using DTW) was the preferred partition-based method for TSCL [88], primarily due to its ability to incorporate elastic distances in both the assignment and centroid computation stages, which resulted in improved clusterings.

However, with the advent of DBA [94], interest in *k*-medoids for TSCL declined. This shift occurred because *k*-means-ba-DTW offered comparable performance to PAM-DTW while being significantly less computationally expensive. Many *k*-medoids algorithms, including PAM, require the computation of a pairwise distance matrix, which often becomes impractical in real-world TSCL scenarios, reducing the feasibility of these approaches.

In the TSCL literature, only two *k*-medoids methods have been explored: PAM and alternate *k*-medoids. We outlined the difference between these models in Section 2.5.1. Moreover, these two clusterers have only been studied using Euclidean and DTW distances. Alternate *k*-medoids with DTW is very similar to *k*-means-DTW, with the primary difference being that alternate *k*-medoids computes medoids as cluster centres, whereas *k*-means-DTW uses the arithmetic mean. In the literature, it has been shown that alternate *k*-medoids-DTW significantly outperform *k*-means-Euclidean [46]. However, as demonstrated in Chapter 5, *k*-means-DTW performs significantly worse than *k*-means-Euclidean.

Given that the only difference between these two algorithms is the method of centroid computation, we hypothesise that clusterers capable of incorporating elastic distances into the centroid computation will demonstrate significantly better performance. Additionally, we hypothesise that the performance improvement from using medoids over the arithmetic mean will be similar for each distance. Therefore, we expect MSM, TWE, and ADTW to be the top-performing *k*-medoids clusterers as they were for the *k*-mean experiments.

To test these hypotheses, we begin by identifying four commonly used *k*-medoids clusterers, two of which have never been applied to TSCL before using elastic distances. We then proceed with experimentation for each model. Our evaluation is divided into two sections. First, we will evaluate each *k*-medoids clusterer in isolation, comparing elastic distances within the same model to identify the best version of each. Baseline clusterers will also be included to provide context for the results. Second, after evaluating each *k*-medoids model individually, we will combine the results to assess how elastic distances impact each model, identify the best-performing *k*-medoids-based clusterer, and comparing them to state-of-the-art clusterers.

## 6.2 TSCL *k*-medoids

We identify and configure four different *k*-medoids clusterers for use with elastic distances. Below, we outline each clusterer, provide pseudocode, and explain the reasoning behind our configuration decisions.

### 6.2.1 Alternate *k*-medoids

Alternate *k*-medoids [75] follows Lloyd's algorithm, but instead of computing the arithmetic mean, it calculates the medoid of each cluster. Therefore, we adapt our previously defined Lloyd's algorithm (Algorithm 26) by updating the centroid computation to use medoids. Additionally, for our experiments, we configure the parameters for alternate *k*-medoids in the same way as we did for our *k*-means elastic experiments, outlined in Table 5.1.

To adapt our Lloyd's algorithm for the alternate *k*-medoids, we make two key changes. First, we modify the objective function to minimise total dissimilarity (TD) instead of inertia. This adjustment also necessitates updating the centroid computation, where we now compute the medoid of each cluster instead of using the arithmetic mean.

Second, we remove the process for handling empty clusters, as this issue does not arise in *k*-medoids algorithms. In *k*-medoids, the medoid is always a data point from the dataset, ensuring that at least one data point is assigned to each cluster.

Algorithm 27 presents the alternate *k*-medoids algorithm used in our experiments.

**Algorithm 27:** Alternate_*k*-medoids(**X**, **k**, **n_init**, **max_iters**, **tol**, **n_init**))

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_init (Number of restarts with different initial medoids),
       max_iters (Maximum number of iterations before forced
       termination), tol (TD variation threshold)
Output: Assignment of each time series to a cluster and corresponding
        medoids
1  $best\_TD \leftarrow \infty$
2  Let best_assignments be an empty array of length n
3  Let best_medoids be an empty array of length k
4  for $i \leftarrow 1$ to n_init do
5      Let medoids be an array of k randomly chosen time series from dataset
       X
6      Let assignments be an empty array of length n
7      Let $prev\_TD \leftarrow \infty$
8      for $j \leftarrow 1$ to max_iters do
9          for each time series $x_i$ in X do
10             Compute the distance between $x_i$ and each of the k medoids
11             Assign $x_i$ to the nearest centre
12         for each centre $c_j$ in medoids do
13             Update $c_j$ to be the medoids of all time series assigned to it
14         Let curr_inner_TD be the of the current clustering
15         if $|curr\_inner\_TD - prev\_TD| < tol$ then
16             break
17         $prev\_TD \leftarrow curr\_inner\_TD$
18     Let curr_TD be the of the current clustering
19     if $curr\_TD < best\_TD$ then
20         $best\_TD \leftarrow curr\_TD$
21         $best\_assignments \leftarrow assignments$
22         $best\_medoids \leftarrow medoids$
23 return $best\_assignments, best\_medoids$
```

### 6.2.2 PAM

PAM [69] is the most popular *k*-medoids algorithm [107]. PAM traditionally consists of two steps: BUILD and SWAP. The BUILD step determines the initial medoids, while the SWAP step refines the initial medoids to reach a local optimum. These steps are detailed in Algorithm 29 and Algorithm 30, respectively, and the complete PAM algorithm that combines them is shown in Algorithm 28.

The BUILD step is designed to select "good" initial medoids, and its effectiveness largely depends on the quality of the distance measure used. In our initial experiments, we chose to use a different initialisation strategy than BUILD. We instead choose to use Forgy initialisation with 10 restarts. This approach allows us to focus specifically on how elastic distances affect the SWAP step within PAM. Additionally, by using the same initialisation method as our Lloyd's-based algorithms, we can make more accurate comparisons and better understand how elastic distances influence the refinement and averaging process.

PAM shares many of the same configuration options as Lloyd's. Specifically it shares max_iters, n_init and init_algorithm. We set these values to matches our Lloyd's-based clusterers for consistency.

**Algorithm 28:** PAM(**X**, **k**, **max_iters**)

```
Input: X (Dataset of objects), k (Number of clusters), max_iters
       (Maximum number of iterations for PAM SWAP), (TD variation
       threshold)
Output: Final assignments of each object to a cluster and corresponding
        medoids
1 $initial\_medoids \leftarrow PAM\_BUILD(X, k)$
2 $assignements, medoids \leftarrow PAM\_SWAP(X, k, max\_iters, initial\_medoids)$
3 return $assignments, medoids$
```

**Algorithm 29:** BUILD(**X**, **k**)

```
Input: X (Dataset of objects), k (Number of clusters)
Output: Initial k medoids
$TD \leftarrow \infty$, $m_1 \leftarrow$ null
for each object $x_c$ in X do
    $TD_j \leftarrow 0$
    for each object $x_o$ in X do
        $TD_j \leftarrow TD_j + d(x_o, x_c)$
    if $TD_j < TD$ then
        $TD \leftarrow TD_j$
        $m_1 \leftarrow x_c$
for each object $x_o$ in X do
    if $x_o = m_1$ then
        continue
    $d_{\text{nearest}}(o) \leftarrow d(m_1, x_o)$
for $i \leftarrow 1$ to $k - 1$ do
    $best\_TD \leftarrow \infty$
    $best\_x \leftarrow$ null
    for each object $x_c$ in X do
        if $x_c$ in $\{m_1, \ldots, m_j\}$ then
            continue
        $curr\_TD \leftarrow 0$
        for each object $x_o$ in X do
            if $x_o$ in $\{m_1, \ldots, m_j\}$ then
                continue
            $change\_in\_TD \leftarrow d(x_o, x_c) - d_{\text{nearest}}(o)$
            if $change\_in\_TD < 0$ then
                $curr\_TD \leftarrow curr\_TD + change\_in\_TD$
        if $curr\_TD < best\_TD$ then
            $best\_TD \leftarrow curr\_TD$
            $best\_x \leftarrow x_c$
    $TD \leftarrow TD + best\_TD$
    $m_{i+1} \leftarrow best\_x$
return $\{m_1, \ldots, m_k\}$
```

**Algorithm 30:** SWAP(**X**, **k**, **max_iters**, **medoids**)

```
Input: X (Dataset of of time series of length n), k (Number of clusters),
       max_iters (Maximum number of iterations), medoids (Initial
       medoids to refine)
Output: Assignments of each time series to a cluster and corresponding
        medoids
1  for each object $x_o$ in X do
2      Compute $nearest(o)$, $d_{\text{nearest}}(o)$, and $d_{\text{second}}(o)$
3  for $i \leftarrow 1$ to max_iters do
4      $best\_TD \leftarrow 0$
5      $best\_m \leftarrow$ null
6      $best\_x \leftarrow$ null
7      for each medoid $m_j$ in medoids do
8          for each non-medoid $x_c$ in X do
9              $curr\_TD \leftarrow 0$
10             for each non-medoid $x_o$ in X do
11                 $curr\_TD \leftarrow curr\_TD + \Delta(x_o, m_j, x_c)$
12             if $curr\_TD < best\_TD$ then
13                 $best\_TD \leftarrow curr\_TD$
14                 $best\_m \leftarrow m_j$
15                 $best\_x \leftarrow x_c$
16     if $best\_TD \geq 0$ then
17         break
18     Swap roles of medoid best_m and non-medoid best_x
19     for each object $x_o$ in X do
20         Update $nearest(o)$, $d_{\text{nearest}}(o)$, and $d_{\text{second}}(o)$
21     $TD \leftarrow TD + best\_TD$
22 Let assignments be an array where each element $x_o$ in X is assigned to the
   nearest medoid in $\{m_1, \ldots, m_k\}$
23 return $assignments, medoids$
```

### 6.2.3 CLARA

CLARA [68] repeatedly applies PAM to random subsets of the dataset. Using the medoids produced for a subset, all values in the entire dataset are then assigned to their nearest medoids, and the TD of the configuration is measured. This process is repeated, and the iteration with the lowest TD is returned.

**Algorithm 31:** PAM_without_BUILD(**X**, **k**, **n_init**, **max_iters**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_init (Number of restarts with different initial centroids),
       max_iters (Maximum number of iterations before forced
       termination)
Output: Assignment of each time series to a cluster and corresponding
        medoids
1  $best\_TD \leftarrow \infty$
2  Let best_assignments be an empty array of length $n$
3  Let best_medoids be an empty array of length $k$
4  for $i \leftarrow 1$ to n_init do
5      Let initial_medoids be an array of $k$ randomly chosen time series from
         dataset X
6      $assignements \leftarrow PAM\_SWAP(X, k, max\_iters, initial\_medoids)$
7      Let $curr\_TD$ be the of the current clustering
8      if $curr\_TD < best\_TD$ then
9          $best\_TD \leftarrow curr\_TD$
10         $best\_assignments \leftarrow assignments$
11         $best\_medoids \leftarrow medoids$
12 return $assignments, medoids$
```

To facilitate this process, CLARA uses two additional parameters: the number of samples to use for each subset (*n_samples*) and the number of runs to perform with different random samples (*n_sampling_iters*). We set *n_samples* to $40 + 2k$ for each dataset as it is recommended in the original paper [68]. Choosing an appropriate value for *n_sampling_iters* is more challenging, as the optimal value depends on factors such as the size of the dataset and the availability of computational resources. For our experiments, we set this value to 10. While this may be higher than necessary for many datasets, we opted for a value that is applicable across all datasets in the UCR archive, thus choosing a higher value than usual.

After a random subset of time series is selected, PAM is executed. Traditionally, CLARA uses PAM with BUILD initialisation (Algorithm 28). However, in our version of CLARA, we employ PAM with Forgy initialisation and 10 restarts

(Algorithm 31). The version of CLARA used in our experimentation is defined in Algorithm 32.

**Algorithm 32:** CLARA(**X**, **k**, **n_samples**, **n_sampling_iters**, **max_iters**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_samples (Number of samples in each subset), n_sampling_iters
       (Number of sampling iterations), max_iters (Maximum number of
       iterations before forced termination)
Output: Assignment of each time series to a cluster and corresponding
        medoids
1  $best\_TD \leftarrow \infty$
2  Let best_assignments be an empty array of length $n$
3  Let best_medoids be an empty array of length $k$
4  for $i \leftarrow 1$ to n_sampling_iters do
5      Let random_subset be an array of n_samples randomly chosen time
         series from dataset X
6      $subset\_assignments, medoids \leftarrow$ PAM$(random\_subset, k, max\_iters$
7      Let assignments be an array of integers where each element represents
         the index of the medoid in medoids closest to the corresponding time
         series in $D$.
8      Let curr_TD be the of the current clustering based on the assignments
         in assignments.
9      if $curr\_TD < best\_TD$ then
10         $best\_TD \leftarrow curr\_TD$
11         $best\_assignments \leftarrow assignments$
12         $best\_medoids \leftarrow medoids$
13 return $best\_assignments, best\_medoids$
```

### 6.2.4 CLARANS

CLARANS [86] updates the PAM SWAP algorithm by randomly choosing a value in the dataset to become a random medoid. If that swap reduces TD, then the swap is performed immediately. If it doesn't reduce TD, a counter is incremented to track the number of failed swaps. If this counter exceeds the value of the *max_neighbours* parameter, then the algorithm terminates and returns the medoids and assignments for the configuration.

The CLARANS algorithm introduces a new parameter, *max_neighbours*, which defines the maximum number of neighbouring solutions the algorithm will explore for each set of medoids. A neighbouring solution is obtained by replacing one of the medoids with a non-medoid and evaluating if this reduces the total cost. It is recommended to set *max_neighbours* to 1.25% of the total number of possible swaps for the dataset [86].

Additionally, CLARANS uses a parameter called *num_local*, which determines the number of restarts the algorithm performs with different initial medoids, selected using Forgy initialisation. In our previous experiments, we already employ this strategy, though we refer to this parameter as *n_init*. For consistency across our experiments, we will use the name *n_init* in our CLARANS implementation instead of *num_local*.

Algorithm 33 outlines the CLARANS SWAP procedure, and Algorithm 34 presents the full CLARANS algorithm.

**Algorithm 33:** CLARANS_SWAP(**X**, **k**, **max_neighbours**, **medoids**)

```
Input: X (Dataset of of time series of length n), k (Number of clusters),
       max_neighbours (Maximum attempt to find a swap), medoids
       (Initial medoids to refine)
Output: Assignments of each time series to a cluster and corresponding
        medoids
1  Let assignments be the assignment of each value in $X$ to its closest medoids
     in medoids
2  Let $TD$ be the TD of the assignments clustering
3  $i \leftarrow 0$
4  while $i < max\_neighbours$ do
5      Let $medoids\_to\_replace$ be a randomly selected medoids from
         medoids
6      Let $non\_medoids\_to\_swap$ be a randomly selected non-medoids from
         $X$
7      Let $candidate\_medoids$ be the medoids array where
         $medoids\_to\_replace$ is swapped $non\_medoids\_to\_swap$
8      Let $candidate\_assignments$ be the assignment of each value in $X$ to its
         closest medoids in $candidate\_medoids$
9      Let $canidate\_TD$ be the TD of the $candidate\_assignments$ clustering
10     if $candidate\_TD < TD$ then
11         $TD \leftarrow candidate\_TD$
12         $assignments \leftarrow candidate\_assignment$
13         $medoids \leftarrow candidate\_medoids$
14     else
15         $i \leftarrow i + 1$
16 return $assignments, medoids$
```

**Algorithm 34:** CLARANS(**X**, **k**, **n_init**, **max_iters**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       n_init (Number of restarts with different initial centroids),
       max_iters (Maximum number of iterations before forced
       termination)
Output: Assignment of each time series to a cluster and corresponding
        medoids
1  $best\_TD \leftarrow \infty$
2  Let $best\_assignments$ be an empty array of length $n$
3  Let $best\_medoids$ be an empty array of length $k$
4  for $i \leftarrow 1$ to $n\_init$ do
5      Let $initial\_medoids$ be an array of $k$ randomly chosen time series from
         dataset X
6      $assignements \leftarrow PAM\_SWAP(X, k, max\_iters, initial\_medoids)$
7      Let $curr\_TD$ be the of the current clustering
8      if $curr\_TD < best\_TD$ then
9          $best\_TD \leftarrow curr\_TD$
10         $best\_assignments \leftarrow assignments$
11         $best\_medoids \leftarrow medoids$
12 return $assignments, medoids$
```

## 6.3 Experiment setup

To evaluate four different $k$-medoids algorithms across 12 elastic distances, we divide our experiments and analysis into two sections. First, we evaluate each $k$-medoids model in isolation, comparing each models performance to itself using different elastic distances and incorporating baseline results. In this first section, our goal is to identify the best-performing elastic distance for each model. Once each model has been analysed individually, we will then compare the $k$-medoids algorithms against one another, including the baseline clusterers, to determine the best-performing $k$-medoids approach for TSCL.

Many of the $k$-medoids algorithms share common parameters. While the specific configurations for each elastic distance model will be defined individually in their respective sections, Table 6.1 provides a general overview of the configuration used across all $k$-medoids models. Our goal is to maintain consistency in as many parameters as possible, altering only the distance measure. This approach ensures that we are evaluating the performance of the distance measures themselves, rather than differences in model configuration.

| | $max\_iters$ | $n\_init$ | $init\_algo$ | additional |
|---|---|---|---|---|
| PAM | 50 | 10 | Forgy | - |
| alternate $k$-medoids | 50 | 10 | Forgy | $tol = 1 \times 10^{-6}$ |
| CLARA | 50 | 10 | Forgy | $n\_samples = 40 + 2k$ |
| CLARANS | 50 | 10 | Forgy | $max\_neighbours = 0.0125(k(n-k))$ |

Table 6.1 Baseline $k$-medoids models parameters. A “-” means the parameter does not apply to the model.

In addition to the model-specific parameters, the elastic distances require default parameter values to be set. Table 5.2 outlines the parameters used for each elastic distance, which are the same as those employed in Chapter 5. Based on our experimentation in Chapter 5, these parameters serve as good default parameters as the performance of each distance was aligned with our expectations.

## 6.4 Alternate *k*-medoids

For our experiment each alternate *k*-medoids model configuration is defined in Table 6.2.

| | *max_iters* | *tol* | *n_init* | *init_algo* | *distance* |
|---|---|---|---|---|---|
| alternate-adtw | 50 | $1\times10^{-6}$ | 10 | Forgy | ADTW |
| alternate-ddtw | 50 | $1\times10^{-6}$ | 10 | Forgy | DDTW |
| alternate-dtw | 50 | $1\times10^{-6}$ | 10 | Forgy | DTW |
| alternate-edr | 50 | $1\times10^{-6}$ | 10 | Forgy | EDR |
| alternate-erp | 50 | $1\times10^{-6}$ | 10 | Forgy | ERP |
| alternate-euclidean | 50 | $1\times10^{-6}$ | 10 | Forgy | Euclidean |
| alternate-lcss | 50 | $1\times10^{-6}$ | 10 | Forgy | LCSS |
| alternate-msm | 50 | $1\times10^{-6}$ | 10 | Forgy | MSM |
| alternate-twe | 50 | $1\times10^{-6}$ | 10 | Forgy | TWE |
| alternate-wddtw | 50 | $1\times10^{-6}$ | 10 | Forgy | WDDTW |
| alternate-wdtw | 50 | $1\times10^{-6}$ | 10 | Forgy | WDTW |
| alternate-shape-dtw | 50 | $1\times10^{-6}$ | 10 | Forgy | shape-DTW |
| alternate-soft-dtw | 50 | $1\times10^{-6}$ | 10 | Forgy | soft-DTW |

Table 6.2 Alternate *k*-medoids model parameters

### 6.4.1 Alternate *k*-medoids Combined test-train split

Figure 6.5 presents the combined test-train split results for alternate *k*-medoids with all elastic distances. However, 36 datasets are missing from this evaluation. The primary reason for this is that certain distances did not complete within the seven-day runtime limit. Specifically, LCSS did not finish for 30 datasets, EDR for 9 datasets, and soft-DTW for 10 datasets. A comprehensive list of the missing datasets is provided in Table A.12.

The number of missing datasets specifically for LCSS is surprisingly high. In previous experiments, LCSS did not exhibit extended convergence times. This discrepancy may suggest a potential issue within our code, although no such issue was observed in the test-train split. Another possible explanation is that LCSS may

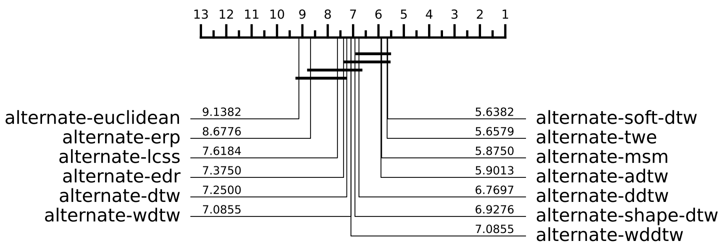


Fig. 6.1 AMI

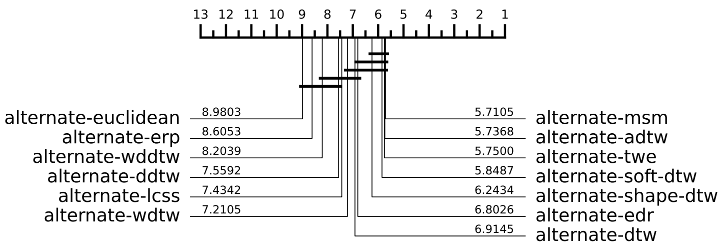


Fig. 6.2 ARI

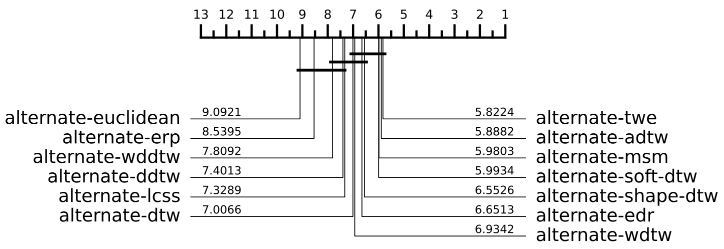


Fig. 6.3 CLACC

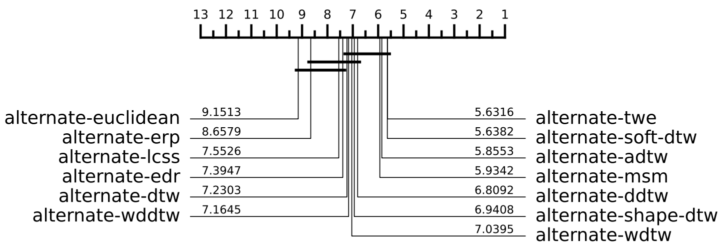


Fig. 6.4 NMI

Fig. 6.5 CD diagrams of alternate *k*-medoids over 76 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.12.

struggle to effectively form medoids, leading the algorithm to consistently reaching the maximum number of iterations, thereby significantly increasing the runtime.

Due to the total of 36 missing datasets for the combined test-train split, it is challenging to draw broad and meaningful conclusions. Therefore, we present the initial results with all elastic distances but then choose to exclude LCSS, EDR and soft-DTW due to multiple missing datasets. Therefore, we will briefly analyse these three distances for the results that have finished and then exclude them from the rest of the results.

Alternate-LCSS always appears in the bottom clique. While it is always better in terms of average rank than alternate-Euclidean it is not significantly better. Alternate-EDR seems to have the most variable rankings. For AMI and NMI EDR appears in the bottom clique. However, for CLACC it appears in the top clique and appears in the second best clique for ARI. This discrepancy may suggest that while there are many pairs of time series accurately assigned, the overall distribution of the values maybe poor and there could be a lot of overlap between clusters.

Alternate-soft-DTW is one of the best performing elastic distances always appear in the top clique. We will focus our evaluation of alternate-soft-DTW when analysing the test-train split as it has completed many more datasets allowing more meaningful analysis. However, for the limited number it has completed for the combined test-train split it is one of the best performing distances.

We now exclude alternate-LCSS, alternate-EDR, and alternate-soft-DTW from our analysis. Figure 6.10 presents the critical difference diagrams for nine different elastic distances across 106 datasets. For all evaluation metrics, alternate-TWE achieves the highest average rank, followed by alternate-MSM and alternate-ADTW.

In Figure 6.10, for every evaluation metric, all elastic distances outperform alternate-Euclidean, except ERP for CLACC. TWE, MSM, ADTW, and shape-DTW consistently feature in the top cliques, whereas Euclidean, ERP, DTW, WDTW, and WDDTW are always in the bottom clique. Additionally, in a surprising result, DDTW appears in the top clique for AMI and NMI, marking the first time in our experiments that DDTW has been ranked so highly.

Table 6.3 provides the average scores for each model across all evaluation metrics. In terms of average score, alternate-TWE is the best-performing model for every metric. However, across different domains (Table 6.4), alternate-TWE ranks highest in only two out of the seven domains. Alternate-ADTW, alternate-MSM, alternate-shape-DTW, and alternate-WDTW each perform best in one or more categories.

To contextualise the results, Figure 6.15 displays the critical difference diagrams with the baseline clusterers included. *k*-means-ba-DTW is the best-performing clusterer across all evaluation metrics. However, alternate-TWE, alternate-MSM, alternate-ADTW, and alternate-shape-DTW consistently appear in the top clique alongside it. Moreover, for every evaluation metric, alternate-MSM, alternate-TWE,

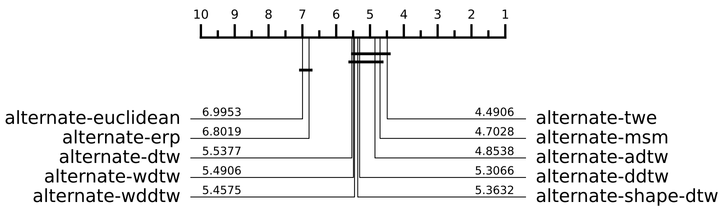


Fig. 6.6 AMI

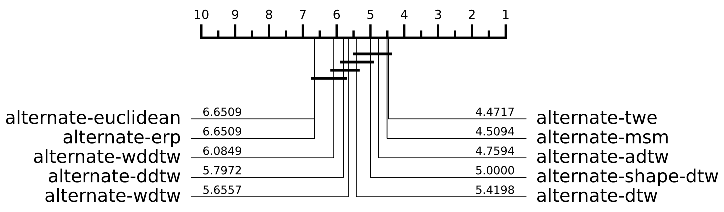


Fig. 6.7 ARI

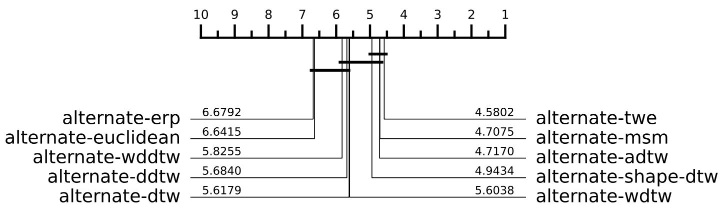


Fig. 6.8 CLACC

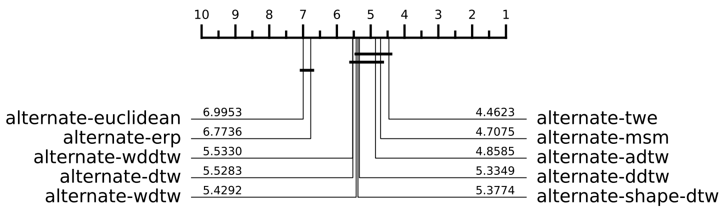


Fig. 6.9 NMI

Fig. 6.10 CD diagrams of alternate *k*-medoids over 106 datasets from the UCR archive using the combined test-train split. Missing datasets are outline in Table A.13.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| alternate-adtw | 0.248 | 0.302 | 0.559 | 0.327 | 0.715 |
| alternate-ddtw | 0.205 | 0.271 | 0.535 | 0.294 | 0.693 |
| alternate-dtw | 0.232 | 0.286 | 0.550 | 0.311 | 0.707 |
| alternate-erp | 0.181 | 0.235 | 0.508 | 0.262 | 0.685 |
| alternate-euclidean | 0.185 | 0.236 | 0.509 | 0.263 | 0.694 |
| alternate-msm | 0.246 | 0.300 | 0.562 | 0.324 | 0.713 |
| alternate-shape-dtw | 0.245 | 0.296 | 0.558 | 0.320 | 0.712 |
| alternate-twe | **0.253** | **0.306** | **0.565** | **0.330** | **0.716** |
| alternate-wddtw | 0.208 | 0.276 | 0.536 | 0.300 | 0.690 |
| alternate-wdtw | 0.231 | 0.287 | 0.550 | 0.313 | 0.705 |

Table 6.3 Summary of average score across multiple evaluation metrics over 106 datasets from the UCR archive using the combined test-train split.

alternate-ADTW, and alternate-shape-DTW outperform *k*-shapes. Overall, while no alternate *k*-medoids approach surpasses the current state-of-the-art, many of the elastic distances are not significantly different from it.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| alternate-adtw | 0.290 | **0.232** | 0.218 | 0.468 | 0.118 | **0.166** | 0.324 |
| alternate-ddtw | 0.291 | 0.164 | 0.212 | 0.242 | 0.091 | 0.096 | 0.250 |
| alternate-dtw | 0.284 | 0.178 | 0.199 | 0.452 | 0.149 | 0.154 | 0.261 |
| alternate-erp | 0.232 | 0.153 | 0.176 | 0.255 | 0.088 | 0.098 | 0.268 |
| alternate-euclidean | 0.243 | 0.197 | 0.178 | 0.251 | 0.045 | 0.099 | 0.260 |
| alternate-msm | **0.323** | 0.169 | 0.217 | 0.346 | 0.174 | 0.148 | 0.390 |
| alternate-shape-dtw | 0.298 | 0.196 | 0.226 | 0.372 | 0.125 | 0.154 | **0.418** |
| alternate-twe | 0.321 | 0.173 | **0.228** | 0.440 | **0.188** | 0.146 | 0.325 |
| alternate-wddtw | 0.315 | 0.163 | 0.203 | 0.264 | 0.063 | 0.089 | 0.252 |
| alternate-wdtw | 0.294 | 0.180 | 0.196 | **0.487** | 0.076 | 0.157 | 0.267 |

Table 6.4 Average ARI score on problems split by problem domain over 106 datasets from the UCR archive using the combined test-train split.

### 6.4.2 Alternate *k*-medoids Test-train split

Figure 6.20 presents the critical difference diagram for alternate *k*-medoids over 102 of the UCR test-train splits. The general ranking order remains consistent with the combined test-train split results. Alternate-soft-DTW, alternate-ADTW, alternate-TWE, alternate-MSM, and alternate-shape-DTW consistently appear in the top clique across all evaluation metrics. In contrast, alternate-ERP, alternate-LCSS, alternate-WDDTW, and alternate-Euclidean are consistently placed in the bottom clique. However, on the test-train split, both alternate-ERP and alternate-LCSS perform worse than alternate-Euclidean for every evaluation metric, which was not the case in the combined test-train split, where alternate-ERP only performed worse for one dataset.

Table 6.5 summarises the average performance of each clusterer across all evaluation metrics. Alternate-soft-DTW is the best-performing clusterer overall, excelling in three out of the seven domains. Alternate-MSM performs best in two domains, alternate-ADTW in one, and alternate-DTW in one. Notably, this is the first time alternate-DTW has been the top-performing distance for any domain.

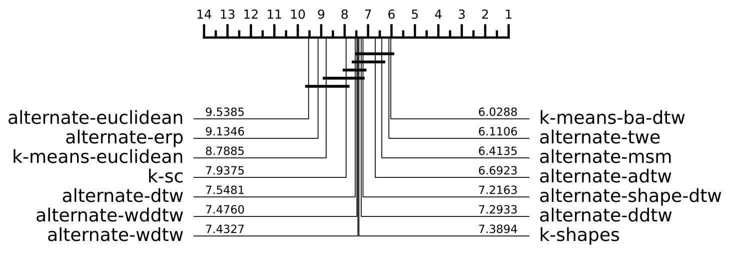


Fig. 6.11 AMI

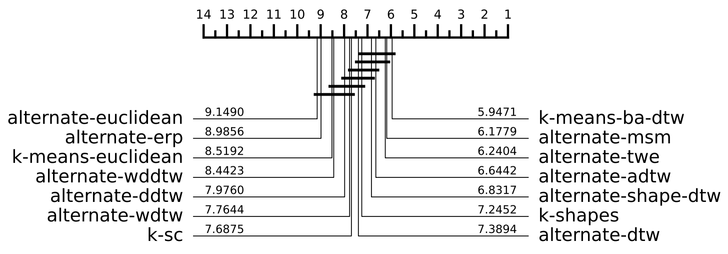


Fig. 6.12 ARI

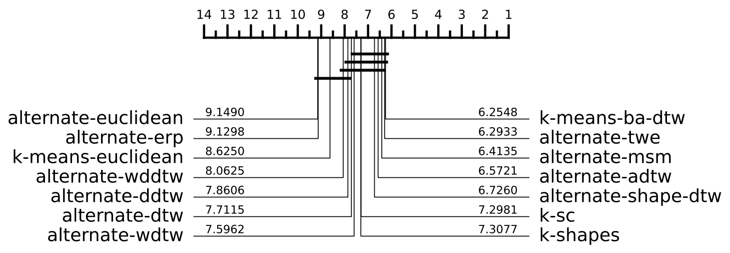


Fig. 6.13 CLACC

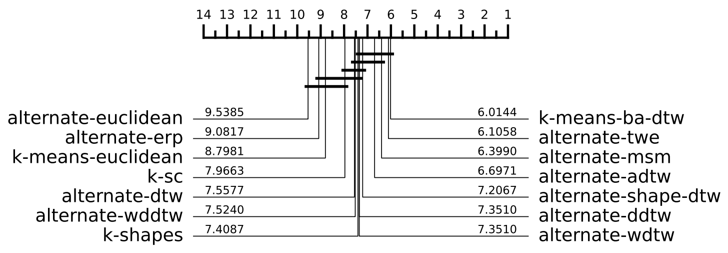


Fig. 6.14 NMI

Fig. 6.15 CD diagrams of alternate *k*-medoids with baseline clusterers over 104 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.14

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| alternate-adtw | 0.223 | 0.279 | 0.564 | 0.312 | 0.701 |
| alternate-ddtw | 0.194 | 0.255 | 0.538 | 0.287 | 0.684 |
| alternate-dtw | 0.197 | 0.257 | 0.536 | 0.292 | 0.692 |
| alternate-edr | 0.194 | 0.242 | 0.533 | 0.275 | 0.688 |
| alternate-erp | 0.170 | 0.220 | 0.509 | 0.257 | 0.674 |
| alternate-euclidean | 0.177 | 0.228 | 0.520 | 0.264 | 0.687 |
| alternate-lcss | 0.141 | 0.198 | 0.498 | 0.233 | 0.639 |
| alternate-msm | 0.214 | 0.270 | 0.545 | 0.303 | 0.697 |
| alternate-shape-dtw | 0.227 | 0.277 | 0.561 | 0.311 | 0.702 |
| alternate-soft-dtw | **0.239** | **0.294** | **0.569** | **0.327** | **0.706** |
| alternate-twe | 0.213 | 0.268 | 0.548 | 0.301 | 0.697 |
| alternate-wddtw | 0.180 | 0.239 | 0.525 | 0.274 | 0.676 |
| alternate-wdtw | 0.194 | 0.254 | 0.533 | 0.289 | 0.689 |

Table 6.5 Summary of average score across multiple evaluation metrics over 102 datasets from the UCR archive using the test-train split.

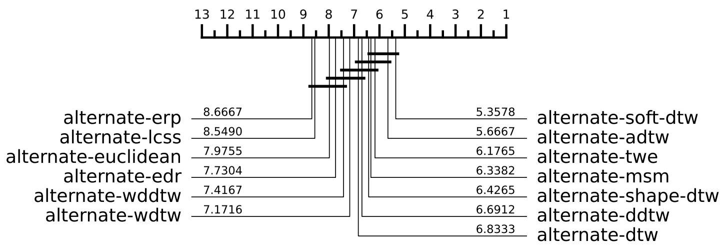


Fig. 6.16 AMI

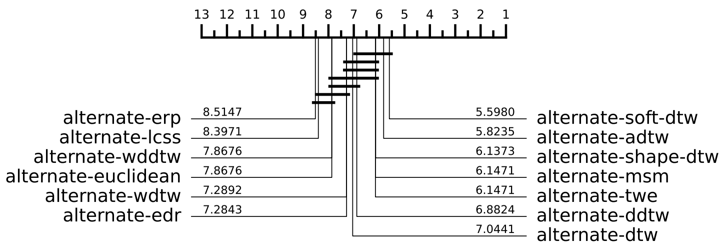


Fig. 6.17 ARI

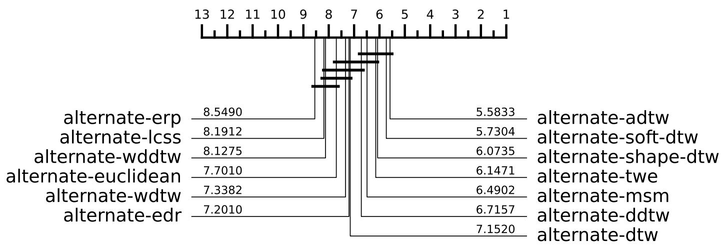


Fig. 6.18 CLACC

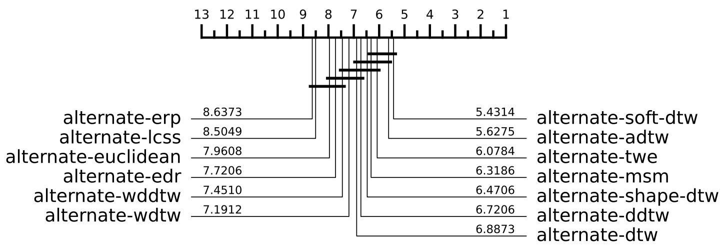


Fig. 6.19 NMI

Fig. 6.20 CD diagrams of alternate *k*-medoids over 102 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.15.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| alternate-adtw | 0.280 | 0.154 | 0.169 | 0.335 | 0.146 | **0.213** | 0.315 |
| alternate-ddtw | 0.255 | 0.120 | 0.182 | 0.257 | 0.105 | 0.145 | 0.249 |
| alternate-dtw | 0.211 | **0.163** | 0.166 | 0.385 | 0.119 | 0.185 | 0.217 |
| alternate-edr | 0.255 | 0.123 | 0.182 | 0.311 | 0.121 | 0.121 | 0.211 |
| alternate-erp | 0.216 | 0.087 | 0.179 | 0.215 | 0.048 | 0.131 | 0.263 |
| alternate-euclidean | 0.225 | 0.151 | 0.158 | 0.228 | 0.064 | 0.140 | 0.275 |
| alternate-lcss | 0.133 | 0.041 | 0.158 | 0.260 | 0.070 | 0.143 | 0.185 |
| alternate-msm | 0.273 | 0.152 | 0.178 | 0.313 | 0.103 | 0.179 | 0.320 |
| alternate-shape-dtw | 0.260 | 0.136 | **0.200** | 0.349 | 0.150 | 0.195 | **0.378** |
| alternate-soft-dtw | **0.283** | 0.156 | 0.189 | **0.468** | **0.172** | 0.206 | 0.273 |
| alternate-twe | 0.281 | 0.121 | 0.187 | 0.273 | 0.132 | 0.160 | 0.341 |
| alternate-wddtw | 0.272 | 0.131 | 0.172 | 0.212 | 0.054 | 0.086 | 0.249 |
| alternate-wdtw | 0.212 | 0.160 | 0.171 | 0.374 | 0.070 | 0.182 | 0.225 |

Table 6.6 Average ARI score on problems split by problem domain. 102 datasets from the UCR archive using the test-train split.

Figure 6.25 presents the critical difference diagram for alternate *k*-medoids with the baseline clusterers over the test-train split. Across all evaluation metrics, alternate-soft-DTW and alternate-ADTW emerge as the top two performing alternate *k*-medoids clusterers, consistently outperforming *k*-means-ba-DTW. While *k*-means-ba-DTW remains in the top clique for both ARI and CLACC, it is surpassed on average by alternate-MSM, alternate-TWE, alternate-shape-DTW, alternate-soft-DTW, and alternate-ADTW. Alternate-ERP and alternate-LCSS, on the other hand, are consistently in the bottom clique and are the only two elastic distances that perform worse than *k*-means-Euclidean.

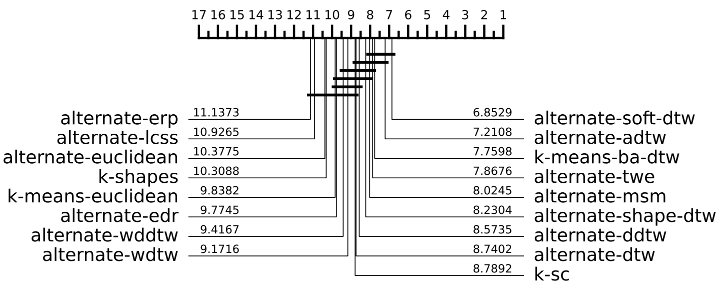


Fig. 6.21 AMI

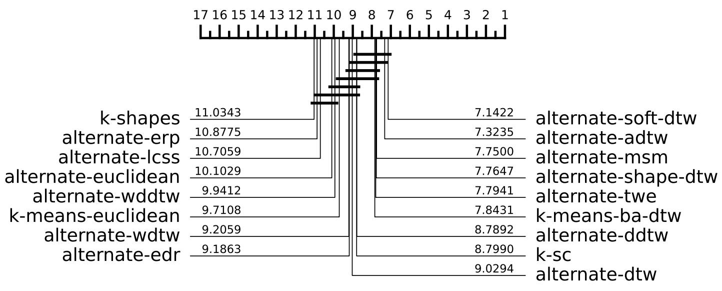


Fig. 6.22 ARI

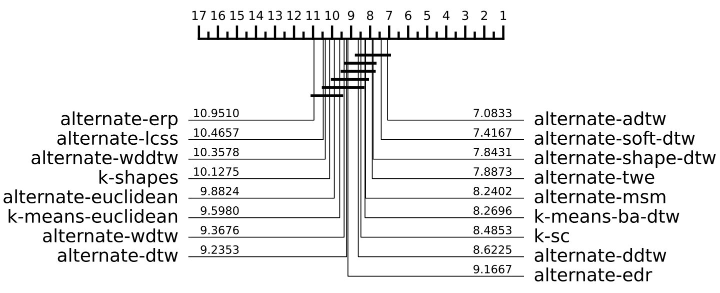


Fig. 6.23 CLACC

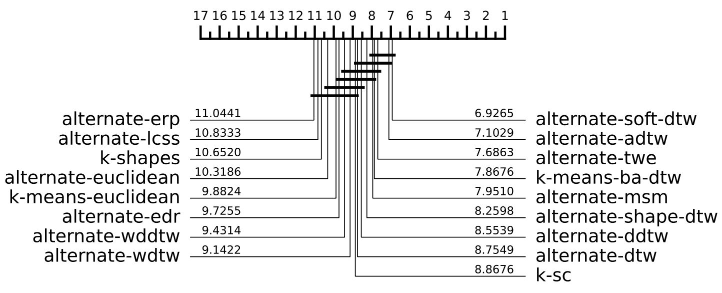


Fig. 6.24 NMI

Fig. 6.25 CD diagrams of alternate *k*-medoids over 102 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.16

### 6.4.3 Comparison to *k*-means

One of our hypothesis was that *k*-medoids algorithms would outperform *k*-means across all elastic distances. Secondly, we proposed that using medoids as centroids would result in similar performance improvements across all distances when compared to *k*-means. We will now test these hypotheses by directly comparing the results of *k*-means with those of alternate *k*-medoids.

Alternate *k*-medoids employs Lloyd's algorithm, similar to *k*-means. As previously outlined, the key difference between alternate *k*-medoids and *k*-means lies in how the centroids are computed. Alternate *k*-medoids uses medoids (which can utilise elastic distances), while *k*-means relies on the arithmetic mean. Comparing alternate *k*-medoids with *k*-means enables us to directly assess the impact of using medoids as cluster centroids.

Figure 6.30 presents the critical difference diagram comparing the performance of alternate *k*-medoids and *k*-means. In this comparison, all elastic distances perform better with alternate *k*-medoids than with *k*-means. On average, we observe a rank improvement of approximately 2.22 when using alternate *k*-medoids instead of *k*-means with the same elastic distance. This is shown in Figure 6.31.

However, while alternate *k*-medoids with elastic distances outperform *k*-means with the same distances, *k*-means-ADTW and *k*-means-MSM still appear in the top clique for ARI, CLACC, and NMI. These two methods are only consistently outperformed by alternate-TWE, alternate-MSM, and alternate-ADTW.

Figure 6.32 compares the difference in average score for each elastic distance in terms of ARI and AMI. The numbers shown above each distances bars shows the difference between scores of alternate *k*-medoids (blue) and *k*-means (red). For both ARI and AMI, the average score improves for all distances except ERP and Euclidean. On average alternate *k*-medoids improves the clusterers performance by 0.031 for ARI and 0.037 AMI.

Figure 6.32 shows that DDTW, DTW and EDR exhibit the most significant improvements for both AMI and ARI. These distances have an implicit warping penalty. In Chapter 5, we hypothesised that elastic distances with implicit warping penalties are more susceptible to pathological warping, especially when compared to an unaligned centroid. Based on Figure 6.32, we theorise that when the centroid

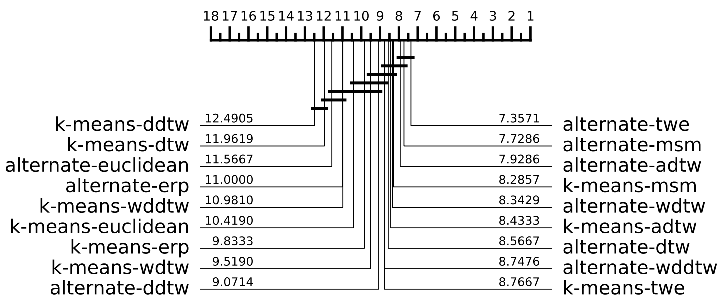


Fig. 6.26 AMI

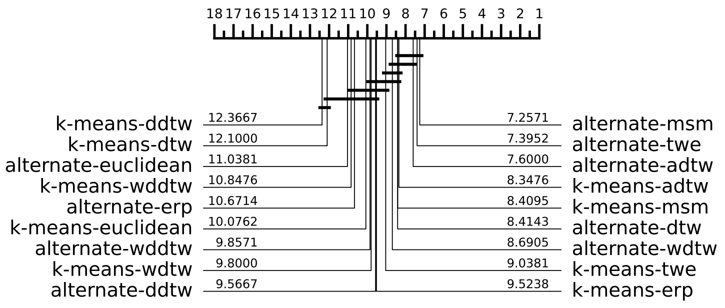


Fig. 6.27 ARI

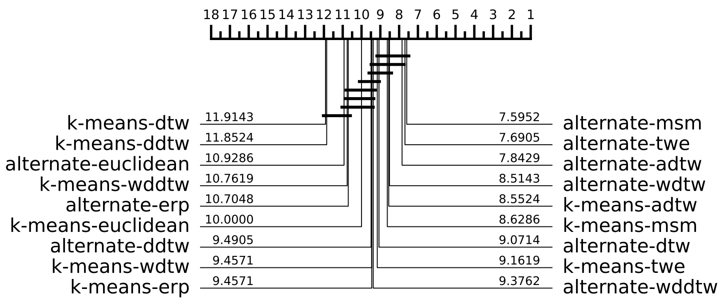


Fig. 6.28 CLACC

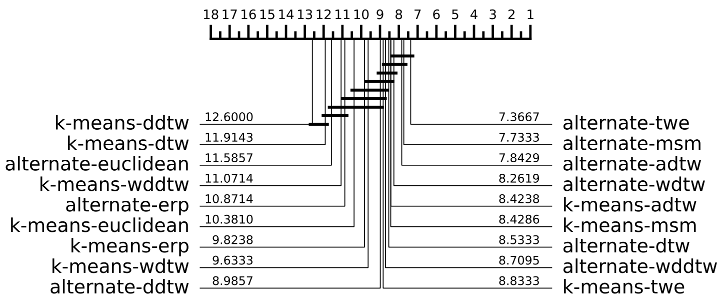


Fig. 6.29 NMI

Fig. 6.30 CD diagrams of alternate *k*-medoids and *k*-means over 105 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.17

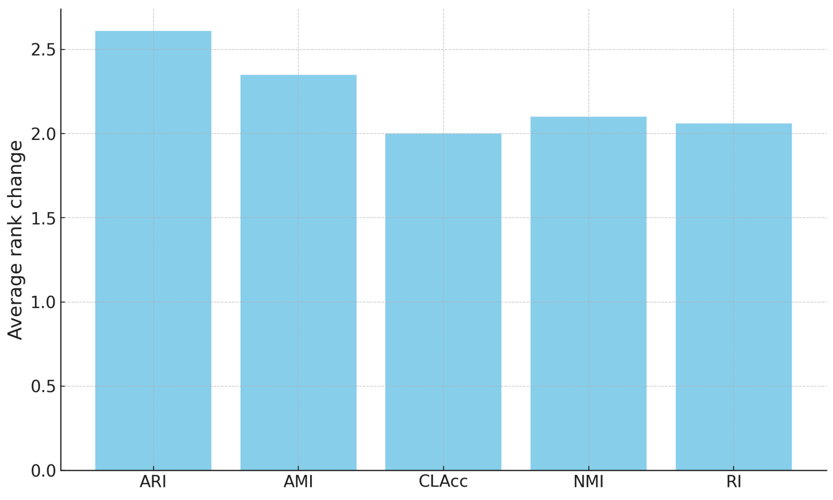
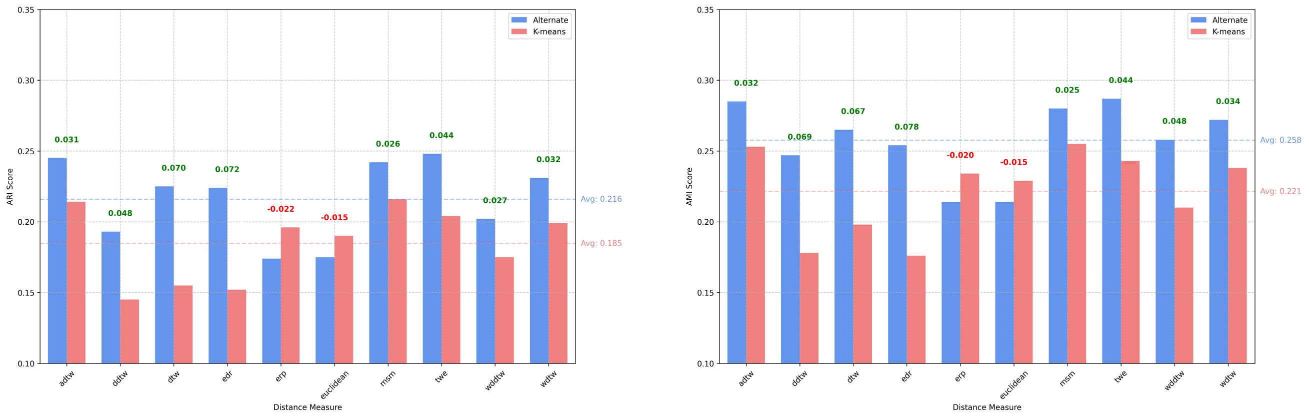


Fig. 6.31 Average rank improvement for each elastic distance when using alternate *k*-medoids over *k*-means over 105 datasets from the UCR archive combined test-train split.

is computed with alignment, less pathological warping occurs, leading to a general improvement for all distances, but particularly for those with an implicit warping penalty, such as DTW, DDTW and EDR.



(a) ARI (b) AMI

Fig. 6.32 Comparison of the performance of alternate *k*-medoids and *k*-means using various elastic distances across 105 datasets from the UCR archive, evaluated on the combined test-train split. The blue bars represent the scores for alternate *k*-medoids, while the red bars indicate the scores for *k*-means. The dashed blue and red lines denote the average scores for alternate *k*-medoids and *k*-means, respectively.

Figure 6.37 presents the comparison between alternate *k*-medoids and *k*-means for the test-train split. In this comparison, we observe that alternate *k*-medoids improves clustering performance for many elastic distances. However, the improvement is not as significant as in the combined test-train split comparison. Additionally, for some evaluation metrics, alternate *k*-medoids performs worse than *k*-means with the same elastic distance. For instance, Figure 6.37 shows that across all four evaluation metrics, *k*-means-MSM consistently achieves a higher average rank than alternate-MSM.

Figure 6.38 shows the ARI and AMI performance for each distance using alternate *k*-medoids and *k*-means over the test-train split. On average, alternate *k*-medoids improved the ARI performance by 0.007. For AMI, the improvement was 0.012. This represents a notably smaller improvement than observed in the combined test-train split.

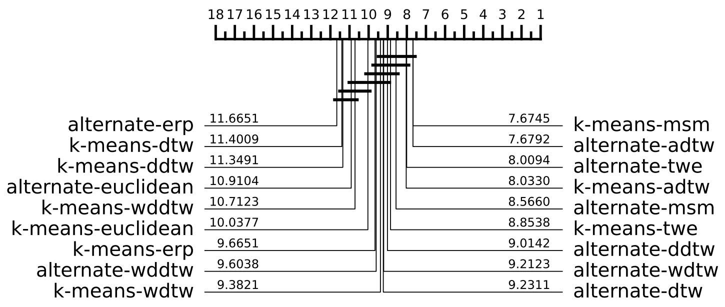


Fig. 6.33 AMI

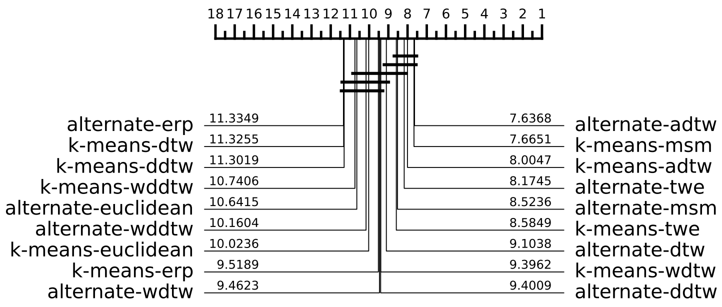


Fig. 6.34 ARI

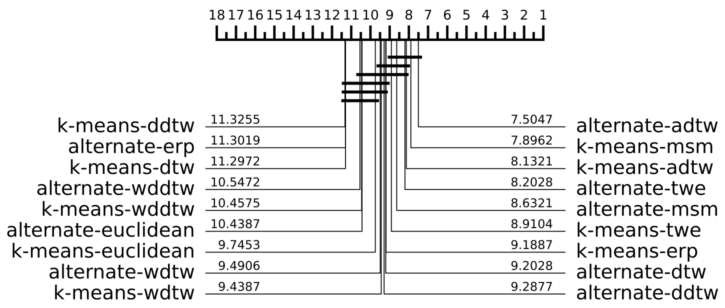


Fig. 6.35 CLACC

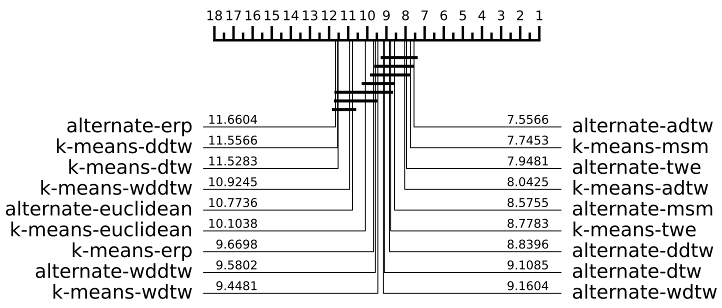
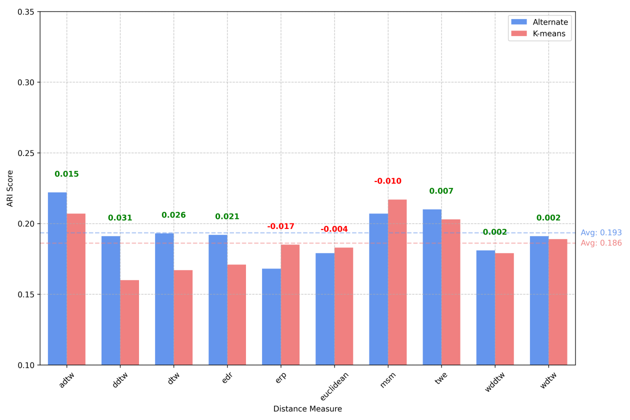


Fig. 6.36 NMI

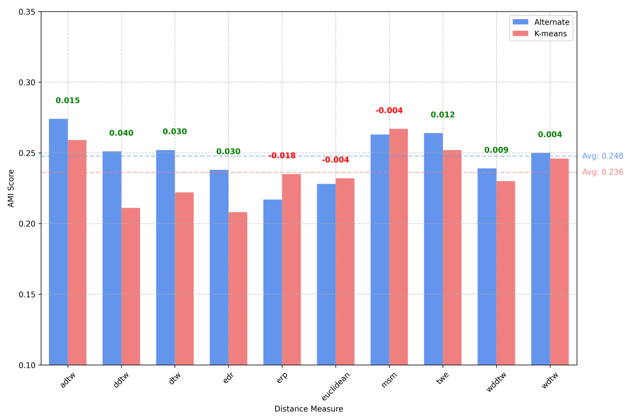
Fig. 6.37 CD diagrams of alternate k-medoids and k-means over 106 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.18.

Similar to the combined test-train split, alternate DTW, DDTW and EDR show the greatest improvement compared to *k*-means. This further supports our theory that distances with an implicit warping penalty depend more on an aligned centroid for improved performance than those with an explicit penalty.

Additionally, Figure 6.38 shows that alternate-ERP performs worse than *k*-means-ERP, consistent with the combined test-train split results. However, for ARI, alternate-MSM perform worse than *k*-means-MSM. This was not observed in the combined test-train split. Though the difference is only $-0.010$ for ARI and $-0.004$ for AMI, in the combined test-train split, alternate-MSM outperformed *k*-means-MSM.



(a) ARI



(b) AMI

Fig. 6.38 Comparison of the performance of alternate *k*-medoids and *k*-means using various elastic distances across 106 datasets from the UCR archive, evaluated on the test-train split. The blue bars represent the scores for alternate *k*-medoids, while the red bars indicate the scores for *k*-means. The dashed blue and red lines denote the average scores for alternate *k*-medoids and *k*-means, respectively.

Overall, for both the combined test-train split and the test-train split, alternate *k*-medoids improves clustering performance compared to *k*-means when using the same elastic distance. This improvement is particularly evident for distances that use an implicit warping penalty. However, the improvement observed in the test-train split is significantly smaller, on average, than in the combined test-train split. We hypothesise that this is due to the reduced amount of data available in the test-train split, which lowers the likelihood of finding good medoids in the training data. As a result, performance degrades, and the improvement is substantially reduced.

### 6.4.4 Alternate *k*-medoids conclusion

Overall, for both the combined test-train split and the test-train split, we observe that elastic distances enhance the performance of alternate *k*-medoids compared to alternate-Euclidean. Additionally, in both splits, the best-performing alternate models perform similarly to, and in some evaluation metrics, even better than the current state-of-the-art approaches.

Furthermore, we directly compared alternate $k$-medoids with elastic distances to $k$-means using the same elastic distances. In the combined test-train split, alternate $k$-medoids improved clustering by an average of 0.027 ARI and 0.032 AMI compared to $k$-means using the same distance. A similar trend was observed in the test-train split, though to a lesser extent, where alternate $k$-medoids improved clustering by an average of 0.0046 ARI and 0.0089 AMI compared to $k$-means.

At the beginning of this section, we also hypothesised that the performance improvement compared to $k$-means using the same elastic distance would be consistent across distances. However, this was not the case. Specifically, distances with implicit warping penalties (e.g., DTW and DDTW) benefited significantly more from using medoids than those with explicit warping penalties (e.g., MSM, TWE, ADTW). This difference was particularly pronounced in the test-train split results.

## 6.5 PAM

For our experiments each PAM model configuration is defined in Table 6.2.

| | $max\_iters$ | $n\_init$ | $init\_algo$ | $distance$ |
|---|---|---|---|---|
| PAM-adtw | 50 | 10 | Forgy | ADTW |
| PAM-ddtw | 50 | 10 | Forgy | DDTW |
| PAM-dtw | 50 | 10 | Forgy | DTW |
| PAM-edr | 50 | 10 | Forgy | EDR |
| PAM-erp | 50 | 10 | Forgy | ERP |
| PAM-euclidean | 50 | 10 | Forgy | Euclidean |
| PAM-lcss | 50 | 10 | Forgy | LCSS |
| PAM-msm | 50 | 10 | Forgy | MSM |
| PAM-twe | 50 | 10 | Forgy | TWE |
| PAM-wddtw | 50 | 10 | Forgy | WDDTW |
| PAM-wdtw | 50 | 10 | Forgy | WDTW |
| PAM-shape-dtw | 50 | 10 | Forgy | shape-DTW |
| PAM-soft-dtw | 50 | 10 | Forgy | soft-DTW |

Table 6.7 PAM model parameters

### 6.5.1 PAM Combined test-train split

Figure 6.43 shows the critical difference diagram for PAM with 12 different elastic distances. In Figure 6.43, all elastic distances outperform PAM-Euclidean for all evaluation metrics. In addition, across all evaluation metrics, PAM-TWE and PAM-MSM consistently appear in the top clique. PAM-soft-DTW, PAM-ADTW, and PAM-EDR are in the top clique for three out of the four evaluation metrics. While all elastic distances perform better than PAM-Euclidean, PAM-LCSS and PAM-ERP consistently appear in the bottom clique alongside PAM-Euclidean across all evaluation metrics.

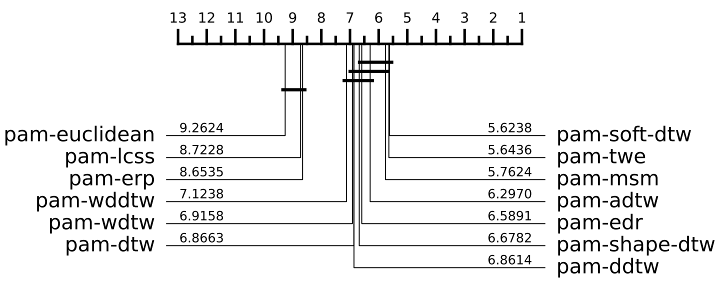


Fig. 6.39 AMI

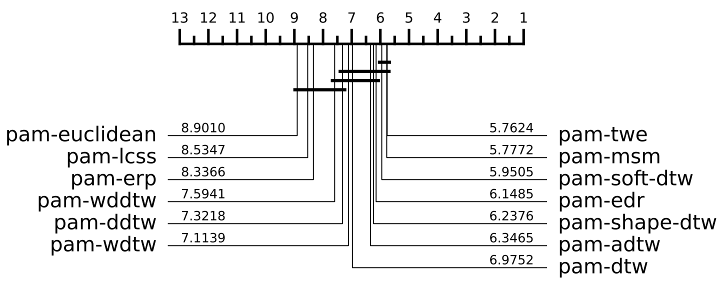


Fig. 6.40 ARI

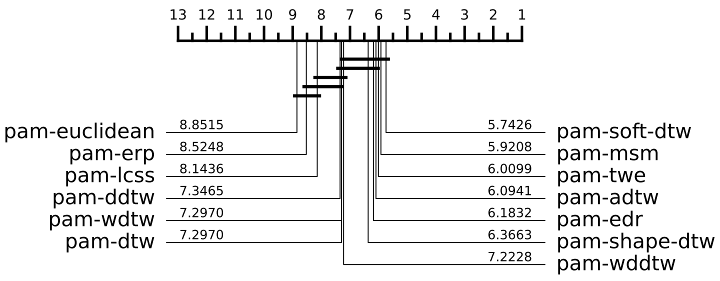


Fig. 6.41 CLACC

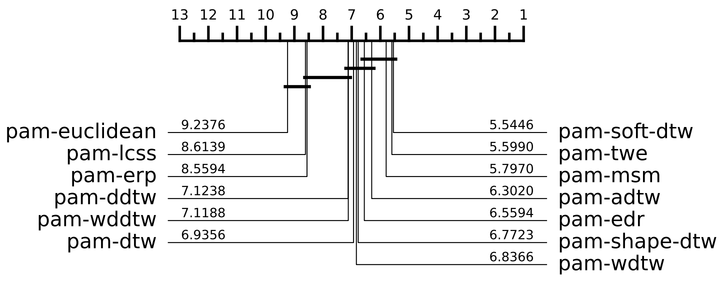


Fig. 6.42 NMI

Fig. 6.43 CD diagrams of PAM over 101 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.19.

Table 6.8 shows the average score for each evaluation metric. For all metrics, PAM-TWE achieves the highest average score. However, when assessing performance by domain (Table 6.9), PAM-TWE only performs best in the Device domain.

Furthermore, no single distance consistently performs best over the the majority of domains. PAM-ADTW performs best in two categories which is the most categories one distance performs best in. This suggests that even though for all the distances the same potential medoids are available, no distance chooses the same medoids.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| pam-adtw | 0.244 | 0.293 | 0.562 | 0.317 | 0.705 |
| pam-ddtw | 0.206 | 0.264 | 0.541 | 0.284 | 0.681 |
| pam-dtw | 0.225 | 0.276 | 0.550 | 0.301 | 0.694 |
| pam-edr | 0.231 | 0.267 | 0.562 | 0.289 | 0.690 |
| pam-erp | 0.177 | 0.224 | 0.512 | 0.252 | 0.681 |
| pam-euclidean | 0.177 | 0.222 | 0.507 | 0.250 | 0.684 |
| pam-lcss | 0.153 | 0.200 | 0.507 | 0.224 | 0.631 |
| pam-msm | 0.255 | 0.299 | 0.573 | 0.322 | 0.711 |
| pam-shape-dtw | 0.243 | 0.290 | 0.564 | 0.314 | 0.703 |
| pam-soft-dtw | 0.251 | 0.300 | 0.571 | 0.324 | 0.703 |
| pam-twe | **0.262** | **0.304** | **0.579** | **0.328** | **0.714** |
| pam-wddtw | 0.203 | 0.258 | 0.541 | 0.283 | 0.680 |
| pam-wdtw | 0.222 | 0.273 | 0.549 | 0.299 | 0.692 |

Table 6.8 Summary of average score across multiple evaluation metrics over 101 datasets from the UCR archive using the combined test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| pam-adtw | 0.312 | **0.232** | 0.229 | 0.368 | 0.106 | **0.171** | 0.291 |
| pam-ddtw | 0.326 | 0.170 | 0.230 | 0.170 | 0.084 | 0.083 | 0.162 |
| pam-dtw | 0.284 | 0.191 | 0.208 | 0.409 | 0.138 | 0.160 | 0.149 |
| pam-edr | 0.341 | 0.101 | **0.254** | 0.304 | 0.146 | 0.124 | 0.206 |
| pam-erp | 0.248 | 0.182 | 0.175 | 0.165 | 0.093 | 0.100 | 0.187 |
| pam-euclidean | 0.247 | 0.217 | 0.180 | 0.155 | 0.064 | 0.099 | 0.152 |
| pam-lcss | 0.154 | 0.055 | 0.190 | 0.241 | 0.064 | 0.146 | 0.282 |
| pam-msm | **0.355** | 0.183 | 0.238 | 0.319 | 0.173 | 0.150 | 0.356 |
| pam-shape-dtw | 0.307 | 0.209 | 0.234 | 0.344 | 0.123 | 0.159 | **0.397** |
| pam-soft-dtw | 0.340 | 0.137 | 0.240 | **0.478** | 0.165 | 0.165 | 0.185 |
| pam-twe | 0.351 | 0.196 | 0.240 | 0.413 | **0.181** | 0.148 | 0.351 |
| pam-wddtw | 0.337 | 0.172 | 0.198 | 0.191 | 0.081 | 0.083 | 0.172 |
| pam-wdtw | 0.283 | 0.203 | 0.193 | 0.470 | 0.076 | 0.164 | 0.172 |

Table 6.9 Average ARI score on problems split by problem domain over 101 datasets from the UCR archive using the combined test-train split.

To contextualise our PAM results, Figure 6.48 shows the critical difference diagrams for PAM with 12 elastic distances compared to the baseline clusterers. In Figure 6.48 PAM-MSM consistently outperforms *k*-means-ba-DTW in average rank across all evaluation metrics, although the difference is not statistically significant for any metric. For all evaluation metrics, PAM-MSM, PAM-soft-DTW, PAM-TWE, *k*-means-ba-DTW, PAM-ADTW, PAM-EDR, and PAM-shape-DTW consistently appear in the top clique. Interestingly, *k*-shapes, which for previous experiments always appeared in the top clique for all evaluation metrics, only appears in the top clique for CLACC and ARI.

Furthermore, in Figure 6.48, all clusterers outperform PAM-Euclidean. However, PAM-ERP is outperformed by *k*-means-Euclidean for CLACC, and PAM-LCSS is outperformed by *k*-means-Euclidean for AMI, ARI, and NMI. PAM-Euclidean, PAM-LCSS, *k*-means-Euclidean, and PAM-ERP consistently appear in the bottom clique.

Another notable observation is that PAM-DTW performs significantly worse than *k*-means-ba-DTW in three out of the four evaluation metrics, only appearing in the same clique for ARI. This suggests that, across the UCR archive, averaging-based centroid methods may outperform medoid-based centroid methods. We will explore this hypothesis further later in the chapter.

Due to the 11 missing datasets in the initial comparison of PAM with all elastic distances, we exclude models with more than 5 missing datasets to ensure that the missing data does not significantly affect our findings. As a result, we exclude PAM-soft-DTW and PAM-shape-DTW, which failed to produce results for 6 and 11 datasets, respectively, due to exceeding the seven-day runtime limit. Figure 6.53 presents the critical difference diagrams for PAM across 105 datasets. The rank order of the distances remains consistent with previous results, even with the additional datasets.

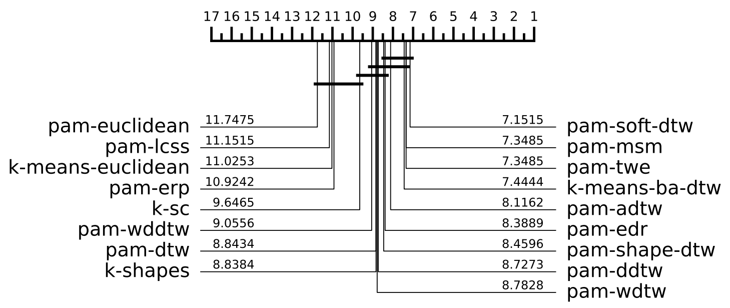


Fig. 6.44 AMI

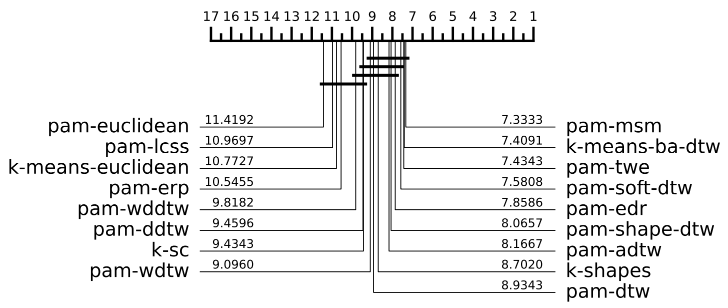


Fig. 6.45 ARI

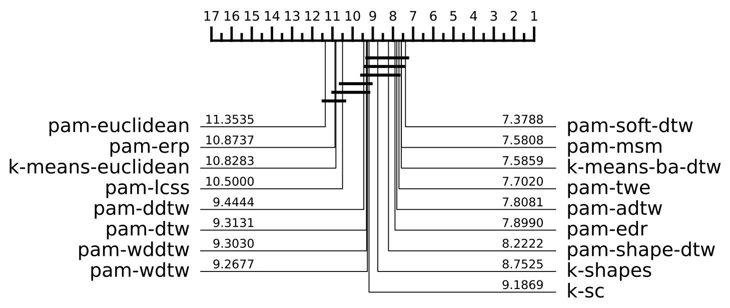


Fig. 6.46 CLACC

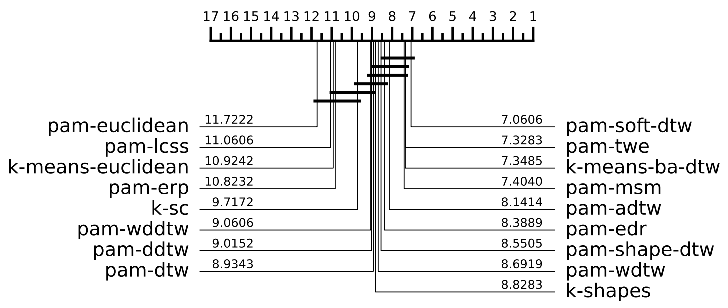


Fig. 6.47 NMI

Fig. 6.48 CD diagrams of PAM with baseline clusterers over 99 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.20.

Figure 6.54 shows scatter plots comparing the results of PAM-MSM and PAM-TWE directly against *k*-means-ba-DTW. Both PAM-MSM and PAM-TWE perform similarly, winning a comparable number of datasets, and often performing well on the same datasets. Additionally, for datasets where both methods excel, their performance is nearly identical, suggesting that PAM-MSM and PAM-TWE likely find many of the same medoids.

An interesting observation is that, compared to *k*-means-ba-DTW, both PAM-MSM and PAM-TWE exhibit significantly better results for some datasets, but significantly worse performance than *k*-means-ba-DTW for others. This reveals a notable weakness of medoid-based clustering algorithms: for strong performance, there must be a representative medoid in the dataset for each cluster. If such a

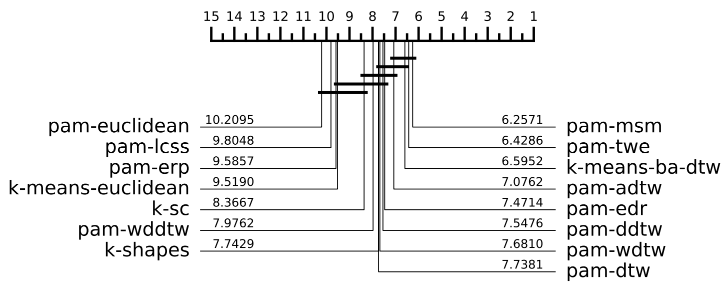


Fig. 6.49 AMI

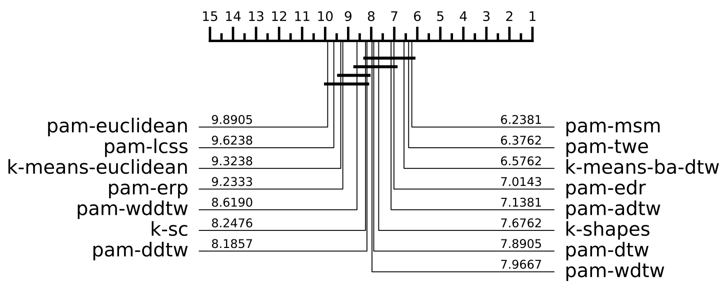


Fig. 6.50 ARI

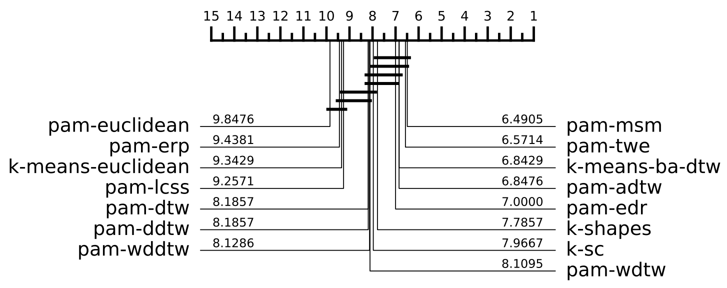


Fig. 6.51 CLACC

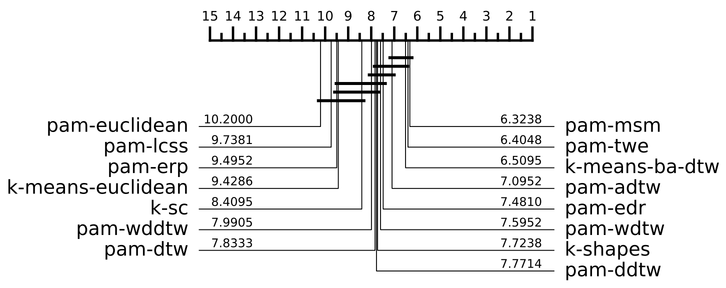
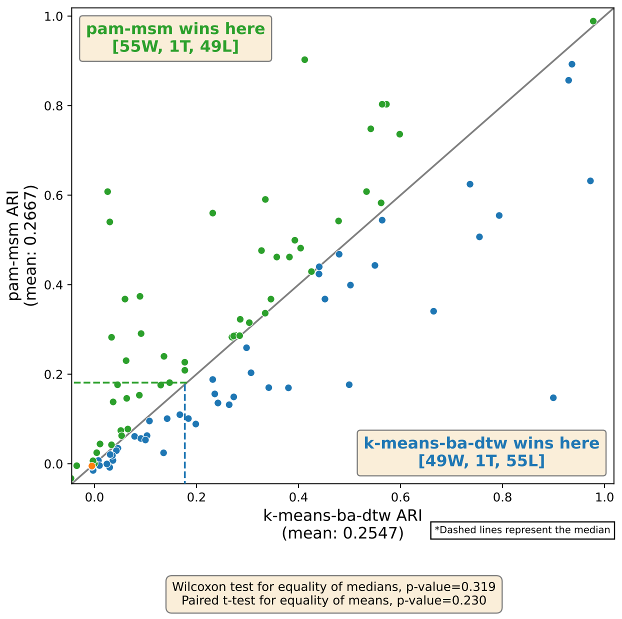


Fig. 6.52 NMI

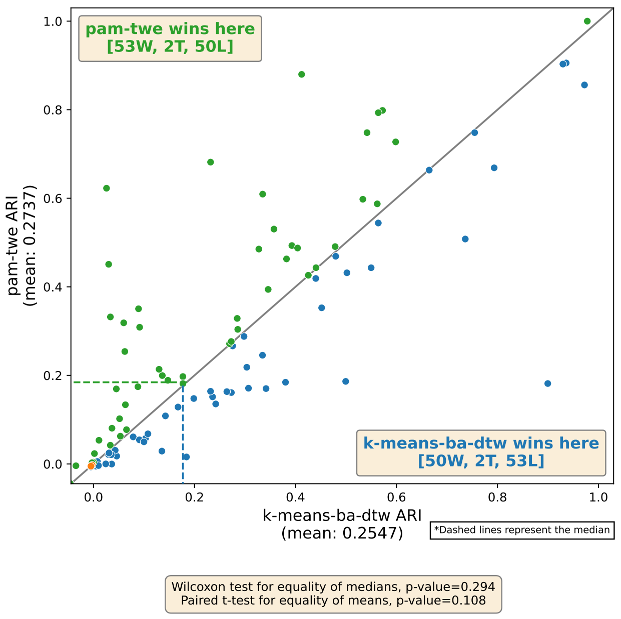
Fig. 6.53 CD diagrams of PAM with 10 elastic distances, exclude PAM-soft-DTW and PAM-shape-DTW with the baseline clusterers over 105 datasets from the UCR archive using the combine test-train split. Missing datasets are outlined in Table A.21.

medoid is absent, the clustering performance degrades significantly. This likely explains the large performance variance when comparing these methods to *k*-means-ba-DTW. For datasets with strong medoids, the PAM models excelled, but for those without suitable medoids, their performance suffered.

We now consider the runtime complexity of PAM. PAM has a computational complexity of $O(kn^2)$. While it is an expensive algorithm to run, for TSCL specifically, the most expensive component is the requirement of a pairwise distance matrix. For example, if we were to compute a pairwise distance matrix for the dataset $X$ of size $n$ using DTW, the computational complexity of DTW is $O(m^2)$, where $m$ is the length of both time series. This means to compute the DTW pairwise



(a) PAM-MSM compared to *k*-means-ba-DTW



(b) PAM-TWE compared to *k*-means-ba-DTW

Fig. 6.54 PAM-MSM and PAM-TWE results compared directly to *k*-means-ba-DTW over 105 datasets from the UCR archive using the combine test-train split.

distance matrix of $X$, it would have a computational complexity of $O(n^2 \times m^2)$ (assuming unique pairs).

We specifically observed this huge runtime complexity as all our PAM models failed to complete within the seven day runtime limit for the Crop dataset. This is not surprising, as Crop is one of the largest datasets, containing 24,000 unique instances. Computing a pairwise distance matrix for the Crop dataset dataset would require 288,000,000 unique distance computations. This highlights one of the key drawbacks of PAM: calculating large pairwise distance matrices for many elastic distances is infeasible.

Figure 6.55 shows the relative FitTime for each PAM model compared to the baseline clusterers. All the considered PAM models exhibited significant runtimes. However, many appear faster than the baseline models such as *k*-means-ba-DTW. This is because in our experimental setup with 10 restarts using different Forgy initialisations, PAM benefits from precomputing the distance matrix and reusing it for all restarts. This means that whether we used 1 or 10 restarts, the runtime would remain similar. In contrast, clusterers such as *k*-means-ba-DTW and *k*-SC must compute the averages and distances in every iteration, as these values cannot be precomputed. Therefore, under our methodology, PAM's runtime might appear more favourable than it actually is. In practice, if *k*-means-ba-DTW were run with a single initialisation, it would be approximately ten times faster than what is shown in Figure 6.55, whereas PAM's runtime would remain largely unchanged.

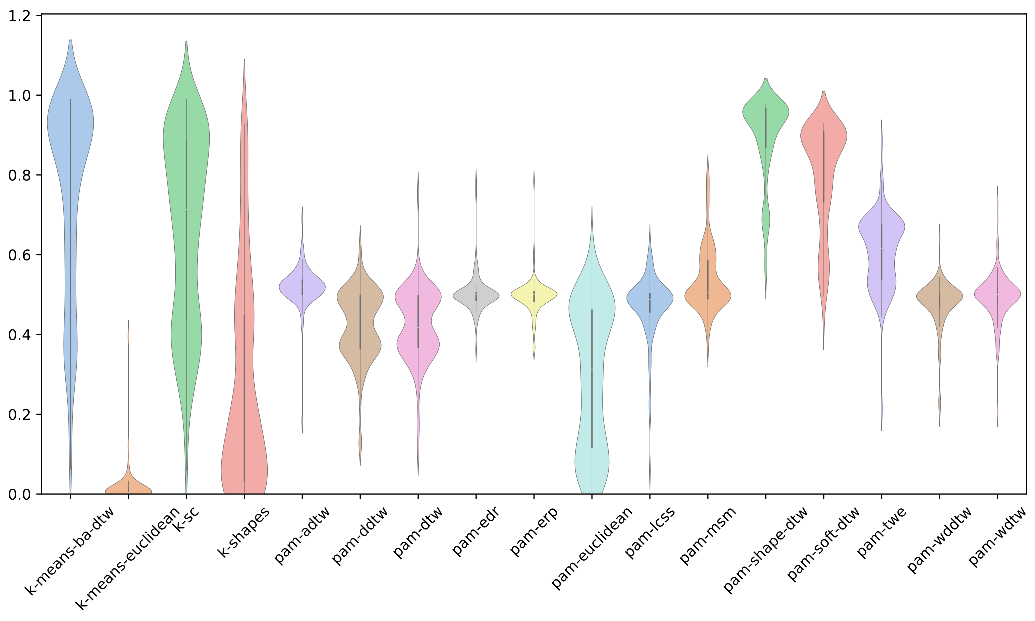


Fig. 6.55 Relative FitTime violin plot for PAM with 13 distances and the baseline clusterers over 99 UCR archive datasets using the combined test-train split.

## 6.5.2 PAM Test-train split

We now examine PAM over the test-train split. Figure 6.60 shows the critical difference diagrams for PAM using 12 elastic distances across 111 datasets from the UCR archive. The rank order of each distance remains largely consistent with the combined test-train split.

PAM-TWE and PAM-MSM consistently appear in the top clique similar to the combined test-train split. However, in addition for the test-train split, PAM-soft-DTW, PAM-ADTW, and PAM-shape-DTW are also in the top clique for every evaluation metric. Furthermore, PAM-TWE achieves the highest average score across all evaluation metrics, as shown in Table 6.10, which is consistent with the combined test-train split.

PAM-Euclidean performs the worst by rank on AMI and NMI but outperforms PAM-WDDTW and PAM-LCSS on ARI. For CLACC, PAM-Euclidean surpasses PAM-DTW, PAM-WDDTW, and PAM-ERP. Notably, PAM-WDDTW, PAM-LCSS, PAM-Euclidean, PAM-ERP, PAM-DDTW, and PAM-DTW consistently appear in the bottom clique across all evaluation metrics.

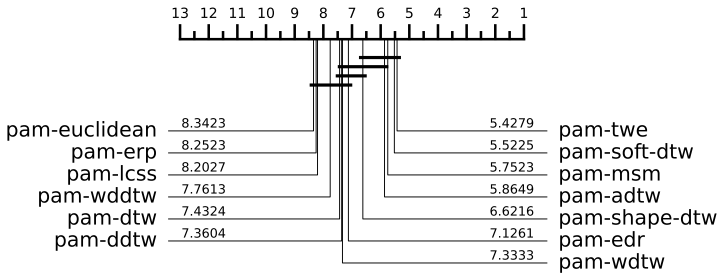


Fig. 6.56 AMI

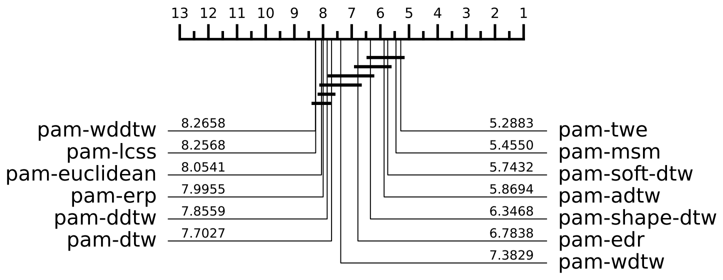


Fig. 6.57 ARI

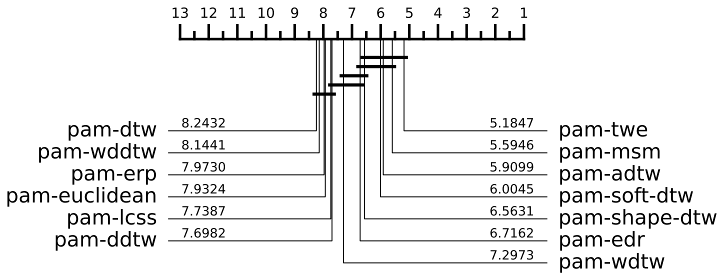


Fig. 6.58 CLACC

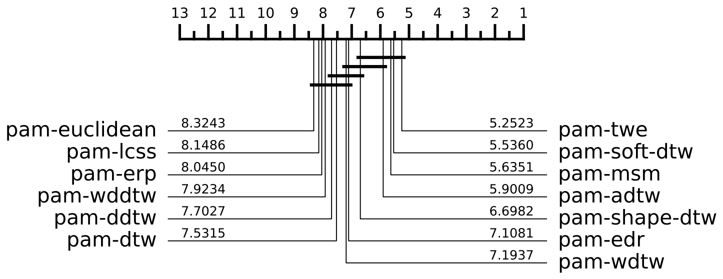


Fig. 6.59 NMI

Fig. 6.60 CD diagrams of PAM over 111 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.22.

Table 6.11 highlights the performance of each model across different domains. A pattern similar to that in the combined test-train split is observed: no single distance dominates multiple domains. Interestingly, although PAM-ADTW was the top performer in two domains for the combined test-train split, for the test-train split PAM-ADTW is not the best in any domain.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| pam-adtw | 0.232 | 0.285 | 0.567 | 0.318 | 0.702 |
| pam-ddtw | 0.191 | 0.256 | 0.535 | 0.287 | 0.675 |
| pam-dtw | 0.209 | 0.268 | 0.541 | 0.303 | 0.692 |
| pam-edr | 0.216 | 0.262 | 0.553 | 0.294 | 0.693 |
| pam-erp | 0.185 | 0.238 | 0.523 | 0.275 | 0.687 |
| pam-euclidean | 0.182 | 0.233 | 0.522 | 0.270 | 0.687 |
| pam-lcss | 0.154 | 0.210 | 0.507 | 0.244 | 0.637 |
| pam-msm | 0.240 | 0.294 | 0.569 | 0.327 | 0.707 |
| pam-shape-dtw | 0.231 | 0.284 | 0.564 | 0.318 | 0.703 |
| pam-soft-dtw | 0.239 | 0.294 | 0.570 | 0.328 | 0.703 |
| pam-twe | **0.246** | **0.299** | **0.576** | **0.333** | **0.711** |
| pam-wddtw | 0.180 | 0.242 | 0.524 | 0.276 | 0.672 |
| pam-wdtw | 0.219 | 0.279 | 0.552 | 0.315 | 0.697 |

Table 6.10 Summary of average score across multiple evaluation metrics over 111 datasets from the UCR archive using the test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| pam-adtw | 0.290 | 0.180 | 0.179 | 0.372 | 0.135 | 0.198 | 0.333 |
| pam-ddtw | 0.268 | 0.171 | 0.176 | 0.220 | 0.098 | 0.093 | 0.279 |
| pam-dtw | 0.250 | 0.167 | 0.164 | 0.417 | 0.138 | 0.175 | 0.215 |
| pam-edr | 0.299 | 0.093 | 0.202 | 0.311 | 0.151 | 0.137 | 0.291 |
| pam-erp | 0.227 | 0.186 | 0.180 | 0.244 | 0.067 | 0.122 | 0.269 |
| pam-euclidean | 0.231 | 0.179 | 0.162 | 0.251 | 0.075 | 0.121 | 0.265 |
| pam-lcss | 0.168 | 0.043 | 0.162 | 0.284 | 0.087 | 0.158 | 0.181 |
| pam-msm | **0.317** | 0.186 | 0.194 | 0.318 | 0.164 | 0.175 | 0.360 |
| pam-shape-dtw | 0.263 | 0.173 | **0.205** | 0.360 | 0.142 | 0.184 | **0.406** |
| pam-soft-dtw | 0.298 | 0.127 | 0.189 | **0.462** | 0.166 | **0.198** | 0.324 |
| pam-twe | 0.314 | 0.187 | 0.203 | 0.364 | **0.187** | 0.174 | 0.358 |
| pam-wddtw | 0.277 | 0.165 | 0.149 | 0.236 | 0.064 | 0.079 | 0.233 |
| pam-wdtw | 0.257 | **0.211** | 0.178 | 0.442 | 0.102 | 0.180 | 0.218 |

Table 6.11 Average ARI score on problems split by problem domain over 111 datasets from the UCR archive using the test-train split.

To contextualise the test-train split results, we include the baseline clusterer in Figure 6.65. The results of PAM with the baseline differ slightly from those for the combined test-train split. PAM-TWE, PAM-MSM, PAM-soft-DTW, PAM-ADTW, and PAM-shape-DTW consistently appear in the top clique. Notably, *k*-means-ba-DTW does not rank in the top clique. The bottom clique remains the same as in the combined test-train split, with the addition of *k*-shapes.

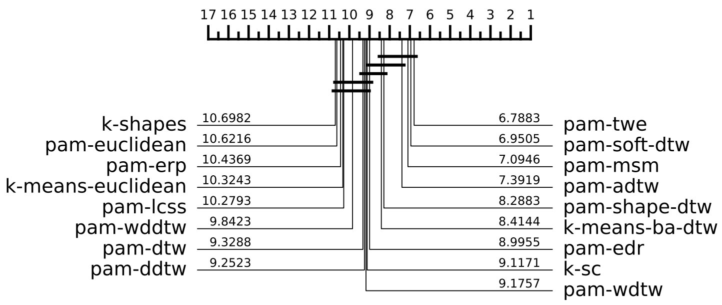


Fig. 6.61 AMI

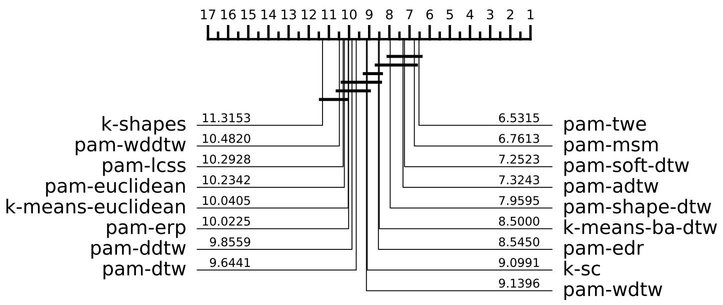


Fig. 6.62 ARI

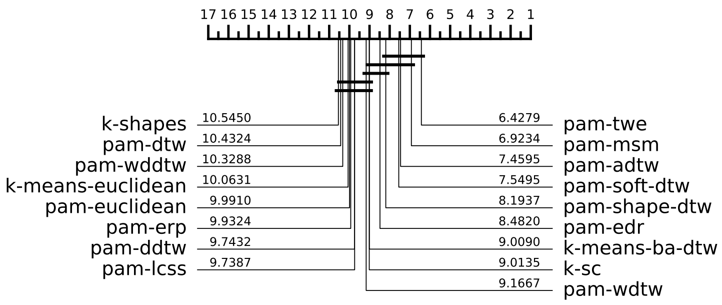


Fig. 6.63 CLACC

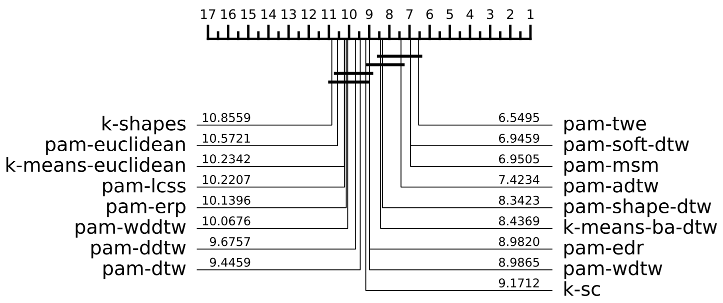
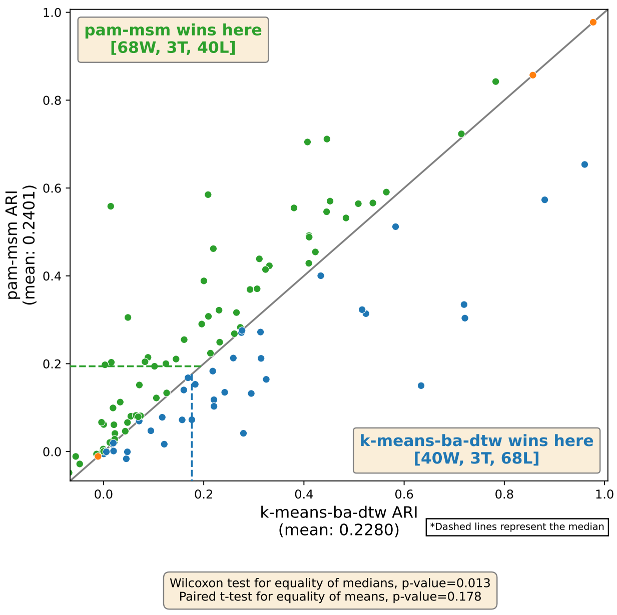


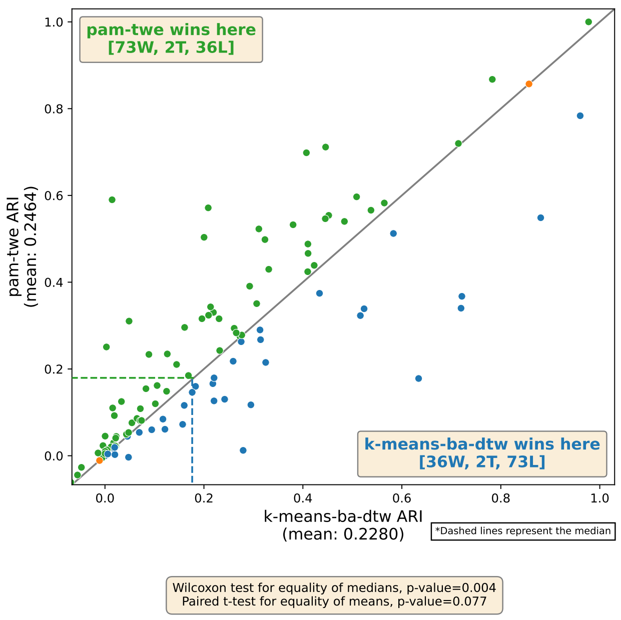
Fig. 6.64 NMI

Fig. 6.65 CD diagrams of PAM with baseline clusterers over 111 datasets from the UCR archive using the test train split split. Missing datasets are outlined in Table A.22.

To investigate why *k*-means-ba-DTW did not appear in the top clique, Figure 6.66 presents a scatter diagram comparing PAM-MSM and PAM-TWE to *k*-means-ba-DTW. For the combined test-train split, *k*-means-ba-DTW demonstrated comparable performance to PAM-MSM and PAM-TWE. However, for the test-train split, both PAM-MSM and PAM-TWE significantly outperformed *k*-means-ba-DTW. This discrepancy may highlight a potential weakness of DBA, suggesting that the averages it produces are not as effective on unseen data as



(a) PAM-MSM compared to *k*-means-ba-DTW



(b) PAM-TWE compared to *k*-means-ba-DTW

Fig. 6.66 PAM-MSM and PAM-TWE results compared directly to *k*-means-ba-DTW over 111 datasets from the UCR archive using the test-train split.

medoids are. Furthermore, similar to the combined test-train split, PAM-MSM and PAM-TWE appear to perform well on the same datasets, obtaining similar scores for them. This may again suggest that similar medoids are being selected for both methods.

### 6.5.3 PAM conclusion

Overall, for both the combined test-train split and the test-train split, we observe that elastic distances significantly enhance the performance of PAM compared to PAM with the Euclidean distance. Additionally, for the combined test-train split, the best-performing elastic distances with PAM are not significant difference from the state-of-the-art *k*-means-ba-DTW algorithm. Furthermore, for the test-train split, several elastic distances with PAM outperform the state-of-the-art by a significant margin.

The rank order of elastic distances remains consistent across both the combined test-train and test-train splits. Specifically, MSM and TWE consistently emerge as the best-performing elastic distances for PAM, followed closely by ADTW, shape-DTW, and soft-DTW. This pattern in rank order is similar to what we observed with $k$-means. This further evidences our hypothesis that elastic distances that explicitly penalise warping off are better for TSCL.

## 6.6 CLARA

For our CLARA experiments, we use the configurations defined in Table 6.12.

| | *max_iters* | *n_init* | *init_algo* | *distance* | *n_samples* |
|---|---|---|---|---|---|
| CLARA-adtw | 50 | 10 | Forgy | ADTW | $40+2k$ |
| CLARA-ddtw | 50 | 10 | Forgy | DDTW | $40+2k$ |
| CLARA-dtw | 50 | 10 | Forgy | DTW | $40+2k$ |
| CLARA-edr | 50 | 10 | Forgy | EDR | $40+2k$ |
| CLARA-erp | 50 | 10 | Forgy | ERP | $40+2k$ |
| CLARA-euclidean | 50 | 10 | Forgy | Euclidean | $40+2k$ |
| CLARA-lcss | 50 | 10 | Forgy | LCSS | $40+2k$ |
| CLARA-msm | 50 | 10 | Forgy | MSM | $40+2k$ |
| CLARA-twe | 50 | 10 | Forgy | TWE | $40+2k$ |
| CLARA-wddtw | 50 | 10 | Forgy | WDDTW | $40+2k$ |
| CLARA-wdtw | 50 | 10 | Forgy | WDTW | $40+2k$ |
| CLARA-shape-dtw | 50 | 10 | Forgy | shape-DTW | $40+2k$ |
| CLARA-soft-dtw | 50 | 10 | Forgy | soft-DTW | $40+2k$ |

Table 6.12 CLARA model parameters. $k$ is the number of clusters for a given datasets.

### 6.6.1 CLARA Combined test-train split

Figure 6.71 presents the critical difference diagrams for CLARA using 12 different elastic distances over 112 datasets. CLARA-soft-DTW, CLARA-MSM, CLARA-TWE, CLARA-shape-DTW, and CLARA-ADTW consistently appear in the top clique across all evaluation metrics. Additionally, although not significantly different, CLARA-soft-DTW consistently ranks the highest. All elastic distances outperform CLARA-Euclidean for all evaluation metrics. However, CLARA-ERP and CLARA-LCSS consistently rank in the bottom clique alongside CLARA-Euclidean.

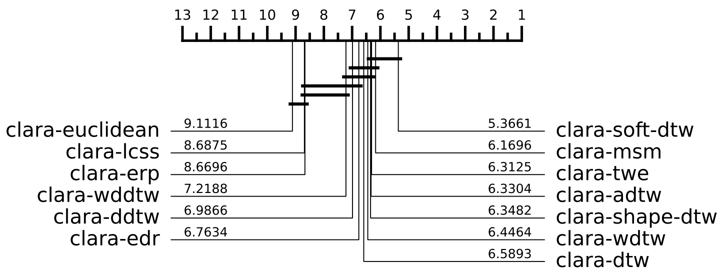


Fig. 6.67 AMI

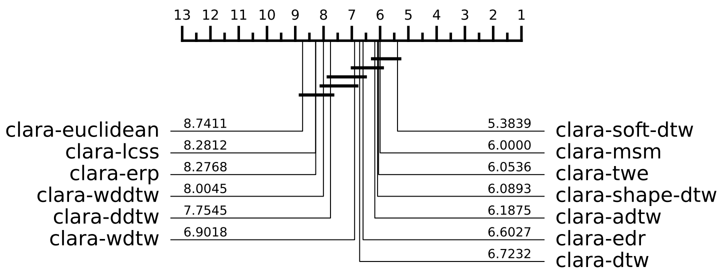


Fig. 6.68 ARI

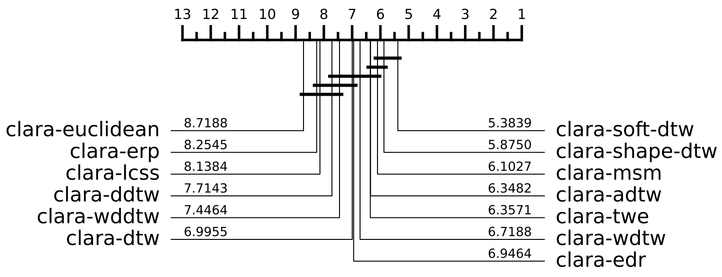


Fig. 6.69 CLACC

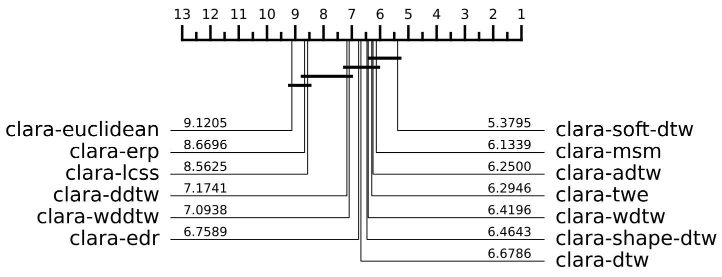


Fig. 6.70 NMI

Fig. 6.71 CD diagrams of CLARA over 112 datasets from the UCR archive using the combined test-train split.

We anticipated the rankings to be similar to those of PAM, as CLARA uses PAM internally. For the most part, the results are comparable, with soft-DTW, MSM, TWE, and ADTW consistently appearing in the top 4–5 positions. However, we observe that shape-DTW performs better with CLARA than with PAM in terms of ranking, while EDR, which appeared in the top clique for PAM, does not for CLARA. Additionally, WDTW performs significantly better with CLARA, ranking in the top clique for AMI and NMI, compared to its performance with PAM. The lower-ranking distances follow a similar pattern, with ERP and LCSS consistently found in the bottom clique alongside Euclidean, as with PAM.

Table 6.13 presents the average scores across 112 datasets for each distance and evaluation metric. CLARA-soft-DTW achieves the highest average score across four evaluation metrics and ties with CLARA-shape-DTW for the best average score in ARI. However, the differences in raw average scores among CLARA-soft-DTW, CLARA-shape-DTW, CLARA-MSM, CLARA-ADTW, and CLARA-TWE are minimal.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| clara-adtw | 0.201 | 0.259 | 0.529 | 0.282 | 0.687 |
| clara-ddtw | 0.164 | 0.232 | 0.507 | 0.254 | 0.653 |
| clara-dtw | 0.187 | 0.247 | 0.519 | 0.271 | 0.682 |
| clara-edr | 0.182 | 0.228 | 0.515 | 0.250 | 0.668 |
| clara-erp | 0.159 | 0.214 | 0.493 | 0.239 | 0.668 |
| clara-euclidean | 0.161 | 0.211 | 0.490 | 0.237 | 0.672 |
| clara-lcss | 0.138 | 0.190 | 0.487 | 0.214 | 0.632 |
| clara-msm | 0.200 | 0.258 | 0.528 | 0.280 | 0.684 |
| clara-shape-dtw | **0.214** | 0.265 | 0.542 | 0.287 | 0.690 |
| clara-soft-dtw | **0.214** | **0.271** | **0.546** | **0.294** | **0.691** |
| clara-twe | 0.200 | 0.258 | 0.527 | 0.281 | 0.685 |
| clara-wddtw | 0.160 | 0.227 | 0.504 | 0.252 | 0.658 |
| clara-wdtw | 0.185 | 0.248 | 0.522 | 0.273 | 0.681 |

Table 6.13 Summary of average score across multiple evaluation metrics over 112 datasets from the UCR archive using the combined test-train split.

Table 6.14 shows CLARA's performance across different problem domains. CLARA-soft-DTW performs best in three domains: Image, Simulated, and Device. CLARA-shape-DTW also leads in three domains: Sensor, Motion, and ECG. Finally, CLARA-DTW performs best in the Spectro domain, tied with CLARA-ADTW.

In our PAM results, presented in Section 6.5, shape-DTW and soft-DTW were the best performers in only one domain each. Specifically, PAM-shape-DTW excelled in the ECG domain, and PAM-soft-DTW in the Simulated domain. With

CLARA, both CLARA-soft-DTW and CLARA-shape-DTW remain top performers in these domains while also excelling in two additional domains each.

PAM-ADTW was the top performer in the Spectro domain, and CLARA-ADTW has maintained that position. However, CLARA-DTW surprisingly shares the same average score in the Spectro domain as CLARA-ADTW. This is unexpected, as ADTW has consistently outperformed DTW in all of our previous experiments, including the PAM results.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| clara-adtw | 0.226 | **0.220** | 0.189 | 0.353 | 0.095 | 0.135 | 0.231 |
| clara-ddtw | 0.218 | 0.164 | 0.166 | 0.208 | 0.046 | 0.096 | 0.171 |
| clara-dtw | 0.203 | **0.220** | 0.184 | 0.324 | 0.111 | 0.120 | 0.174 |
| clara-edr | 0.228 | 0.092 | 0.204 | 0.279 | 0.132 | 0.101 | 0.188 |
| clara-erp | 0.184 | 0.161 | 0.182 | 0.196 | 0.078 | 0.099 | 0.157 |
| clara-euclidean | 0.184 | 0.207 | 0.171 | 0.192 | 0.031 | 0.097 | 0.230 |
| clara-lcss | 0.139 | 0.028 | 0.187 | 0.216 | 0.062 | 0.120 | 0.176 |
| clara-msm | 0.239 | 0.172 | 0.216 | 0.238 | 0.096 | 0.127 | 0.280 |
| clara-shape-dtw | 0.206 | 0.186 | **0.227** | 0.322 | 0.116 | **0.173** | **0.367** |
| clara-soft-dtw | **0.242** | 0.155 | 0.220 | **0.395** | **0.137** | 0.137 | 0.243 |
| clara-twe | 0.222 | 0.190 | 0.218 | 0.295 | 0.117 | 0.117 | 0.248 |
| clara-wddtw | 0.217 | 0.159 | 0.178 | 0.180 | 0.044 | 0.073 | 0.166 |
| clara-wdtw | 0.198 | 0.182 | 0.195 | 0.353 | 0.056 | 0.134 | 0.200 |

Table 6.14 Average ARI score on problems split by problem domain over 112 datasets from the UCR archive using the combined test-train split.

Figure 6.76 presents the critical difference diagram for CLARA with 12 elastic distances, including the baseline clusterers. Across all evaluation metrics, *k*-means-ba-DTW ranks the highest and consistently appears in the top clique alongside *k*-shapes. However, for CLACC, CLARA-soft-DTW also appears in the top clique. Beyond CLACC, no other CLARA models are present in a top clique.

Additionally, several CLARA models perform worse than *k*-means-Euclidean. Specifically, CLARA-DDTW, CLARA-WDDTW, CLARA-ERP, and CLARA-LCSS consistently rank below *k*-means-Euclidean for all evaluation metrics. Furthermore, CLARA-LCSS and CLARA-ERP consistently appear in the bottom clique and, for NMI, ARI, and AMI, perform significantly worse than *k*-means-Euclidean.

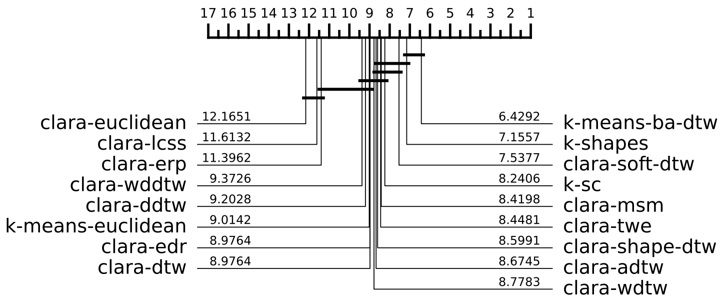


Fig. 6.72 AMI

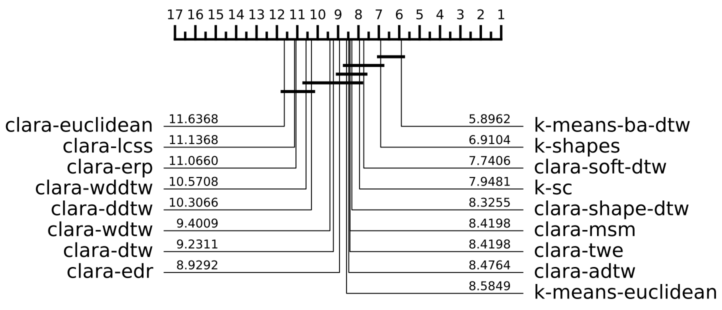


Fig. 6.73 ARI

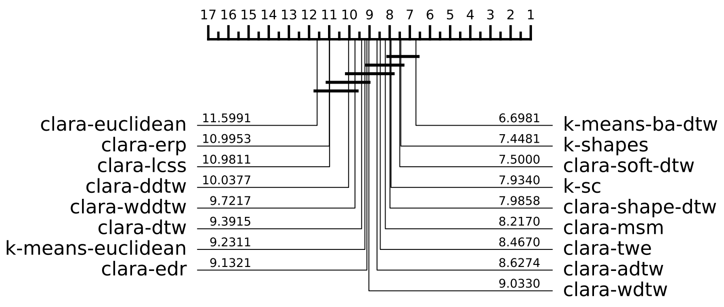


Fig. 6.74 CLACC

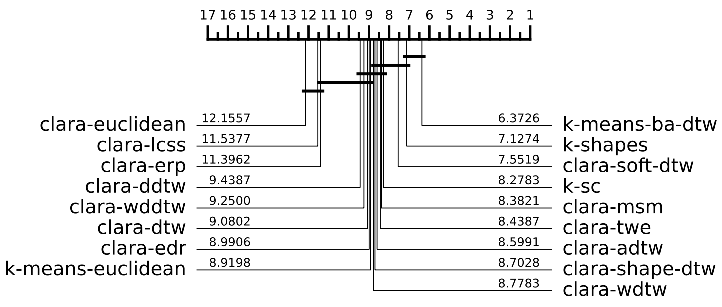


Fig. 6.75 NMI

Fig. 6.76 CD diagrams of CLARA with baseline clusterers over 106 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.24.

Figure 6.77 presents the FitTime critical difference diagram for CLARA, including the baseline clusterers. Most CLARA models are faster than *k*-shapes, which was previously the fastest state-of-the-art TSCL model. Notably, CLARA-MSM, CLARA-TWE, and CLARA-ADTW, as shown in Figure 6.76, do not differ significantly from *k*-shapes in terms of clustering performance. However, Figure 6.77 shows that they are significantly faster. CLARA-soft-DTW is the best-performing CLARA clusterer, but is considerably slower than *k*-shapes. Therefore, a potential alternative to *k*-shapes, which offers similar performance but is faster, is CLARA with MSM, TWE, or ADTW.

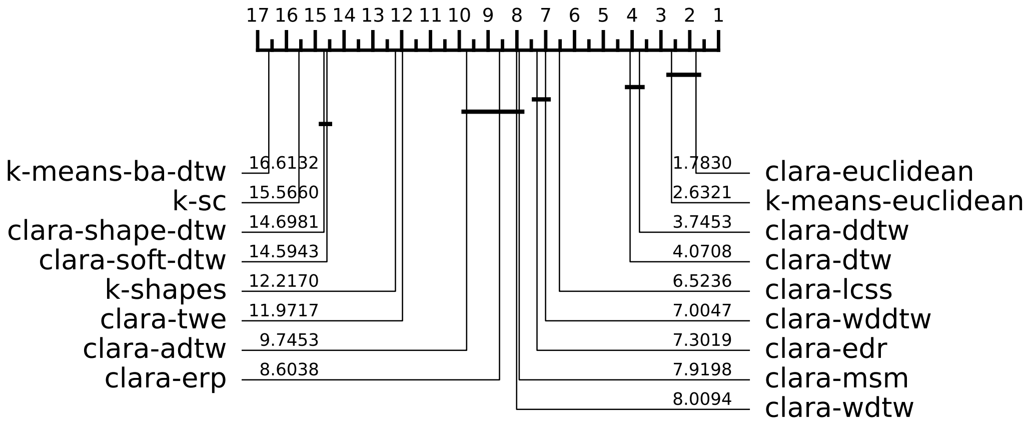


Fig. 6.77 CD diagram for FitTime of CLARA with 12 elastic distance and the baseline clusterers over 106 UCR archive datasets using the combined test-train split.

### 6.6.2 CLARA Test-train split

Figure 6.82 shows the critical difference diagrams for CLARA using the test-train split across 12 elastic distances. The rank order is very similar to the combined test-train split, where CLARA-soft-DTW, CLARA-MSM, CLARA-TWE, and CLARA-ADTW consistently appear in the top clique. However, CLARA-shape-DTW does not appear in the top clique for NMI, whereas it was included in the top clique for all evaluation metrics in the combined test-train split. While all CLARA clusterers outperform CLARA-Euclidean, CLARA-ERP and CLARA-LCSS continue to rank in the bottom clique for every evaluation metric. Additionally, in the test-train split, CLARA-DDTW and CLARA-WDDTW also consistently rank in the bottom clique for every evaluation metric.

In Figure 6.82, we observe that, similar to the combined test-train split, CLARA-soft-DTW is the best-performing clusterer. However, the gap in absolute average rank is much smaller compared to the combined test-train split. In the combined test-train split, CLARA-soft-DTW was on average 0.6662 ranks higher than the next best CLARA clusterer for each evaluation metric (as shown in Figure 6.71). In contrast, for the test-train split, CLARA-soft-DTW is only 0.3186 ranks higher on average than the second-best CLARA clusterer.

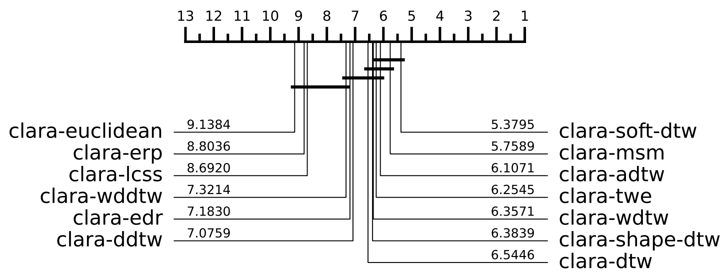


Fig. 6.78 AMI

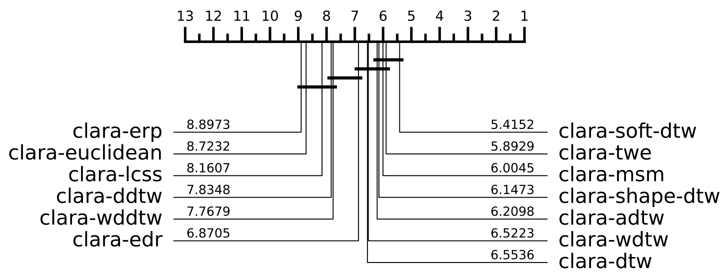


Fig. 6.79 ARI

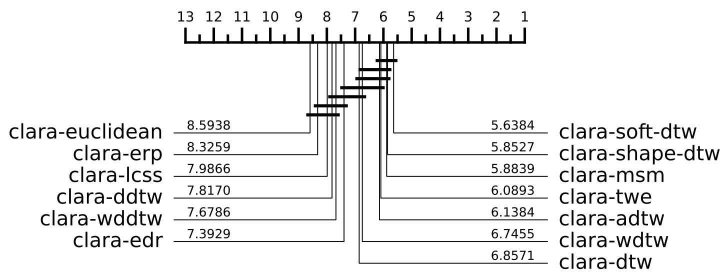


Fig. 6.80 CLACC

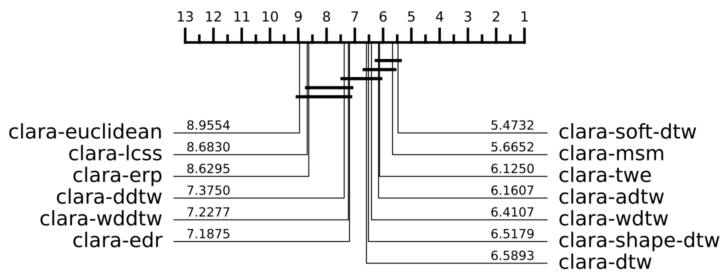


Fig. 6.81 NMI

Fig. 6.82 CD diagrams of CLARA over 112 datasets from the UCR archive using the test train split.

This reduction in CLARA-soft-DTW's dominance for the test-train split is further highlighted in Table 6.15, where it only achieves the highest average score for CLACC. In the combined test-train split, CLARA-soft-DTW had the best average score across all evaluation metrics. In the test-train split, CLARA-MSM has the highest average scores for ARI, AMI, and NMI, and ties with CLARA-TWE for the best score in RI.

Table 6.16 shows the best-performing CLARA clusterers by domain for the test-train split. The same distances continue to excel in similar domains as in the combined test-train split, with CLARA-soft-DTW still leading in the Simulated and Device domains and CLARA-shape-DTW performing best in Motion and ECG.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| clara-adtw | 0.190 | 0.244 | 0.533 | 0.279 | 0.681 |
| clara-ddtw | 0.153 | 0.222 | 0.504 | 0.254 | 0.646 |
| clara-dtw | 0.181 | 0.240 | 0.523 | 0.275 | 0.678 |
| clara-edr | 0.182 | 0.227 | 0.520 | 0.260 | 0.670 |
| clara-erp | 0.153 | 0.205 | 0.497 | 0.243 | 0.664 |
| clara-euclidean | 0.154 | 0.200 | 0.495 | 0.238 | 0.666 |
| clara-lcss | 0.125 | 0.176 | 0.484 | 0.210 | 0.623 |
| clara-msm | **0.203** | **0.259** | 0.539 | **0.293** | **0.685** |
| clara-shape-dtw | 0.196 | 0.247 | 0.541 | 0.281 | 0.681 |
| clara-soft-dtw | 0.200 | 0.256 | **0.545** | 0.291 | 0.683 |
| clara-twe | 0.200 | 0.254 | 0.535 | 0.288 | **0.685** |
| clara-wddtw | 0.151 | 0.221 | 0.506 | 0.256 | 0.653 |
| clara-wdtw | 0.184 | 0.244 | 0.528 | 0.280 | 0.680 |

Table 6.15 Summary of average score across multiple evaluation metrics over 112 datasets from the UCR archive using the test-train split

However, CLARA-MSM performs best in the Image domain, and CLARA-TWE leads in the Sensor domain, both of which were close to being the best in the combined test-train split. CLARA-WDTW emerges as the best performer in the Spectro domain, whereas CLARA-DTW and CLARA-ADTW were the best in this domain in the combined test-train split. CLARA-DTW remains the second-best performer for Spectro.

An interesting observation from Tables 6.15 and 6.16 is that the average values in the test-train split are not significantly lower than those in the combined test-train split. For example, CLARA-soft-DTW achieved an average score of 0.546 in the combined test-train split and 0.545 in the test-train split.

Furthermore, when comparing the average ARI for the Image, Spectro, and Device domains between the combined test-train split and the test-train split, we find that the test-train split yields a higher average ARI for these three domains. For instance, in the Image domain, the best-performing CLARA clusterer achieved an average ARI of 0.242 in the combined split, compared to 0.257 in the test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| clara-adtw | 0.221 | 0.186 | 0.170 | 0.310 | 0.096 | 0.144 | 0.231 |
| clara-ddtw | 0.187 | 0.165 | 0.156 | 0.182 | 0.062 | 0.108 | 0.167 |
| clara-dtw | 0.202 | 0.224 | 0.168 | 0.293 | 0.114 | 0.122 | 0.170 |
| clara-edr | 0.240 | 0.076 | 0.202 | 0.263 | 0.133 | 0.096 | 0.200 |
| clara-erp | 0.182 | 0.166 | 0.169 | 0.201 | 0.047 | 0.098 | 0.153 |
| clara-euclidean | 0.184 | 0.190 | 0.152 | 0.207 | 0.018 | 0.096 | 0.225 |
| clara-lcss | 0.120 | 0.023 | 0.167 | 0.212 | 0.064 | 0.121 | 0.137 |
| clara-msm | **0.257** | 0.182 | 0.199 | 0.268 | 0.086 | 0.131 | 0.267 |
| clara-shape-dtw | 0.211 | 0.170 | 0.182 | 0.298 | 0.102 | **0.168** | **0.324** |
| clara-soft-dtw | 0.229 | 0.127 | 0.183 | **0.371** | **0.152** | 0.152 | 0.248 |
| clara-twe | 0.253 | 0.182 | **0.208** | 0.230 | 0.111 | 0.118 | 0.232 |
| clara-wddtw | 0.179 | 0.170 | 0.173 | 0.196 | 0.056 | 0.076 | 0.162 |
| clara-wdtw | 0.195 | **0.231** | 0.179 | 0.337 | 0.062 | 0.131 | 0.188 |

Table 6.16 Average ARI score on problems split by problem domain over 112 datasets from the UCR archive using the test-train split.

For many other clusterers, this outcome would be unexpected. However, since CLARA uses a randomly selected subset of data, these results may suggest that the training split contains more “good” potential medoids than the test split. As a result, when the test split is excluded from training, the likelihood of identifying one of these better medoids increases compared to the combined test-train split.

Figure 6.87 presents the critical difference diagrams for CLARA using 12 elastic distances, including the baseline clusterers. The ranking is very similar to that of the combined test-train split, with the exception of $k$-shapes performing poorly on the test-train split. We also observe that the performance gap between the best CLARA clusterers and the state-of-the-art $k$-means-ba-DTW is very small. Additionally, while only CLARA-soft-DTW appeared in the top clique for the combined test-train split, Figure 6.87 shows that CLARA-MSM consistently appears in the top clique for every evaluation metric.

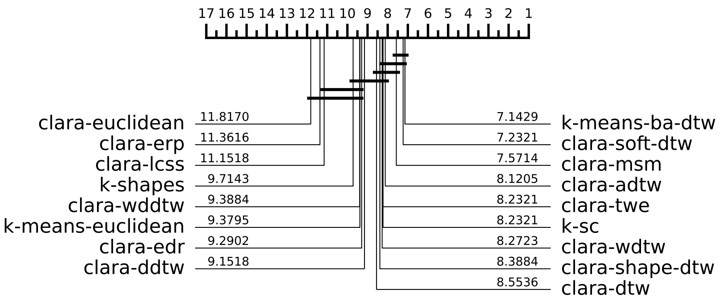


Fig. 6.83 AMI

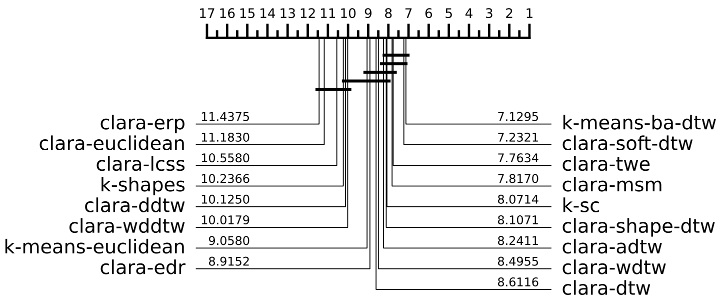


Fig. 6.84 ARI

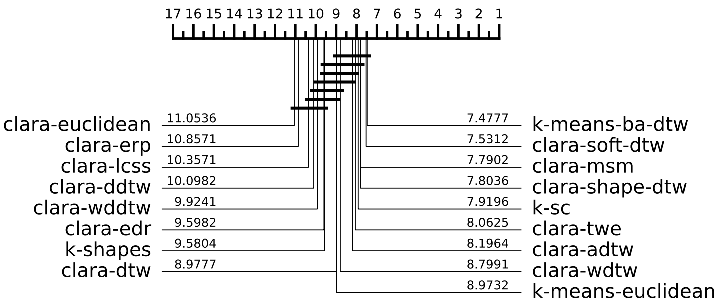


Fig. 6.85 CLACC

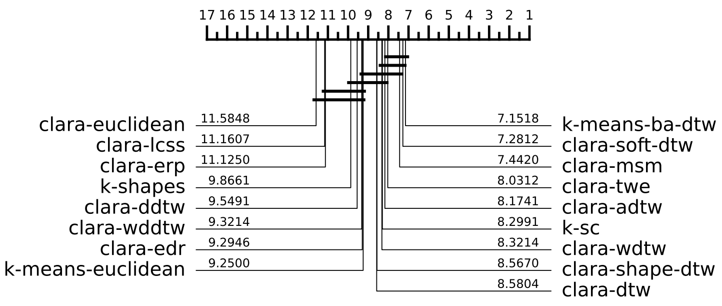


Fig. 6.86 NMI

Fig. 6.87 CD diagrams of CLARA with baseline clusterers over 112 datasets from the UCR archive using the test train split.

## 6.6.3 CLARA Conclusion

Overall, CLARA exhibits a similar rank ordering of elastic distances to what was observed in PAM and alternate *k*-medoids. For the combined test-train split, CLARA-soft-DTW was the best-performing clusterer across all evaluation metrics in terms of both average rank and absolute scores. In the test-train split, CLARA-soft-DTW also had the highest average rank across all evaluation metric, but it was consistently outperformed by CLARA-MSM in terms of average scores.

Most of the CLARA clusterers performed significantly worse than the current state-of-the-art. While CLARA-soft-DTW was not consistently significantly different from *k*-shapes, it was much slower, greatly reducing its utility. Overall, our CLARA experiments suggest that CLARA is not well-suited for TSCL, even with elastic distances. However, the results do help further support our hypothesis on which elastic distances perform best.

## 6.7 CLARANS

For each of our CLARANS experiments we use the model configurations defined in Table 6.17.

| | *max_iters* | *n_init* | *init_algo* | *distance* | *max_neighbours* |
|---|---|---|---|---|---|
| CLARANS-adtw | 50 | 10 | Forgy | ADTW | $0.0125(k(n-k))$ |
| CLARANS-ddtw | 50 | 10 | Forgy | DDTW | $0.0125(k(n-k))$ |
| CLARANS-dtw | 50 | 10 | Forgy | DTW | $0.0125(k(n-k))$ |
| CLARANS-edr | 50 | 10 | Forgy | EDR | $0.0125(k(n-k))$ |
| CLARANS-erp | 50 | 10 | Forgy | ERP | $0.0125(k(n-k))$ |
| CLARANS-euclidean | 50 | 10 | Forgy | Euclidean | $0.0125(k(n-k))$ |
| CLARANS-lcss | 50 | 10 | Forgy | LCSS | $0.0125(k(n-k))$ |
| CLARANS-msm | 50 | 10 | Forgy | MSM | $0.0125(k(n-k))$ |
| CLARANS-twe | 50 | 10 | Forgy | TWE | $0.0125(k(n-k))$ |
| CLARANS-wddtw | 50 | 10 | Forgy | WDDTW | $0.0125(k(n-k))$ |
| CLARANS-wdtw | 50 | 10 | Forgy | WDTW | $0.0125(k(n-k))$ |
| CLARANS-shape-dtw | 50 | 10 | Forgy | shape-DTW | $0.0125(k(n-k))$ |
| CLARANS-soft-dtw | 50 | 10 | Forgy | soft-DTW | $0.0125(k(n-k))$ |

Table 6.17 CLARANS model parameters. $k$ is the number of clusters for a given datasets and $n$ is the number of instances for a given dataset.

### 6.7.1 CLARANS Combined test-train split

Figure 6.92 presents the critical difference diagrams for CLARANS using 12 elastic distances over the combined test-train split. For every evaluation metric, CLARANS-ADTW, CLARANS-soft-DTW, CLARANS-MSM, and CLARANS-TWE consistently appear in the top clique, just as they did for PAM and CLARA. Additionally, all CLARANS clusterers outperform CLARANS-Euclidean. However, similar to CLARA and PAM, CLARANS-LCSS and CLARANS-ERP consistently rank in the bottom clique along with CLARANS-Euclidean.

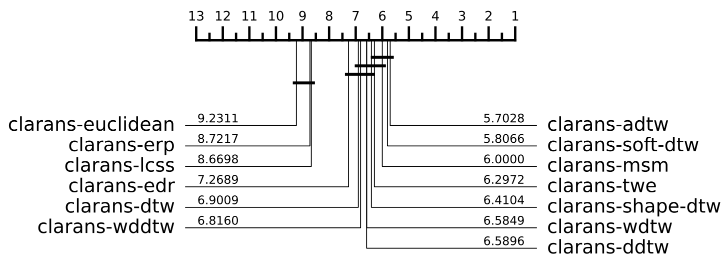


Fig. 6.88 AMI

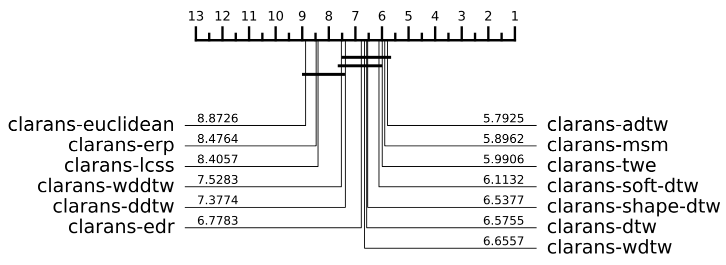


Fig. 6.89 ARI

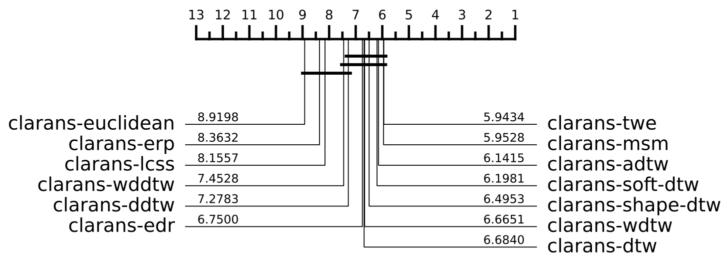


Fig. 6.90 CLACC

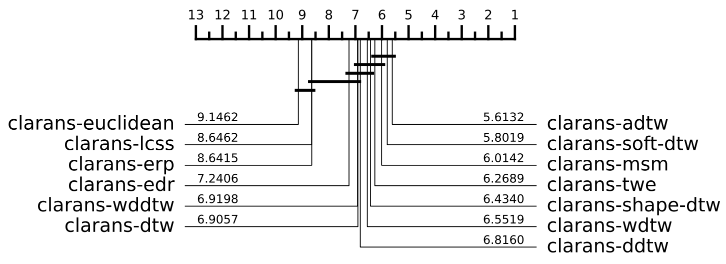


Fig. 6.91 NMI

Fig. 6.92 CD diagrams of CLARANS over 106 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.23.

Table 6.18 shows the average score for each evaluation metric for CLARANS. CLARANS-ADTW is the best-performing clusterer in four out of five evaluation metrics, while CLARANS-TWE performs best in one. This differs from what we observed with PAM and CLARA, where TWE was the best-performing distance for PAM, and soft-DTW led for CLARA across all metrics. However, the differences between the top distances in CLARA and PAM were minimal. We observe the same trend for CLARANS where CLARANS-ADTW, CLARANS-MSM, CLARANS-TWE, and CLARANS-soft-DTW show only minimal differences in total average scores.

Table 6.19 presents CLARANS' performance across different domains. Similar to PAM, no single distance dominates across a majority of domains. In fact, six different distances are best across the seven domains, with only CLARANS-shape-DTW achieving the highest average score in more than one domain. Additionally, like CLARA, DTW performs best in the Spectro domain. In the Simulated domain, it is noteworthy how much better CLARANS-WDTW performs, with an ARI score 0.092 higher than the next best CLARANS clusterer.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| clarans-adtw | **0.252** | **0.298** | 0.565 | **0.322** | **0.711** |
| clarans-ddtw | 0.207 | 0.268 | 0.538 | 0.288 | 0.686 |
| clarans-dtw | 0.231 | 0.275 | 0.557 | 0.300 | 0.700 |
| clarans-edr | 0.222 | 0.259 | 0.549 | 0.281 | 0.690 |
| clarans-erp | 0.178 | 0.224 | 0.512 | 0.251 | 0.681 |
| clarans-euclidean | 0.175 | 0.220 | 0.507 | 0.247 | 0.684 |
| clarans-lcss | 0.151 | 0.201 | 0.505 | 0.225 | 0.639 |
| clarans-msm | 0.240 | 0.290 | 0.564 | 0.313 | 0.706 |
| clarans-shape-dtw | 0.235 | 0.284 | 0.555 | 0.308 | 0.703 |
| clarans-soft-dtw | 0.239 | 0.291 | 0.562 | 0.315 | 0.699 |
| clarans-twe | 0.250 | 0.295 | **0.568** | 0.318 | 0.711 |
| clarans-wddtw | 0.204 | 0.258 | 0.535 | 0.282 | 0.685 |
| clarans-wdtw | 0.233 | 0.284 | 0.553 | 0.308 | 0.703 |

Table 6.18 Summary of average score across multiple evaluation metrics over 106 datasets from the UCR archive using the combined test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| clarans-adtw | 0.330 | 0.188 | 0.218 | 0.436 | 0.143 | **0.173** | 0.281 |
| clarans-ddtw | 0.305 | 0.151 | 0.229 | 0.234 | 0.091 | 0.088 | 0.175 |
| clarans-dtw | 0.297 | **0.219** | 0.184 | 0.436 | 0.154 | 0.158 | 0.153 |
| clarans-edr | 0.325 | 0.078 | 0.212 | 0.347 | 0.165 | 0.121 | 0.222 |
| clarans-erp | 0.236 | 0.167 | 0.164 | 0.264 | 0.111 | 0.102 | 0.168 |
| clarans-euclidean | 0.247 | 0.207 | 0.153 | 0.234 | 0.040 | 0.101 | 0.160 |
| clarans-lcss | 0.155 | 0.049 | 0.169 | 0.293 | 0.080 | 0.130 | 0.267 |
| clarans-msm | 0.325 | 0.170 | 0.216 | 0.368 | 0.167 | 0.144 | 0.270 |
| clarans-shape-dtw | 0.305 | 0.191 | 0.203 | 0.358 | 0.134 | 0.142 | **0.416** |
| clarans-soft-dtw | **0.333** | 0.134 | 0.195 | 0.436 | **0.189** | 0.156 | 0.200 |
| clarans-twe | 0.332 | 0.172 | **0.236** | 0.364 | 0.171 | 0.148 | 0.336 |
| clarans-wddtw | 0.316 | 0.167 | 0.203 | 0.248 | 0.081 | 0.079 | 0.176 |
| clarans-wdtw | 0.292 | 0.216 | 0.192 | **0.528** | 0.075 | 0.165 | 0.164 |

Table 6.19 Average ARI score on problems split by problem domain over 106 datasets from the UCR archive using the test-train split.

Figure 6.97 shows the critical difference diagrams for CLARANS alongside the baseline clusterers. For AMI and NMI, CLARANS-ADTW outperforms the state-of-the-art *k*-means-ba-DTW. However, for CLACC and ARI, *k*-means-ba-DTW ranks higher. Still, CLARANS-ADTW, *k*-means-ba-DTW, CLARANS-MSM, CLARANS-soft-DTW, and CLARANS-TWE consistently appear in the top clique. Notably, CLARANS knocks *k*-shapes out of the top clique for AMI and NMI. Furthermore, only two CLARANS estimators—CLARANS-ERP and CLARANS-LCSS—perform worse than *k*-means-Euclidean.

Figure 6.98 presents the FitTime comparison for CLARANS and the baseline clusterers. With the exception of shape-DTW and soft-DTW, CLARANS is significantly faster than *k*-means-ba-DTW while achieving similar performance. Additionally CLARANS on average is fastest than *k*-shapes, while consistently offering better clustering performance. This suggests that CLARANS could be a good choice when computational constraints prevent the use of *k*-means-ba-DTW.

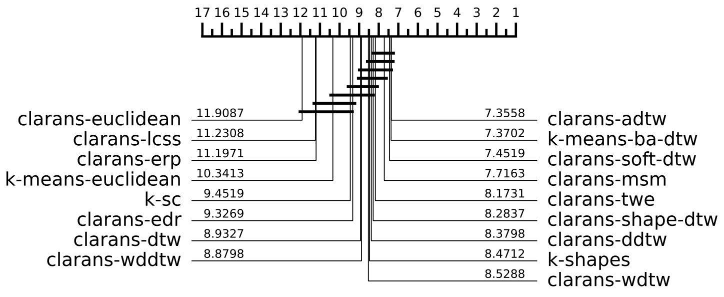


Fig. 6.93 AMI

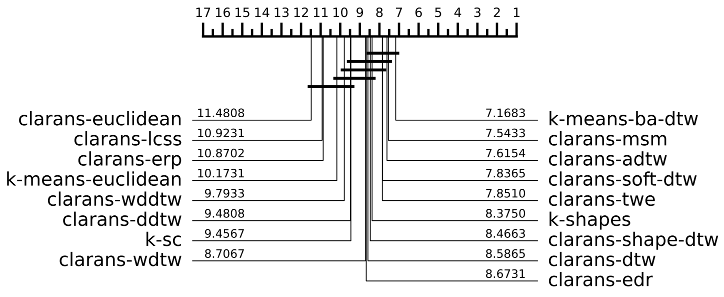


Fig. 6.94 ARI

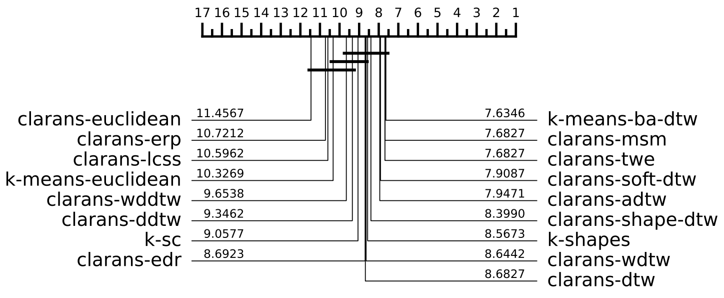


Fig. 6.95 CLACC

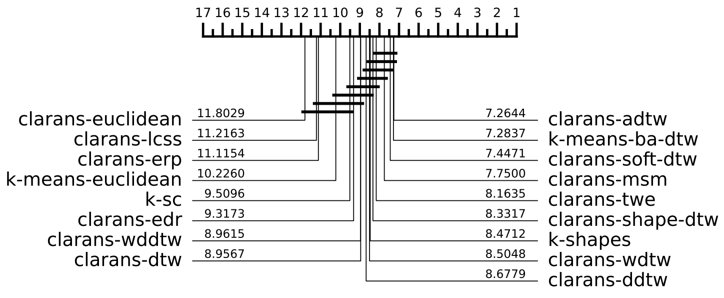


Fig. 6.96 NMI

Fig. 6.97 CD diagrams of CLARANS with baseline clusterers over 104 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.25.

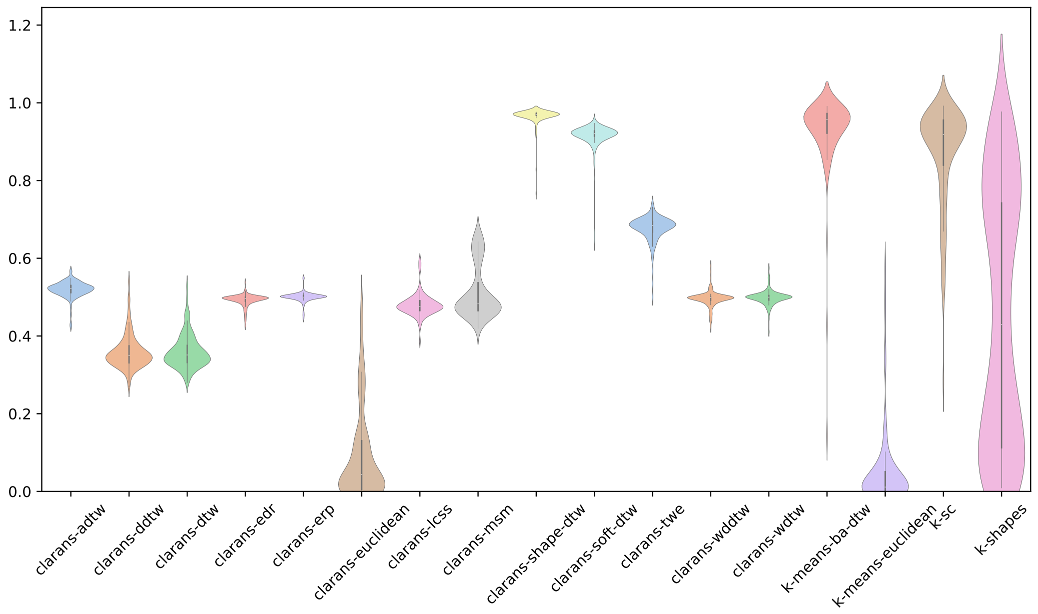


Fig. 6.98 Relative FitTime violin plot for CLARANS with 13 distances and the baseline clusterer over 104 UCR archive datasets using the combined test-train split.

### 6.7.2 CLARANS Test-train split

Figure 6.103 presents the critical difference diagrams for CLARANS using 12 different elastic distances over the test-train split. CLARANS-soft-DTW, CLARANS-MSM, CLARANS-TWE, and CLARANS-ADTW consistently appear in the top clique, similar to our previous findings. CLARANS-Euclidean, CLARANS-LCSS, CLARANS-DDTW, CLARANS-WDDTW, and CLARANS-ERP consistently rank in the bottom clique, with CLARANS-LCSS performing worse than CLARANS-Euclidean for both AMI and NMI. The overall ranking aligns with our combined test-train CLARANS results, as well as the PAM and CLARA results.

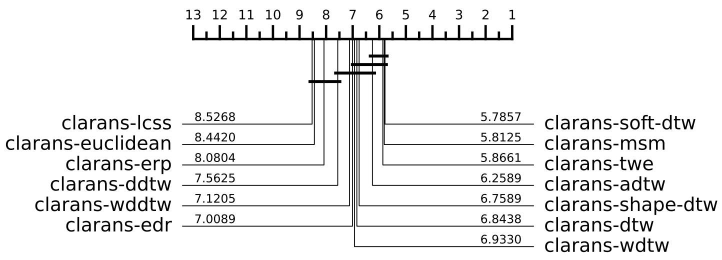


Fig. 6.99 AMI

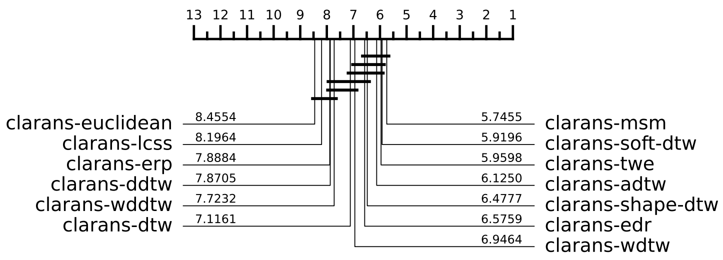


Fig. 6.100 ARI

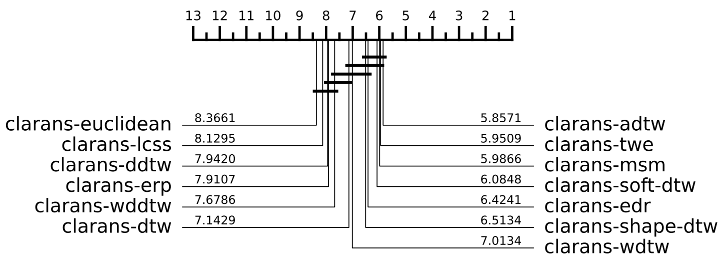


Fig. 6.101 CLACC

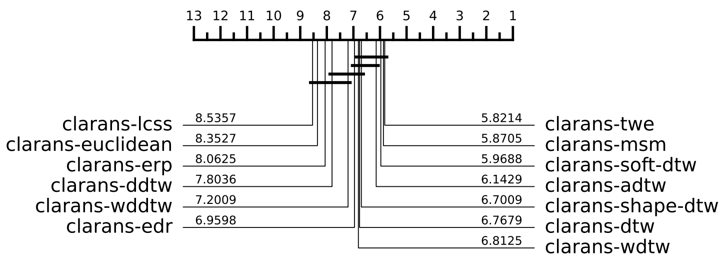


Fig. 6.102 NMI

Fig. 6.103 CD diagrams of CLARANS over 112 datasets from the UCR archive using the test-train split.

Tables 6.20 and 6.21 show a similar trend, where CLARANS-MSM, CLARANS-soft-DTW, CLARANS-TWE, and CLARANS-ADTW are the best-performing clusterers. Interestingly, CLARANS-DTW performs best in the Simulated domain, as shown in Table 6.21.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| clarans-adtw | 0.219 | 0.275 | 0.561 | 0.310 | 0.699 |
| clarans-ddtw | 0.178 | 0.248 | 0.524 | 0.281 | 0.670 |
| clarans-dtw | 0.217 | 0.273 | 0.551 | 0.309 | 0.697 |
| clarans-edr | 0.206 | 0.255 | 0.550 | 0.287 | 0.689 |
| clarans-erp | 0.174 | 0.221 | 0.519 | 0.259 | 0.680 |
| clarans-euclidean | 0.175 | 0.223 | 0.515 | 0.261 | 0.685 |
| clarans-lcss | 0.148 | 0.202 | 0.504 | 0.236 | 0.635 |
| clarans-msm | **0.232** | 0.284 | 0.562 | 0.318 | **0.705** |
| clarans-shape-dtw | 0.217 | 0.271 | 0.554 | 0.305 | 0.697 |
| clarans-soft-dtw | 0.228 | 0.284 | **0.564** | 0.317 | 0.697 |
| clarans-twe | 0.229 | **0.286** | 0.560 | **0.320** | 0.704 |
| clarans-wddtw | 0.188 | 0.254 | 0.531 | 0.288 | 0.675 |
| clarans-wdtw | 0.210 | 0.267 | 0.546 | 0.303 | 0.696 |

Table 6.20 Summary of average score across multiple evaluation metrics over 112 datasets from the UCR archive using the test-train split.

To contextualise our findings, Figure 6.108 shows the critical difference diagrams for CLARANS with 12 elastic distances and the baseline clusterers. CLARANS-soft-DTW, CLARANS-TWE, CLARANS-MSM, and CLARANS-ADTW consistently outperform *k*-means-ba-DTW. Notably, the CLARANS clusterers significantly outperform *k*-means-ba-DTW in CLACC and ARI, pushing it out of the top clique. Overall, CLARANS demonstrates significantly better performance over the test-train split compared to the state-of-the-art, while remaining considerably less computationally expensive.

### 6.7.3 CLARANS conclusion

Overall, CLARANS demonstrates strong clustering performance. The rank order for both the combined and test-train splits is consistent with previous observations for PAM, alternate *k*-medoids, and CLARA. Specifically, for the combined test-train split, CLARANS-ADTW achieved the best average rank across all evaluation

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| clarans-adtw | 0.263 | **0.219** | 0.169 | 0.330 | 0.149 | 0.177 | 0.288 |
| clarans-ddtw | 0.228 | 0.131 | 0.155 | 0.339 | 0.119 | 0.081 | 0.258 |
| clarans-dtw | 0.241 | 0.185 | 0.157 | **0.474** | 0.202 | 0.173 | 0.233 |
| clarans-edr | 0.271 | 0.100 | 0.201 | 0.271 | 0.127 | 0.141 | 0.296 |
| clarans-erp | 0.214 | 0.139 | 0.162 | 0.225 | 0.112 | 0.117 | 0.266 |
| clarans-euclidean | 0.214 | 0.206 | 0.168 | 0.221 | 0.030 | 0.118 | 0.250 |
| clarans-lcss | 0.151 | 0.026 | 0.159 | 0.291 | 0.105 | 0.159 | 0.167 |
| clarans-msm | **0.289** | 0.195 | 0.206 | 0.284 | 0.167 | 0.168 | 0.320 |
| clarans-shape-dtw | 0.241 | 0.213 | 0.173 | 0.334 | 0.154 | 0.168 | **0.380** |
| clarans-soft-dtw | 0.281 | 0.116 | 0.191 | 0.446 | 0.162 | 0.180 | 0.279 |
| clarans-twe | 0.288 | 0.137 | **0.212** | 0.331 | **0.202** | 0.157 | 0.281 |
| clarans-wddtw | 0.275 | 0.149 | 0.166 | 0.253 | 0.071 | 0.092 | 0.263 |
| clarans-wdtw | 0.267 | 0.169 | 0.160 | 0.408 | 0.084 | **0.189** | 0.201 |

Table 6.21 Average ARI score on problems split by problem domain over 112 datasets from the UCR archive using the test-train split.

metrics and had the highest average score for all but one metric. Additionally, for AMI and NMI, CLARANS-ADTW outperformed *k*-means-ba-DTW while maintaining significantly lower computational time.

Similarly, in the test-train split, multiple CLARANS models showed strong clustering performance. CLARANS-soft-DTW, CLARANS-MSM, CLARANS-TWE, and CLARANS-ADTW delivered very similar results across all evaluation metrics and all achieved the highest average rank for one evaluation metric. Furthermore, all four of these CLARANS clusterers consistently outperformed the current state-of-the-art while remaining significantly less computationally expensive.

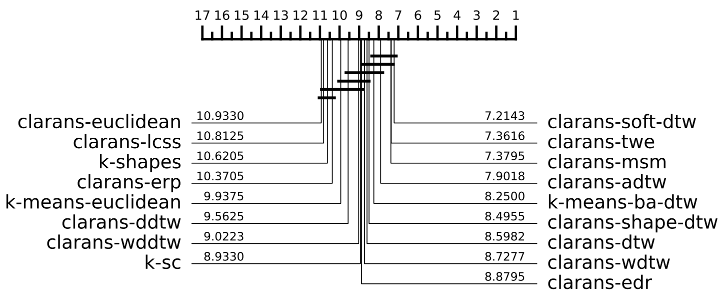


Fig. 6.104 AMI

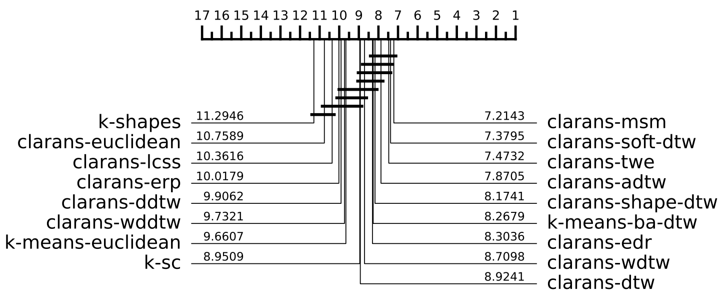


Fig. 6.105 ARI

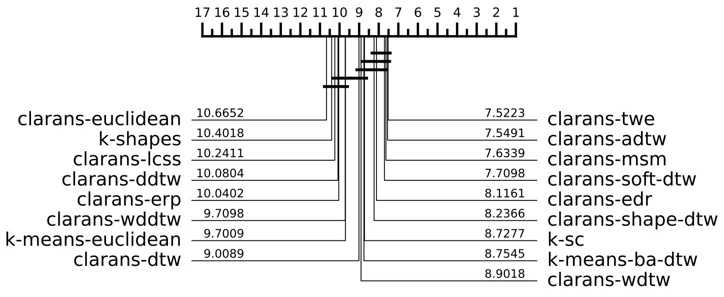


Fig. 6.106 CLACC

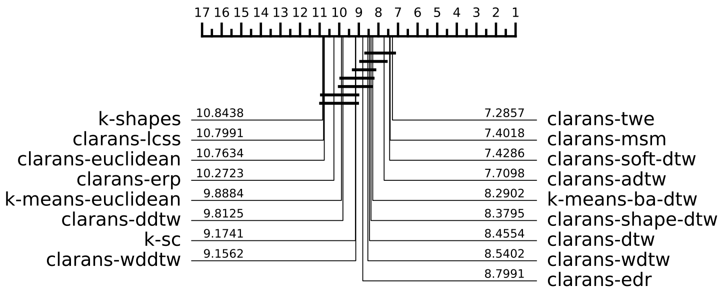


Fig. 6.107 NMI

Fig. 6.108 CD diagrams of CLARANS with baseline clusterers over 112 datasets from the UCR archive using the test-train split.

## 6.8 Analysis

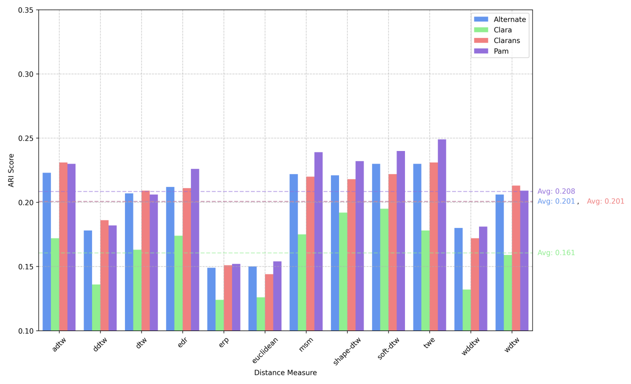
We have now presented the results for each *k*-medoids approach with different elastic distances, comparing them to their own results and to the baseline clusterers. However, we have not yet compared each *k*-medoids approache against one another. In this section, we will compare each *k*-medoids approach to each other to highlight the strengths and weaknesses of each method and identify the best-performing *k*-medoids approach.

### 6.8.1 *k*-medoids with elastic distances

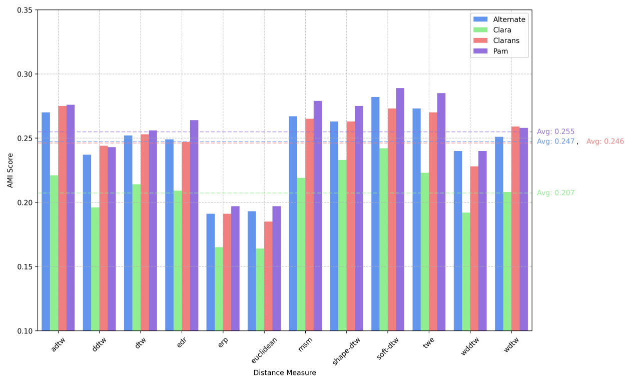
We first examine the performance of each *k*-medoids model across different elastic distances. Figure 6.109 presents a bar graph displaying the average ARI and

AMI scores for each *k*-medoids clusterer with different distance measures over the combined test-train split. Similarly, Figure 6.110 provides the same representation for the test-train split.

From Figures 6.109 and 6.110, we observe that PAM generally achieves the highest average scores in both the combined and test-train splits. CLARANS and alternate exhibit very similar performance across all distances, ranking just below PAM, while CLARA consistently performs the worst among the four models.



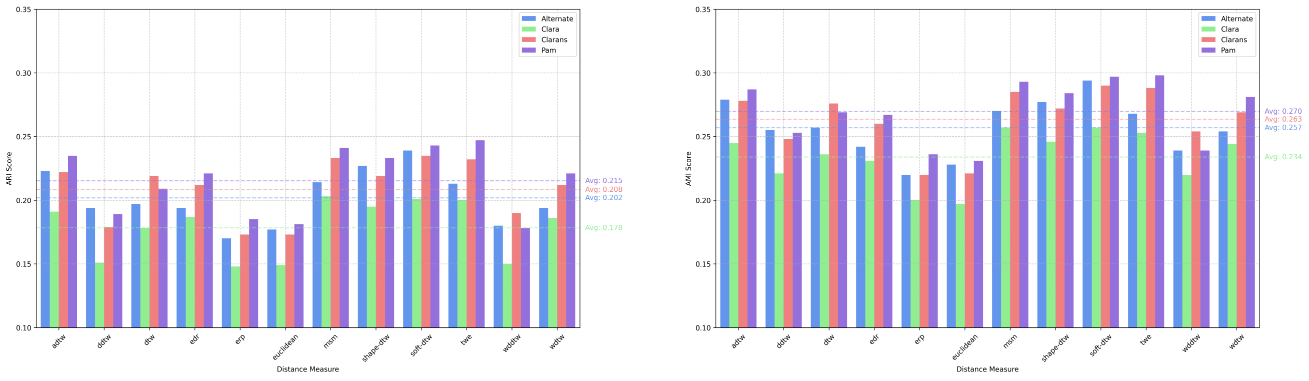
(a) ARI



(b) AMI

Fig. 6.109 Comparison of the performance of four *k*-medoids algorithms across 11 elastic distances using 90 datasets from the UCR archive, evaluated on the combined test-train split. The blue bars represent the scores for alternate *k*-medoids, green for CLARA, red for CLARANS, and purple for PAM. The dashed lines indicate the average scores for each clustering algorithm, with colours matching the corresponding bars. LCSS was excluded due to its failure to complete a significant number of datasets. The missing datasets are listed in Table A.26.

In both figures, TWE, soft-DTW, MSM, shape-DTW, and ADTW are consistently the top-performing distances for every model. ERP, on the other hand, is consistently the worst-performing distance (aside from LCSS, which had to be excluded). Notably, for the combined test-train split, alternate *k*-medoids and CLARANS perform very similarly across all distances. Additionally, for the test-train split, CLARANS is the only model that occasionally outperforms PAM with the same distance. For instance, CLARANS-DTW and CLARANS-WDDTW outperform PAM-DTW and PAM-WDDTW in ARI and AMI over the test-train split.



(a) ARI (b) AMI

Fig. 6.110 Comparison of the performance of four *k*-medoids algorithms across 11 elastic distances using 105 datasets from the UCR archive, evaluated on the test-train split. The blue bars represent the scores for alternate *k*-medoids, green for CLARA, red for CLARANS, and purple for PAM. The dashed lines indicate the average scores for each clustering algorithm, with colours matching the corresponding bars. LCSS was excluded due to failing to complete a large number of datasets. Missing datasets are outlined in Table A.27.

Overall, both figures demonstrate the importance of choosing the right elastic distance. Additionally, we have shown that, in general, each elastic distance affects the models similarly, as no single combination of distance and model significantly alters the rank order of the models.

### 6.8.2 *k*-medoids clustering performance

We now narrow the focus of our experiment to the top five distances for each *k*-medoids model: ADTW, MSM, TWE, shape-DTW, and soft-DTW. We will compare each model with these five elastic distances to find the best performing *k*-medoids-based clusterer with elastic distances. Figure 6.115 shows the critical difference diagrams for the four *k*-medoids clusterers, comparing their top five elastic distances to each other and the baseline clusterers over the combined test-train split.

Figure 6.115 demonstrates that multiple *k*-medoids models outperform the current state-of-the-art in terms of average rank, though the differences are not statistically significant. PAM-soft-DTW is the best-performing clusterer across all evaluation metrics, followed by PAM-TWE and PAM-MSM. Notably, alternate-soft-DTW ranks in the top three for CLACC and NMI. Additionally, CLARANS-ADTW appears in the top clique for AMI, ARI, and NMI, although it does not perform as well for CLACC. Finally, the best-performing CLARA clusterer is CLARA-soft-DTW, which consistently appears in the bottom clique, but its performance across all evaluation metrics is not significantly different from *k*-shapes, which was previously considered state-of-the-art.

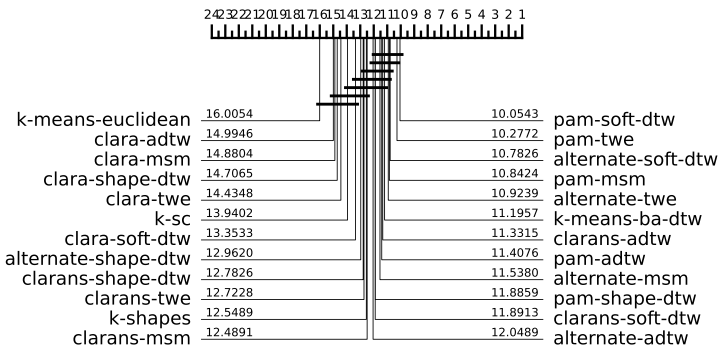


Fig. 6.111 AMI

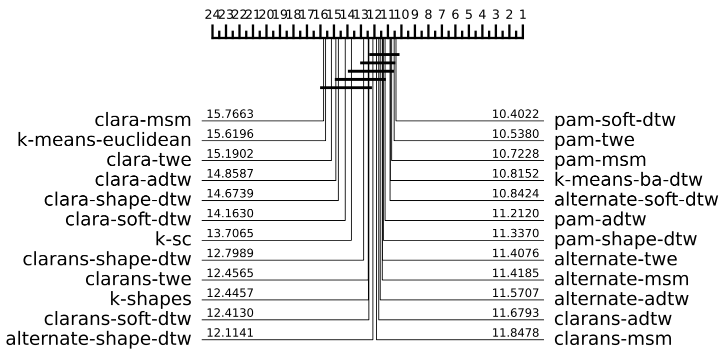


Fig. 6.112 ARI

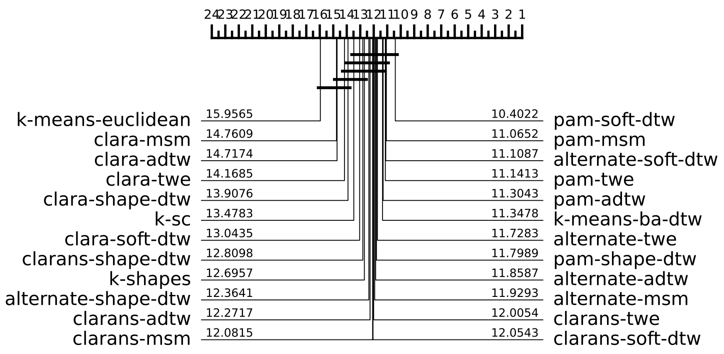


Fig. 6.113 CLACC

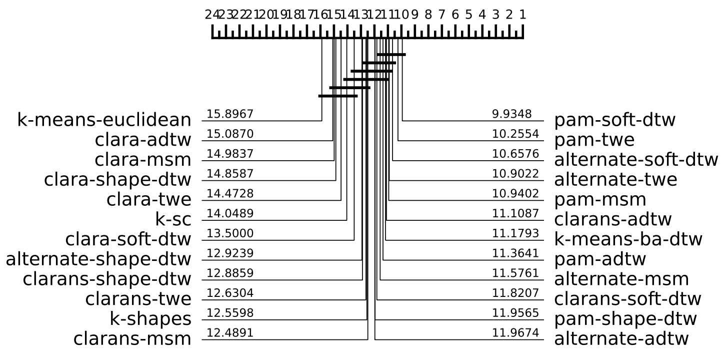


Fig. 6.114 NMI

Fig. 6.115 CD diagrams of 4 *k*-medoids clusterers with their top 5 distances with baseline clusterers over 92 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.28

Table 6.23 presents the average scores achieved by each clusterer across 92 datasets over the combined test-train split. PAM-soft-DTW and PAM-TWE achieve the highest average scores across all datasets. However, Table 6.22 shows that, by domain, PAM models are only the best performers in the Image and Spectro domains. Interestingly, CLARANS-based models perform best in the Device and ECG domains, while *k*-means-ba-DTW leads in the Simulated and Motion domains, and *k*-SC remains the top performer for the Sensor domain.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| pam-adtw | 0.233 | 0.277 | 0.570 | 0.300 | 0.695 |
| pam-msm | 0.241 | 0.279 | 0.578 | 0.301 | 0.698 |
| pam-shape-dtw | 0.232 | 0.273 | 0.570 | 0.296 | 0.693 |
| pam-soft-dtw | 0.242 | **0.286** | 0.578 | **0.310** | 0.694 |
| pam-twe | **0.250** | 0.285 | **0.585** | 0.308 | **0.702** |
| alternate-adtw | 0.228 | 0.272 | 0.565 | 0.296 | 0.695 |
| alternate-msm | 0.225 | 0.267 | 0.567 | 0.290 | 0.692 |
| alternate-shape-dtw | 0.225 | 0.264 | 0.564 | 0.289 | 0.692 |
| alternate-soft-dtw | 0.236 | 0.283 | 0.573 | 0.307 | 0.693 |
| alternate-twe | 0.232 | 0.274 | 0.571 | 0.297 | 0.695 |
| clara-adtw | 0.177 | 0.224 | 0.530 | 0.248 | 0.667 |
| clara-msm | 0.175 | 0.220 | 0.530 | 0.243 | 0.663 |
| clara-shape-dtw | 0.192 | 0.233 | 0.545 | 0.257 | 0.672 |
| clara-soft-dtw | 0.196 | 0.242 | 0.552 | 0.266 | 0.674 |
| clara-twe | 0.178 | 0.222 | 0.532 | 0.247 | 0.666 |
| clarans-adtw | 0.238 | 0.277 | 0.573 | 0.301 | 0.698 |
| clarans-msm | 0.223 | 0.265 | 0.568 | 0.288 | 0.690 |
| clarans-shape-dtw | 0.221 | 0.264 | 0.563 | 0.288 | 0.691 |
| clarans-soft-dtw | 0.223 | 0.271 | 0.569 | 0.295 | 0.687 |
| clarans-twe | 0.233 | 0.271 | 0.573 | 0.294 | 0.696 |
| k-means-ba-dtw | 0.232 | 0.273 | 0.570 | 0.297 | 0.694 |
| k-means-euclidean | 0.169 | 0.210 | 0.513 | 0.237 | 0.672 |
| k-sc | 0.187 | 0.223 | 0.537 | 0.244 | 0.639 |
| k-shapes | 0.208 | 0.260 | 0.555 | 0.282 | 0.685 |

Table 6.22 Summary of average score across multiple evaluation metrics over 92 datasets from the UCR archive using the combined test-train split.

Figure 6.116 presents the FitTime for each clusterer. The most computationally expensive model is PAM-shape-DTW, followed by alternate-MSM. However, all

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| pam-adtw | 0.298 | **0.165** | 0.236 | 0.368 | 0.109 | 0.157 | 0.291 |
| pam-msm | **0.342** | 0.100 | 0.245 | 0.319 | 0.167 | 0.136 | 0.356 |
| pam-shape-dtw | 0.290 | 0.136 | 0.240 | 0.344 | 0.127 | 0.147 | 0.397 |
| pam-soft-dtw | 0.311 | 0.110 | 0.247 | 0.478 | 0.176 | 0.152 | 0.185 |
| pam-twe | 0.338 | 0.116 | 0.247 | 0.413 | 0.179 | 0.136 | 0.351 |
| alternate-adtw | 0.277 | 0.163 | 0.230 | 0.404 | 0.121 | 0.153 | 0.263 |
| alternate-msm | 0.310 | 0.117 | 0.225 | 0.281 | 0.166 | 0.134 | 0.342 |
| alternate-shape-dtw | 0.283 | 0.134 | 0.240 | 0.309 | 0.117 | 0.140 | 0.390 |
| alternate-soft-dtw | 0.311 | 0.125 | 0.225 | 0.447 | 0.193 | 0.150 | 0.170 |
| alternate-twe | 0.306 | 0.121 | 0.237 | 0.388 | 0.181 | 0.133 | 0.231 |
| clara-adtw | 0.218 | 0.146 | 0.191 | 0.291 | 0.099 | 0.100 | 0.161 |
| clara-msm | 0.236 | 0.090 | 0.221 | 0.170 | 0.090 | 0.092 | 0.240 |
| clara-shape-dtw | 0.190 | 0.106 | 0.237 | 0.255 | 0.124 | 0.150 | 0.339 |
| clara-soft-dtw | 0.230 | 0.131 | 0.231 | 0.323 | 0.142 | 0.106 | 0.172 |
| clara-twe | 0.225 | 0.112 | 0.222 | 0.236 | 0.114 | 0.082 | 0.187 |
| clarans-adtw | 0.314 | 0.137 | 0.236 | 0.380 | 0.140 | 0.164 | 0.281 |
| clarans-msm | 0.312 | 0.090 | 0.234 | 0.305 | 0.164 | 0.132 | 0.270 |
| clarans-shape-dtw | 0.281 | 0.139 | 0.224 | 0.295 | 0.143 | 0.133 | **0.416** |
| clarans-soft-dtw | 0.291 | 0.108 | 0.215 | 0.387 | **0.199** | 0.144 | 0.200 |
| clarans-twe | 0.316 | 0.091 | 0.256 | 0.310 | 0.165 | 0.135 | 0.336 |
| k-means-ba-dtw | 0.283 | 0.141 | 0.210 | **0.536** | 0.173 | **0.164** | 0.137 |
| k-means-euclidean | 0.217 | 0.147 | 0.199 | 0.221 | 0.039 | 0.104 | 0.174 |
| k-sc | 0.221 | 0.160 | **0.279** | 0.085 | 0.032 | 0.080 | 0.395 |
| k-shapes | 0.228 | 0.122 | 0.208 | 0.390 | 0.096 | 0.152 | 0.407 |

Table 6.23 Average ARI score on problems split by problem domain over 92 datasets from the UCR archive using the test-train split.

PAM models, except PAM-shape-DTW, are faster than *k*-means-ba-DTW (although, as discussed, our methodology favours models that allow for precomputation). Notably, as shown in Figure 6.115, CLARANS-ADTW stands out as one of the fastest algorithms, and as mentioned earlier, it ranks in the top clique for AMI, ARI, and NMI, potentially providing a fast alternative with very good performance.

Finally, Figure 6.121 presents the critical difference diagram for 102 datasets, comparing each *k*-medoids clusterer with the top five elastic distances, along with the baseline clusterers. We observe similar rankings to the combined test-train split,

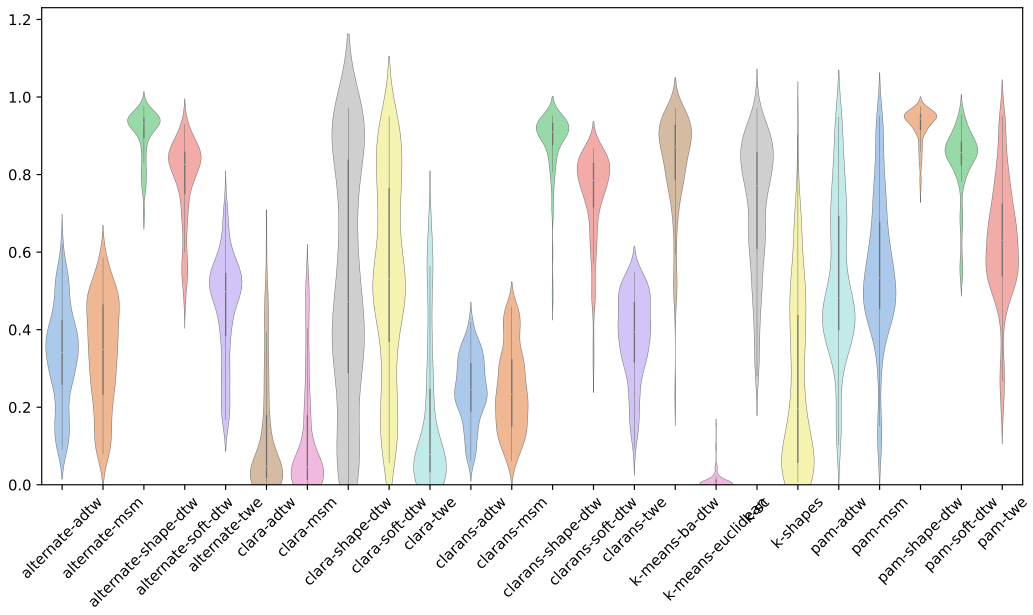


Fig. 6.116 Relative FitTime violin plot for 4 *k*-medoids clusterers with their top 5 distances and the baseline clusterers over 92 UCR archive datasets using the combined test-train split.

where PAM-TWE, PAM-soft-DTW, and PAM-MSM consistently rank among the top four clusterers across all evaluation metrics.

However, for the test-train split, CLARANS-based clusterers perform significantly better and consistently rank higher than alternate *k*-medoids clusterers. Additionally, more than half of the *k*-medoids approaches considered in the experiment outperform the current state-of-the-art method, *k*-means-ba-DTW.

Overall, our experimental comparison demonstrates that *k*-medoids-based clusterers with elastic distances provide superior clustering performance for both the combined test-train split and the test-train split. In particular, we identify PAM-TWE, PAM-soft-DTW, and PAM-MSM as the new state-of-the-art clusterers for both splits. Additionally, we highlight CLARANS-ADTW as a fast alternative that consistently delivers strong performance, although it does not match the effectiveness of the PAM clusterers.

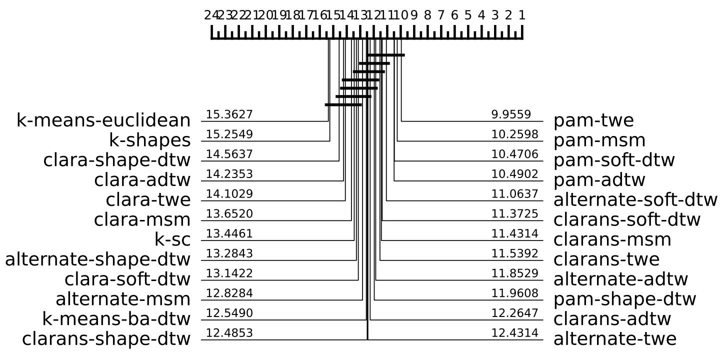


Fig. 6.117 AMI

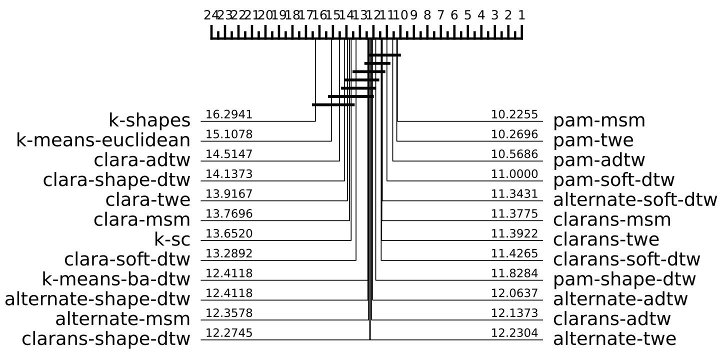


Fig. 6.118 ARI

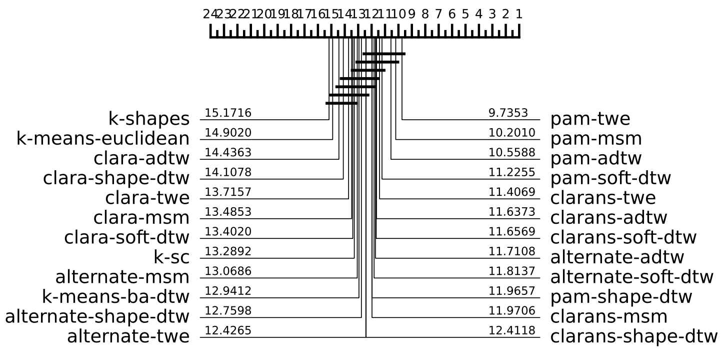


Fig. 6.119 CLACC

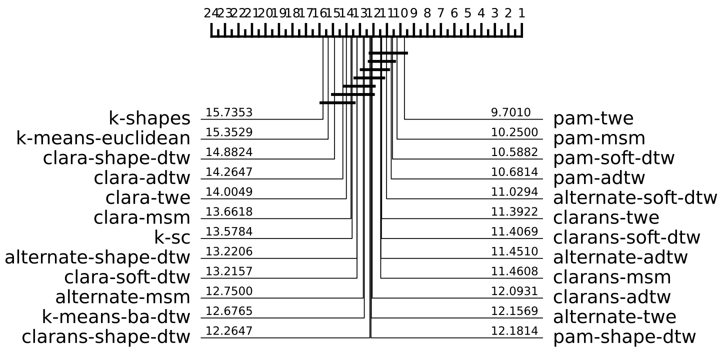


Fig. 6.120 NMI

Fig. 6.121 CD diagrams of 4 *k*-medoids clusterers with their top 5 distances with baseline clusterers over 102 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.29.

## 6.9 Conclusion

In this chapter, we presented and evaluated results for four different *k*-medoids models across 12 elastic distances. We found that TWE, MSM, ADTW, soft-DTW, and shape-DTW consistently performed best across all *k*-medoids models. Conversely, ERP, LCSS, DDTW, and WDDTW were consistently the worst-performing distances for each approach.

Our analysis showed that PAM is the best-performing *k*-medoids model for TSCL, followed by CLARANS and alternate *k*-medoids, which jointly take second place depending on the distance measure used. CLARA performed significantly worse than the other *k*-medoids models. The overall best-performing models were PAM-TWE, PAM-soft-DTW, and PAM-MSM for both the combined test-train split and the test-train split.

Additionally, we demonstrated that using elastic distances for centroid computation is crucial for the success of any model. By comparing alternate *k*-medoids to *k*-means using the same elastic distance and the arithmetic mean, we observed that using medoids consistently improved performance for all distances. However, we also found that when an average was computed with the alignment path of an elastic distance (e.g., *k*-means-ba-DTW), the average version significantly outperformed the medoids variant. This suggests that further improvements to the state-of-the-art could be achieved by using an averaging technique with better elastic distances, such as TWE, MSM, or ADTW.

Regarding our hypotheses, we first showed that incorporating elastic distances into centroid computation greatly enhances clustering performance. We initially hypothesised that the performance improvement would be similar across all distances, but we found this was not the case. Distances with an implicit warping penalty benefited more from medoids than those with an explicit penalty, though not to the extent of outperforming the explicit warping penalty distances. Finally, we hypothesised that the best-performing distances from Chapter 5 would, when paired with medoids, be the best performing distances for *k*-medoids. We confirmed this, showing that PAM-ADTW, PAM-MSM, PAM-TWE, and PAM-soft-DTW all surpassed the performance of the current state-of-the-art *k*-means-ba-DTW for both the combined test-train split and the combined test-train split.

# Chapter 7

# Elastic Barycentre Averaging

## Contributing Publications

- Holder, C.; Guijo-Rubio, D. and Bagnall, A. (2023). Barycentre Averaging for the Move-Split-Merge Time Series Distance Measure. In Proceedings of the 15th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - KDIR; ISBN 978-989-758-671-2; ISSN 2184-3228, SciTePress, pages 51-62. DOI: 10.5220/0012164900003598

*k*-means-ba-DTW is one of the best-performing clusterers we have considered. As demonstrated in our experiments in Chapter 5 and Chapter 6, it consistently ranks among the top TSCL approaches, significantly outperforming both *k*-means-DTW and PAM-DTW. Our experiments suggest that this superior performance is due to the use of the DBA algorithm for centroid computation; however, this algorithm is currently limited to DTW only.

To address this limitation, we develop a new averaging technique, which is a generalised form of the DBA algorithm that can be applied to any elastic distance that computes a complete optimal alignment path. We call this averaging technique the Elastic Barycentre Average. In this chapter we outline the Elastic Barycentre Average algorithm, visually demonstrate how it differs from DBA, and conduct an empirical evaluation to showcase its superiority and ability to achieve state-of-the-art clustering performance with various elastic distances.

## 7.1 Introduction

In Chapter 5, we observed that while some elastic distances significantly outperformed *k*-means-Euclidean, others performed notably worse. We hypothesised that this discrepancy was due to pathological warping in distances that did not explicitly penalise warping off the diagonal, a problem exacerbated when using an average time series generated without considering alignment (i.e., the arithmetic mean).

The importance of incorporating alignment during centroid computation was further highlighted in Section 5, where we directly compared *k*-means-DTW to *k*-means-ba-DTW. We found that using an elastic distance in the centroid computation dramatically improved clustering performance. Moreover, this observation was consistent across all elastic distances when comparing alternate *k*-medoids to *k*-means using the same elastic distances in Section 5.4.

However, despite the improvements seen with PAM, when PAM-DTW was compared to *k*-means-ba-DTW in Section 6.5, we found *k*-means-ba-DTW significantly outperformed PAM-DTW across all evaluation metrics. Moreover, *k*-means-ba-DTW was one of the best-performing clusterers overall, even when compared to PAM variants such as PAM-MSM and PAM-TWE. This led us to hypothesise that an average representation, could be superior to a medoid-based representation for TSCL. One possible reason for this is that medoid-based algorithms rely on the existence of a “good” representative within the dataset for each cluster. If such representatives do not exist, *k*-medoids’ performance can suffer significantly.

In this chapter, we aim to develop an elastic averaging technique that can be applied to a variety of elastic distances, rather than just one (e.g. DBA). We hypothesise that if we can create an averaging method tailored to each elastic distance, it will significantly enhance the performance of those distances in *k*-means clustering, compared to using the arithmetic mean and the medoids. Furthermore, we expect that distances like MSM, TWE, and ADTW, which have already demonstrated strong clustering performance, could outperform the current state-of-the-art when used with this averaging technique.

## 7.2 Elastic Barycentre Averaging

Previously, we defined DBA and outlined how it minimises the DTW Fréchet function in Section 2.5.1. However, to the best of our knowledge, the minimisation of Fréchet functions for other elastic distances has not been explored in the literature. To address this gap, we propose a generalisation of the DTW Fréchet function to accommodate any elastic distance. Formally, we define this generalisation as follows:

$$F(z) := \frac{1}{n} \sum_{i=1}^{n} d(z, x_i)^2 \tag{7.1}$$

where $d$ represents an elastic distance.

We will now use this generalised formulation to adapt DBA to work with any elastic distance that computes a complete optimal alignment path which we refer to as the *Elastic Barycentre Average*. The Elastic Barycentre Average approximates the minimum of Equation 7.1 by adapting DBA with other elastic distance alignment path.

Formally, the Elastic Barycentre Average algorithm is presented in Algorithm 35. The key modification to the original DBA algorithm is the substitution of the DTW alignment path with that of another elastic distance, provided that it computes a complete optimal warping path.

**Algorithm 35:** elastic_barycentre(**X**, **max_iters**, **tol**)

```
Input: X (Dataset of time series of length n. Each time series is of length
       m), max_iters (Maximum number of iterations before forced
       termination), tol (Change in barycentre threshold)
Output: Elastic Barycentre Average of X for a given elastic distance
1  $barycentre \leftarrow mean(X)$
2  $previous\_dist \leftarrow \infty$
3  for $i \leftarrow 1$ to $max\_iters$ do
4      $barycentre \leftarrow elastic\_barycentre\_update(barycentre, X)$
5      $curr\_distance \leftarrow 0$
6      for each time series $curr\_ts$ in $X$ do
7          $curr\_distance \leftarrow$
             $curr\_distance + elastic\_distance(barycentre, curr\_ts)$
8      if $|previous\_dist - curr\_distance| < tol$ then
9          break
10     $previous\_dist \leftarrow curr\_distance$
11 return $barycentre$
```

The *elastic_barycentre* algorithm, shown in Algorithm 35, computes the Elastic Barycentre of the dataset $X$. The algorithm takes three parameters: first, the dataset $X$, consisting of $n$ time series, each of length $m$; second, *max_iters*, an integer that defines the maximum number of update iterations before the algorithm terminates and returns the barycentre; and finally, *tol*, a threshold for early stopping, which halts the algorithm if the total distance to the barycentre changes by less than *tol* between consecutive iterations.

Algorithm 35 begins by calculating the mean time series of the dataset $X$ (line 1). The mean average of a collection of time series is defined in Equation 2.35. A variable is then initialised to track the total distance to the barycentre from the previous iteration, starting at $\infty$ (line 2). The refinement loop follows, running for up to *max_iters* iterations (line 3). During each iteration, the barycentre is updated by calling the *elastic_barycentre_update* function (line 4). After the barycentre is updated, the total distance between each time series and the new barycentre is calculated (lines 6 and 7). The *elastic_distance* function can apply any elastic distance that produces a complete optimal warping path.

A complete optimal warping path is one that forms a continuous path through a cost matrix, starting at the index (m-1, m-1) and finishing at (0,0), assuming 0 indexing. This means that distances like LCSS, which allow "gaps" in their optimal warping path, cannot be used with the Elastic Barycentre Average.

Once the total distance to the new barycentre is obtained, the early stopping condition is checked. If the difference between the previous total distance to the previous barycentre and the current iterations total distance to the current barycentre is less than *tol*, the loop terminates early (lines 8 and 9). Otherwise, the previous distance is updated to the current iteration's distance (line 10). Once all iterations are completed (or the early stopping condition is met), the final refined barycentre is returned (line 11).

**Algorithm 36:** elastic_barycentre_update(**barycentre**, **X**)

```
Input: barycentre (Current estimate of barycentre), X (Dataset of time
       series of length n. Each time series is of length m)
Output: Updated Elastic Barycentre
1  Initialise num_alignments as an array of zeros of length $m$
2  Initialise sum_barycentre as a matrix of zeros of size $m$
3  for each time series curr_ts in X do
4      $CM \leftarrow elastic\_distance\_CM(barycentre, curr\_ts)$
5      $alignment\_path \leftarrow optimal\_warping\_path(CM)$
6      for each pair of indices $(j,k)$ in alignment_alignment do
7          $sum\_barycentre[k] \leftarrow sum\_barycentre[k] + curr\_ts[j]$
8          $num\_alignments[k] \leftarrow num\_alignments[k] + 1$
9  $new\_barycentre \leftarrow summed\_barycentre / num\_alignments$
10 return $new\_barycentre$
```

The *elastic_barycentre_update* function, shown in Algorithm 36, performs a single update of the current barycentre. It takes two parameters: *barycentre*, a time series of length *m* representing the current barycentre to be refined, and *X*, a dataset containing *n* time series, each of length *m*.

The function starts by initialising two arrays. The first, $num\_alignments$, is an array of length $m$ that tracks how many times each index in the barycentre time series is aligned to (line 1). This will later be used to divide the corresponding values in $sum\_barycentre$. The second array, $sum\_barycentre$, also of length $m$, accumulates the values that align to each specific index in the barycentre. For example, if time series $T_i$ of length 3 has an alignment path of $(0,0),(0,1),(1,1),(2,2)$ with the barycentre, $sum\_barycentre$ would be updated as follows: $sum\_barycentre_0 = T_0$, $sum\_barycentre_1 = T_0 + T_1$, and $sum\_barycentre_2 = T_2$.

This illustrates how the barycentre average differs from the arithmetic mean. Specifically, $sum\_barycentre_1$ is updated to $T_0 + T_1$ because the alignment path maps the first index of the barycentre to two different values in $T$.

With these initial variables, the update process begins by iterating through each time series in $X$ (line 3). First, an elastic distance cost matrix is computed (line 4). The algorithms for obtaining the cost matrix for a given elastic distance are outlined in Section 2.4, but instead of returning the final distance (which is the last value in the cost matrix), the entire cost matrix is returned. From this cost matrix, the optimal alignment path is extracted using the $optimal\_warping\_path$ algorithm (line 5). The $optimal\_warping\_path$ algorithm is defined in Algorithm 2. Any elastic distance that can compute its optimal warping path using Algorithm 2 may be used in line 4 (i.e. not LCSS).

Next, using the extracted alignment path (a list of tuples mapping the optimal alignment), each tuple is iterated over (line 6). In each tuple, $j$ represents the index of the current time point in the time series, which is aligned to the $kth$ time point (index) in the barycentre. Using this mapping, the value of $sum\_barycentre[k]$ is incremented by the value of $curr\_t[j]$ (line 7). To track how many times each point in $sum\_barycentre[k]$ has been aligned, the corresponding value in $num\_alignments[k]$ is incremented by 1.

This process is repeated for every time series in the dataset $X$. Once all time series have been processed, the new barycentre is calculated by dividing $sum\_barycentre$ by $num\_alignments$ (line 9). This yields the $new\_barycentre$, which is then returned (line 10).

## 7.3 Elastic Barycentre analysis

Although the update to the original DBA algorithm is straightforward, we observe significant variation in the averages produced for each distance measure. Figure 7.1 illustrates the Elastic Barycentre Averages for 10 different elastic distances over class 1 of the GunPoint dataset. Additionally, the figure includes the arithmetic mean (in red) for reference.

One of the most notable features of Figure 7.1 is the distinct variation in the barycentres produced by each elastic distance. While we anticipated some differences, the extent of the variation is greater than expected. Starting at the top of Figure 7.1, the arithmetic mean, which does not account for alignment, appears generally smooth, exhibiting a dome-like shape.

BA-DTW (DBA) forms a similar dome shape but with a lower peak and a longer, sharper plateau. Additionally, there is a noticeable peak at the centre. BA-DTW closely resembles BA-WDTW, although the peak in BA-WDTW occurs later along the time axis. BA-ADTW, BA-MSM, BA-ERP, and BA-TWE show a similar overall structure, characterised by a sharper incline to a peak followed by a smooth descent. The primary differences among them lie in the finer local variations. For instance, BA-TWE exhibits subtle fluctuations near the peak before a smooth decline, whereas BA-MSM shows numerous local fluctuations as the barycentre descends.

BA-shape-DTW stands out with a dome-like structure but features a depression in the middle. BA-DDTW appears shifted along the time axis compared to the

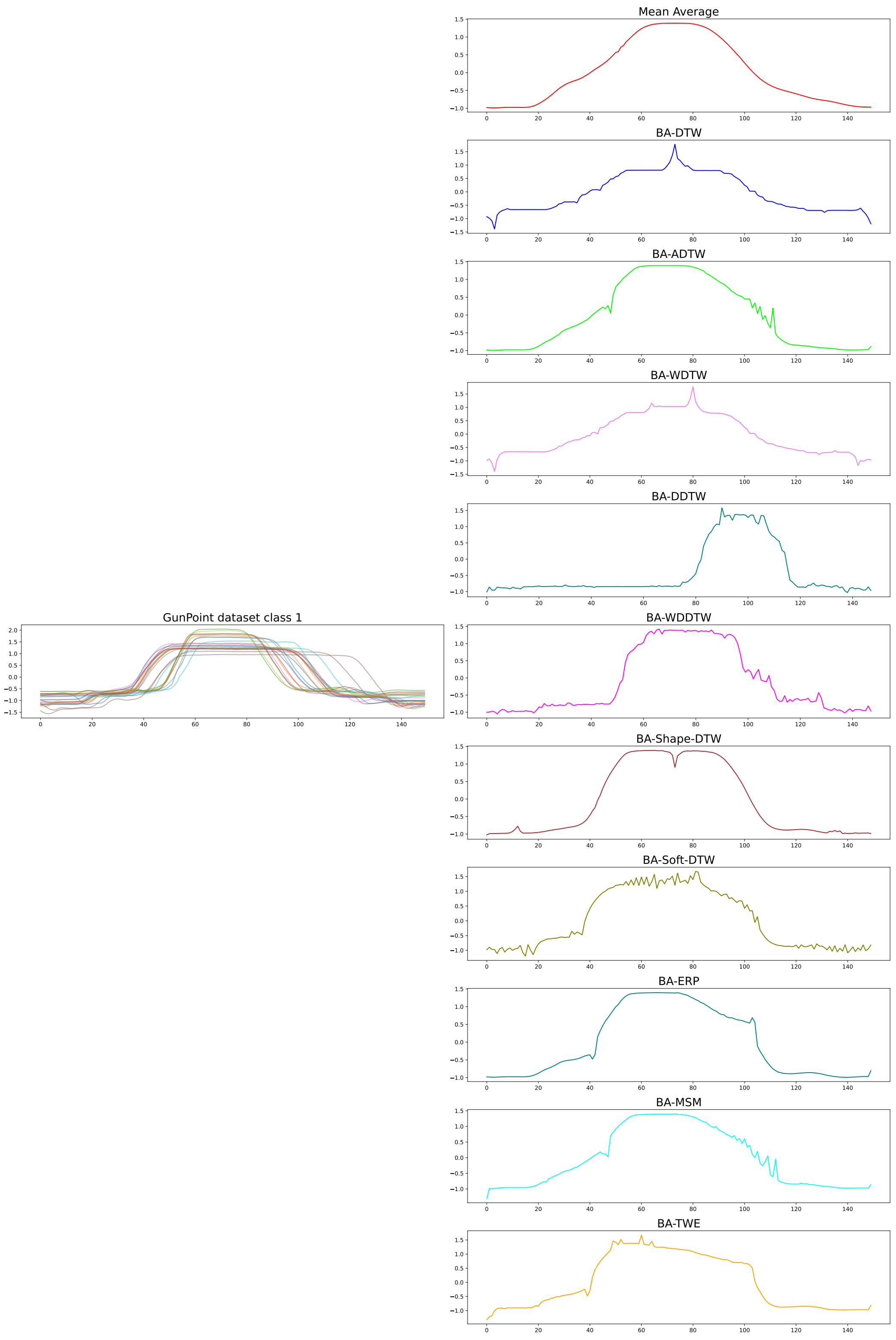


Fig. 7.1 Different Elastic Barycentre Averages for Gunpoint class 1.

other barycentres and has a more compressed overall shape. BA-soft-DTW shares several characteristics with BA-MSM and BA-ADTW but exhibits more localised fluctuations. Finally, BA-WDDTW shows a steep ascent and descent around the dome, with significant localised fluctuations.

While visually inspecting each barycentre provides some insight, it does not offer a quantitative evaluation of their quality. Therefore, we aim to design an experiment to assess their effectiveness using the *k*-means clusterer. In the next section, we outline our experimental approach to evaluate each Elastic Barycentre.

## 7.4 Experiment setup

We will now outline our experiments to evaluate the Elastic Barycentre Average. Each Elastic Barycentre will be assessed using the *k*-means clusterer, where the Elastic Barycentre technique serves as the centroid computation method, and the same elastic distance is used in the assignment computation.

We will compare the results of each Elastic Barycentre clusterer to each other to determine the best Elastic Barycentre averaging technique and assess whether the rank order of each distance is consistent with our previous experiments. Additionally, we will compare the results to the baseline clusterers and the best-performing *k*-medoids techniques identified in Chapter 6 to contextualise results.

Next, we will conduct a more detailed analysis of each Elastic Barycentre Average to evaluate how much the averaging technique contributes to the success of each clusterer. This will involve comparing the Elastic Barycentre Average results to those from Chapter 5 and Chapter 6. By comparing the Elastic Barycentre results to other Lloyd's-based algorithm using the same elastic distances but with different centroid computation techniques (arithmetic mean and medoids), we can isolate the specific impact each Elastic Barycentre has on the final clustering performance.

Table 7.1 shows the configuration of each model used in our experiments. Four elastic distances were excluded from the analysis. LCSS was omitted because it is incompatible with the Elastic Barycentre Average, as it does not produce a complete alignment path (i.e., it allows gaps). DDTW and WDDTW were excluded because the resulting barycentre averages are two time points shorter than the input time series, which introduces complexity when computing distances between time series of different lengths in the *k*-means algorithm. Finally, EDR was excluded due to computational resource limitations, which required us to narrow the scope of our experiments.

| | *max_iters* | *tol* | *n_init* | *init_algo* | *distance* | *averaging* |
|---|---|---|---|---|---|---|
| *k*-means-ba-adtw | 50 | $1\times10^{-6}$ | 10 | Forgy | ADTW | BA-ADTW |
| *k*-means-ba-dtw | 50 | $1\times10^{-6}$ | 10 | Forgy | DTW | BA-DTW |
| *k*-means-ba-erp | 50 | $1\times10^{-6}$ | 10 | Forgy | ERP | BA-ERP |
| *k*-means-ba-msm | 50 | $1\times10^{-6}$ | 10 | Forgy | MSM | BA-MSM |
| *k*-means-ba-twe | 50 | $1\times10^{-6}$ | 10 | Forgy | TWE | BA-TWE |
| *k*-means-ba-wdtw | 50 | $1\times10^{-6}$ | 10 | Forgy | WDTW | BA-WDTW |
| *k*-means-ba-shape-dtw | 50 | $1\times10^{-6}$ | 10 | Forgy | shape-DTW | BA-shape-DTW |
| *k*-means-ba-soft-dtw | 50 | $1\times10^{-6}$ | 10 | Forgy | soft-DTW | BA-soft-DTW |

Table 7.1 Elastic Barycentre model parameters

The parameters set in Table 7.1 are the same as those used in our *k*-means experiments in Chapter 5 and alternate *k*-medoids experiments in Chapter 6. The only change made for our Elastic Barycentre averaging experiments is the averaging technique which is specified in the "averaging" column.

We have adopted a unique naming convention for each barycentre averaging technique, differing from the original Dynamic Time Warping Barycentre Average (DBA). This is because multiple distances share the same first character, leading to potential ambiguity in their names. Therefore, for clarity, our naming convention starts with "BA" (Barycentre Average), followed by a "-" and then the specific distance used.

For instance, the clusterer $k$-means-ba-DTW refers to the $k$-means algorithm that uses the barycentre average ("ba") with the DTW distance for both the assignment stage and the barycentre averaging method. Similarly, when the barycentre average is used with the TWE distance in $k$-means, the clusterer is called $k$-means-ba-TWE.

We also produce a new barycentre average for soft-DTW. As outlined in Section 2.5.1, the soft-DBA averaging technique already exists. The key difference is that soft-DBA exactly minimises the barycentre average function, while BA-soft-DTW uses an estimate of the minimum. Although we do not expect BA-soft-DTW to outperform soft-DBA, it serves as an interesting point of comparison to evaluate the accuracy of the estimates it produces.

Each elastic distance used in both the assignment and averaging phases requires specific parameters. We use the elastic distance parameters outlined in Table 5.2, applying the same parameters for both the assignment and averaging stage. These parameters are consistent with those used in all previous experiments, allowing for direct comparison.

| | *max_iters* | *tol* | *init_barycentre* | *distance* |
|---|---|---|---|---|
| BA-ADTW | 50 | $1 \times 10^{-6}$ | Arithmetic mean | ADTW |
| BA-DTW | 50 | $1 \times 10^{-6}$ | Arithmetic mean | DTW |
| BA-ERP | 50 | $1 \times 10^{-6}$ | Arithmetic mean | ERP |
| BA-MSM | 50 | $1 \times 10^{-6}$ | Arithmetic mean | MSM |
| BA-TWE | 50 | $1 \times 10^{-6}$ | Arithmetic mean | TWE |
| BA-WDTW | 50 | $1 \times 10^{-6}$ | Arithmetic mean | WDTW |
| BA-shape-DTW | 50 | $1 \times 10^{-6}$ | Arithmetic mean | shape-DTW |
| BA-soft-DTW | 50 | $1 \times 10^{-6}$ | Arithmetic mean | soft-DTW |

Table 7.2 Elastic Barycentre averaging parameters

Finally, each barycentre averaging method has its own set of parameters. Table 7.2 outlines the parameters used for each Elastic Barycentre Average. We set `max_iters` to 50, as this value has been shown to ensure sufficient convergence [108]. The `tol` parameter is set to a very small value, ensuring that early convergence occurs only if very minor changes are observed. We use the arithmetic mean as the initial barycentre, following the recommendation in the original paper [94]. Although we could use a medoid as the initial barycentre, doing so would introduce significant computational overhead and could disproportionately affect certain distances. Lastly, the "distance" column represents our independent variable, which is changed for each Elastic Barycentre Average.

We now conduct our Elastic Barycentre Averaging experiments using the methodology and parameters defined above.

## 7.5 Elastic Barycentre clusterer results

### 7.5.1 Combined test-train split

Figure 7.6 presents the critical difference diagram for eight different Elastic Barycentre clusterers. Across all evaluation metrics, *k*-means-ba-TWE is the best-performing clusterer and consistently ranks in the top clique. Joining *k*-means-ba-TWE in the top clique are *k*-means-ba-MSM and *k*-means-ba-shape-DTW for all evaluation metrics. Additionally, *k*-means-ba-soft-DTW is in the top clique for all evaluation metrics except ARI. Notably, five Elastic Barycentre clusterers outperform *k*-means-ba-DTW across all evaluation metrics.

However, Figure 7.6 only includes results from only 83 datasets. This is because *k*-means-ba-shape-DTW and *k*-means-ba-soft-DTW failed to complete over 17 datasets each within our seven-day runtime limit. This issue is highlighted in Figure 7.7, which shows the FitTime of each model.

Due to the large number of missing datasets, we will exclude *k*-means-ba-shape-DTW and *k*-means-ba-soft-DTW from further analysis to allow for evaluation over a larger number of datasets. However, we will include them when comparing results to *k*-means-soft-DBA, as it also has a similar number of missing datasets.

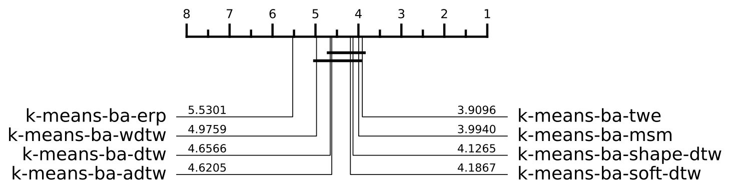


Fig. 7.2 AMI

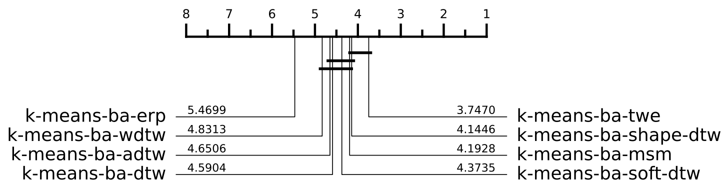


Fig. 7.3 ARI

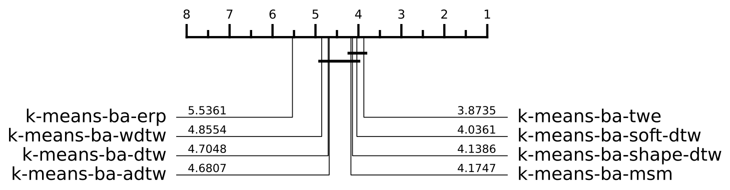


Fig. 7.4 CLACC

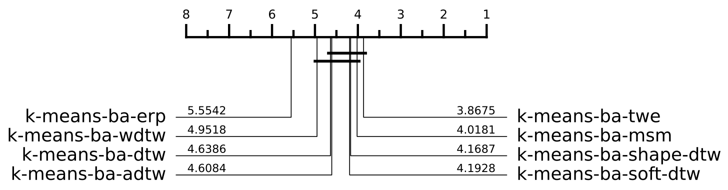


Fig. 7.5 NMI

Fig. 7.6 CD diagrams of Elastic Barycentre *k*-means over 83 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.30

Additionally, for the test-train split, where both *k*-means-ba-shape-DTW and *k*-means-ba-soft-DTW have significantly more completed results, we will conduct further analysis.

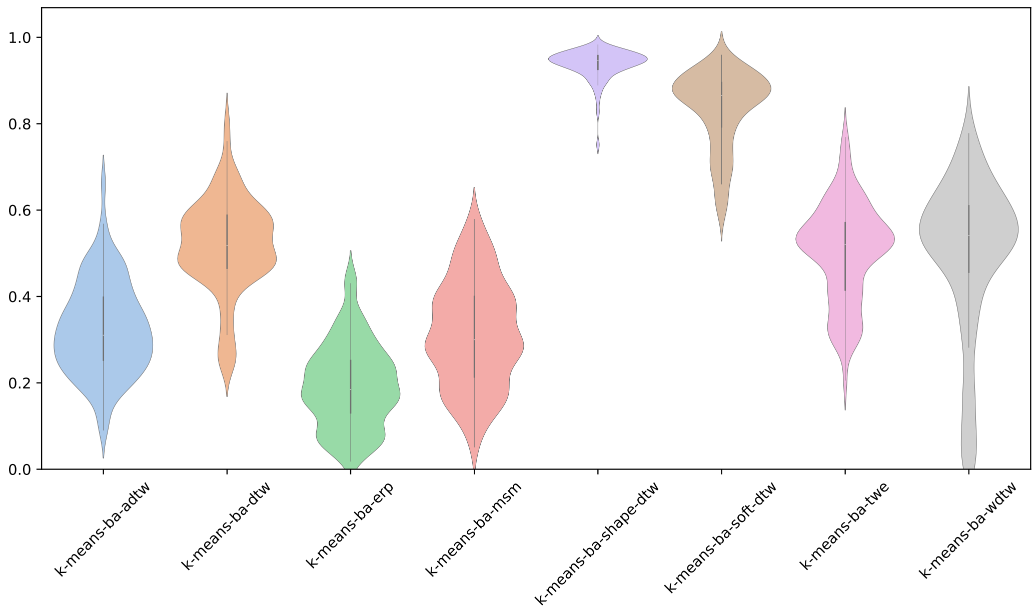


Fig. 7.7 Relative FitTime violin plot for each Elastic Barycentre clusterer over 83 of the UCR archive using the combined test-train split.

Figure 7.12 presents the critical difference diagram for six Elastic Barycentre clusterers over 106 datasets. We observe a similar rank order results, but *k*-means-ba-MSM achieves the highest average ranking for AMI, CLACC, and NMI, while *k*-means-ba-TWE ranks highest for ARI. *k*-means-ba-TWE, *k*-means-ba-MSM, *k*-means-ba-ADTW, and *k*-means-ba-DTW consistently appear in the top clique for all evaluation metrics, while *k*-means-ba-ERP is the worst-performing clusterer across all metrics.

Table 7.3 presents the average score summary for each Elastic Barycentre clusterer across five evaluation metrics. *k*-means-ba-TWE achieves the highest average score across all metrics, with *k*-means-ba-MSM closely following in second place. Additionally, Table 7.4 shows the average ARI scores for different domains. *k*-means-ba-TWE performs best in the Image, Sensor, and ECG domains, while *k*-means-ba-DTW leads in the Simulated, Device, and Motion domains, and *k*-means-ba-ADTW excels in the Spectro domain. Overall, no single Elastic Barycentre

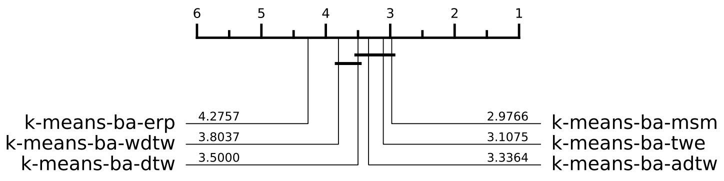


Fig. 7.8 AMI

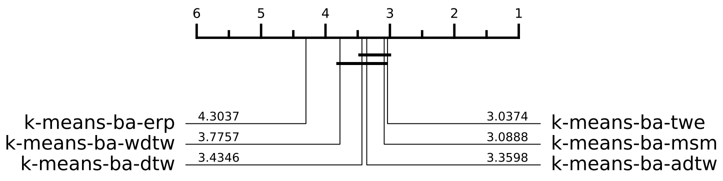


Fig. 7.9 ARI

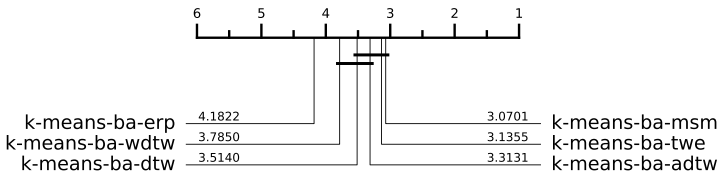


Fig. 7.10 CLACC

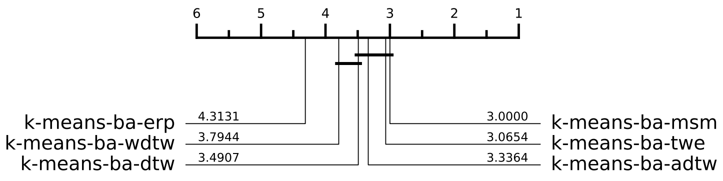


Fig. 7.11 NMI

Fig. 7.12 CD diagrams of Elastic Barycentre *k*-means over 106 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.31

clusterer dominates across all domains, although *k*-means-ba-DTW ranks best in three domains and *k*-means-ba-TWE in two.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-adtw | 0.252 | 0.305 | 0.566 | 0.328 | 0.711 |
| k-means-ba-dtw | 0.256 | 0.304 | 0.569 | 0.328 | 0.712 |
| k-means-ba-erp | 0.207 | 0.257 | 0.530 | 0.282 | 0.695 |
| k-means-ba-msm | 0.256 | 0.309 | 0.571 | 0.331 | 0.709 |
| k-means-ba-twe | **0.273** | **0.317** | **0.580** | **0.340** | **0.716** |
| k-means-ba-wdtw | 0.241 | 0.294 | 0.557 | 0.318 | 0.707 |

Table 7.3 Summary of average score across multiple evaluation metrics over 106 datasets from the UCR archive using the combined test-train split.

To contextualise our Elastic Barycentre clusterer results, we include all baseline clusterers as well as PAM results using the same elastic distances. Figure 7.18 shows the critical difference diagrams for the Elastic Barycentre clusterers alongside the baseline clusterers and PAM.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-adtw | 0.320 | **0.220** | 0.222 | 0.380 | 0.146 | 0.170 | 0.312 |
| k-means-ba-dtw | 0.307 | 0.209 | 0.195 | **0.586** | **0.173** | **0.177** | 0.246 |
| k-means-ba-erp | 0.259 | 0.212 | 0.199 | 0.306 | 0.079 | 0.113 | 0.279 |
| k-means-ba-msm | 0.338 | 0.217 | 0.233 | 0.340 | 0.142 | 0.158 | 0.335 |
| k-means-ba-twe | **0.364** | 0.211 | **0.255** | 0.398 | 0.158 | 0.142 | **0.373** |
| k-means-ba-wdtw | 0.304 | 0.204 | 0.209 | 0.493 | 0.080 | 0.157 | 0.262 |

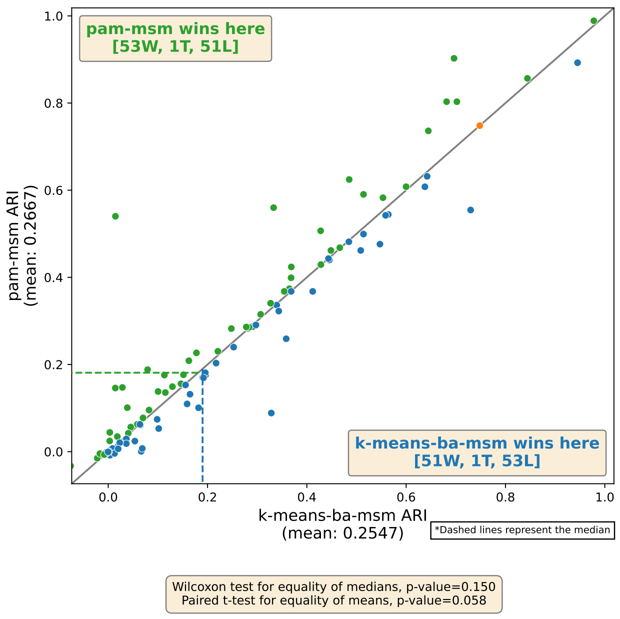
Table 7.4 Average ARI score on problems split by problem domain over 106 datasets from the UCR archive using the combined test-train split.

Figure 7.18 demonstrates that Elastic Barycentre Average clusterers outperform PAM with the same elastic distance across all evaluation metrics. Additionally, all Elastic Barycentre clusterers outperform *k*-means-Euclidean.

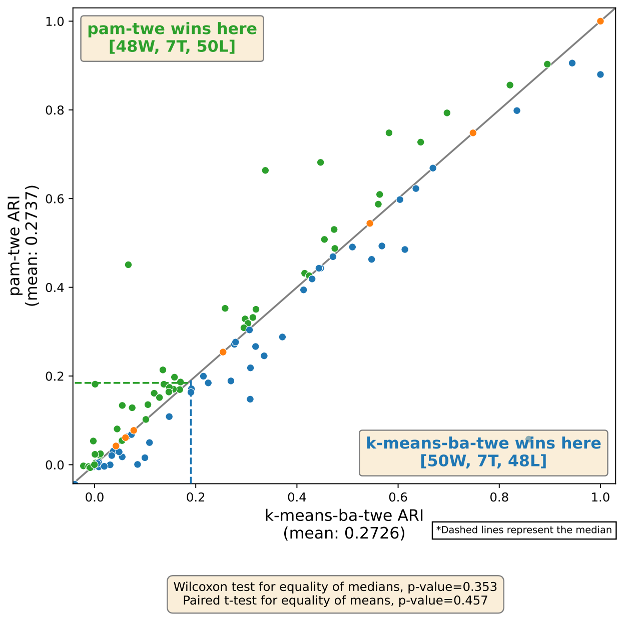
The top clique across all evaluation metrics includes *k*-means-ba-TWE, *k*-means-ba-MSM, *k*-means-ba-DTW, *k*-means-ba-ADTW, PAM-MSM, PAM-TWE, and PAM-ADTW. The differences between *k*-means-ba-TWE and PAM-TWE, as well as *k*-means-ba-MSM and PAM-MSM, are very small, as shown in Figure 7.13. For both PAM-TWE and PAM-MSM, their average ARI is higher than their corresponding Elastic Barycentre clusterer counterparts, primarily because PAM-TWE and PAM-MSM significantly outperform *k*-means-ba-TWE and *k*-means-ba-MSM for certain datasets.

Figure 7.23 shows the critical difference diagrams for all Elastic Barycentre clusterers (including *k*-means-ba-shape-DTW and *k*-means-ba-soft-DTW) with PAM and the baseline clusterers, in addition to *k*-means-soft-DBA. While no clusterer is better than *k*-means-soft-DBA on average, for all evaluation metrics, *k*-means-ba-TWE, *k*-means-ba-shape-DTW, *k*-means-ba-MSM, PAM-TWE, PAM-soft-DTW appear in the same clique and are therefore not significantly different from *k*-means-soft-DBA.

Furthermore, if we analyse the average scores shown in Table 7.5 while *k*-means-soft-DBA has the highest average score for every evaluation metric, the gap



(a) $k$-means-ba-MSM compared to PAM-MSM



(b) $k$-means-ba-TWE compared to PAM-TWE

Fig. 7.13 $k$-means-ba-MSM and $k$-means-ba-TWE results compared directly to PAM-MSM and PAM-TWE respectively over 105 datasets from the UCR archive using the combined test-train split.

has significantly shrunk. For ARI the gap between the previous state-of-the-art ($k$-means-ba-DTW) was 0.04 ARI. However, compared to $k$-means-ba-shape-DTW the gap is only 0.015 and for $k$-means-ba-TWE the gap is only 0.017.

Figure 7.6 shows the average ARI score by domain. Notably when we consider the average ARI by domain $k$-means-soft-DBA is only best for one domain: Device. Between the different Elastic Barycentre clusterers, they perform best for four of the seven domains. Specifically $k$-means-ba-TWE is best at two domains: Image and Spectro.

Finally, we analyse the FitTime of each Elastic Barycentre clusterer compared to the baseline clusterers and PAM. Figure 7.24 shows the violin plot for FitTime. We observe that the relative FitTime for Elastic Barycentre Average is higher than PAM using the same distance; as previously explained, this difference arises from our experimental methodology. Specifically, with Forgy initialisation and 10 restarts, PAM benefits from precomputing the distance matrix and reusing it across

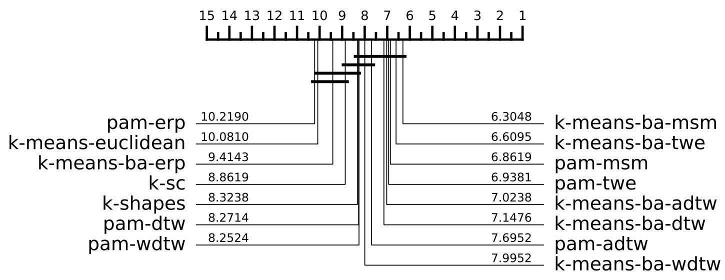


Fig. 7.14 AMI

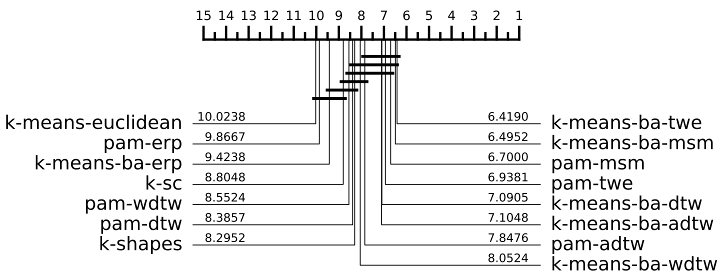


Fig. 7.15 ARI

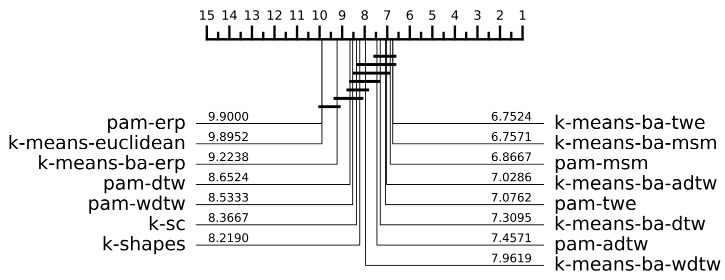


Fig. 7.16 CLACC

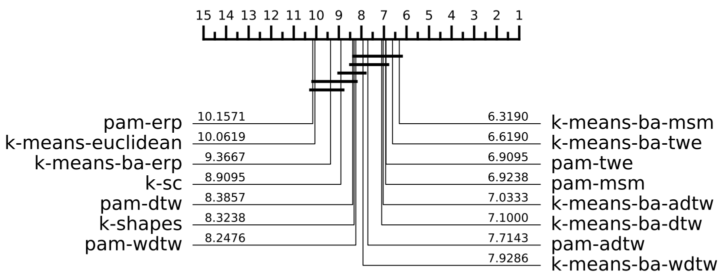


Fig. 7.17 NMI

Fig. 7.18 CD diagrams of Elastic Barycentre *k*-means with the baseline clusterers over 105 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.32.

all restarts. In contrast, the Elastic Barycentre Average methods must recompute distances and averages at every iteration and restart. Therefore, if we conducted the same experiment using only one restart, the Elastic Barycentre Average methods would likely be faster than PAM when employing the same elastic distance.

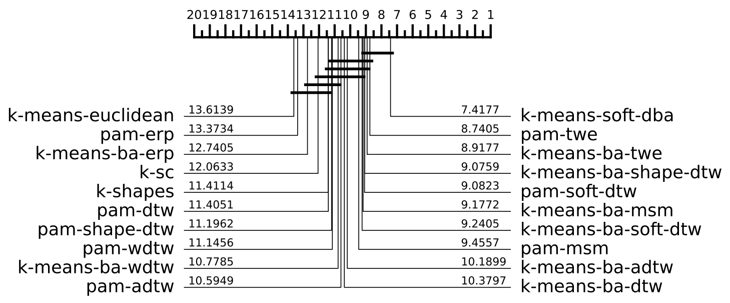


Fig. 7.19 AMI

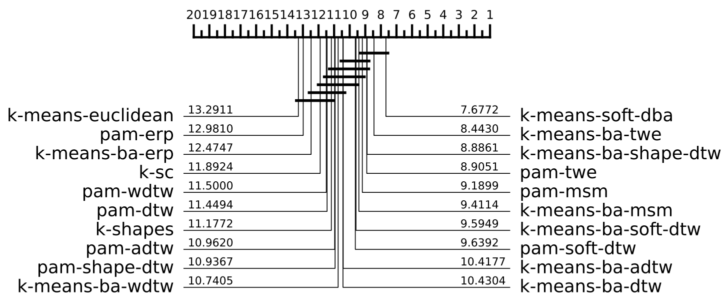


Fig. 7.20 ARI

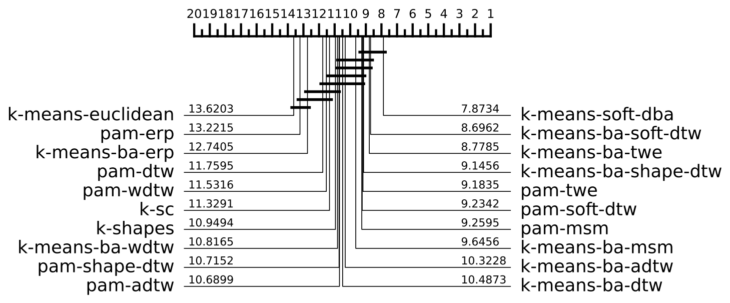


Fig. 7.21 CLACC

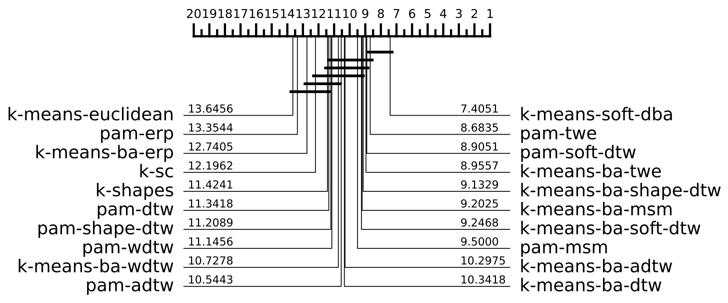


Fig. 7.22 NMI

Fig. 7.23 CD diagrams of Elastic Barycentre *k*-means with the baseline clusterers and soft-DBA over 79 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.33.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-adtw | 0.259 | 0.301 | 0.597 | 0.311 | 0.685 |
| k-means-ba-dtw | 0.265 | 0.303 | 0.602 | 0.313 | 0.687 |
| k-means-ba-erp | 0.207 | 0.245 | 0.555 | 0.256 | 0.666 |
| k-means-ba-msm | 0.260 | 0.300 | 0.600 | 0.310 | 0.683 |
| k-means-ba-shape-dtw | 0.290 | 0.325 | 0.617 | 0.335 | 0.698 |
| k-means-ba-soft-dtw | 0.267 | 0.303 | 0.614 | 0.314 | 0.683 |
| k-means-ba-twe | 0.288 | 0.319 | 0.618 | 0.329 | 0.694 |
| k-means-ba-wdtw | 0.249 | 0.291 | 0.592 | 0.301 | 0.680 |
| k-means-euclidean | 0.200 | 0.239 | 0.543 | 0.251 | 0.664 |
| k-means-soft-dba | **0.305** | **0.338** | **0.632** | **0.347** | **0.704** |
| k-sc | 0.220 | 0.251 | 0.573 | 0.262 | 0.643 |
| k-shapes | 0.240 | 0.286 | 0.587 | 0.297 | 0.678 |
| pam-adtw | 0.268 | 0.308 | 0.602 | 0.318 | 0.687 |
| pam-dtw | 0.246 | 0.288 | 0.590 | 0.299 | 0.676 |
| pam-erp | 0.191 | 0.232 | 0.546 | 0.244 | 0.660 |
| pam-msm | 0.280 | 0.314 | 0.616 | 0.324 | 0.695 |
| pam-shape-dtw | 0.267 | 0.305 | 0.603 | 0.316 | 0.685 |
| pam-soft-dtw | 0.277 | 0.317 | 0.614 | 0.327 | 0.686 |
| pam-twe | 0.292 | 0.324 | 0.625 | 0.334 | 0.699 |
| pam-wdtw | 0.243 | 0.285 | 0.589 | 0.296 | 0.672 |

Table 7.5 Summary of average score across multiple evaluation metrics over 79 datasets from the UCR archive using the combined test-train split.

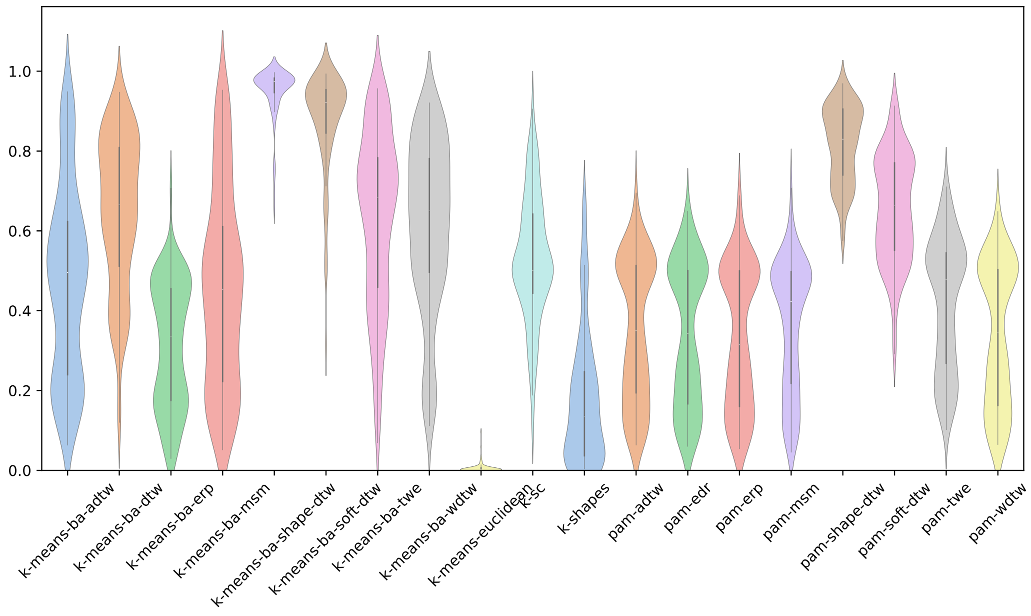


Fig. 7.24 Relative FitTime violin plot for Elastic Barycentre clusterers, PAM and the baseline clusterers over 105 datasets from the UCR archive using the combined test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-adtw | 0.321 | 0.267 | 0.273 | 0.298 | 0.192 | 0.130 | 0.236 |
| k-means-ba-dtw | 0.307 | 0.257 | 0.234 | **0.536** | 0.217 | 0.140 | 0.137 |
| k-means-ba-erp | 0.254 | 0.258 | 0.249 | 0.216 | 0.108 | 0.068 | 0.184 |
| k-means-ba-msm | 0.341 | 0.267 | 0.278 | 0.254 | 0.181 | 0.120 | 0.224 |
| k-means-ba-shape-dtw | 0.328 | 0.238 | 0.317 | 0.369 | 0.247 | **0.155** | 0.375 |
| k-means-ba-soft-dtw | 0.306 | 0.188 | 0.294 | 0.416 | 0.237 | 0.146 | 0.207 |
| k-means-ba-twe | **0.372** | 0.263 | **0.320** | 0.320 | 0.213 | 0.113 | 0.267 |
| k-means-ba-wdtw | 0.304 | 0.258 | 0.257 | 0.447 | 0.093 | 0.113 | 0.164 |
| k-means-euclidean | 0.242 | 0.272 | 0.238 | 0.221 | 0.072 | 0.074 | 0.174 |
| k-means-soft-dba | 0.371 | 0.265 | 0.314 | 0.471 | **0.273** | 0.144 | 0.192 |
| k-sc | 0.244 | 0.291 | 0.313 | 0.085 | 0.041 | 0.097 | 0.395 |
| k-shapes | 0.264 | 0.225 | 0.226 | 0.390 | 0.110 | 0.141 | **0.407** |
| pam-adtw | 0.320 | **0.297** | 0.270 | 0.368 | 0.149 | 0.132 | 0.291 |
| pam-dtw | 0.292 | 0.245 | 0.246 | 0.409 | 0.182 | 0.123 | 0.149 |
| pam-erp | 0.247 | 0.256 | 0.213 | 0.165 | 0.128 | 0.052 | 0.187 |
| pam-msm | 0.365 | 0.248 | 0.287 | 0.319 | 0.214 | 0.110 | 0.356 |
| pam-shape-dtw | 0.317 | 0.269 | 0.276 | 0.344 | 0.156 | 0.124 | 0.397 |
| pam-soft-dtw | 0.350 | 0.166 | 0.289 | 0.478 | 0.222 | 0.130 | 0.185 |
| pam-twe | 0.361 | 0.263 | 0.295 | 0.413 | 0.224 | 0.113 | 0.351 |
| pam-wdtw | 0.290 | 0.255 | 0.233 | 0.470 | 0.090 | 0.126 | 0.172 |

Table 7.6 Average ARI score on problems split by problem domain over 79 datasets from the UCR archive using the combined test-train split.

### 7.5.2 Test-train split

Figure 7.29 shows the critical difference diagrams for eight Elastic Barycentre clusterers across 98 datasets using the test-train split. The ranking is similar to that observed in the combined test-train split, with *k*-means-ba-MSM and *k*-means-ba-TWE being the top two performers, consistently appearing in the top clique. Additionally, *k*-means-ba-ADTW, *k*-means-ba-shape-DTW, and *k*-means-ba-soft-DTW are also in the top clique across all evaluation metrics. However, in the test-train split, *k*-means-ba-DTW does not appear in the top clique for any evaluation metric, unlike in the combined test-train split.

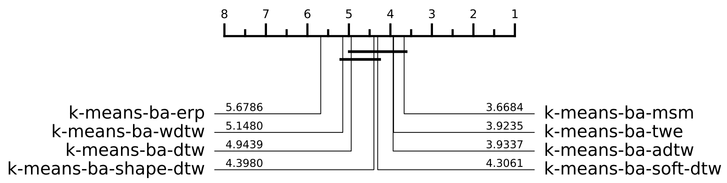


Fig. 7.25 AMI

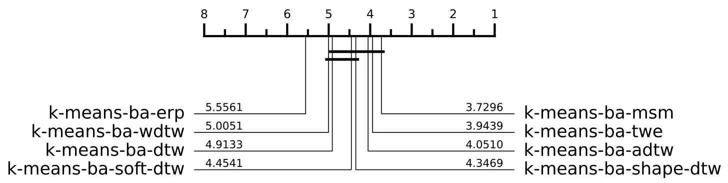


Fig. 7.26 ARI

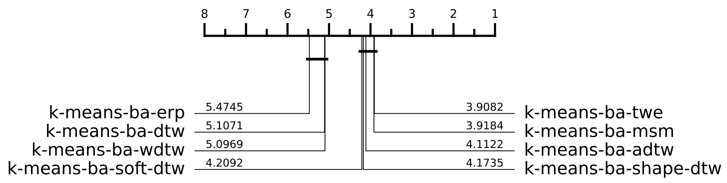


Fig. 7.27 CLACC

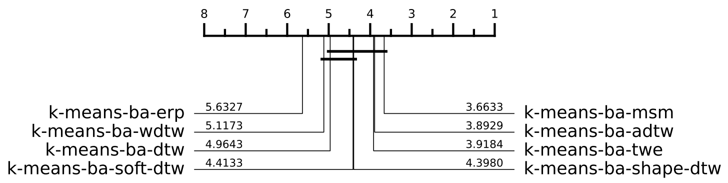


Fig. 7.28 NMI

Fig. 7.29 CD diagrams of Elastic Barycentre *k*-means over 98 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.34.

Table 7.7 shows the average score for each Elastic Barycentre clusterer. Similar to the combined test-train split, *k*-means-ba-TWE is the best-performing clusterer across all evaluation metrics, closely followed by *k*-means-ba-MSM.

Table 7.8 shows the performance of each Elastic Barycentre clusterer by problem domain. The same clusterers perform well across domains, consistent with the combined test-train split. Notably, *k*-means-ba-TWE performs best in only one domain: Sensor, although it is just 0.001 ARI from being the top performer in the Image domain. Additionally, in the Device and ECG domains, *k*-means-ba-shape-DTW shows dominant performance, significantly outperforming the second-best clusterer in both cases.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-adtw | 0.228 | 0.279 | 0.569 | 0.313 | 0.694 |
| k-means-ba-dtw | 0.226 | 0.274 | 0.562 | 0.309 | 0.694 |
| k-means-ba-erp | 0.179 | 0.223 | 0.529 | 0.260 | 0.674 |
| k-means-ba-msm | 0.238 | 0.285 | 0.572 | 0.319 | 0.696 |
| k-means-ba-shape-dtw | 0.237 | 0.284 | 0.575 | 0.319 | 0.697 |
| k-means-ba-soft-dtw | 0.229 | 0.282 | 0.572 | 0.317 | 0.690 |
| k-means-ba-twe | **0.247** | **0.294** | **0.576** | **0.328** | **0.698** |
| k-means-ba-wdtw | 0.217 | 0.267 | 0.555 | 0.303 | 0.688 |

Table 7.7 Summary of average score across multiple evaluation metrics over 98 datasets from the UCR archive using the test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-adtw | 0.293 | 0.225 | 0.166 | 0.346 | 0.192 | 0.171 | 0.195 |
| k-means-ba-dtw | 0.268 | 0.228 | 0.179 | **0.454** | 0.175 | 0.156 | 0.129 |
| k-means-ba-erp | 0.230 | 0.215 | 0.167 | 0.257 | 0.047 | 0.119 | 0.173 |
| k-means-ba-msm | **0.319** | 0.212 | 0.193 | 0.329 | 0.161 | 0.159 | 0.267 |
| k-means-ba-shape-dtw | 0.268 | 0.213 | 0.200 | 0.358 | **0.205** | 0.173 | **0.357** |
| k-means-ba-soft-dtw | 0.291 | 0.179 | 0.156 | 0.412 | 0.196 | **0.192** | 0.181 |
| k-means-ba-twe | 0.318 | 0.228 | **0.246** | 0.306 | 0.178 | 0.136 | 0.268 |
| k-means-ba-wdtw | 0.262 | **0.245** | 0.177 | 0.417 | 0.110 | 0.155 | 0.136 |

Table 7.8 Average ARI score on problems split by problem domain over 98 datasets from the UCR archive using the test-train split.

We contextualise our results by incorporating the baseline clusterers, soft-DBA and PAM using the same elastic distances as in the Elastic Barycentre clusterers. Figure 7.34 presents the critical difference diagrams for the Elastic Barycentre clusterers alongside the baseline clusterers, soft-DBA and PAM.

Figure 7.34 shows results similar to the combined test-train split, with the exception that PAM-TWE outperforms *k*-means-soft-DBA in CLACC and ARI. Additionally, while the top clique remains consistent with the combined test-train split, it includes fewer clusterers: PAM-TWE, *k*-means-soft-DBA, PAM-MSM, *k*-means-ba-TWE, and *k*-means-ba-MSM across all evaluation metrics.

Furthermore, we observe that all Elastic Barycentre clusterers outperform *k*-means-Euclidean. Additionally, all Elastic Barycentre clusterers outperform their PAM counterparts with the same elastic distance, except for PAM-TWE and PAM-MSM. For AMI, ARI, and NMI, *k*-means-ba-MSM outperforms PAM-MSM, but for CLACC, PAM-MSM outperforms *k*-means-ba-MSM. Notably, PAM-TWE consistently outperforms *k*-means-ba-TWE across all evaluation metrics.

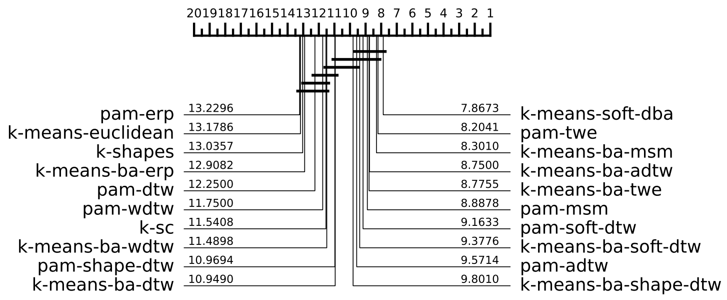


Fig. 7.30 AMI

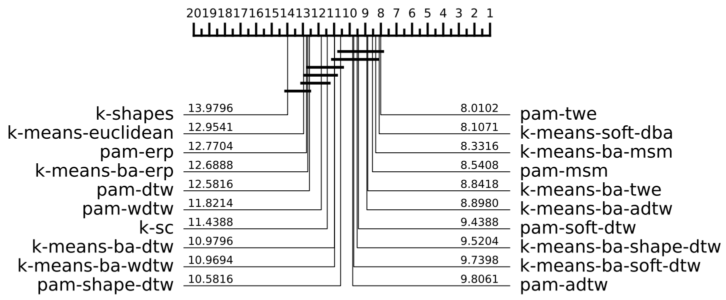


Fig. 7.31 ARI

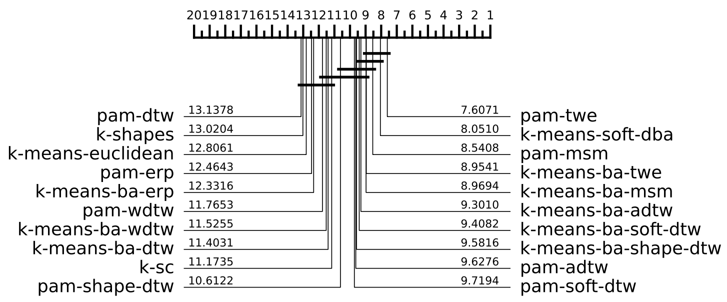


Fig. 7.32 CLACC

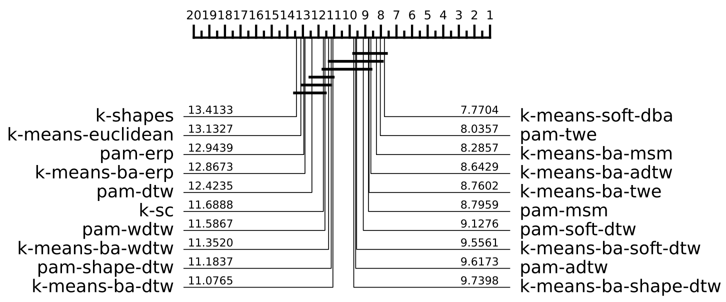


Fig. 7.33 NMI

Fig. 7.34 CD diagrams of Elastic Barycentre *k*-means with the baseline clusterers and soft-DBA over 98 datasets from the UCR archive using the test train split. Missing datasets are outlined in Table A.35.

Table 7.9 presents the average scores for the test-train split, comparing the Elastic Barycentre clusterers, baseline clusterers, soft-DBA, and PAM. *k*-means-soft-DBA is the best performing clusterer for three of the five evaluation metrics. *k*-means-ba-soft-TWE achieves the highest average ARI, while PAM-TWE achieves the highest RI. This differs from the combined test-train split where *k*-means-soft-DBA had the highest average score for every evaluation metric.

Table 7.10 shows the performance of the Elastic Barycentre clusterers, baseline clusterers, soft-DBA, and PAM across different problem domains. The results are similar to those observed in the combined test-train split.

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-adtw | 0.228 | 0.279 | 0.569 | 0.313 | 0.694 |
| k-means-ba-dtw | 0.226 | 0.274 | 0.562 | 0.309 | 0.694 |
| k-means-ba-erp | 0.179 | 0.223 | 0.529 | 0.260 | 0.674 |
| k-means-ba-msm | 0.238 | 0.285 | 0.572 | 0.319 | 0.696 |
| k-means-ba-shape-dtw | 0.237 | 0.284 | 0.575 | 0.319 | 0.697 |
| k-means-ba-soft-dtw | 0.229 | 0.282 | 0.572 | 0.317 | 0.690 |
| k-means-ba-twe | **0.247** | 0.294 | 0.576 | 0.328 | 0.698 |
| k-means-ba-wdtw | 0.217 | 0.267 | 0.555 | 0.303 | 0.688 |
| k-means-euclidean | 0.177 | 0.219 | 0.523 | 0.257 | 0.675 |
| k-means-soft-dba | 0.246 | **0.300** | **0.588** | **0.334** | 0.701 |
| k-sc | 0.185 | 0.230 | 0.537 | 0.263 | 0.658 |
| k-shapes | 0.110 | 0.175 | 0.485 | 0.206 | 0.587 |
| pam-adtw | 0.229 | 0.275 | 0.576 | 0.309 | 0.692 |
| pam-dtw | 0.206 | 0.257 | 0.551 | 0.293 | 0.682 |
| pam-erp | 0.179 | 0.224 | 0.528 | 0.262 | 0.676 |
| pam-msm | 0.238 | 0.286 | 0.580 | 0.319 | 0.698 |
| pam-shape-dtw | 0.227 | 0.273 | 0.572 | 0.307 | 0.693 |
| pam-soft-dtw | 0.238 | 0.285 | 0.581 | 0.320 | 0.695 |
| pam-twe | 0.245 | 0.291 | 0.586 | 0.326 | **0.702** |
| pam-wdtw | 0.215 | 0.269 | 0.561 | 0.304 | 0.687 |

Table 7.9 Summary of average score across multiple evaluation metrics over 98 datasets from the UCR archive using the test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| k-means-ba-adtw | 0.293 | 0.225 | 0.166 | 0.346 | 0.192 | 0.171 | 0.195 |
| k-means-ba-dtw | 0.268 | 0.228 | 0.179 | 0.454 | 0.175 | 0.156 | 0.129 |
| k-means-ba-erp | 0.230 | 0.215 | 0.167 | 0.257 | 0.047 | 0.119 | 0.173 |
| k-means-ba-msm | **0.319** | 0.212 | 0.193 | 0.329 | 0.161 | 0.159 | 0.267 |
| k-means-ba-shape-dtw | 0.268 | 0.213 | 0.200 | 0.358 | 0.205 | 0.173 | 0.357 |
| k-means-ba-soft-dtw | 0.291 | 0.179 | 0.156 | 0.412 | 0.196 | **0.192** | 0.181 |
| k-means-ba-twe | 0.318 | 0.228 | **0.246** | 0.306 | 0.178 | 0.136 | 0.268 |
| k-means-ba-wdtw | 0.262 | 0.245 | 0.177 | 0.417 | 0.110 | 0.155 | 0.136 |
| k-means-euclidean | 0.219 | 0.241 | 0.164 | 0.274 | 0.040 | 0.115 | 0.153 |
| k-means-soft-dba | 0.304 | 0.216 | 0.166 | **0.493** | **0.212** | 0.187 | 0.185 |
| k-sc | 0.217 | 0.214 | 0.188 | 0.250 | 0.046 | 0.110 | **0.369** |
| k-shapes | 0.126 | 0.191 | 0.110 | 0.104 | 0.073 | 0.078 | 0.041 |
| pam-adtw | 0.291 | 0.213 | 0.175 | 0.372 | 0.135 | 0.176 | 0.257 |
| pam-dtw | 0.244 | 0.201 | 0.158 | 0.417 | 0.138 | 0.163 | 0.112 |
| pam-erp | 0.222 | 0.236 | 0.181 | 0.244 | 0.067 | 0.096 | 0.173 |
| pam-msm | 0.315 | 0.231 | 0.200 | 0.318 | 0.164 | 0.151 | 0.277 |
| pam-shape-dtw | 0.258 | 0.208 | 0.205 | 0.360 | 0.142 | 0.166 | 0.352 |
| pam-soft-dtw | 0.295 | 0.151 | 0.190 | 0.462 | 0.166 | 0.180 | 0.235 |
| pam-twe | 0.311 | 0.228 | 0.212 | 0.364 | 0.187 | 0.151 | 0.271 |
| pam-wdtw | 0.251 | **0.254** | 0.176 | 0.442 | 0.102 | 0.153 | 0.143 |

Table 7.10 Average ARI score on problems split by problem domain over 98 datasets from the UCR archive using the test-train split.

### 7.5.3 Elastic Barycentre clusterer conclusion

Overall we have presented results for 8 different Elastic Barycentre Average clusterers and compared the results to the baseline clusterers as well as PAM clusterers. We showed that Elastic Barycentre clusterers offer some of the best clustering performance outperforming the current state-of-the-art consistently over multiple evaluation metrics.

We note for the combined test-train split the Elastic Barycentre clusterers, specifically *k*-means-ba-TWE and *k*-means-ba-MSM are not significantly different from *k*-means-soft-DBA which was previously the best performing clusterer by a large margin (no other clusterer appeared in the same clique as it previously). Furthermore, we showed that the for every Elastic Barycentre clusterer they outperformed

PAM with the same elastic distance while being computationally significantly cheaper.

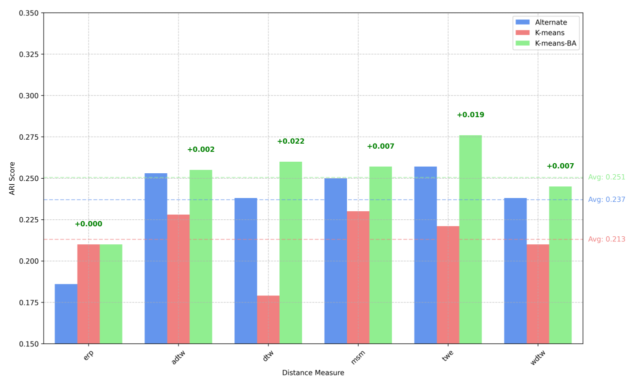
Similarly for the test-train split we observe similar performance though the improvement between PAM and the respective Elastic Barycentre clusterer was much smaller. *k*-means-ba-TWE was consistently one of the best performing clusterers in and is not significantly different from the previous state-of-the-art *k*-means-soft-DBA.

## 7.6 Elastic Barycentre evaluation

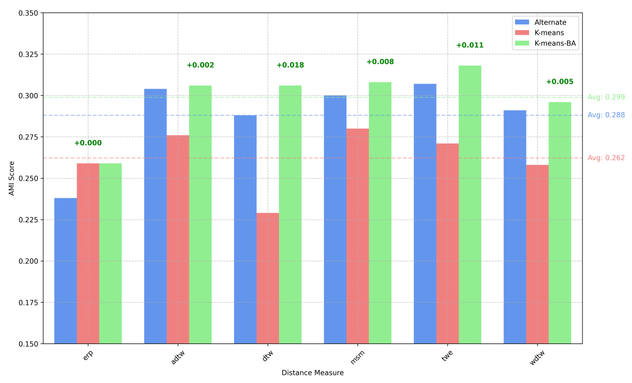
We have now evaluated each Elastic Barycentre clusterer and have shown that it achieves state-of-the-art performance for TSCL. We now seek to understand how much the Elastic Barycentre Average contributes to the performance of each model. To do this, we will compare each Elastic Barycentre clusterer to two other Lloyd's-based techniques that use the same elastic distances. Specifically, for each elastic distance, we compare the results of the following methods: *k*-means with an elastic distance using the arithmetic mean; alternate *k*-medoids with an elastic distance; and *k*-means-ba with an elastic distance using the Elastic Barycentre Average.

Figures 7.35 and 7.36 present a bar graph displaying the average ARI and AMI scores for each clusterer over the combined test-train split and the separate test-train splits, respectively. We have excluded shape-DTW and soft-DTW because, for *k*-means, each of these clusterers failed to converge on over 10 datasets.

For the combined test-train split, we observe that Elastic Barycentre clusterers improve clustering over alternate *k*-medoids for all elastic distances for both ARI and AMI. However, the degree of improvement differs greatly between distances. Table 7.11a shows the percentage increase in ARI and AMI for each distance between alternate *k*-medoids and *k*-means-ba. We observe from the table and visually in Figure 7.35 that the performance improvement varies significantly



(a) ARI



(b) AMI

Fig. 7.35 Comparison of the performance of *k*-means-ba, *k*-means, and alternate *k*-medoids across six elastic distances using 103 datasets from the UCR archive, evaluated on the combined test-train split. The blue bars represent the scores for alternate *k*-medoids, the red bars correspond to *k*-means using the arithmetic mean, and the green bars represent *k*-means-ba using the Elastic Barycentre Average. The values above each set of bars indicate the difference in scores between *k*-means-ba and the second best method. The dashed lines denote the average scores for each clustering algorithm, with colours matching the corresponding bars.

across different distances. For distances such as ERP and DTW, we observe the largest increases in both ARI and AMI. This can likely be explained by the fact that the ERP and DTW alternate *k*-medoids methods start from much lower ARI and AMI scores, and therefore there is greater potential for improvement. However, it is noteworthy that TWE, which was already one of the best-performing alternate *k*-medoids clusterers, improves by 7.39%. For distances that started with similar ARI scores, such as alternate-ADTW and alternate-MSM, we observed much smaller increases in performance.

For both AMI and ARI, we observe a similar trend in performance gain between the clustering models. However, we note that ARI increases by more than AMI. Since ARI measures the number of pairs of samples that are assigned to the same or different clusters correctly, this suggests that the Elastic Barycentre algorithm improves the clustering by more accurately grouping similar pairs and separating dissimilar pairs of samples. In other words, the Elastic Barycentre enhances the local pairwise relationships between data points, leading to a higher agreement in pairwise assignments as captured by ARI.

| Distance | % Increase ARI | % Increase AMI |
|---|---|---|
| ERP | 12.90% | 8.82% |
| ADTW | 0.79% | 0.66% |
| DTW | 9.24% | 6.25% |
| MSM | 2.80% | 2.67% |
| TWE | 7.39% | 3.58% |
| WDTW | 2.94% | 1.72% |
| **Average** | **6.68%** | **3.95%** |

(a) Combined test-train split

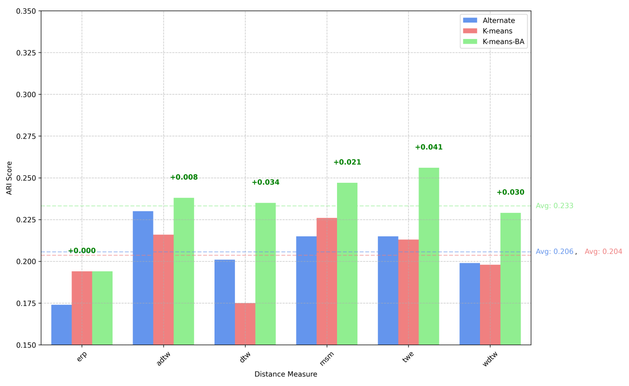
| Distance | % Increase ARI | % Increase AMI |
|---|---|---|
| ERP | 11.49% | 8.37% |
| ADTW | 3.48% | 3.87% |
| DTW | 16.92% | 10.27% |
| MSM | 14.88% | 9.85% |
| TWE | 19.07% | 13.55% |
| WDTW | 15.08% | 10.38% |
| **Average** | **13.49%** | **9.38%** |

(b) Combined test-train split

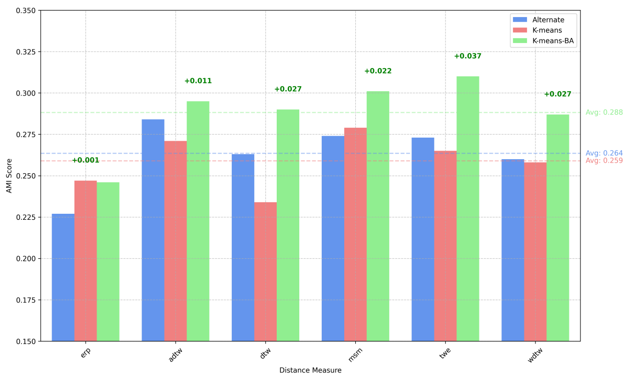
Table 7.11 Percentage increase of ARI and AMI for *k*-means-ba over alternate *k*-medoids over datasets from the UCR archive: 103 combined test-train splits and 107 test-train splits.

On the other hand, AMI measures the agreement between the entire clusterings by considering the mutual information and the distribution of cluster assignments. The smaller increase in AMI indicates that while there is an improvement in the overall cluster structure, it is less pronounced compared to the improvements in pairwise sample agreements. Therefore, the greater increase in ARI suggests that the Elastic Barycentre Average particularly enhances clustering performance in terms of correctly assigning pairs of samples to the same or different clusters, which is more sensitively reflected by ARI than by AMI.

For the test-train split, we find that the average improvement is significantly greater than for the combined test-train split. Table 7.11 shows that the average ARI improvement for the combined test-train split was 6.68%, while for AMI, it was 3.95%. However, in the test-train split, we observe a much larger average improvement of 13.49% in ARI and 9.38% in AMI. Notably, the difference between alternate-TWE and *k*-means-ba-TWE for the test-train split is 0.041 in ARI and 0.037 in AMI, highlighting a substantial improvement. This indicates that centroids computed using the Elastic Barycentre Average are much better general represen-



(a) ARI



(b) AMI

Fig. 7.36 Comparison of the performance of *k*-means-ba, *k*-means, and alternate *k*-medoids across six elastic distances using 107 datasets from the UCR archive, evaluated on the test-train split. The blue bars represent the scores for alternate *k*-medoids, the red bars correspond to *k*-means using the arithmetic mean, and the green bars represent *k*-means-ba using the Elastic Barycentre Average. The values above each set of bars indicate the difference in scores between *k*-means-ba and the second best method. The dashed lines denote the average scores for each clustering algorithm, with colours matching the corresponding bars.

tations of the dataset compared to those produced by medoids or the arithmetic mean.

Overall, we have shown that the Elastic Barycentre Average yields significantly better results than using medoids or the arithmetic mean for both the combined test-train and test-train splits. The improvement for each distance is not linear, with different amounts of improvement observed for each elastic distance, ranging from a 0.79% increase to a 19.07% increase. However, we have demonstrated that the improvement is directly attributable to the use of the Elastic Barycentre Average.

## 7.7 Conclusion

In this chapter, we proposed a new averaging technique for time series data called the Elastic Barycentre Average. This technique is inspired by the widely used Dynamic Time Warping Barycentre Average, which exclusively works with the

DTW elastic distance. However, the Elastic Barycentre Average can be applied to any elastic distance that produces a complete warping path.

We have provided pseudocode, default parameters, and clear examples demonstrating how to compute the Elastic Barycentre Average. Visual examples of nine new Elastic Barycentres were presented, highlighting the significant differences between them, even though they were all derived from the same data.

Next, we conducted an experiment using the *k*-means clustering algorithm to evaluate the performance of the Elastic Barycentre Average. We began by comparing the Elastic Barycentre clusterers using different elastic distances against one another to identify the best-performing variant. Then, we contextualised the results by incorporating the baseline clusterers, PAM using the same elastic distances, and *k*-means-soft-DBA. We demonstrated that multiple Elastic Barycentre clusterers achieve state-of-the-art performance and outperform *k*-medoids models with the same elastic distance. Furthermore, we showed that across the combined test-train split and test-train split for all evaluation metrics, the best Elastic Barycentre clusterers: *k*-means-ba-TWE and *k*-means-ba-MSM, performance was not significantly different from *k*-means-soft-DBA and in some instances surpasses it.

Finally, to isolate and demonstrate the contribution of the Elastic Barycentre Average to improving Lloyd's-based algorithms, we conducted an experiment comparing alternate *k*-medoids and *k*-means to *k*-means-ba. These algorithms are structurally identical, with the only difference being how centroids are computed: alternate *k*-medoids computes medoids, *k*-means uses the arithmetic mean, and the Elastic Barycentre clusterer employs the Elastic Barycentre Average. By comparing their performance, we were able to measure the impact of the Elastic Barycentre Average. We observed that for all elastic distances considered, the Elastic Barycentre Average always improves clustering performance over *k*-means and alternate *k*-medoids with the same elastic distance. In some cases, the Elastic Barycentre Average resulted in an improvement of up to 19.07% in ARI. Overall, across the considered elastic distances, we observed an average improvement of 6.68% in ARI and 3.95% in AMI for the combined test-train split compared to alternate *k*-medoids. For the test-train split, we observed an even greater improvement, with an average increase of 13.49% in ARI and 9.38% in AMI compared to alternate *k*-medoids.

In summary, we have introduced a new time series averaging method that, when used with *k*-means, achieves and even surpasses the performance of current state-of-the-art clustering methods. We have empirically shown that the primary factor contributing to the success of the Elastic Barycentre clusterer is the Elastic Barycentre Average.

# Chapter 8

# KESBA: A Fast and Scalable End-to-End Elastic Distance Clustering Algorithm

In Chapter 7, we proposed the Elastic Barycentre Average, a method for computing improved averages of time series data by utilising elastic distances that produce a complete alignment path.

When combined with the *k*-means algorithm, the Elastic Barycentre Average significantly improves clustering performance of all the elastic distances considered, outperforming methods that apply the same elastic distance with *k*-means, as well as alternative approaches like alternate *k*-medoids or PAM. Notably, TWE and MSM, when paired with the Elastic Barycentre Average, achieved state-of-the-art performance, even surpassing *k*-means-soft-DBA in some evaluation metrics.

However, while clustering performance is the primary focus of an experimental evaluation such as this thesis, real-world practitioners must also consider the runtime of TSCL algorithms, which can be as important as the clustering performance. In response to this, we propose a substantially faster version of the Elastic Barycentre

Average *k*-means clusterer: the *k*-means (K) end-to-end elastic (E) stochastic subgradient (S) Barycentre (B) Average (A) (KESBA).

KESBA leverages an accelerated version of the Elastic Barycentre Average alongside various *k*-means optimisations, achieving state-of-the-art performance with a runtime that is significantly faster than the original method and other state-of-the-art clusterers. KESBA offers practitioners a versatile, highly scalable clusterer specifically designed for real-world TSCL applications.

## 8.1 Introduction

TSCL techniques are notoriously computationally expensive. Throughout this thesis, we have had to exclude several datasets from various experiments because certain clusterers were unable to complete within our seven-day runtime limit. While the UCR archive contains some large datasets, many real-world applications involve datasets that are similar in size or significantly larger, both in terms of the number of instances and the length of the time series. This poses a significant limitation to the practical applicability of many TSCL approaches discussed in this work.

Within the TSCL literature, there are numerous examples of clusterers that prioritise lower computational runtime over achieving the best performance. These methods offer practitioners fast alternatives that trade some clustering performance for the ability to handle large datasets efficiently. Examples include *k*-shapes [89], TADPole [9], SOMTimeS [52], JET [127], and many others. Among these, *k*-shapes has emerged as one of the most widely used TSCL approaches.

In Chapter 4, we observed that *k*-shapes is orders of magnitude faster than *k*-means-ba-DTW, while still delivering competitive performance relative to the state-of-the-art. However, after conducting our new experiments throughout this thesis using both *k*-means and *k*-medoids clusterers, we found that *k*-shapes no longer holds it place as a state-of-the-art clusterer.

Compared to the top-performing clusterer identified in Chapter 7, *k*-shapes was, on average, 0.048 ARI and 0.033 AMI worse than *k*-means-ba-TWE for the combined test-train split. For the test-train split, this performance gap was even larger, with *k*-shapes averaging 0.136 ARI and 0.119 AMI worse than *k*-means-ba-TWE. Furthermore, when compared to all of the Elastic Barycentre Average *k*-means clusterers and PAM clusterers, *k*-shapes never ranked among the top four cliques.

However, achieving superior clustering performance with the best-performing PAM and Elastic Barycentre Average *k*-means clusterers comes at a significant computational cost. We have demonstrated that PAM and the Elastic Barycentre Average are orders of magnitude slower than *k*-shapes and *k*-means-Euclidean.

A substantial portion of the computational time for Elastic Barycentre Average *k*-means clusterers is due to the Elastic Barycentre Average computation. This is illustrated in Figure 8.1, which presents the FitTime CD diagram comparing various Lloyd's-based algorithms that differ only in their centroid computation methods (e.g., medoids, arithmetic mean for *k*-means, and the Elastic Barycentre Average for *k*-means-ba).

The computational complexity of the Elastic Barycentre Average significantly limits its practicality for real-world clustering applications. Therefore, in this chapter, we aim to develop a faster version of the Elastic Barycentre Average and build a clusterer around it, enabling it to achieve state-of-the-art accuracy while being significantly less computationally expensive. Moreover, this new approach will be customisable to meet practitioners' specific needs for runtime efficiency and clustering performance.

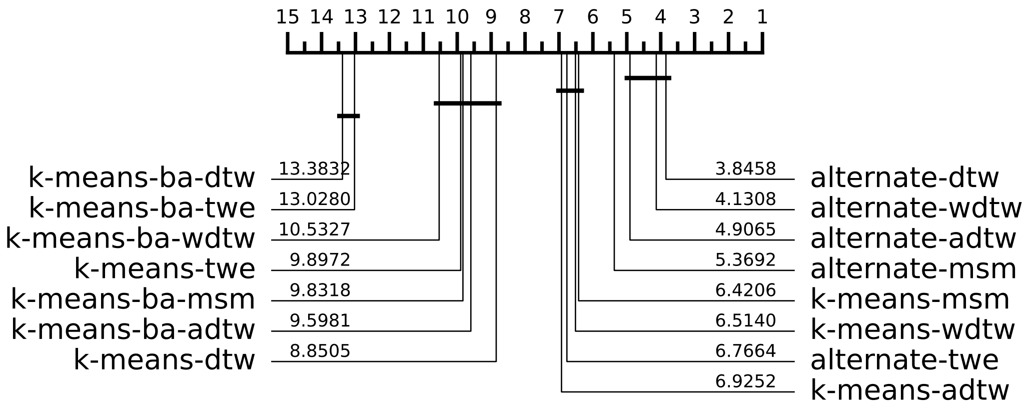


Fig. 8.1 CD diagram for the FitTime of Lloyd's-based clusterers with different centroid computation algorithms over the UCR archive using the test-train split. Missing datasets are outlined in Table A.36.

## 8.2 Stochastic Subgradient Elastic Barycentre Average

The Stochastic Subgradient Dynamic Barycentre Average (SSG-DBA) [108] shares similarities with DBA, as both methods attempt to compute a subgradient. Subgradient methods are a form of nonsmooth optimisation [7] that operate similarly to gradient descent but replace the gradient with a subgradient. As discussed in Section 2.4.7, DTW (and, similarly, other elastic distances) is not differentiable everywhere, which prevents the use of gradient descent. Consequently, subgradient methods provide a generalisation of the gradient under mild conditions that hold for the Fréchet function [108].

The primary difference between SSG-DBA and DBA lies in their optimisation approach: DBA is a batch optimisation method, whereas SSG-DBA is a stochastic optimisation method. DBA computes an exact subgradient based on all the time series in the collection, while SSG-DBA estimates the subgradient using a single randomly selected time series from the collection. As a result, SSG-DBA updates the current barycentre at every iteration, whereas DBA only updates the barycentre average after completing a full pass through the collection of time series. Consequently, after processing the entire collection, SSG-DBA performs *n* updates, whereas DBA performs a single update [108].

Since SSG-DBA updates the average *n* times within a single iteration, it converges to a solution that satisfies the necessary conditions more rapidly than DBA. Like DBA, SSG-DBA is a heuristic method that approximates the optimal solution, meaning its estimation may differ from that of DBA. However, SSG-DBA satisfies the local minimisation criterion of the Fréchet function, which ensures its convergence [108].

To begin, we propose an adaptation to the SSG-DBA algorithm in the same way we previously adapted DBA in Chapter 7: replace the DTW alignment path with any other elastic distance that computes a complete warping path. Algorithm 37 and Algorithm 38 present the Elastic SSG Barycentre Average.

Algorithm 37 closely resembles the Elastic Barycentre Average shown in Algorithm 35. Many lines of code are identical to those explained in Section 7.2, so we will focus only on the new lines introduced for the Elastic SSG Barycentre Average.

Algorithm 37 requires two additional parameters that were not needed for the Elastic Barycentre Average: $initial\_step\_size$ and $end\_step\_size$. These parameters control the gradient descent rate. The $initial\_step\_size$ is the starting step size for the gradient. On line 3, a new variable, $current\_step\_size$, is defined and set to the value of $initial\_step\_size$. This variable tracks the current gradient descent size. The refinement iteration begins on line 4. For clarity, we refer to the iterations in Algorithm 37 as "refinement iterations" and the iterations in Algorithm 38 as the "update iterations."

Within the refinement iteration, a new conditional step is introduced on lines 5 to 8, controlling the reduction of the gradient size within the update iteration in Algorithm 38. If it is the first refinement iteration, the variable $step\_size\_reduction$ is set to linearly decrease the gradient size from $initial\_step\_size$ to $end\_step\_size$

over the course of the $n$ update iterations. For subsequent refinement iterations, *step_size_reduction* is set to 0 since the step size has already reduced to *end_step_size*.

Next, the *elastic_ssg_barycentre_update* function is called to perform the Elastic SSG Barycentre update iterations on line 9. It takes four parameters: *barycentre* (the current barycentre), $X$ (the current dataset of time series), *current_step_size* (the starting gradient descent size), and *step_size_reduction* (which controls how much the *current_step_size* is reduced during each iteration). Once the *elastic_ssg_barycentre_update* completes, two variables are returned: the updated *barycentre* and the *current_step_size*. After the first set of update iterations, *current_step_size* will be equal to *end_step_size* and will remain at this value for the rest of the algorithm's execution.

Beyond this, the algorithm functions the same way as the Elastic Barycentre Average. The refinement iterations will continue until *max_iters* is reached, and once completed, the Elastic SSG Barycentre Average is returned. We will now outline the *elastic_ssg_barycentre_update* algorithm in Algorithm 38.

Algorithm 38 presents the procedure to iteratively update the Elastic SSG Barycentre Average for one pass through the dataset $X$. The process begins by setting *new_barycentre* to the initial barycentre, *barycentre* (line 1), and starting the iteration from 1 to $n$ (line 2). A random time series, not previously selected, is chosen from the dataset $X$ (line 3). Using the randomly selected time series, the cost matrix and alignment path are computed using a elastic distance that generates a complete warping path (lines 4 and 5). A temporary barycentre, *temp_barycentre*, is then created to store the intermediate results (line 6).

The generated alignment path is then used to update the barycentre. The *jth* time point in the *random_time_series* is subtracted from the *kth* time point in the *new_barycentre*, and the result is assigned to the *kth* time point in *temp_barycentre* (line 8).

**Algorithm 37:** elastic_ssg_barycentre(**X**, **max_iters**, **tol**, **initial_step_size**, **end_step_size**)

```
Input: X (Dataset of time series of length n. Each time series is of length
       m), max_iters (Maximum number of iterations before forced
       termination), tol (Change in barycentre threshold),
       initial_step_size (Initial step size), end_step_size (End step size)
Output: Elastic SSG Barycentre Average of X for a given elastic distance
1  $barycentre \leftarrow mean(X)$
2  $previous\_dist \leftarrow \infty$
3  $current\_step\_size \leftarrow initial\_step\_size$
4  for $i \leftarrow 1$ to $max\_iters$ do
5      if $i == 1$ then
6          $step\_size\_reduction \leftarrow (initial\_step\_size - end\_step\_size)/n$
7      else
8          $step\_size\_reduction \leftarrow 0$
9      $barycentre, current\_step\_size \leftarrow$
         $elastic\_ssg\_barycentre\_update(barycentre, X, current\_step\_size, step\_size\_reduction)$
10     $curr\_distance \leftarrow 0$
11     for each time series $curr\_ts$ in $X$ do
12         $curr\_distance \leftarrow$
             $curr\_distance + elastic\_distance(barycentre, curr\_ts)$
13     if $|previous\_dist - curr\_distance| < tol$ then
14         break
15     $previous\_dist \leftarrow curr\_distance$
16 return $barycentre$
```

Once all alignments have been processed, a *new_barycentre* is created by multiplying *temp_barycentre* by $2.0 \times current\_step\_size$ (line 9). As the algorithm progresses, *current_step_size* gradually decreases, causing the *new_barycentre* to change by progressively smaller amounts with each iteration. Finally, *current_step_size* is reduced by the value of *step_size_reduction* (line 10). The update iteration repeats until all time series have been processed (in a random order). Once complete, the algorithm returns the updated *new_barycentre* and *current_step_size* (line 11).

```
Algorithm 38: elastic_ssg_barycentre_update(barycentre, X, current_step_size, step_size_reduction)
Input: barycentre (Current estimate of barycentre), X (Dataset of time
       series of length n. Each time series is of length m),
       current_step_size (Step-size for the subgradient descent),
       step_size_reduction (Amount current_step_size should reduce by
       each iteration)
Output: Updated Elastic SSG Barycentre and the current_step_size after
        reductions.
1  $new\_barycentre \leftarrow barycentre$
2  for $i \leftarrow 1$ to $n$ do
3      $random\_time\_series \leftarrow$ choose a random time series from X (not
         already selected)
4      $CM \leftarrow elastic\_distance\_CM(barycentre, random\_time\_series)$
5      $alignment\_path \leftarrow optimal\_warping\_path(CM)$
6      Initialise temp_barycentre as a zeros array of size $m$.
7      for each pair of indices $(j,k)$ in alignment_path do
8          $temp\_barycentre[k] \leftarrow temp\_barycentre[k] +$
             $(new\_barycentre[k] - random\_time\_series[j])$
9      $new\_barycentre \leftarrow (2.0 \times current\_step\_size) \times temp\_barycentre$
10     $current\_step\_size \leftarrow current\_step\_size - step\_size\_reduction$
11 return $new\_barycentre, current\_step\_size$
```

## 8.3 Elastic SSG Barycentre analysis

Figure 8.2 presents the Elastic SSG Barycentre Averages for 10 different elastic distances applied to class 1 of the GunPoint dataset. For reference, the arithmetic mean is also included (in red).

Similar to the Elastic Barycentre Average, it is noteworthy how distinct each Elastic SSG Barycentre is from one another depending on the elastic distance used. However, when compared to the Elastic Barycentre Averages for the same data shown in Figure 7.1, the barycentres produced by the Elastic SSG method are very similar. The global structures are nearly identical for most elastic distances, with the exception of SSG-BA-DDTW (turquoise line), which deviates significantly from the Elastic Barycentre DDTW Average.

The key difference between the Elastic SSG Barycentre Average and the Elastic Barycentre Average lies in the local structures. For instance, in the case of SSG-BA-TWE (yellow line), the peak is much more pronounced compared to BA-TWE in Figure 7.1. Additionally, more local fluctuations are evident in all the Elastic SSG Barycentre Averages. This is likely due to the Elastic SSG Barycentre Average updating the barycentre at each iteration, whereas the Elastic Barycentre Average updates the barycentre only once after all update iterations have been completed.

Additionally, we recorded the number of iterations required for each Elastic Barycentre Average and Elastic SSG Barycentre Average to converge when producing an average for GunPoint class 1. The convergence criterion was either reaching the maximum number of iterations or satisfying the early stopping condition, where the barycentre change between iterations was less than $tol$. Table 8.1 presents the number of iterations taken by each averaging technique to converge.

We observed that the Elastic SSG Barycentre Average required fewer iterations to converge across all distances compared to the Elastic Barycentre Average. For some distances, such as DTW, WDTW, DDTW, and WDDTW, the difference was particularly significant. This indicates that similar Elastic Barycentre Averages can be produced with much less computational effort.

| **Distance** | **SSG-BA Iterations** | **BA Iterations** |
|---|---|---|
| DTW | 8 | 184 |
| ADTW | 5 | 2 |
| WDTW | 8 | 21 |
| DDTW | 10 | 500 |
| WDDTW | 5 | 500 |
| Shape DTW | 3 | 11 |
| Soft DTW | 2 | 3 |
| ERP | 2 | 3 |
| MSM | 2 | 3 |
| TWE | 2 | 3 |
| **Average** | **4.7** | **123** |

Table 8.1 Number of Iterations for SSG-BA and BA to produce the barycentres in Figure 8.2 and 7.1 respectively. The maximum number of iterations was set to 500.

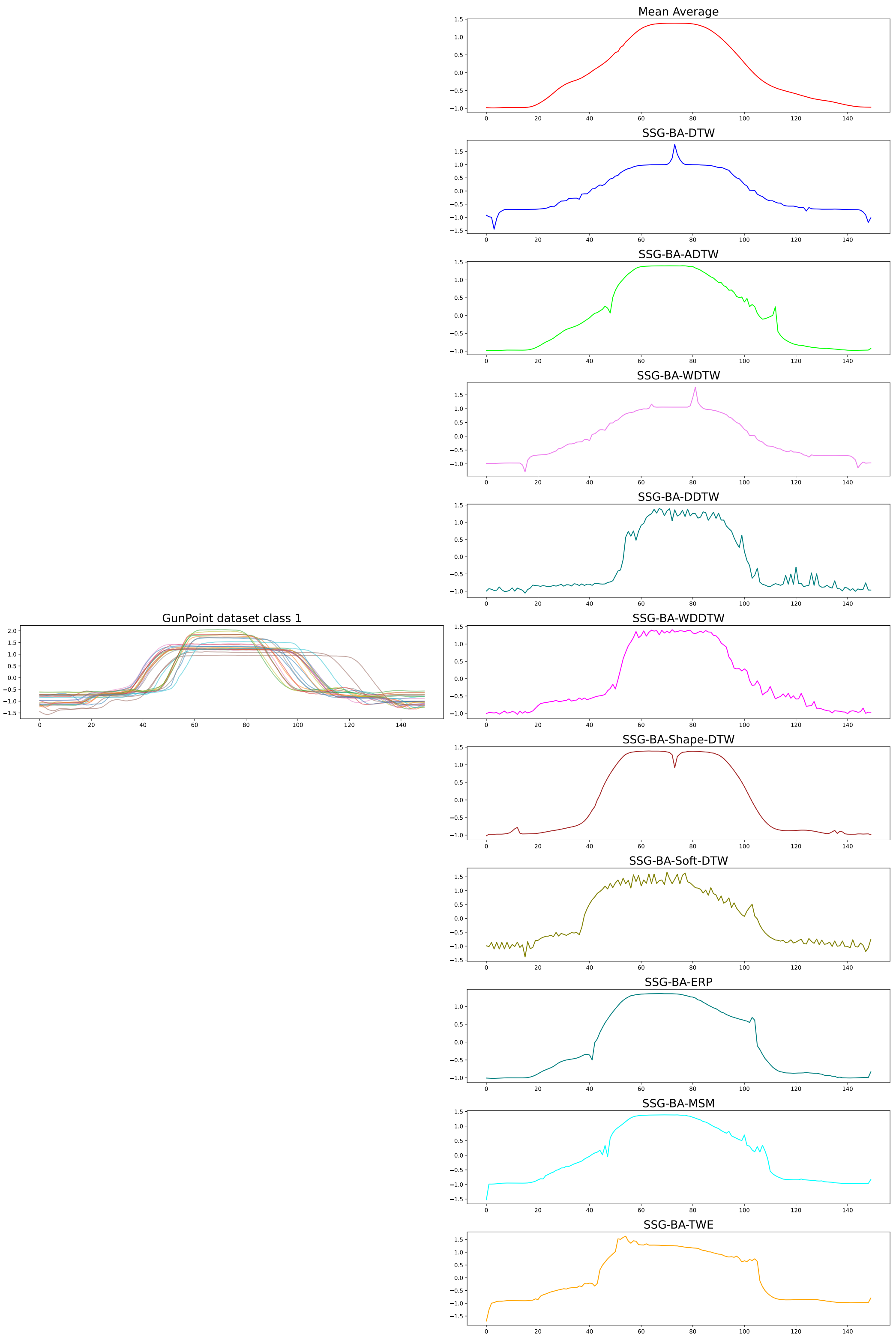


Fig. 8.2 Different Elastic SSG Barycentre Averages for GunPoint class 1.

## 8.4 The KESBA clustering algorithm

We now propose the *k*-means (K) end-to-end elastic (E) stochastic subgradient (S) Barycentre (B) Average (A) (KESBA) clustering algorithm. The goal of KESBA is to deliver competitive clustering performance comparable to the newly identified state-of-the-art TSCL algorithms, while significantly reducing computational runtime.

KESBA is built around the Elastic SSG Barycentre Average proposed previously. Previous work has shown that SSG-DBA converges much faster than the traditional DBA while producing similar results [108]. In Section 8.3, we demonstrated that the adapted Elastic SSG Barycentre Average shares these characteristics, producing a similar average to the Elastic Barycentre Average but with significantly less iterations required.

However, integrating the Elastic SSG Barycentre Average into Lloyd's algorithm requires specific updates to be made. While the speed-up provided by the Elastic SSG Barycentre Average is substantial, we introduce additional optimisations to enhance the algorithm's speed and scalability further.

Algorithm 39 outlines the KESBA clustering algorithm. We will now provide a detailed explanation of all the enhancements we have made to Lloyd's algorithm to make the KESBA clustering algorithm as fast and scalable as possible.

**Algorithm 39:** KESBA (**X**, **k**, **max_iters**, **ba_subset_size**, **window**, **initial_step_size**, **end_step_size**)

```
Input: X (Dataset of time series of length n), k (Number of clusters),
       max_iters (Maximum number of iterations before forced
       termination), ba_subset_size (Percentage of time series to use each
       iteration of Random Subset Elastic SSG Barycentre Average),
       window (Sakoe-chiba bounding window for distance computation.),
       initial_step_size (Initial SSG step size.), end_step_size (End SSG
       step size.)
Output: Assignment of each time series to a cluster
1  $centres \leftarrow elastic\_kmeans\_plus\_plus(X, k, window)$
2  Let $assignments$ be an empty array of length $n$
3  Let $prev\_assignments$ be an empty array of length $n$
4  for $j \leftarrow 1$ to $max\_iters$ do
5      for each time series $x_i$ in X do
6          Compute the distance between $x_i$ and each of the $k$ centres using the
             $window$ as a parameter
7          Assign $x_i$ to the nearest centre
8      if $prev\_assignment == assignment$ then
9          break
10     if any cluster has no assignments then
11         repeat
12             for each cluster $c_j$ in $centres$ do
13                 if cluster $c_j$ has no assignments then
14                     Set $best\_candidate$ to the time series that reduces
                         inertia the most and is not currently a centroid
15                     $c_j \leftarrow best\_candidate$
16                     Recompute cluster assignments
17         until every cluster has at least one assignment;
18     for each centre $c_j$ in $centres$ do
19         Update $c_j$ to be the Random Subset Elastic SSG Barycentre
             Average of the time series assigned to cluster $c_j$
20     $prev\_assignments \leftarrow assignments$
21 return $assignments$
```

### 8.4.1 Random Subset Elastic SSG Barycentre Average

The first and most significant modification we apply to Lloyd's algorithm is the development of a faster Elastic SSG Barycentre Average, which we call the Random Subset Elastic SSG Barycentre Average, specifically designed for KESBA. This method incorporates optimisations inspired by CLARA and CLARANS to reduce the algorithm's runtime.

At a high level, the Random Subset Elastic SSG Barycentre Average computes the Elastic SSG Barycentre Average on a random subset of the data selected during each refinement iteration. This strategy produces a similar average but significantly reduces the computational cost compared to the full Elastic Barycentre Average. The Random Subset Elastic SSG Barycentre Average is applied in line 19 of the KESBA algorithm, as shown in Algorithm 39.

Algorithm 40 introduces the Random Subset Elastic SSG Barycentre Average, which adds a new parameter compared to the standard Elastic SSG Barycentre Average: *ba_subset_size*. This parameter, set between 0 and 1, controls the percentage of the dataset used in each refinement iteration. A value of 1 will use all of the data whereas a value of 0.5 will use 50% of the data. During each iteration, a random subset of the dataset is selected to update the current barycentre, with *ba_subset_size* determining the number of time series included in the subset.

The *ba_subset_size* parameter gives practitioners the ability to balance runtime with the quality of the computed barycentre. Our hypothesis is that a higher value will improve the quality of the average but increase runtime, whereas a lower value will reduce runtime at the potential cost of average quality. However, as we will demonstrate in Section 8.7, we find that a lower *ba_subset_size* not only significantly reduces runtime but also improves KESBA's overall performance.

In Algorithm 40, the *ba_subset_size* parameter is applied on line 4 to calculate the number of time series used in each refinement iteration, stored as *num_ts_to_use*.

This value is rounded to determine the exact number of instances selected for each iteration. For example, if *ba_subset_size* is set to 0.2 for a dataset with 100 instances, 20 instances will be randomly chosen for each refinement iteration.

To ensure sufficient data is used for the barycentre computation, we set a minimum of 10 instances (or $n$ if the cluster has fewer than 10 instances) and a maximum of $n$ to be used. This is necessary because Lloyd's algorithm can produce both very large and very small clusters. If a small cluster of only 10 instances is passed to the Random Subset Elastic SSG Barycentre function with *ba_subset_size* set to 0.2, only two time series would be selected, leading to a poor-quality barycentre. Setting a minimum ensures enough data is used to produce a more reliable barycentre.

Next, on line 7, the $step\_size\_reduction$ variable is adjusted based on the *num_ts_to_use* variable, as this dictates the number of iterations over which the step size must reduce to $end\_step\_size$. Then, on line 10, *num_ts_to_use* time series are randomly selected from the dataset $X$, ensuring both selection and order are randomised. These time series are passed to the $random\_subset\_elastic\_$ $ssg\_barycentre\_update$ function on line 11.

The remainder of the algorithm follows the Elastic SSG Barycentre Average. On line 14, although only a random subset of the dataset is used to update the barycentre, all time series are included when computing the total distance to the barycentre. This ensures a consistent measure of total distance across iterations, allowing for reliable convergence checks.

Algorithm 41 presents the update function for the Random Subset Elastic SSG Barycentre. The primary difference from the original Elastic SSG Barycentre update function is the inclusion of a random subset of $X$, denoted as *random_subset_X*, which is passed as a parameter. Because this subset is already randomised, the logic for shuffling the sequence has been removed, simplifying the function and improving efficiency compared to the original.

**Algorithm 40:** random_subset_elastic_ssg_barycentre(**X**, **max_iters**, **tol**, **initial_step_size**, **end_step_size**, **ba_subset_size**)

```
Input: X (Dataset of n time series. Each time series is of length m),
       max_iters (Maximum number of iterations before forced
       termination), tol (Change in barycentre threshold),
       initial_step_size (Initial step size), end_step_size (End step size),
       ba_subset_size (Percentage of time series to use for an update
       iteration)
Output: Random Subset Elastic SSG Barycentre Average of X for a given
        elastic distance
1  $barycentre \leftarrow mean(X)$
2  $previous\_dist \leftarrow \infty$
3  $current\_step\_size \leftarrow initial\_step\_size$
4  $num\_ts\_to\_use \leftarrow \min(n, \max(10, (ba\_subset\_size \times n)))$
5  for $i \leftarrow 1$ to $max\_iters$ do
6      if $i == 1$ then
7          $step\_size\_reduction \leftarrow (initial\_step\_size - end\_step\_size)/num\_ts\_to\_use$
8      else
9          $step\_size\_reduction \leftarrow 0$
10     Let random_subset_X be an array of $num\_ts\_to\_use$ randomly
         selected time series from X.
11     $barycentre, current\_step\_size \leftarrow$
         $random\_subset\_elastic\_ssg\_barycentre\_update(barycentre, random\_subset\_X,$
         $current\_step\_size, step\_size\_reduction)$
12     $curr\_distance \leftarrow 0$
13     for each time series $curr\_ts$ in X do
14         $curr\_distance \leftarrow curr\_distance + elastic\_distance(barycentre, curr\_ts)$
15     if $|previous\_dist - curr\_distance| < tol$ then
16         break
17     $previous\_dist \leftarrow curr\_distance$
18 return $barycentre$
```

In Section 8.7, we conduct an extensive evaluation of the *ba_subset_size* parameter and its impact on clustering performance and runtime for KESBA. While we recommend a default value between 0.4 and 0.5, we encourage practitioners to adjust this parameter based on their specific computational runtime needs. Our

**Algorithm 41:** random_subset_elastic_ssg_barycentre_update(**barycentre**, **random_subset_X**, **current_step_size**, **step_size_reduction**)

```
Input: barycentre (Current estimate of barycentre), random_subset_X
       (Dataset of randomly selected time series of size n. Each time series
       is of length m), current_step_size (Step size for the subgradient
       descent), step_size_reduction (Amount by which
       current_step_size should reduce each iteration)
Output: Updated Random Subset Elastic SSG Barycentre and the
        current_step_size after reductions
1  $new\_barycentre \leftarrow barycentre$
2  for each time series random_time_series in random_subset_X do
3      $CM \leftarrow elastic\_distance\_CM(new\_barycentre, random\_time\_series)$
4      $alignment\_path \leftarrow optimal\_warping\_path(CM)$
5      Initialise temp_barycentre as a zero array of size $m$.
6      for each pair of indices $(j,k)$ in alignment_path do
7          $temp\_barycentre[k] \leftarrow temp\_barycentre[k] + (new\_barycentre[k] - random\_time\_series[j])$
8      $new\_barycentre \leftarrow new\_barycentre - (2.0 \times current\_step\_size) \times temp\_barycentre$
9      $current\_step\_size \leftarrow current\_step\_size - step\_size\_reduction$
10 return $new\_barycentre, current\_step\_size$
```

evaluation in Section 8.7 is intended to provide practitioners with the necessary insights to make informed decisions tailored to their requirements.

### 8.4.2 Elastic *k*-means++ initialisation

One of the most significant computational improvements we introduce for KESBA is the adaptation of *k*-means++ to work with elastic distances, specifically TWE and MSM. While we have found instances of *k*-means++ being used with DTW in the open-source TSCL community [118], we were unable to locate any literature that evaluates its performance with elastic distances. Therefore, we believe we are the first to integrate and systematically evaluate *k*-means++ with any elastic distances.

*k*-means++ [6] is widely regarded as one of the most popular and recommended methods for cluster initialisation in traditional clustering literature [17]. However, as shown in Chapter 4, *k*-means++ performs worse than Forgy, Random with 10 restarts, and Forgy with 10 restarts for time series data. We hypothesise that this is because $k$-means++ relies heavily on the Euclidean distance to measure similarity between time series. As demonstrated through numerous experiments in this thesis, the Euclidean distance is not a suitable distance measure for TSCL. In contrast, elastic distances have proven to be significantly more effective. Therefore, we hypothesise that incorporating elastic distances into $k$-means++ could substantially improve the initial centroids, providing a viable alternative to Forgy with 10 restarts.

In the literature, the only reference we found to $k$-means++ with an elastic distance (specifically DTW) was in the tslearn open-source repository [118]. Additionally, we were unable to find any evaluations of the impact of using $k$-means++ with elastic distances. While we do not conduct a review of initialisation methods, we do compare $k$-means++ with elastic distances against Forgy with restarts.

Our results show that $k$-means++ with an elastic distance achieves clustering performance comparable to Forgy with 10 restarts, while significantly reducing runtime—by a factor of 10—since no restarts are required. This is because $k$-means++ avoids local optima more effectively than random initialisation strategies [17], eliminating the need for multiple restarts. Although computing the initial centroids using $k$-means++ is more expensive than using Forgy, the centroids selected by $k$-means++ are theoretically “better,” meaning that Lloyd’s algorithm requires fewer iterations to converge [17]. Thus, the initial cost of $k$-means++ is offset by the reduced number of distance computations needed for convergence. Overall, elastic $k$-means++ offers a significant reduction in computational time while delivering clustering performance similar to Forgy with 10 restarts.

Algorithm 42 presents the Elastic $k$-means++ algorithm. This algorithm is identical to that proposed by [6], but it uses an elastic distance in place of the squared Euclidean distance. The Elastic $k$-means++ algorithm is used on line 1 of the KESBA algorithm shown in Algorithm 39.

**Algorithm 42:** elastic_kmeans_plus_plus(**X**, **n_clusters**)

```
Input: X (Dataset of n time series, each of length m),
n_clusters (Number of clusters)
Output: centers (Array of initial cluster centers)
1  Select a random initial center index $initial\_center\_idx$ from $\{1, 2, \ldots, n\}$
2  Initialise $indexes$ as a list containing $initial\_center\_idx$
3  Compute $min\_distances$ as the elastic distances between each time series in
     $X$ and $X[initial\_center\_idx]$
4  for $k = 2$ to $n\_clusters$ do
5      $total\_distance \leftarrow 0$
6      for $i = 1$ to $n$ do
7          $total\_distance = total\_distance + min\_distance[i]$
8      Let probabilities be an array of zeros of length $n$
9      for $i = 1$ to $n$ do
10         $probabilities[i] \leftarrow min\_distances[i] / total\_distance$
11     Randomly select the next centroid index $next\_center\_idx$ from
         $\{1, 2, \ldots, n\}$ with the weighted probability distribution probabilities
12     Append $next\_center\_idx$ to $indexes$
13     Compute distances_new_center as the distances between each time
         series in $X$ and $X[next\_center\_idx]$ using the elastic distance
14     $min\_distances[i] \leftarrow \min(min\_distances[i], distances\_new\_center[i])$
15 $centers \leftarrow X[indexes]$
16 return $centers$
```

Overall, Elastic *k*-means++ will be shown to achieve clustering performance comparable to Forgy with 10 restarts, but without the need for restarts, resulting in significantly reduced computational runtime. In Section 8.7, we show that while Elastic *k*-means++ is slightly outranked by Forgy with 10 restarts, the difference is not statistically significant. Moreover, the runtime savings are substantial—over eight times faster than Forgy with 10 restarts—making it a worthwhile trade-off between a minimal reduction in clustering performance and significantly faster runtimes.

### 8.4.3 Lloyd's-stopping condition

In Section 4.6, we defined a convergence criteria for Lloyd's algorithm as measuring the change in inertia between iterations. If the change in inertia falls below the threshold $tol$, the algorithm is considered to have converged.

However, when using the Random Subset Elastic SSG Barycentre Average within the $k$-means algorithm, measuring inertia change between iterations is no longer applicable. This is because every time the Random Subset Elastic SSG Barycentre Average computes the average of a collection (or cluster) of time series, a different barycentre average is produced, even if the same time series are used. This variability is due to the random order and random subset of data that the Random Subset Elastic SSG Barycentre Average iteratively refines. As a result, the inertia of the same set of time series will almost always differ between iterations, causing the inertia to change even when the assignments remain unchanged. This means measuring change in inertia between iterations is no longer an accurate way to measure convergence.

One potential solution is to set the $tol$ parameter higher. However, it is difficult to determine a value that is high enough to account for the random inertia changes caused by the Random Subset Elastic SSG Barycentre Average, yet low enough to prevent premature convergence. Therefore, instead of relying on inertia which is relative to the centroids, we measure the change in assignments to each cluster between iterations. If the assignments do not change, the algorithm is considered to have converged. This is demonstrated in Algorithm 39 on lines 8 and 9.

In Chapter 4, we noted that using a convergence condition based on unchanged assignments between iterations could significantly increase runtime. However, due to the other optimisations implemented in KESBA, this trade-off is acceptable.

### 8.4.4 Increased iterations

In Chapter 4, we discussed the *max_iters* parameter for Lloyd's algorithm, suggesting a default value of 50 for all experiments in this thesis. We noted with unlimited computational resources we would set this value higher. However, when using Forgy initialisation with 10 restarts, as we have done throughout this thesis, we had to carefully balance the number of iterations to ensure sufficient convergence while allowing the algorithms to finish within our seven-day runtime limit.

However, for KESBA, the optimisations we implemented allow the algorithm to afford a higher maximum number of iterations without exceeding the runtime limit. Typically, as dataset size and the number of clusters increase, Lloyd's algorithm requires more iterations to converge. In Chapter 4, we demonstrated that while 50 iterations were sufficient for most datasets in the UCR archive, some of the larger datasets did exceed this limit. Since KESBA is designed for very large datasets, it is appropriate to significantly increase the *max_iters* value to 300 by default to ensure convergence, even for datasets much larger than those in the UCR archive. For many datasets in the UCR archive, we do not expect to approach this number of iterations, and the larger value is set to provide redundancy for the largest datasets.

For the same reason, we can also increase the *max_iters* parameter for the Random Subset Elastic SSG Barycentre Average. We set *max_iters* for this averaging technique to 300. Again, while we do not expect most datasets to reach this number of iterations, the higher limit is set for redundancy.

Although increasing the number of iterations may lead to longer runtimes, the optimisations made to KESBA make this an acceptable trade-off, providing a more scalable algorithm for even the largest datasets. The *max_iters* for both the Lloyd's iteration and Random Subset Elastic SSG Barycentre Average is provided as a parameter to the KESBA algorithm.

### 8.4.5 Sakoe-Chiba bounding window

The final optimisation we propose is the use of a Sakoe-Chiba bounding window. The same Sakoe-chiba bounding window is used for every elastic distance computation within the KESBA. This includes within *k*-means++, the assignment stage, the empty cluster stage and the averaging stage. This greatly reduces the computational cost of using a elastic distance.

While the Sakoe-Chiba bounding window has been used in the assignment phase of Lloyd's algorithm [118], we have found no examples of it being used with a barycentre average. A large portion of the computation time for the KESBA is within the Random Subset Elastic SSG Barycentre Average. Specifically computing the elastic cost matrix (line 3 in Algorithm 41).

However, it is a non trivial task to set a value for the bounding window. As such in Section 8.7 we conduct extensive experiments aimed at identifying a value that strikes the best balance between computational efficiency and maintaining the algorithm's strong clustering performance. This will provide practitioners with a window size that maximises performance benefits while preserving clustering quality.

## 8.5 KESBA cluster configuration

KESBA is designed to be versatile, giving practitioners full control over balancing runtime and clustering performance. We recommend that practitioners primarily parameterise KESBA based on their specific runtime requirements. However, we provide default KESBA parameters that strike a balance between computational efficiency and clustering quality.

Table 8.2 presents our recommended default parameters. While these serve as a starting point, we strongly suggest practitioners consult our detailed KESBA tuning experiments in Section 8.7. These experiments are not intended to optimise clustering performance, but to demonstrate how adjusting KESBA's parameters can influence both runtime and performance. This should help practitioners make informed decisions by illustrating the impact each parameter has on computational time and clustering quality.

For our KESBA experiments, we have chosen two elastic distances: TWE and MSM. TWE is selected because of its consistent performance in both the PAM and Elastic Barycentre Averaging experiments. However, as highlighted throughout this thesis, TWE is also one of the most computationally expensive elastic distances. To balance this, we include MSM, which also performs well in the PAM and Elastic Barycentre Averaging experiments but is generally much faster than TWE. These two distances should illustrate how the choice of elastic distance affects both clustering and runtime performance in KESBA. Practitioners may opt for any elastic distance that computes a complete optimal alignment path, as outlined in Chapter 7.

| | *max_iters* | *ba_subset_size* | *window* | *init_algo* | *distance* |
|---|---|---|---|---|---|
| KESBA-TWE | 300 | 0.4 | 0.4 | TWE-$k$-means++ | TWE |
| KESBA-MSM | 300 | 0.5 | 0.5 | MSM-$k$-means++ | MSM |

Table 8.2 Default parameters for KESBA.

Table 8.3 shows the default parameters for the Random Subset Elastic SSG Barycentre Average used across all KESBA clusterers and experiments. We follow the default parameters suggested by [108] for the initial and final step sizes. The *window* and *ba_subset_size* parameters are excluded from this table, as they vary depending on the specific KESBA configuration. For example, in the KESBA-TWE configuration, a *window* size of 0.4 and a *ba_subset_size* of 0.4 would be used for the Random Subset Elastic SSG Barycentre Average.

| *tol* | *max_iters* | *initial_step_size* | *end_step_size* |
|---|---|---|---|
| $1 \times 10^{-6}$ | 300 | 0.005 | 0.05 |

Table 8.3 KESBA Random Subset Elastic SSG Barycentre Average parameters

## 8.6 KESBA experiment

### 8.6.1 Combined Test-Train Split

Figure 8.7 presents the CD diagrams for KESBA, compared to the baseline and state-of-the-art clusterers identified in this thesis. For every evaluation metric, both KESBA-TWE and KESBA-MSM are consistently ranked in the top clique. For AMI and NMI, KESBA-MSM and KESBA-TWE achieve nearly identical rankings. However, for ARI and CLACC, KESBA-TWE performs slightly better than KESBA-MSM. While neither KESBA variant outperforms their Elastic Barycentre Average counterparts, *k*-means-ba-TWE and *k*-means-ba-MSM, they come very close in all evaluation metrics except ARI, where the Elastic Barycentre Average demonstrates considerably better performance. Additionally, for AMI, CLACC and NMI KESBA is significantly better than *k*-shapes, although for ARI, *k*-shapes narrowly makes it into the top clique.

Looking at the raw average scores in Table 8.4, the difference between KESBA-MSM and KESBA-TWE is minimal, with KESBA-TWE achieving slightly higher average scores across all evaluation metrics, though the margin is small. KESBA-MSM shows similar average scores to *k*-means-ba-MSM, except for CLACC, where *k*-means-ba-MSM performs noticeably better. However, *k*-means-ba-TWE appears stronger than KESBA-TWE in terms of average score, particularly for ARI and CLACC.

In Figure 8.8, we directly compare KESBA to its Elastic Barycentre Average *k*-means counterpart for ARI. KESBA-TWE and *k*-means-ba-TWE perform similarly,

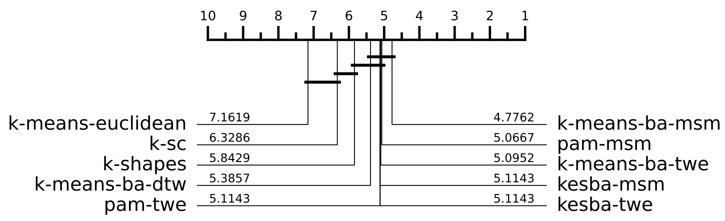


Fig. 8.3 AMI

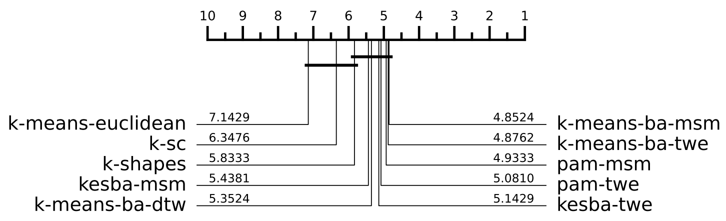


Fig. 8.4 ARI

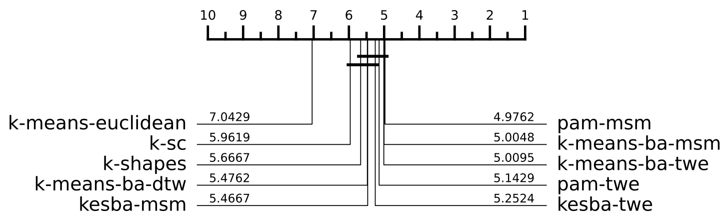


Fig. 8.5 CLACC

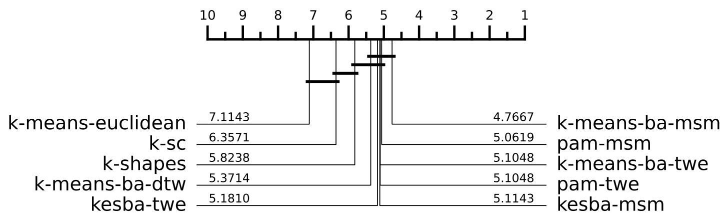


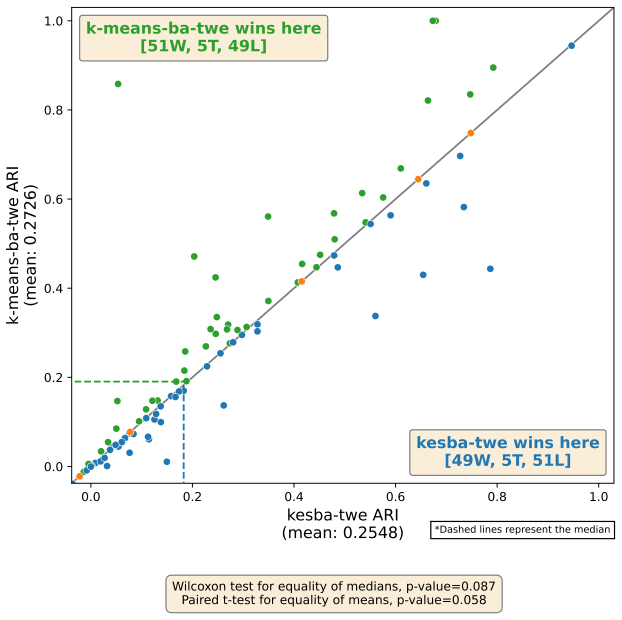
Fig. 8.6 NMI

Fig. 8.7 CD diagrams of KESBA experiment over 105 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.37.

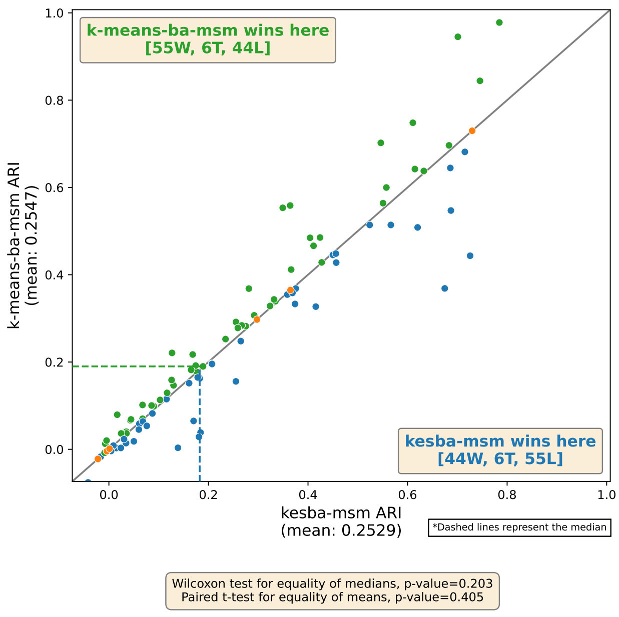
| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| k-means-ba-dtw | 0.255 | 0.301 | 0.571 | 0.326 | 0.709 |
| k-means-ba-msm | 0.255 | 0.306 | 0.573 | 0.328 | 0.706 |
| k-means-ba-twe | 0.273 | 0.315 | 0.583 | 0.338 | 0.713 |
| k-means-euclidean | 0.199 | 0.248 | 0.522 | 0.274 | 0.690 |
| k-sc | 0.215 | 0.259 | 0.543 | 0.281 | 0.660 |
| k-shapes | 0.231 | 0.288 | 0.555 | 0.310 | 0.700 |
| kesba-msm | 0.253 | 0.303 | 0.566 | 0.325 | 0.705 |
| kesba-twe | 0.255 | 0.306 | 0.570 | 0.328 | 0.702 |
| pam-msm | 0.267 | 0.312 | 0.582 | 0.334 | 0.714 |
| pam-twe | **0.274** | **0.317** | **0.588** | **0.340** | **0.717** |

Table 8.4 Summary of average score across multiple evaluation metrics over 105 datasets from the UCR archive using the combined test-train split.

with *k*-means-ba-TWE winning 51 datasets and KESBA-TWE winning 49. The gap is slightly larger for KESBA-MSM and *k*-means-ba-MSM, where *k*-means-ba-MSM wins on 55 datasets compared to KESBA-MSM's 44. In both cases, KESBA's wins are by small margins, suggesting that even with optimisations that reduce the amount of data used in the averaging phase, KESBA still performs comparably across most datasets. Overall, this comparison indicates that KESBA offers a viable alternative to the Elastic Barycentre Average, delivering very similar results while being significantly faster.



(a) KESBA-TWE compared to *k*-means-ba-TWE



(b) KESBA-MSM compared to *k*-means-ba-MSM

Fig. 8.8 KESBA-TWE and KESBA-MSM results compared directly to *k*-means-ba-TWE and *k*-means-ba-MSM, respectively, for ARI over 105 datasets from the UCR archive using the combined test-train split.

We have demonstrated that KESBA's clustering performance is not significantly different from its Elastic Barycentre Averaging counterpart, and it remains competitive with the best-performing PAM clusterers. KESBA achieves this while having a significantly lower runtime than both PAM and the Elastic Barycentre Average clusterers. Figure 8.9 presents the relative FitTime violin plots, illustrating that KESBA is consistently and significantly faster than its Elastic Barycentre Average counterparts.

To further showcase KESBA's FitTime superiority, we analysed the values used to construct the Violin plots in Figure 8.9. Table 8.5 presents the total, mean,

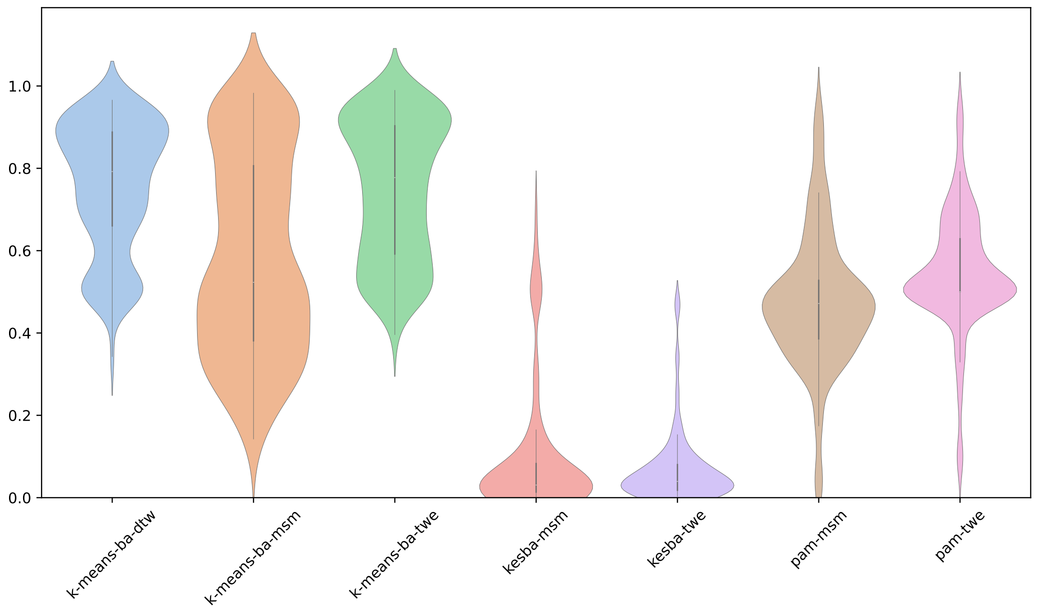


Fig. 8.9 Relative FitTime violin plot comparison for KESBA, PAM and Elastic Barycentre Average clusterers over 105 datasets from the UCR archive using the combined test-train split.

median, and maximum runtimes in hours for each clusterer across 105 datasets. While PAM exhibits faster overall total, mean, and median runtimes compared to KESBA, KESBA-MSM recorded the fastest maximum runtime, with KESBA-TWE following closely as the second fastest. This underscores KESBA's key advantage: its scalability.

Although KESBA's overall runtime statistics—such as total, mean, and median—are very similar to PAM, the maximum runtime statistic highlights PAM's scalability issue. While PAM performs well on smaller datasets, its runtime grows exponentially due to the need to compute a pairwise distance matrix, which has a time complexity of $O(n^2)$. This exponential growth, coupled with the high computational cost of elastic distances, renders PAM impractical for larger datasets.

A clear example of KESBA's efficiency is its performance on the Crop dataset, the largest dataset in the UCR archive with 240,000 unique time series instances. Using the combined test-train split, KESBA-MSM completed the dataset in just 48 minutes, while KESBA-TWE finished in 69 minutes. In contrast, none of the PAM clusterers using elastic distances were able to produce results for the Crop dataset within the seven-day runtime limit. Given that PAM would require 288,000,000 unique elastic distance computations, its runtime becomes computationally infeasible. Although we do not know exactly how long PAM would take to complete, it is evident that it would exceed 168 hours (and likely far more), whereas KESBA finishes in around one hour. This stark difference underscores the superior scalability of KESBA compared to PAM.

Furthermore, if the Crop dataset had been included in the overall experiment, KESBA would have achieved the lowest total, mean, median, and maximum runtimes among all the clusterers considered.

| **Metric** | **k-means-ba-msm** | **k-means-ba-twe** | **kesba-msm** | **kesba-twe** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|
| **Total** | 829.95 | 1193.84 | 386.71 | 485.61 | **321.15** | 391.54 |
| **Mean** | 7.90 | 11.37 | 3.68 | 4.62 | **3.06** | 3.73 |
| **Median** | 0.33 | 0.62 | 0.17 | 0.12 | **0.07** | 0.12 |
| **Max** | 84.33 | 121.71 | **40.32** | 77.56 | 77.83 | 83.12 |

Table 8.5 Four FitTime statistics for completing clustering on 105 datasets from the UCR archive using the combined test-train split. “Total” refers to the cumulative hours required to process all datasets, “Mean” represents the average time taken per dataset, “Median” is the midpoint time to complete a dataset, and “Max” is the longest time taken to complete any single dataset.

Additionally, Figure 8.10 presents the CD diagram for the FitTime of our KESBA experiment. KESBA-MSM ranks among the fastest clusterers, only surpassed by *k*-shapes and *k*-means-Euclidean. However, we have demonstrated that KESBA's performance is significantly better than both *k*-shapes and *k*-means-Euclidean. Moreover, as we will show later in this chapter, practitioners can further reduce KESBA's runtime through its versatile set of parameters, while still maintaining superior performance compared to *k*-shapes.

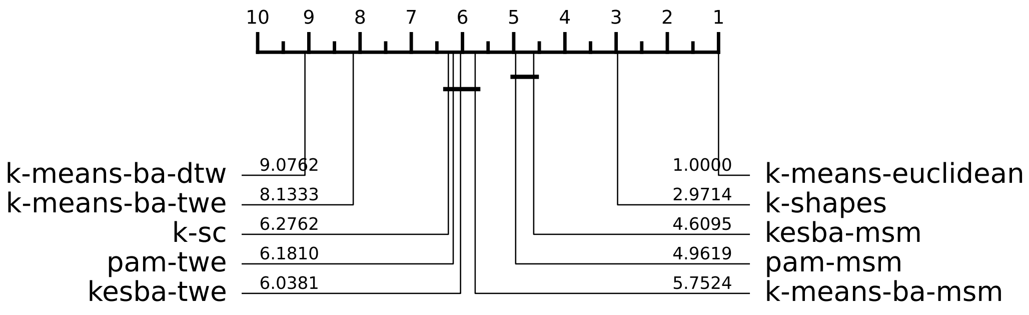


Fig. 8.10 CD diagram for KESBA FitTime compared to other clusterers for 105 UCR archive datasets using the combined test-train split.

Finally, we compare KESBA's performance to *k*-means-soft-DBA. Initially, we excluded *k*-means-soft-DBA from our analysis because it failed to complete 27 datasets within the seven-day runtime limit. Including it would have significantly reduced the quality of the evaluation. However, since *k*-means-soft-DBA has achieved the best clustering performance of any method considered in this thesis, we believe it is important to include in the comparison.

Figure 8.15 shows the CD diagram for KESBA, including *k*-means-soft-DBA, over 84 datasets from the combined test-train split. When including *k*-means-soft-DBA, KESBA falls out of the top clique for every evaluation metric except ARI. Given KESBA's significantly shorter runtime compared to *k*-means-soft-DBA, this is still impressive. Overall, Figure 8.15 shows that for ARI, KESBA-TWE is not significantly different from *k*-means-soft-DBA. However, for CLACC, AMI, and NMI, both KESBA-TWE and KESBA-MSM are significantly different.

Overall, we have presented KESBA for the combined test-train split. KESBA demonstrates state-of-the-art performance while requiring significantly less runtime. Compared to the Elastic Barycentre Averaging *k*-means clusterer, KESBA is considerably faster. While its runtime is comparable to PAM on small to medium-sized UCR datasets, KESBA is orders of magnitude faster than PAM on very large datasets, such as Crop, underscoring its scalability.

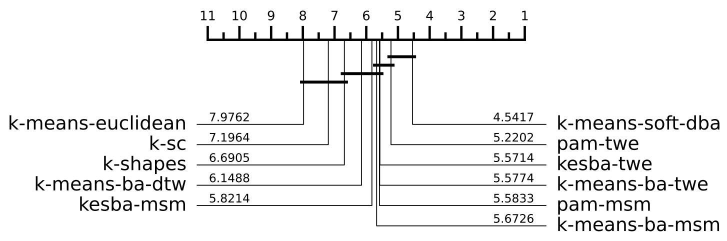


Fig. 8.11 AMI

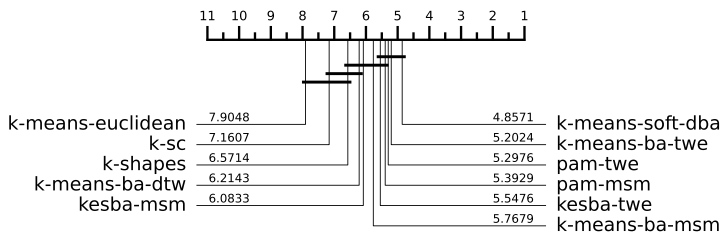


Fig. 8.12 ARI

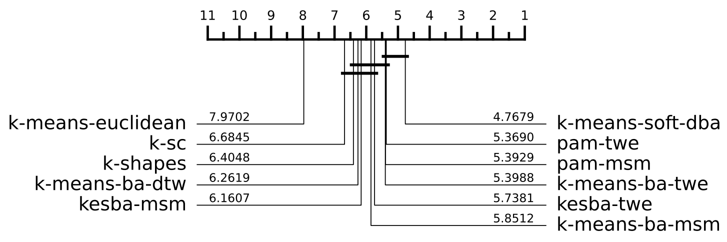


Fig. 8.13 CLACC

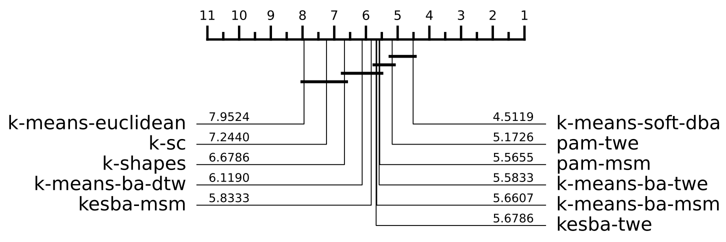


Fig. 8.14 NMI

Fig. 8.15 CD diagrams of KESBA experiment with soft-DBA over 84 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.38.

### 8.6.2 Test-train split

Figure 8.20 presents the CD diagrams for KESBA compared to the baseline and state-of-the-art clusterers outlined in this thesis, using the test-train split. Both KESBA-TWE and KESBA-MSM appear in the top clique for AMI, ARI, and NMI. However, for CLACC, KESBA-MSM and KESBA-TWE falls just short of the top clique. Interestingly, KESBA-MSM consistently outperforms KESBA-TWE in the test-train split, whereas, for the combined test-train split, KESBA-TWE generally performed better. In our Elastic Barycentre Average experiments we also observe that *k*-means-ba-MSM consistently outperforms *k*-means-ba-TWE for the test-train split, so it is not surprising that the same pattern holds for KESBA. Overall, our findings for the test-train split align closely with those from the combined test-train split.

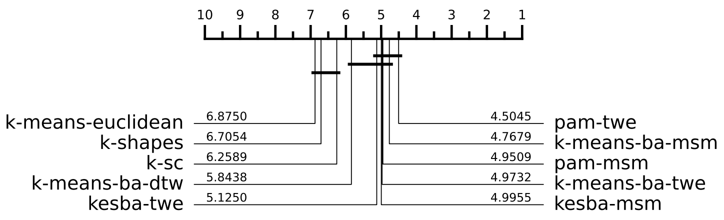


Fig. 8.16 AMI

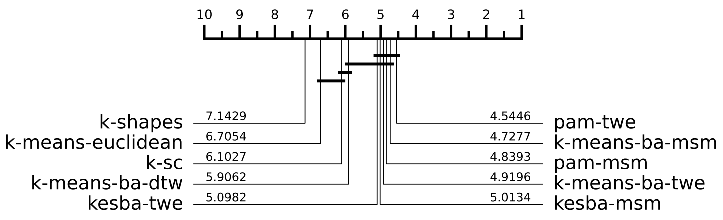


Fig. 8.17 ARI

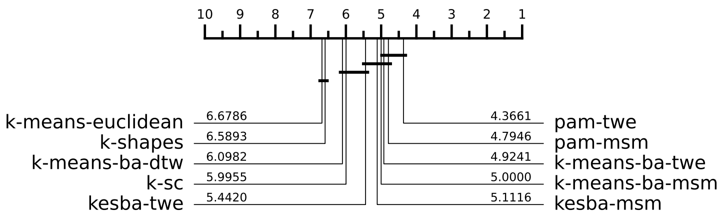


Fig. 8.18 CLACC

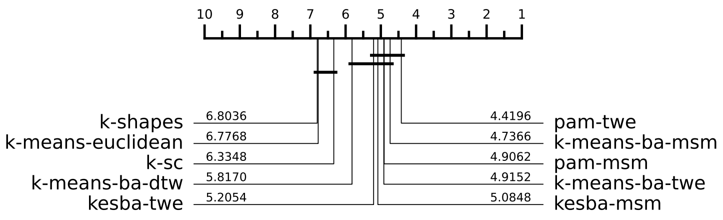


Fig. 8.19 NMI

Fig. 8.20 CD diagrams of KESBA experiment over 112 datasets from the UCR archive using the test-train split.

Figure 8.21 shows the CD diagram for KESBA's FitTime compared to other clusterers. For the test-train split, KESBA-MSM achieves a higher average FitTime rank than *k*-shapes, although the difference is not statistically significant. It is noteworthy that KESBA-MSM is the second fastest clusterer while still appearing in the top clique for AMI, ARI, and NMI in terms of clustering performance.

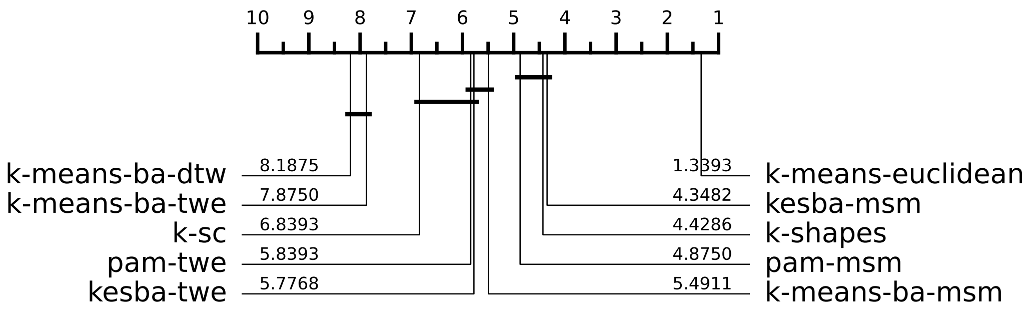


Fig. 8.21 CD diagram for KESBA FitTime compared to other clusterers for 112 UCR archive datasets using the test-train split.

To provide further context on the runtime of each clusterer relative to one another, Table 8.6 outlines various FitTime statistics for each clusterer. In Table 8.6, KESBA-MSM has the lowest FitTime for all four statistics. Compared to the combined test-train split, the PAM clusterers show a significantly higher total runtime. This is mainly due to the inclusion of the Crop dataset in the test-train split experiments (as PAM finished Crop for the test-train split). Table 8.7 shows the runtime (in hours) it took each clusterer to complete the Crop dataset. KESBA is significantly faster than the other clusterers in terms of total runtime hours, being almost twice as fast as the next closest, *k*-means-ba-MSM. This further demonstrates KESBA's scalability.

| **Metric** | **k-means-ba-msm** | **k-means-ba-twe** | **kesba-msm** | **kesba-twe** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|
| **Total hours** | 226.06 | 398.76 | **145.94** | 227.20 | 426.00 | 480.17 |
| **Mean hours** | 2.15 | 3.80 | **1.39** | 2.16 | 4.06 | 4.57 |
| **Median hours** | **0.03** | 0.07 | **0.03** | 0.05 | 0.05 | 0.06 |
| **Max hours** | 27.84 | 46.70 | **21.02** | 38.95 | 234.26 | 190.57 |

Table 8.6 Four FitTime statistics for completing clustering on 112 datasets from the UCR archive using the test-train split. "Total" refers to the cumulative hours required to process all datasets, "Mean" represents the average time taken per dataset, "Median" is the midpoint time to complete a dataset, and "Max" is the longest time taken to complete any single dataset.

| | **k-means-ba-msm** | **k-means-ba-twe** | **kesba-msm** | **kesba-twe** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|
| Crop | 0.39 | 0.87 | **0.22** | 0.34 | 234.26 | 190.57 |

Table 8.7 Total time each clusterer took to complete Crop dataset in hours for the test-train split. We note that our PAM clusterers were able to exceed our normal seven-day runtime limit as they were run before our HPC introduced a runtime limit.

Finally, similar to the combined test-train split, we also introduce *k*-means-soft-DBA to evaluate KESBA against the best-performing clusterer. For the test-train split, when *k*-means-soft-DBA is included in the evaluation, KESBA remains in the top clique for AMI and NMI. However, for ARI and CLACC, KESBA falls into the second-best clique. However, KESBA is significantly better than *k*-shapes for the test-train split where it always appears in the bottom clique.

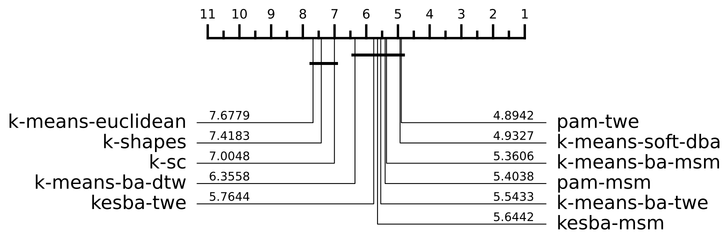


Fig. 8.22 AMI

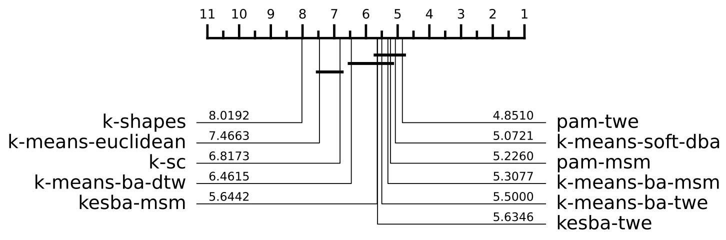


Fig. 8.23 ARI

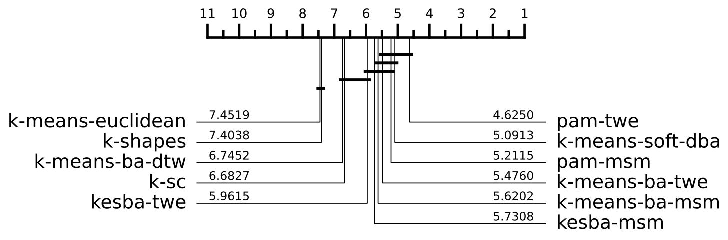


Fig. 8.24 CLACC

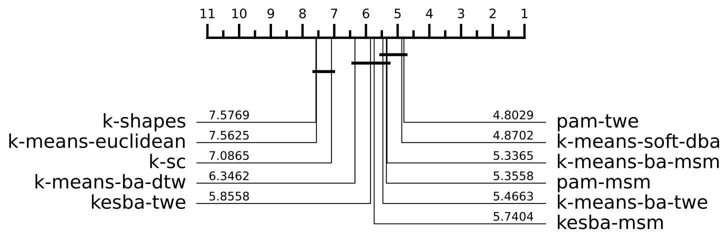


Fig. 8.25 NMI

Fig. 8.26 CD diagrams of KESBA experiment with soft-DBA over 104 datasets from the UCR archive using the test-train split.

### 8.6.3 Conclusion: KESBA

We have introduced KESBA, a fast and scalable TSCL algorithm that achieves state-of-the-art performance while being significantly more computationally efficient than comparable clusterers. Our results demonstrate that KESBA, using TWE and MSM, delivers similar results to the Elastic Barycentre Average *k*-means clusterers, but with substantially reduced runtime. Furthermore, we showed that while PAM can achieve faster runtimes on small to medium-sized datasets, KESBA proves to be orders of magnitude faster on large datasets, all while maintaining comparable clustering performance. Overall, KESBA is a state-of-the-art, versatile, and highly scalable clusterer, purpose-built for real-world TSCL applications.

## 8.7 KESBA Runtime Versatility

One of the key advantages of KESBA is its adaptable runtime, allowing it to meet a variety of computational and clustering performance requirements. Every practitioner has unique constraints, whether prioritising computational speed or clustering performance. KESBA offers several parameters that enable practitioners to balance runtime and performance based on their specific needs.

In this section, we explore how each of KESBA's parameters affects runtime and clustering performance. Our objective is not to optimise clustering performance, but rather to identify reasonable default settings that offer a good balance between the two. Additionally, we aim to demonstrate how practitioners can leverage KESBA's parameters to achieve the desired balance between runtime efficiency and clustering accuracy.

All of the experiments presented are for the combined test-train split because we are focused on the runtime rather than optimisation clustering performance. As such by combing the test-train split this gives KESBA the most amount of data to evaluate it's runtime and scalability over.

### 8.7.1 Elastic $k$-means++

Initialisation is critical for the success of any Lloyd's-based algorithm. In Chapter 4, we evaluated five different initialisation techniques and concluded that using Forgy with 10 restarts provided the most consistent results. However, rerunning each clusterer 10 times with different initial centroids is computationally expensive. To address this, we developed Elastic $k$-means++, which ensures consistency without requiring multiple restarts, significantly reducing runtime while maintaining stable clustering performance.

To demonstrate the effectiveness of Elastic *k*-means++, we conducted an experiment comparing KESBA with Forgy initialisation (using 10 restarts) against KESBA with Elastic *k*-means++ (whcih does not use any restarts). For this experiment, we set both KESBA models to use a full *window* (1.0) and a full *ba_subset_size* (1.0), ensuring that window and subset size do not influence the results. Table 8.8 details the parameters used for each KESBA model in this experiment. Note that "KESBA-full" is not the final KESBA model presented previously, as it uses a full window and does not employ a subset for averaging. Later, we will demonstrate how adding window and averaging parameters improves KESBA's performance while reducing runtime. This experiment focuses solely on the impact of initialisation strategies on KESBA's runtime.

| | *max_iters* | *ba_subset_size* | *window* | *init_algo* | *distance* |
|---|---|---|---|---|---|
| KESBA-full-TWE | 300 | 1.0 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-full-MSM | 300 | 1.0 | 1.0 | MSM-*k*-means++ | MSM |
| KESBA-forgy-restarts-TWE | 300 | 1.0 | 1.0 | Forgy 10 restarts | TWE |
| KESBA-forgy-restarts-MSM | 300 | 1.0 | 1.0 | Forgy 10 restarts | MSM |

Table 8.8 KESBA initialisation experiment parameters.

Figure 8.9 presents the CD diagrams comparing the impact of these initialisation strategies on overall clustering performance. Across all evaluation metrics, Forgy with 10 restarts marginally outperforms Elastic *k*-means++, though the difference is not statistically significant. KESBA-full-TWE consistently ranks in the top clique with KESBA-forgy-restarts-MSM and KESBA-forgy-restarts-TWE, indicating that Elastic *k*-means++ provides comparable performance. KESBA-full-MSM is also in the top clique for AMI, CLACC, and NMI, though it falls slightly behind in ARI.

Where Elastic *k*-means++ truly excels is in reducing runtime. Table 8.9 highlights the runtime savings with different initialisation strategies. KESBA-forgy-

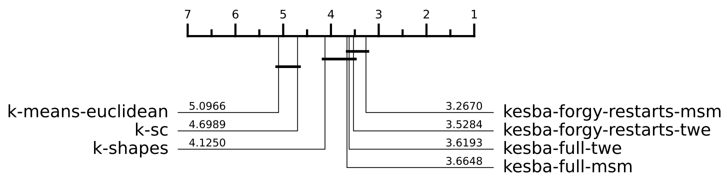


Fig. 8.27 AMI

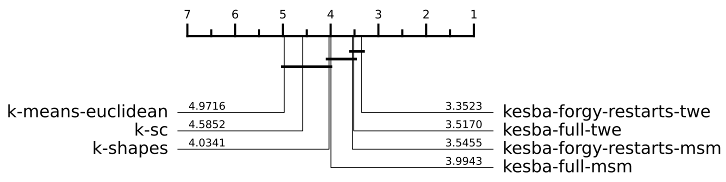


Fig. 8.28 ARI

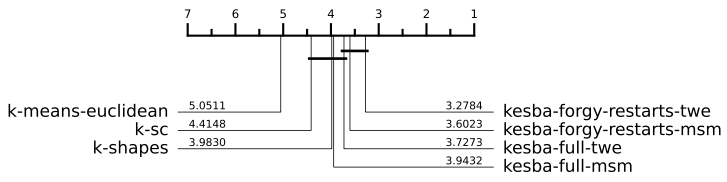


Fig. 8.29 CLACC

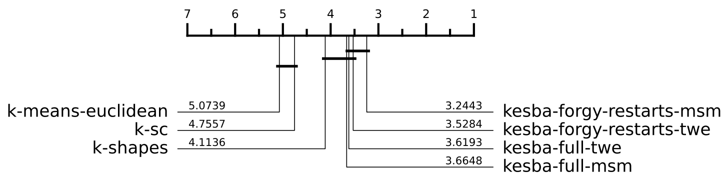


Fig. 8.30 NMI

Fig. 8.31 CD diagrams of KESBA with different initialisation strategies over 88 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.40

restarts-TWE takes over 500 more total hours than KESBA-full-TWE, while KESBA-full-MSM completes all datasets 395 hours faster than KESBA-forgy-restarts-MSM. The mean, median, and maximum runtimes also show that KESBA-full models are significantly faster than Forgy with 10 restarts. On average, KESBA-full-MSM is approximately eight times faster, as Forgy requires nine additional runs due to the restarts.

| **Metric** | **kesba-forgy-restarts-msm** | **kesba-forgy-restarts-twe** | **kesba-full-msm** | **kesba-full-twe** |
|---|---|---|---|---|
| **Total hours** | 449.97 | 613.66 | **53.26** | 110.65 |
| **Mean hours** | 5.11 | 6.97 | **0.61** | 1.26 |
| **Median hours** | 0.49 | 0.94 | **0.06** | 0.10 |
| **Max hours** | 35.48 | 56.98 | **6.08** | 13.09 |

Table 8.9 FitTime statistics for clustering 88 datasets from the UCR archive using the combined test-train split.

Overall, the choice of initialisation strategy significantly impacts runtime. While Forgy with 10 restarts provides consistent performance, it comes at a high computational cost. Elastic *k*-means++ delivers comparable stability and clustering performance at a fraction of the runtime. For practitioners with limited computational resources, Elastic *k*-means++ is a more efficient option. If resources allow, Forgy may offer marginally better results, but running Elastic *k*-means++ with 10 restarts could potentially outperform Forgy with the same number of restarts.

### 8.7.2 Random Subset Elastic SSG Barycentre Subset Size

A key feature of KESBA is the Random Subset Elastic SSG Barycentre, where the *ba_subset_size* parameter controls the amount of data used in each update iteration of the SSG barycentre average computation. Since the averaging computation accounts for a significant portion of KESBA's runtime, this parameter gives practitioners considerable control over the algorithm's runtime. Interestingly, adjusting this parameter not only reduces runtime but can also improves clustering performance. Below, we present experiments demonstrating how different *ba_subset_size* values affect both runtime and clustering performance.

To ensure consistency across the experiments, we kept all other parameters constant. We used *k*-means++ for initialisation, given its previously demonstrated strength, and set the bounding window to 1.0 to avoid influencing the runtime. Additionally, we focused solely on TWE for runtime analysis of different barycentre subset sizes.

Table 8.10 explicitly defines the parameters used for each clusterer. When only changing the *ba_subset_size* parameter, we label the models as "KESBA-average," followed by the subset size (e.g., "10-TWE") and the distance metric used. We also include "KESBA-full," which does not use a subset and applies a full bounding window.

Figure 8.36 presents the CD diagrams for clustering performance with different *ba_subset_size* values. Across all evaluation metrics, there is no statisti-

| | *max_iters* | *ba_subset_size* | *window* | *init_algo* | *distance* |
|---|---|---|---|---|---|
| KESBA-full-TWE | 300 | 1.0 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-average-10-TWE | 300 | 0.1 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-average-20-TWE | 300 | 0.2 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-average-30-TWE | 300 | 0.3 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-average-40-TWE | 300 | 0.4 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-average-50-TWE | 300 | 0.5 | 1.0 | TWE-*k*-means++ | TWE |

Table 8.10 KESBA subset size experiment parameters.

cally significant difference between the subset sizes. While using the full subset (*ba_subset_size* $= 1.0$) yields the best performance for AMI, ARI, and NMI, the improvement in rank is marginal. Interestingly, for CLACC, using *ba_subset_size* $=$ $0.4$ actually outperforms the full window setting.

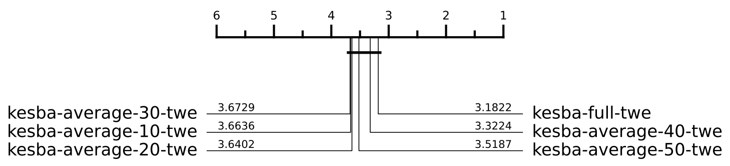


Fig. 8.32 AMI

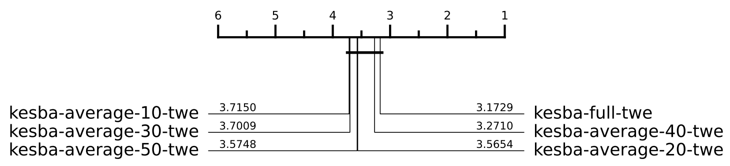


Fig. 8.33 ARI

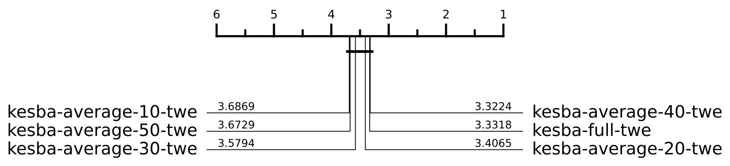


Fig. 8.34 CLACC

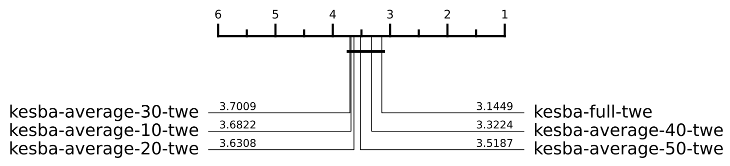


Fig. 8.35 NMI

Fig. 8.36 CD diagrams of KESBA with different *ba_subset_size* over 107 datasets from the UCR archive using the combine test train split. Missing datasets outlined in Table A.41.

Additionally, Figure 8.37 shows the runtime of each *ba_subset_size* value compared to the corresponding ARI score. A smaller *ba_subset_size* leads to a faster runtime, with a significant improvement between using 10% and 20%. The difference in runtime between 20% and 50% is minimal, but between 50%

and 100%, the reduction is substantial, demonstrating the strong impact of the *ba_subset_size* parameter on runtime.

An interesting observation from Figure 8.37 is that when using a *ba_subset_size* of 40%, the average ARI score is almost identical to the score achieved when using all of the data. This was unexpected, as we initially hypothesised that reducing the amount of data would lead to a more linear decrease in performance. However, with TWE, setting *ba_subset_size* to 0.4 achieves nearly the same ARI performance as using the full dataset, while also saving over 100 hours of total runtime.

Finally, Figure 8.37 also shows that using a *ba_subset_size* of 10% or 20% results in the lowest average ARI performance. However, the raw difference in average ARI between using 10% and 100% of the data is minimal — only 0.0044 ARI, while saving approximately 180 hours of total runtime. This suggests that using a very small *ba_subset_size* still performs competitively.

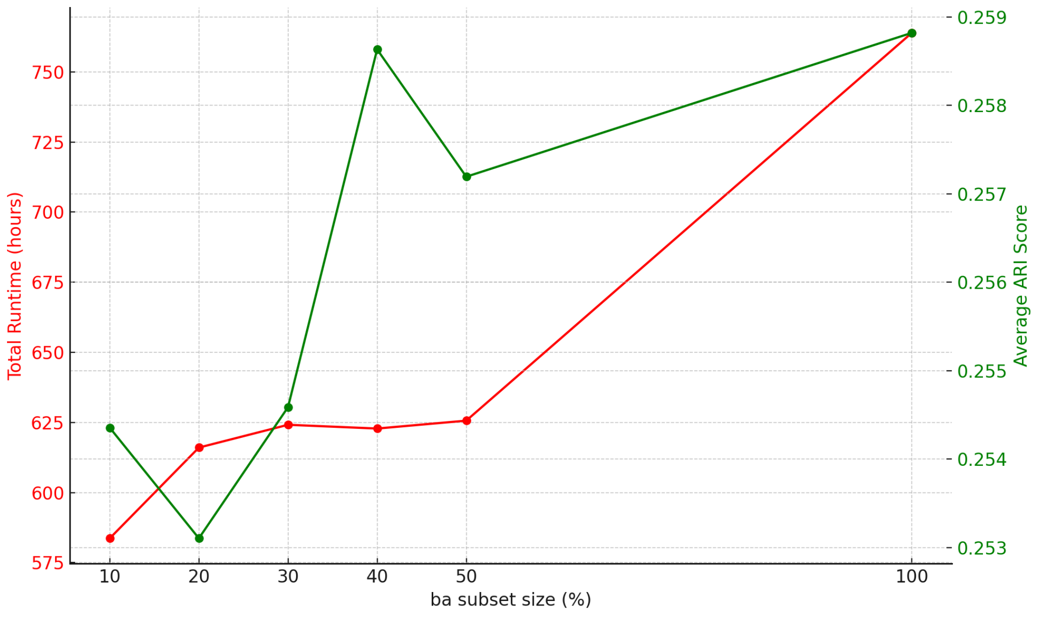


Fig. 8.37 KESBA runtime (red line) compared to average ARI score (green line) for different *ba_subset_sizes* over 106 datasets from the UCR combined test-train split. Missing datasets are outlined in Table A.42.

In summary, our experiments demonstrate that using KESBA with smaller *ba_subset_sizes* dramatically reduces runtime while maintaining comparable clustering performance.

### 8.7.3 Bounding Window

In TSC, bounding windows are commonly used to enhance classification performance and reduce computational runtime [74]. However, their use in TSCL is less frequent, and we found no examples in the literature where bounding windows were applied during barycentre average computation. In this section, we present experiments with different window sizes to explore their effects on clustering performance and runtime. Our findings indicate that using a bounding window not only improves clustering performance but also significantly reduces computational runtime.

Table 8.11 details the configuration for our window experiment. To ensure that the *ba_subset_size* does not influence the runtime, we set it to 1.0 across all clusterers.

| | *max_iters* | *ba_subset_size* | *window* | *init_algo* | *distance* |
|---|---|---|---|---|---|
| KESBA-full-TWE | 300 | 1.0 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-window-10-TWE | 300 | 1.0 | 0.1 | TWE-*k*-means++ | TWE |
| KESBA-window-20-TWE | 300 | 1.0 | 0.2 | TWE-*k*-means++ | TWE |
| KESBA-window-30-TWE | 300 | 1.0 | 0.3 | TWE-*k*-means++ | TWE |
| KESBA-window-40-TWE | 300 | 1.0 | 0.4 | TWE-*k*-means++ | TWE |
| KESBA-window-50-TWE | 300 | 1.0 | 0.5 | TWE-*k*-means++ | TWE |

Table 8.11 KESBA window size experiment parameters.

Figure 8.38 compares the runtime of each window size to the average ARI score achieved. Generally, the runtime (red line) increases fairly linearly as the window size increases. However, the average ARI score (green line) shows that a window size of 30% outperforms using a full window while being over 316 hours faster.

Moreover, using a 10% window reduces the total runtime by over 500 hours, with an average ARI difference of only 0.005 compared to using a full window.

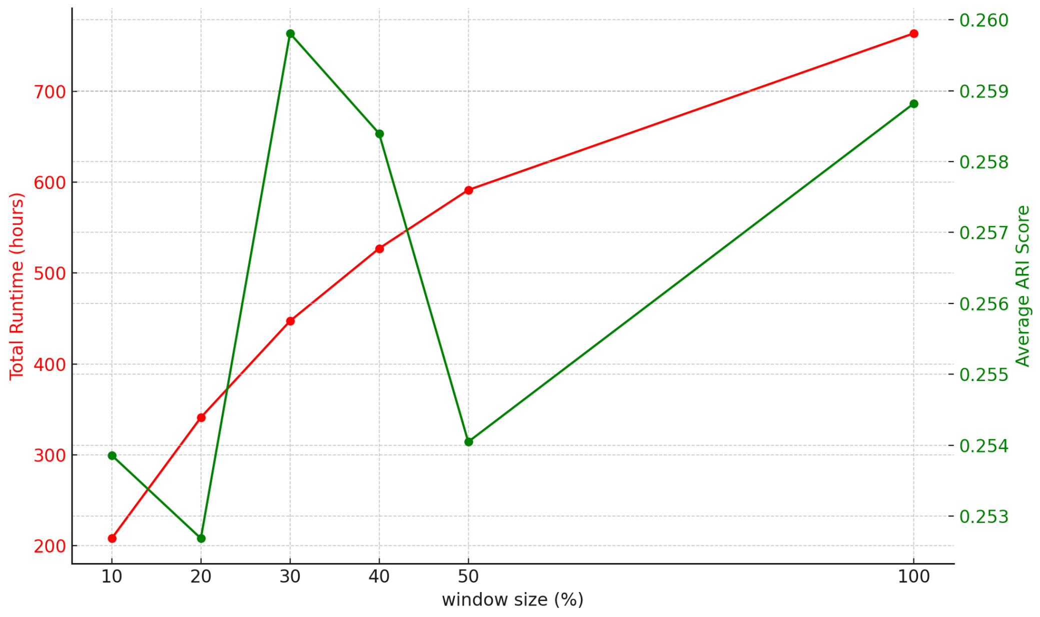


Fig. 8.38 KESBA runtime (red line) compared to average ARI score (green line) for different *window* sizes over 106 datasets from the UCR combined test-train split. Missing datasets are outlined in Table A.42.

Overall, this shows the bounding window is an incredibly powerful parameter for reducing runtime when used in all stages of the algorithm. Interestingly, we also found that smaller window sizes can sometimes improve clustering performance. Even in cases where the clustering performance is slightly worse than using a full window, the raw ARI difference when using a 10% window is remarkably small, making it a highly efficient option.

Overall, we recommend setting a bounding window size between 30% and 40% for optimal clustering performance. However, significant runtime improvements can be achieved by reducing the window to 10%, with only a small trade-off in clustering performance. For extremely large datasets, we suggest using a window size between 10% and 20%.

### 8.7.4 Bounding Window and Barycentre Subset Size

We have demonstrated the independent benefits of using the Random Subset Elastic SSG Barycentre subset size and a bounding window. For the final KESBA clusterer, we propose applying both techniques simultaneously. However, tuning these values separately is not practical in real-world clustering scenarios. To simplify the process, we suggest setting the window size equal to the subset size.

To investigate the impact of using matching *window* and *ba_subset_size* values, we have designed an experiment with various size settings. Table 8.12 outlines the parameters used in this experiment, showing how the *window* and *ba_subset_size* interact. In all cases, the *window* parameter matches the *ba_subset_size*, and we recommend practitioners adopt this approach.

| | *max_iters* | *ba_subset_size* | *window* | *init_algo* | *distance* |
|---|---|---|---|---|---|
| KESBA-full-TWE | 300 | 1.0 | 1.0 | TWE-*k*-means++ | TWE |
| KESBA-both-10-TWE | 300 | 0.1 | 0.1 | TWE-*k*-means++ | TWE |
| KESBA-both-20-TWE | 300 | 0.2 | 0.2 | TWE-*k*-means++ | TWE |
| KESBA-both-30-TWE | 300 | 0.3 | 0.3 | TWE-*k*-means++ | TWE |
| KESBA-both-40-TWE | 300 | 0.4 | 0.4 | TWE-*k*-means++ | TWE |
| KESBA-both-50-TWE | 300 | 0.5 | 0.5 | TWE-*k*-means++ | TWE |

Table 8.12 KESBA window size and barycentre average subset size experiment parameters.

Figure 8.39 presents the results of this experiment. The figure compares the average ARI score (green line) for different window and *ba_subset_size* settings to the runtime (red line). The x-axis represents the matching *window* and *ba_subset_size* percentages.

In Figure 8.39, the runtime increases linearly as the window and subset sizes increase. A similar trend was observed in previous experiments when the window and subset sizes were evaluated independently, though it was less linear than when using both parameters together. This supports our recommendation to set the window and subset sizes to the same value, as it ensures the algorithm scales linearly with each parameter setting.

Interestingly, in Figure 8.39, we find that using a 40% window and *ba_subset_size* actually yields an average ARI score that is 0.003 higher than when using a full window and the entire dataset for averaging. Additionally, this configuration reduces the total runtime by over 250 hours compared to using a full window and the entire dataset.

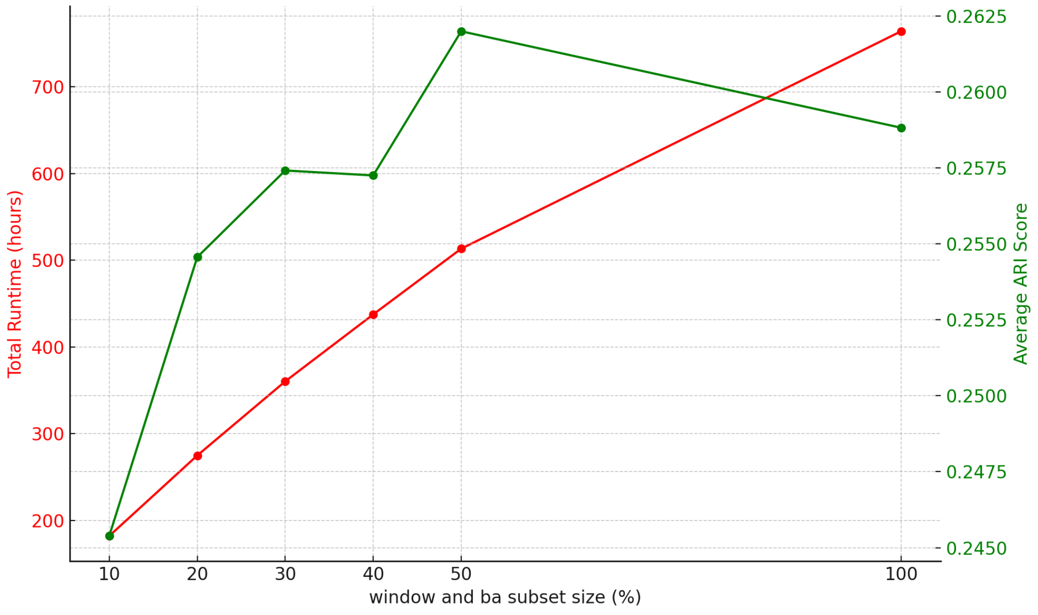


Fig. 8.39 KESBA runtime (red line) compared to average ARI score (green line) for different *window* sizes over 106 datasets from the UCR combined test-train split. Missing datasets are outlined in Table A.42.

The CD diagrams in Figure 8.44 illustrate the performance of using matching window and *ba_subset_size* values. The figure shows that the best-performing KESBA configuration uses a 40% window and a 40% *ba_subset_size* for AMI, CLACC, and NMI. However, for ARI, KESBA-full-TWE slightly outperforms KESBA-both-40-TWE. Despite this, for AMI, ARI, and NMI, there is no statistically significant difference between the various configurations. The only notable exception is CLACC, where KESBA-both-10-TWE performs significantly worse than the other configurations.

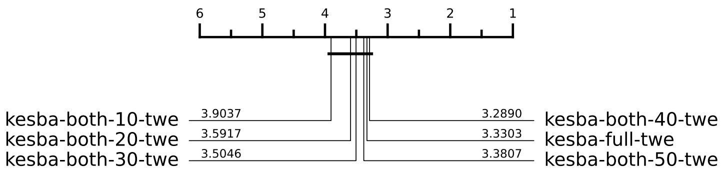


Fig. 8.40 AMI

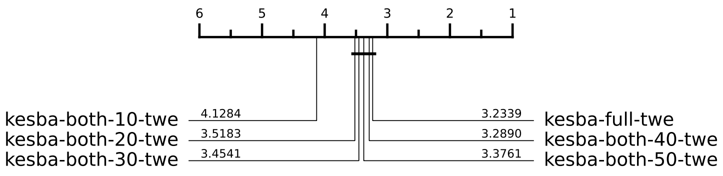


Fig. 8.41 ARI

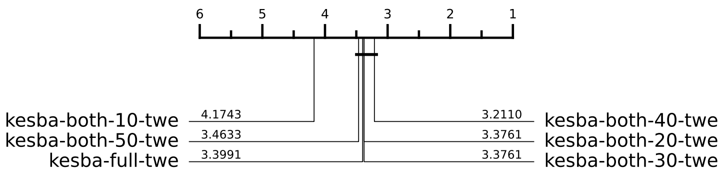


Fig. 8.42 CLACC

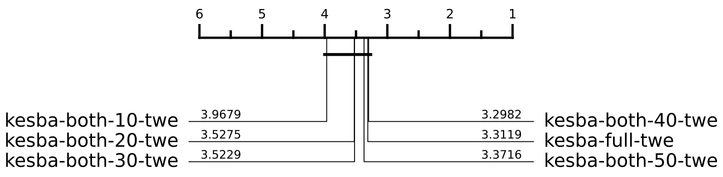


Fig. 8.43 NMI

Fig. 8.44 CD diagrams of KESBA with equal window and ba subset size over 109 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.43.

When comparing the runtime from our previous experiment using only a bounding window, the difference in runtime appears small. However, when we compare KESBA-window-50-TWE with KESBA-both-50-TWE, we find that KESBA-both-50-TWE is 78 hours in total runtime faster. Additionally, KESBA-both-50-TWE achieves a higher average ARI score compared to using only a bounding window. This demonstrates that the bounding window should be used in combination with the *ba_subset_size* for optimal performance.

In conclusion, we have shown that using both the *ba_subset_size* and *window* together improves clustering performance and reduces runtime compared to using either parameter in isolation or using a full window and all the data for averaging. We recommend that practitioners set both the *window* and *ba_subset_size* to a value between 0.4 and 0.5. However, if lower runtime is required, smaller sizes still offer comparable clustering performance while significantly reducing runtime.

## 8.8 Conclusion

In this chapter, we have presented KESBA: a state-of-the-art, versatile, and highly scalable clusterer, designed for real-world TSCL applications. Our empirical results demonstrate that KESBA with TWE performs as well as the current state-of-the-art clusterers, while requiring significantly less computational runtime.

To develop KESBA, we introduced an elastic version of *k*-means++ (Elastic *k*-means++), a novel averaging algorithm (Random Subset Elastic SSG Barycentre Average), and implemented the use of a bounding window within the barycentre averaging process and throughout the entire KESBA algorithm.

Our experiments demonstrate that KESBA's performance is comparable to both PAM and the Elastic Barycentre Average when using the same elastic distances. Furthermore, KESBA consistently outperforms the Elastic Barycentre Average in terms of runtime and achieves significantly faster performance than PAM, especially on medium to large datasets, where the runtime improvement is on the scale of orders of magnitude.

We have provided practitioners with a comprehensive set of experiments exploring KESBA's flexible parameters, which allow for fine-tuning of runtime and clustering performance based on their requirements. Our results indicate that using a bounding window, combined with a matching value for the *ba_subset_size*, outperforms using either technique independently. Specifically, we found that a window and *ba_subset_size* of 40% provided the best clustering performance, while significantly reducing runtime compared to using a full window and the entire dataset for averaging.

# Chapter 9

# The Elastic Clustering Ensemble (ECE) algorithm

In Chapter 6, we experimented with PAM using 12 different elastic distances. We found that the best-performing elastic distances with PAM outperformed the current state-of-the-art clusterers. However, when analysing the performance of each elastic distance across different time series domains, we discovered that no single elastic distance was the best across all domains. In our experiments, spanning seven time series domains, six different elastic distances performed best in at least one domain.

Based on these findings, we hypothesise that an ensemble model capable of selecting the most appropriate distance measure, or weighting predictions based on the suitability of certain elastic distances for specific data, will significantly improve clustering performance. Therefore, in this chapter, we introduce the Elastic Clustering Ensemble (ECE) clusterer, which combines eight different elastic distance PAM clusterers using a novel proportionally weighted ensemble scheme. The ECE clusterer achieves state-of-the-art performance, consistently outperforming each of the individual PAM clusterers that it comprises it.

## 9.1 Introduction

An ensemble of clusterers is a group of base clusterers whose individual decisions are combined through a fusion process to cluster data. In the TSC literature, ensemble models consistently rank among the top-performing classifiers, providing substantial performance improvements [84]. One of the most well-known ensemble models in TSC is the Elastic Ensemble (EE) classifier [74], which consists of 10 1-NN classifiers, each using a different distance measure: Euclidean, DTWCV, MSM, WDTW, ERP, TWE, LCSS, WDDTW, DDTWCV, DTW, and DDTW. The EE employs a proportionally weighted ensemble scheme, which prioritises the predictions of 1-NN classifiers that perform well on the training data, as measured by classification accuracy. The EE has been shown to significantly outperform any individual 1-NN classifier within the ensemble. We hypothesise that a similar ensemble model based on elastic distances could be developed for TSCL, yielding more consistent and superior overall clustering performance compared to using any single elastic distance.

Ensemble diversity is a critical factor in the success of ensembling strategies [74]. In the TSCL literature, several ensemble clustering algorithms have been proposed, such as RandomNet [72] and the Symbolic Pattern Forest (SPF) [71]. However, none have leveraged multiple different elastic distance based clusterers to achieve ensemble diversity. Broadly speaking, ensemble diversity can be introduced in various ways: using different clustering algorithms to form a heterogeneous ensemble, selecting different data attributes for each clusterer (often randomly), or modifying each clusterer internally by re-weighting the training data or incorporating randomisation [74]. We intend to introduce ensemble diversity by using a range of different elastic distances with PAM and weighting them based on their unsupervised clustering performance.

In addition to considering ensemble diversity, clustering ensemble models must also consider the label correspondence problem [4]. When combining the predictions of multiple clusterers, cluster labels often do not align. For example, one clusterer might label a group of points as "cluster 1", while another might label the same group as "cluster 4". Although the labels differ, they refer to the same set of data points. Therefore, in clustering ensembles, label alignment before predictions are made is an important consideration. Various methods exist to address the label correspondence problem, which will be outlined shortly.

In this chapter, we begin by outlining six popular cluster ensemble schemes from the literature. We then propose our own cluster ensemble scheme, inspired by the EE and incorporating concepts from existing cluster ensemble approaches. Following this, we identify the base clusterers we will use to compose our ensemble models. Using these base clusterers and the proposed ensemble scheme, we evaluate the performance of our initial ensemble clusterer against both the baseline clusterers and the individual PAM models that constitute it.

Next, we compare the performance of our ensemble scheme with the six clustering ensemble schemes from the literature. We also experiment with different parameters for our ensemble scheme to assess their impact. Finally, we use the best-performing configuration of our ensemble scheme to create a new ensemble clusterer, the Elastic Clustering Ensemble (ECE), which we evaluate against the state-of-the-art clusterers identified previously in this thesis.

## 9.2 Clustering Ensemble Schemes

To begin, we present existing ensemble schemes that are used throughout the literature. We will use these to evaluate our proposed ensemble scheme against.

### 9.2.1 Simple Vote (SV)

The simple vote (SV) ensemble is one of the simplest method for combining clustering prediction. In this approach, each time series is assigned to the cluster that it appears most frequently in across the base clusterers. For example, if five base clusterers are used, and three assign a time series to "cluster 3" while the other two assign it to "cluster 1", the ensemble will predict "cluster 3" because it has the majority of votes. In the event of a tie, where multiple clusters receive the same number of votes, the final label is randomly selected from the tied options.

However, before a prediction can be made, the label correspondence problem must be addressed. SV employs a cost matrix that quantifies the misalignment between clusters produced by different clusterers. The cost matrix is constructed by comparing the frequency with which points from a cluster in one clustering overlap with points from a cluster in a reference clustering. The objective is to minimise this misalignment and find the optimal correspondence between clusters.

To find the optimal correspondence between clusters, the Hungarian algorithm [62], is applied to the cost matrix. This algorithm finds the optimal mapping of cluster labels from one clustering to another, minimising the total cost, which in this case reflects how well clusters align across different clusterers. Once the optimal mapping is found, the labels of the current clustering are reassigned according to the reference clustering. By repeating this process for each clustering, the algorithm ensures that all clusters are consistently aligned.

### 9.2.2 Iterative Voting (IVC)

The Iterative Voting Consensus (IVC) [74] algorithm maps each data point to a vector, where each element indicates its cluster membership across different clusterings in an ensemble. The algorithm works iteratively, starting with an initial set of cluster centres, each represented by a vector. Each dimension of these vectors corresponds to the cluster memberships of points in the ensemble. A cluster centre, therefore, is a vector that summarises the most common cluster assignments for all the points currently assigned to that cluster.

In each iteration, the algorithm performs two steps. First, it updates the cluster centres by computing the majority value for each dimension across the vectors of the points assigned to the cluster. This means that for each feature (or position in the vector), the centre takes on the value that most of the assigned points have in that dimension. As a result, the cluster centre represents the most frequent cluster memberships across the different clusterings in the ensemble. Second, the algorithm reassigns each data point to the cluster whose centre has the smallest Hamming distance to the point's vector, which measures the number of differing entries between the two vectors.

The process repeats until the clusters stabilise. The IVC algorithm mitigates the correspondence problem through this iterative process.

### 9.2.3 Cluster-based Similarity Partitioning Algorithm (CSPA)

The Cluster-based Similarity Partitioning Algorithm (CSPA) [115] computes a similarity matrix to address the issue of cluster label correspondence across different base clusterers. From the similarity matrix, a similarity graph is constructed where the vertices correspond to data points, and the edges represent the similarity measures between these points. This graph is then partitioned into $k$ clusters using the METIS clustering algorithm [57], which produces the final ensemble predictions.

A cluster label similarity matrix can be computed by comparing the predictions of different base clusterers. Specifically, the similarity between two data points is calculated as the ratio of the number of clusterings in which both points appear in the same cluster to the total number of clusterings in the ensemble [87]. This results in a fraction that represents how frequently two points are clustered together across the ensemble.

### 9.2.4 Meta-CLustering Algorithm (MCLA)

The Meta-CLustering Algorithm (MCLA) [115] is a ensemble schema which mitigates the correspondence problem by transforming the predicted labels for each base clusterer into a hypergraph. Using the hypergraph a voting scheme is applied to produce the final predictions.

Table 9.1 presents an example of how cluster labels from multiple base clusterers are represented as a hypergraph. A hypergraph consists of vertices and hyperedges. In a regular graph, an edge connects exactly two vertices, whereas a hyperedge is a generalisation of an edge that can connect any number of vertices [4].

| | $\lambda(1)$ | $\lambda(2)$ | $\lambda(3)$ |
|---|---|---|---|
| $x_1$ | 1 | 2 | 1 |
| $x_2$ | 1 | 2 | 1 |
| $x_3$ | 1 | 2 | 2 |
| $x_4$ | 2 | 3 | 1 |

(a) Original label vectors

| | $H(1)$ | | | $H(2)$ | | | $H(3)$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | $h_1$ | $h_2$ | $h_3$ | $h_4$ | $h_5$ | $h_6$ | $h_7$ | $h_8$ | $h_9$ |
| $v_1$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $v_2$ | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $v_3$ | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| $v_4$ | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

(b) Hypergraph representation

Table 9.1 Illustrative cluster ensemble problem with $r = 3$ and $k = 3$: Original label vectors (left) and equivalent hypergraph representation with 9 hyperedges (right), where $r$ is the number of base clusterers and $k$ is the number of clusters. Each cluster is transformed into a hyperedge.

In Table 9.1, three base clusterers ($\lambda(1), \lambda(2), \lambda(3)$) are shown, each predicting clusters for four instances ($x_1, x_2, x_3, x_4$). The base clusterers and their predictions are listed in Table 9.1(a). Next, a binary membership indicator matrix $H(q)$ is constructed, with a column representing each cluster (now corresponding to a hyperedge), as shown in Table 9.1(b). Each base clusterer generates a block in this table, for example, $\lambda(1)$ corresponds to the binary membership indicator matrix block $H(1)$.

Each block for each base clusterer is concatenated to form $H = (H(1), H(2), H(3))$, which defines the adjacency matrix of a hypergraph with $n$ vertices and $\sum_{q=1}^{r} k(q)$ hyperedges, where $r$ is the number of base clusterers and $q$ denotes the cluster labels for the $qth$ base clusterer. This process maps each cluster to a hyperedge and the set of clusterings to a hypergraph. By transforming the predicted clusters to a hypergraph the correspondence problem is mitigated.

Using the computed hypergraph, MCLA generates the final predictions by constructing a Meta-Graph, where each hyperedge is treated as a node. This Meta-Graph is then clustered using the METIS graph clustering algorithm to form meta-clusters, grouping together similar hyperedges. For each meta-cluster, an average of each cluster is computed, which is later used to assess similarity when assigning new cluster objects. This process mitigates the label correspondence problem.

### 9.2.5 Hybrid Bipartite Graph Formulation (HBGF)

The Hybrid Bipartite Graph Formulation (HBGF) [34] models instances and clusters simultaneously in a graph. The graph edges can only connect instance vertices to cluster vertices, resulting in a bipartite graph [34]. This means the HBGF bipartite graph has two types of vertices:

1. Instance vertices: representing the data point being clustered
2. Cluster vertices: representing each cluster from the ensemble of base clusterers.

An edge is drawn between an instance vertex and a cluster vertex if the instance belongs to that cluster. The weight of each edge is set to 1 to represent this membership. No edges exist between vertices of the same type: instance vertices cannot be directly connected to other instance vertices, and cluster vertices cannot be connected to other cluster vertices. This process produces a graph that can be partitioned using a graph partition algorithm such as METIS to produce the final ensemble clusterings.

### 9.2.6 Nonnegative Matrix Factorisation (NMF)

Nonnegative Matrix Factorisation (NMF) [70] creates a collective similarity matrix and decomposes it into lower-dimensional factors that capture the underlying cluster structure shared among the base clusterers. This approach addresses the label correspondence problem while extracting similarities to form a final cluster consensus.

NMF begins by constructing a connectivity matrix $M$ from the base clusterers' predictions. This $n \times n$ matrix, where $n$ is the number of data instances, contains entries $M_{ij}$ that represent the frequency with which instances $i$ and $j$ are assigned to the same cluster across all base clusterers. The matrix encapsulates the co-association information between pairs of instances.

The connectivity matrix $M$ is then factorised using an iterative algorithm that ensures non-negativity in the resulting matrices. Specifically, NMF decomposes $M$ into two non-negative matrices: $Q$, which represents cluster membership indicators, and $S$, a scaling matrix that adjusts the influence of each cluster.

To generate the final ensemble clustering predictions, the product of $Q$ and the square root of $S$ is computed. Each data instance is then assigned to the cluster corresponding to the highest value in its representation, producing the final ensemble predictions.

## 9.3 Elastic Unsupervised Proportional Weighting (EUPW)

The Elastic Ensemble (EE) for TSC [74] proposes a proportionally weighted voting scheme based on the classification accuracy of each base classifier, normalised over the number of transformations. In this scheme, a classifier's weight in the ensemble is equal to its normalised training accuracy. When using this approach with 10 1-NN classifiers with different elastic distances, [74] showed that EE significantly outperforms any individual 1-NN classifier that composes EE. We hypothesise that by developing a similar ensemble scheme tailored for elastic distance-based clusterers, we could achieve comparable performance improvements for clustering.

However, the EE proportional ensemble scheme cannot be directly applied to TSCL ensembling because TSCL is an unsupervised task, meaning there are no ground truth labels to calculate classification accuracy. Additionally, the EE scheme lacks a strategy to address the label correspondence problem. To overcome these challenges, we propose the Elastic Unsupervised Proportional Weighting (EUPW) ensemble scheme, which adapts the EE proportional weighting for unsupervised tasks and includes an additional step to mitigate the label correspondence issue.

The first issue we tackle is the use of a supervised metric. Our proposal is straightforward: instead of using a supervised evaluation metric, we use an unsupervised one. Several unsupervised metrics exist, such as the Calinski-Harabasz Index (CHI) [15], Davies-Bouldin Index (DBI) [25], and Silhouette Coefficient [58]. Many of these metrics rely on a distance measure and an averaging technique, typically the Euclidean distance and the arithmetic mean. However, as demonstrated in Chapters 5 and 6, the Euclidean distance is not suitable for TSCL, and in Chapter 7, we showed that the Elastic Barycentre Average is superior to the arithmetic mean for time series data. Therefore, we hypothesise that traditional unsupervised evaluation metrics may not perform well for TSCL.

To address this, we propose elastic unsupervised evaluation metrics for EUPW by adapting existing metrics, replacing the Euclidean distance with an elastic distance. Additionally, if the evaluation metric relies on the arithmetic mean, we substitute it with the Elastic Barycentre Average, using the same elastic distance.

For our initial experiments, we adapt the Calinski-Harabasz Index (CHI) to create the Elastic Calinski-Harabasz Index (ECHI). Later in this chapter, we also experiment with the Davies-Bouldin Index (DBI), creating the Elastic Davies-Bouldin Index (EDBI). However, our initial focus remains on the using ECHI.

Adapting an unsupervised measure to be "elastic" involves replacing the Euclidean distance and arithmetic mean with an elastic distance and the Elastic Barycentre Average, respectively. Specifically, for our experiments, we use MSM and TWE, as these distances produced the best results with the Elastic Barycentre Average in Chapter 7. In principle, any elastic distance that generates a complete warping path could be used with an elastic unsupervised measure.

When using TWE with the ECHI, we refer to the measure as ECHI-TWE, which uses the TWE distance and its corresponding Elastic Barycentre Average. Similarly, when using MSM, we refer to it as ECHI-MSM, which incorporates the MSM distance and its Elastic Barycentre Average.

For ECHI, a higher value indicates better clustering, whereas a lower value indicates worse clustering. This allows easy integration of the measure into the EE proportional weighting scheme. However, for some unsupervised evaluation metrics, such as DBI, lower scores indicate better clustering. To ensure consistency, we invert such scores by dividing them by 1, allowing these metrics to be seamlessly used within the EUPW ensemble scheme.

In addition to updating the weighting evaluation metric, we address the label correspondence problem by using the same alignment algorithm from the SV ensemble scheme. Specifically, when predicting the final cluster labels, a cost matrix is constructed to quantify cluster misalignment, and the Hungarian algorithm is applied to find optimal label mappings.

In summary, the EUPW scheme weights each clusterer based on an unsupervised elastic evaluation metric like ECHI, normalised by the number of transformations. Once a base clusterer's prediction is selected according to its weighting, a cost matrix is constructed to account for misalignment between base clusterers. The Hungarian algorithm is then applied to determine the optimal label mappings, and the final prediction is returned.

## 9.4 Elastic Clustering Ensemble Experiment

### 9.4.1 Base PAM Clusterers

For our elastic clustering ensemble experiments, we chose PAM as our base clusterer. PAM was selected because it was one of the top-performing clusterers in our previous experiments, demonstrating strong results on both the combined test-train and test-train splits. Additionally, as highlighted in Chapter 6, we observed that various elastic distances with PAM performed best for different time series domains. This suggests that PAM with varying elastic distances provides diverse clustering results, making it an ideal candidate for ensembling.

In the original Elastic Ensemble (EE) [74], 10 1-NN classifiers using different elastic distances were combined to form an ensemble. The specific distances used were: DTWCV, MSM, WDTW, ERP, TWE, LCSS, WDDTW, DDTWCV, DTW, and DDTW. DTWCV and DDTWCV are tuned versions of the respective distances incorporating a bounding window. These distances were selected for their ability to produce diverse results across the UCR archive. Following a similar rationale, we have chosen elastic distances for our PAM ensemble.

Our first choice is TWE, which was the top-performing distance with PAM across all evaluation metrics in Section 6.5, making it an obvious selection. Next, we selected distances that performed best in individual domains, either in the test-train split or the combined test-train split, as described in Chapter 6. Across both splits, MSM, WDTW, soft-DTW, shape-DTW, and ADTW were identified as the top performers in one or more domains. However, we exclude shape-DTW as a candidate due to its failure to complete for many datasets, which hinders evaluation. Therefore, MSM, WDTW, soft-DTW, and ADTW were chosen next as base cluster candidates.

To further increase diversity within the ensemble, we also included ERP, DTW, and WDDTW. Although these three distances did not perform as well in our previous clustering experiments, their clustering results were significantly different from the core five distances, thereby adding diversity and potentially enhancing the ensemble's overall performance.

In summary, we selected the following base clusterers for all our ensemble schemes: PAM-DTW, PAM-MSM, PAM-TWE, PAM-ERP, PAM-WDTW, PAM-ADTW, PAM-WDDTW, and PAM-soft-DTW. The clusterings used for each base clusterer are the same as those produced and evaluated in Chapter 6. We have two less elastic distances than the EE, however, two elastic distances in the EE are tuned versions of DTW and DDTW which we are are unable to do in clustering.

## 9.5 EUPW intial experiments

Our initial evaluation aims to assess the effectiveness of the proposed EUPW ensemble scheme. To do this, we conduct an experiment using the eight base PAM clusterers, which are ensembled with EUPW using three different distances: Euclidean, MSM, and TWE. EUPW-Euclidean employs the standard CHI score, while EUPW-TWE and EUPW-MSM use the adapted ECHI evaluation score. We then compare the performance of these ensemble models against the baseline clusterers and the two best-performing PAM variants. The goal is to determine the relative strength of EUPW and evaluate whether the use of ECHI leads to improved clustering performance.

### 9.5.1 Combined Test-Train Split

Figure 9.5 presents the critical difference diagrams for the EUPW ensemble over the combined test-train split. We observe that for all four evaluation metrics, EUPW-TWE and EUPW-MSM are the best performing clusterers and consistently outperform the best-performing PAM models, PAM-TWE and PAM-MSM. Notably, EUPW-TWE ranks as the top-performing clusterer across all evaluation metrics, although the improvement is not statistically significant.

Additionally, EUPW-Euclidean is consistently outperformed by EUPW variants that use elastic distances. This supports our hypothesis that elastic distances enhance the CHI evaluation metric. EUPW-Euclidean only surpasses the best PAM variant in terms of NMI, but lags behind in AMI, ARI, and CLACC.

Table 9.2 presents the average score for each clusterer across all evaluation metrics. EUPW-TWE performs best in ARI and CLACC, while PAM-TWE achieves the highest average scores for AMI, NMI, and RI. Although the ARI and CLACC scores between EUPW-TWE and PAM-TWE are quite similar, there is a noticeable difference in median performance.

Figure 9.6 provides a direct comparison between EUPW-TWE, PAM-TWE, and PAM-MSM. The figure shows that while EUPW-TWE achieves scores similar to both PAM-TWE and PAM-MSM, it exhibits a significantly higher median score, indicating that it performs more consistently across datasets with considerably less variance in its results.

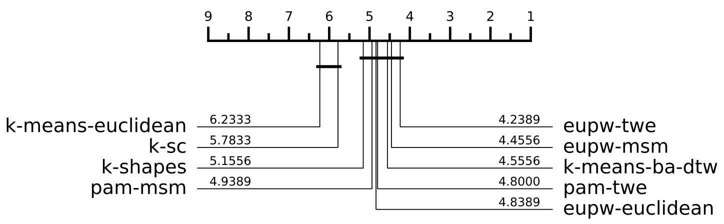


Fig. 9.1 AMI

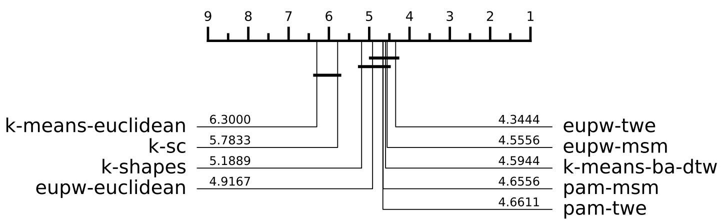


Fig. 9.2 ARI

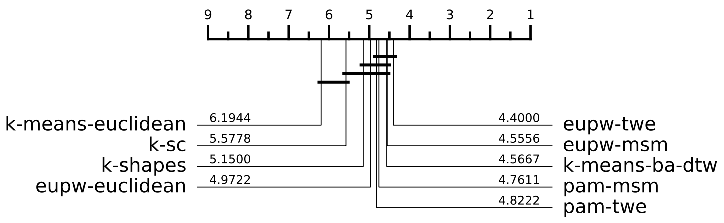


Fig. 9.3 CLACC

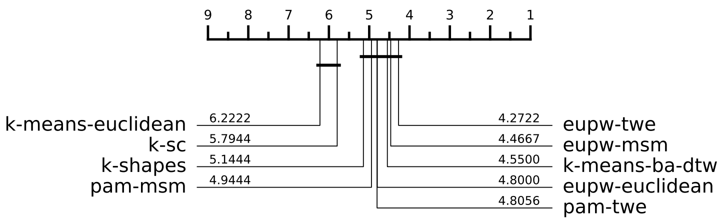


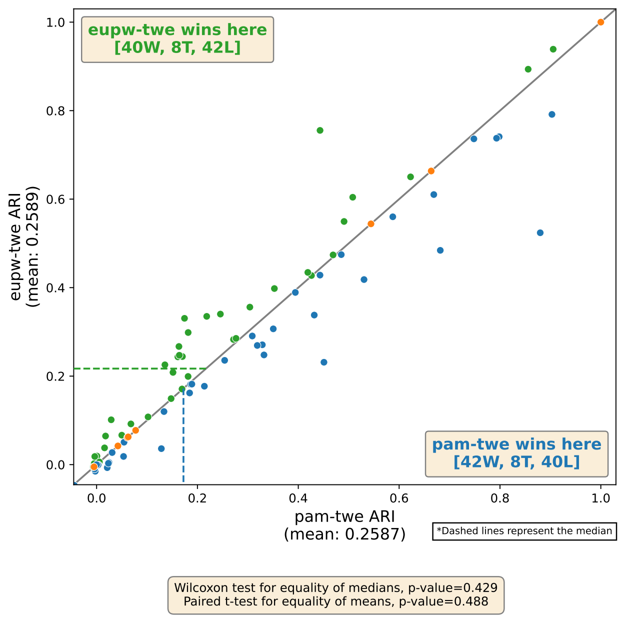
Fig. 9.4 NMI

Fig. 9.5 CD diagrams of EUPW experiment using the baseline clusterers and the two best performing PAM variants over 90 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.44.

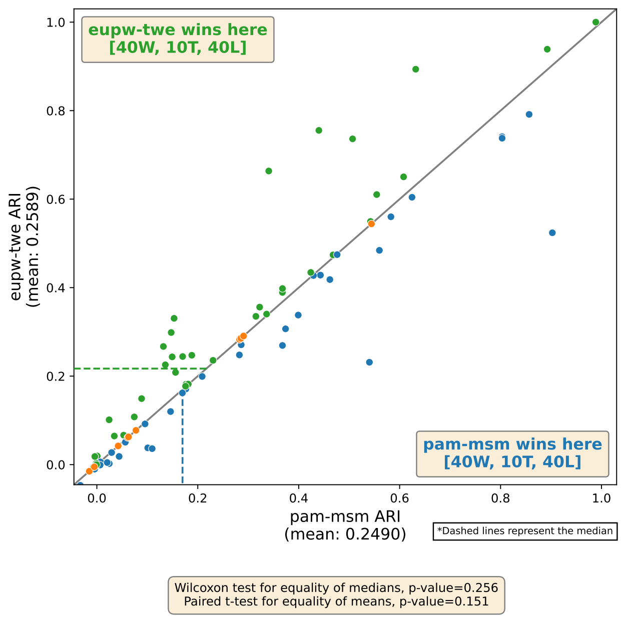
| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| eupw-euclidean | 0.240 | 0.270 | 0.583 | 0.277 | 0.672 |
| eupw-msm | 0.258 | 0.286 | 0.598 | 0.292 | 0.681 |
| eupw-twe | **0.2589** | 0.286 | **0.5985** | 0.292 | 0.681 |
| k-means-ba-dtw | 0.246 | 0.277 | 0.591 | 0.284 | 0.678 |
| k-means-euclidean | 0.186 | 0.222 | 0.535 | 0.229 | 0.655 |
| k-sc | 0.196 | 0.224 | 0.552 | 0.231 | 0.622 |
| k-shapes | 0.229 | 0.268 | 0.579 | 0.274 | 0.673 |
| pam-msm | 0.249 | 0.279 | 0.590 | 0.286 | 0.680 |
| pam-twe | 0.2587 | **0.287** | 0.5984 | **0.293** | **0.685** |

Table 9.2 Summary of average score across multiple evaluation metrics over 90 datasets from the UCR archive using the combined test-train split.

Table 9.3 compares the EUPW results to the baseline clusterers and PAM across different time series domains. Although EUPW does not achieve the highest average ARI score in any single domain, it consistently performs well across all domains. This is emphasised in Table 9.4, which shows the average rank of each clusterer for



(a) EUPW-TWE compared to PAM-TWE



(b) EUPW-TWE compared to PAM-MSM

Fig. 9.6 EUPW-TWE results compared directly to PAM-MSM and PAM-TWE, respectively, over 90 datasets from the UCR archive using the combined test-train split.

ARI over the seven time series domains. Table 9.4 shows PAM-TWE's lowest rank is 2.5 for the ECG domain, but its highest rank is 6.273 for the Spectro domain. In contrast, EUPW-TWE's lowest rank is 3.625 for the Simulated domain, with a highest rank of 4.909 for the Spectro domain. This results in PAM-TWE having an average rank difference of 3.774 between its best and worst ranks, while EUPW-TWE has a smaller average rank difference of only 1.284. This demonstrates the superior consistency of EUPW.

For this experiment (and subsequent EUPW experiments), we regrettably had to exclude 20 datasets due to an issue we identified in our EUPW implementation, specifically when using the ECHI evaluation metric. This problem caused the algorithm to fail on certain datasets. We suspect the issue may be related to the formation of empty clusters when applying the label alignment algorithm. In future iterations of the algorithm, we aim to resolve this issue to ensure a complete set of results can be obtained.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| eupw-euclidean | 0.324 | 0.148 | 0.260 | 0.415 | 0.103 | 0.125 | 0.184 |
| eupw-msm | 0.312 | 0.165 | 0.258 | 0.472 | 0.178 | 0.157 | 0.271 |
| eupw-twe | 0.317 | 0.166 | 0.261 | 0.464 | 0.164 | 0.157 | 0.280 |
| k-means-ba-dtw | 0.291 | 0.178 | 0.212 | **0.586** | 0.173 | 0.155 | 0.137 |
| k-means-euclidean | 0.224 | 0.173 | 0.210 | 0.306 | 0.039 | 0.107 | 0.174 |
| k-sc | 0.222 | **0.185** | **0.270** | 0.187 | 0.032 | 0.074 | 0.395 |
| k-shapes | 0.274 | 0.155 | 0.203 | 0.429 | 0.096 | **0.160** | **0.407** |
| pam-msm | **0.333** | 0.132 | 0.246 | 0.391 | 0.167 | 0.134 | 0.356 |
| pam-twe | 0.325 | 0.146 | 0.252 | 0.475 | **0.179** | 0.136 | 0.351 |

Table 9.3 Average ARI score on problems split by problem domain over 90 datasets from the UCR archive using the combined test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| eupw-euclidean | **3.913** | 6.409 | 4.957 | 5.375 | 6.000 | 4.154 | 5.750 |
| eupw-msm | 4.543 | 5.227 | 4.870 | 4.000 | 4.125 | **3.962** | 4.875 |
| eupw-twe | 4.543 | 4.909 | **4.283** | 3.625 | 4.250 | 4.115 | 4.375 |
| k-means-ba-dtw | 4.848 | **3.364** | 5.609 | **2.625** | **2.750** | 4.846 | 7.500 |
| k-means-euclidean | 7.022 | 3.909 | 5.739 | 7.562 | 7.500 | 6.154 | 7.500 |
| k-sc | 6.413 | 3.818 | 5.043 | 8.188 | 7.375 | 5.615 | 4.375 |
| k-shapes | 4.804 | 4.818 | 5.435 | 5.250 | 5.875 | 5.615 | 4.125 |
| pam-msm | 3.935 | 6.273 | 4.370 | 5.375 | 3.750 | 5.385 | 4.000 |
| pam-twe | 4.978 | 6.273 | 4.696 | 3.000 | 3.375 | 5.154 | **2.500** |

Table 9.4 Average ARI rank performance on problems split by problem domain over 90 datasets from the UCR archive using the combined test-train split

### 9.5.2 Test-train split

Figure 9.11 presents the critical difference diagrams for the EUPW ensemble, the baselines clusterers and the two best PAM variants. For the test-train split, EUPW performs poorly. Additionally, we find that ECHI does not consistently outperform CHI. For AMI and CLACC, EUPW with all distances performs worse than PAM-TWE, but all three outperform PAM-MSM. For ARI, PAM-TWE outperforms all three EUPW distances, and PAM-MSM outperforms EUPW-TWE and EUPW-MSM. Overall, EUPW does not appear suitable for the test-train split.

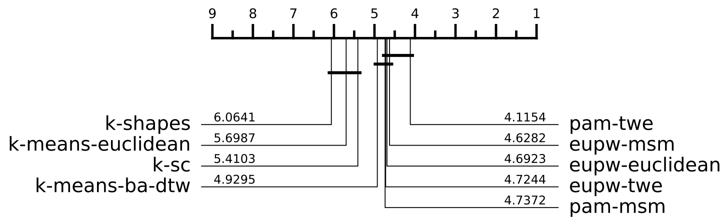


Fig. 9.7 AMI

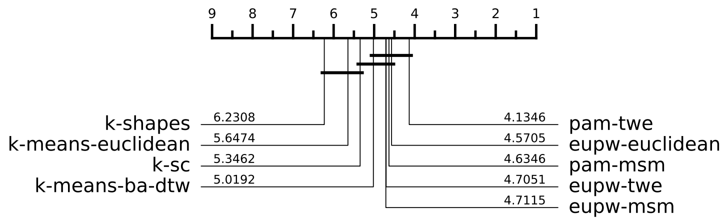


Fig. 9.8 ARI

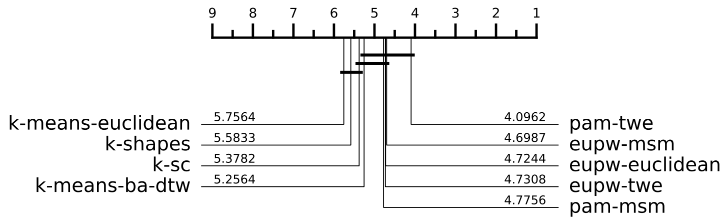


Fig. 9.9 CLACC

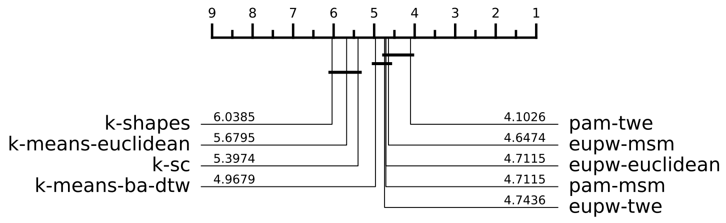


Fig. 9.10 NMI

Fig. 9.11 CD diagrams of EUPW experiment using the baseline clusterers and the two best performing PAM variants over 78 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.45.

### 9.5.3 Conclusion: EUPW initial experiments

We have presented the our initial results for EUPW over both the combined test-train split and the test-train split, comparing EUPW against the baseline clusterers and the two best-performing PAM variants. For the combined test-train split, EUPW with TWE consistently emerged as the top-performing clusterer across all evaluation metrics, although the difference was not statistically significant.

Throughout our combined test-train experiments, we identified one of EUPW's key strengths: its consistency. Compared to the best-performing PAM variants, EUPW's median ARI score was notably higher. Additionally, in our domain-based evaluation, while EUPW did not achieve the highest average score for any domain, it displayed the smallest difference in rank change across datasets, further highlighting its consistent performance.

However, when evaluating EUPW on the test-train split, its performance was significantly weaker. PAM-TWE consistently outperformed all three EUPW distances across every evaluation metric. Moreover, for the test-train split, the original CHI outperformed the ECHI. However, as will be shown, this problem does not seem exclusive to the EUPW ensemble scheme.

In summary, EUPW with TWE shows promise as a strong ensemble scheme for the combined test-train split, however, EUPW does not appear to be a suitable choice for the test-train split.

## 9.6 EUPW Compared to Other Ensemble Schemes

We have evaluated our proposed ensemble scheme against the baseline clusterers and the best-performing PAM variants. Next, we explore how EUPW compares to other popular ensemble techniques from the literature. To do this, we design an evaluation experiment comparing EUPW with six other ensemble techniques previously outlined.

### 9.6.1 Combined Test-Train Split

Figure 9.16 presents the CD diagrams for the EUPW ensemble compared to six other ensemble schemes and the two best-performing PAM variants. For AMI and NMI, EUPW is, on average, the best-performing ensemble scheme. However, for ARI and CLACC, EUPW-TWE is outperformed by both HBGF and CSPA. Notably, EUPW-TWE is the only ensemble scheme that does not rank lower than PAM-TWE for any evaluation metric. Although CSPA and HBGF perform well for CLACC and NMI, they perform worse than PAM-TWE for AMI and NMI, and HBGF performs worse than PAM-MSM for both AMI and NMI. Importantly, none of the clusterers included in the comparison are statistically significantly different from each other.

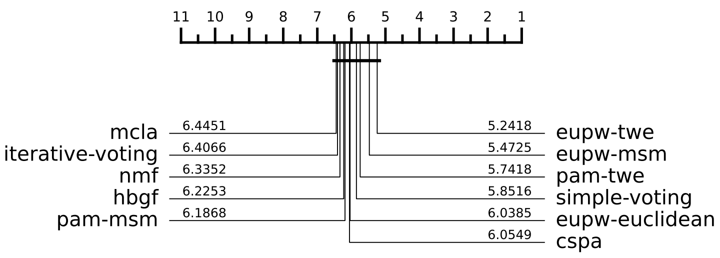


Fig. 9.12 AMI

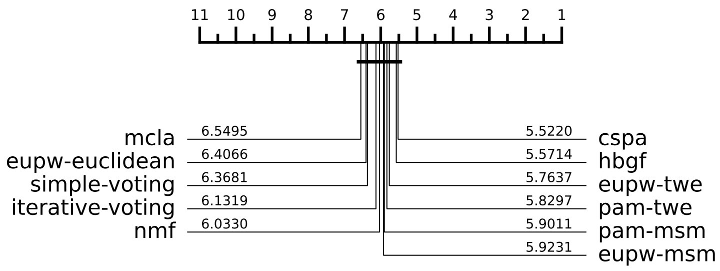


Fig. 9.13 ARI

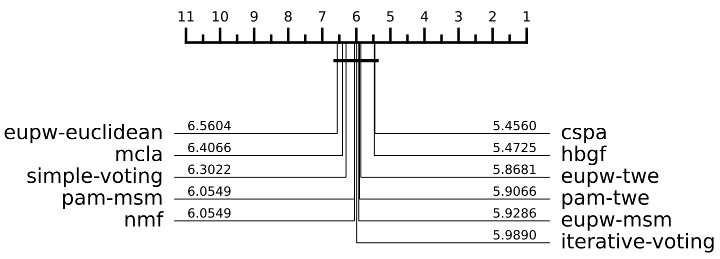


Fig. 9.14 CLACC

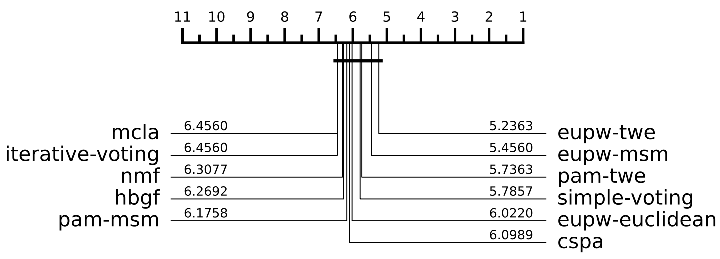


Fig. 9.15 NMI

Fig. 9.16 CD diagrams of EUPW compared to other ensemble schemes over 91 datasets from the UCR archive using the combine test-train split. Missing datasets are outlined in Table A.46

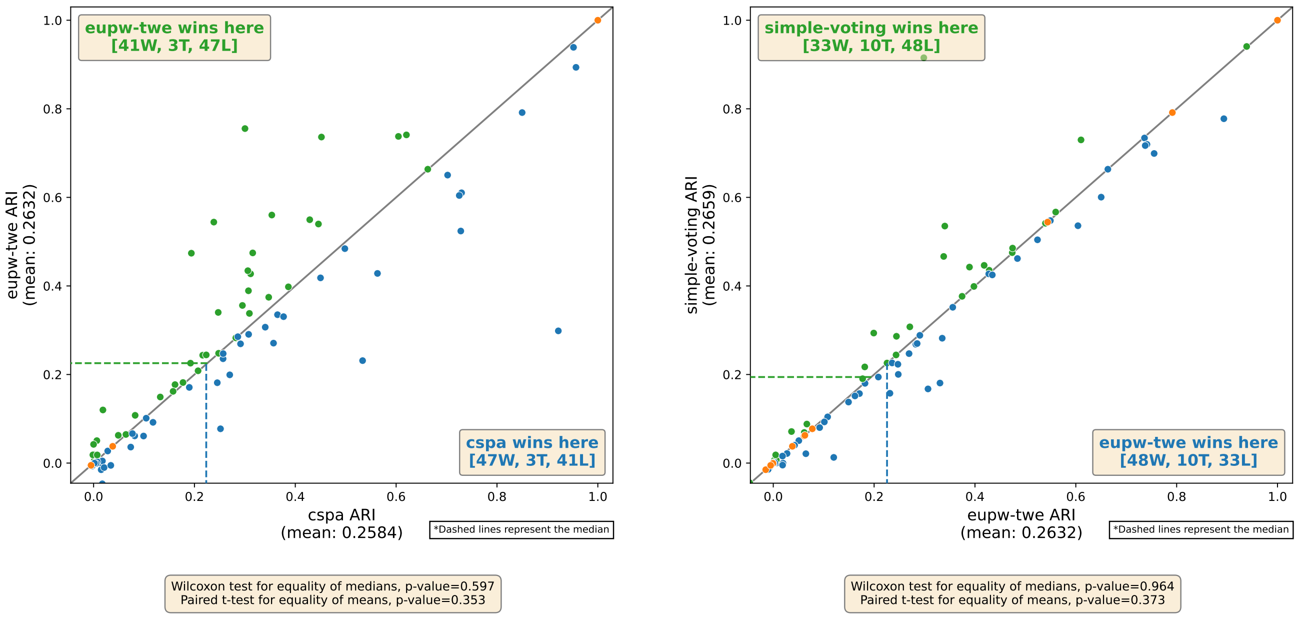
Table 9.5 shows the average score for each ensemble scheme. One interesting observation is that the SV scheme has the highest average score for ARI, AMI, CLACC, and NMI. However, in Figure 9.16, SV never surpasses PAM-TWE in terms of average rank, suggesting it is very inconsistent. To investigate this further, we directly compare EUPW-TWE and SV in Figure 9.17(b). The figure shows that EUPW-TWE consistently outperforms SV, and EUPW-TWE's median is significantly higher, indicating much greater consistency. However, SV for some datasets considerably outperforms EUPW-TWE.

CSPA outperformed EUPW-TWE for ARI and CLACC, as shown in Figure 9.16. Figure 9.17(a) highlights some interesting differences between the two. Notably,

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| cspa | 0.258 | 0.283 | 0.598 | 0.289 | **0.689** |
| eupw-euclidean | 0.245 | 0.276 | 0.586 | 0.282 | 0.675 |
| eupw-msm | 0.263 | 0.291 | 0.600 | 0.297 | 0.684 |
| eupw-twe | 0.263 | 0.291 | 0.600 | 0.297 | 0.684 |
| hbgf | 0.249 | 0.275 | 0.595 | 0.281 | 0.685 |
| iterative-voting | 0.253 | 0.282 | 0.593 | 0.288 | 0.684 |
| mcla | 0.258 | 0.288 | 0.594 | 0.294 | 0.683 |
| nmf | 0.251 | 0.284 | 0.594 | 0.290 | 0.682 |
| pam-msm | 0.255 | 0.284 | 0.593 | 0.290 | 0.684 |
| pam-twe | 0.263 | 0.291 | 0.600 | 0.297 | 0.687 |
| simple-voting | **0.266** | **0.292** | **0.600** | **0.298** | 0.684 |

Table 9.5 Summary of average score across multiple evaluation metrics over 91 datasets from the UCR archive using the combine test-train split.

CSPA performs better than EUPW-TWE on six additional datasets, but CSPA's average ARI score and median is lower than EUPW-TWE. This suggests that while EUPW-TWE may not achieve the best score for every datasets, on average is performs much more consistently.



(a) EUPW-TWE compared to CSPA (b) EUPW-TWE compared to Simple-Voting

Fig. 9.17 EUPW-TWE results compared directly to CSPA and Simple-Voting, respectively, over 91 datasets from the UCR archive using the combined test-train split.

Table 9.6 shows the average ARI score for each clusterer across different time series domains. EUPW-MSM performs best in the Motion domain, CSPA performs best in the Spectro and Sensor domain, and SV performs best in the Simulated domain. However, PAM-MSM performs best in the Image and ECG domains and PAM-TWE performs best in the Device domain. This shows no ensemble outperforms the best PAM variant in all time series domain.

Furthermore, Table 9.7 shows the average ARI rank for each clusterer across each time series domain. CSPA performs best in the most domains in terms of average rank, but it also has a wide range between its best and worst rank performance. CSPA's worst rank performance is 7.271 for the Image domain. Additionally CSPA ranks seventh or higher in three domains: Image, Device, and Motion. In contrast, EUPW-TWE's highest rank is 6.417 for the Image domain, and it only ranks sixth or higher in two domains: Image and Device. This indicates that while EUPW-TWE may not always achieve the top rank, it remains one of the most consistently performing clusterers across all domains.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| cspa | 0.270 | **0.204** | **0.276** | 0.523 | 0.133 | 0.147 | 0.317 |
| eupw-euclidean | 0.333 | 0.148 | 0.260 | 0.415 | 0.088 | 0.143 | 0.184 |
| eupw-msm | 0.321 | 0.165 | 0.258 | 0.472 | 0.166 | **0.173** | 0.271 |
| eupw-twe | 0.326 | 0.166 | 0.261 | 0.464 | 0.149 | 0.172 | 0.280 |
| hbgf | 0.264 | 0.178 | 0.275 | 0.524 | 0.154 | 0.140 | 0.199 |
| iterative-voting | 0.305 | 0.170 | 0.268 | 0.407 | 0.144 | 0.161 | 0.294 |
| mcla | 0.320 | 0.159 | 0.253 | 0.473 | 0.143 | 0.169 | 0.262 |
| nmf | 0.289 | 0.173 | 0.255 | 0.468 | 0.151 | 0.158 | 0.293 |
| pam-msm | **0.342** | 0.132 | 0.246 | 0.391 | 0.172 | 0.150 | **0.356** |
| pam-twe | 0.332 | 0.146 | 0.252 | 0.475 | **0.181** | 0.149 | 0.351 |
| simple-voting | 0.335 | 0.157 | 0.249 | **0.524** | 0.149 | 0.171 | 0.265 |

Table 9.6 Average ARI score on problems split by problem domain over 91 datasets from the UCR archive using the combine test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| cspa | 7.271 | **2.455** | **4.217** | **4.125** | 7.071 | 7.036 | 5.750 |
| eupw-euclidean | 5.208 | 7.773 | 6.478 | 8.250 | 9.571 | 4.536 | 6.750 |
| eupw-msm | 6.062 | 6.136 | 6.391 | 6.438 | 5.500 | **4.464** | 6.625 |
| eupw-twe | 6.417 | 5.636 | 5.630 | 5.625 | 6.286 | 4.750 | 5.875 |
| hbgf | 6.979 | 3.273 | 4.565 | **4.125** | 5.429 | 6.321 | 9.750 |
| iterative-voting | 5.146 | 6.818 | 5.913 | 7.062 | 7.071 | 6.857 | 5.375 |
| mcla | 5.917 | 6.591 | 7.196 | 7.000 | 6.000 | 6.214 | 7.750 |
| nmf | 6.292 | 4.545 | 6.457 | 5.188 | 5.786 | 6.929 | 5.125 |
| pam-msm | **4.875** | 7.636 | 5.783 | 7.250 | 3.571 | 7.143 | 5.000 |
| pam-twe | 6.333 | 7.636 | 5.717 | 4.625 | **3.429** | 6.643 | **2.250** |
| simple-voting | 5.500 | 7.500 | 7.652 | 6.312 | 6.286 | 5.107 | 5.750 |

Table 9.7 Average ARI rank performance on problems split by problem domain over 91 datasets from the UCR archive using the combine test-train split.

### 9.6.2 Test-Train Split

Figure 9.6 presents the CD diagrams for EUPW compared to other ensemble schemes for the test-train split. In our previous evaluation, we found EUPW is not well-suited for the test-train split. However, this appears to be the case for all ensemble schemes. Across all evaluation metrics, PAM-TWE consistently outperforms each ensemble scheme. This suggests that none of the ensemble methods can learn generalised representations of the data that translate effectively for the test-train split.

Although EUPW is outperformed by PAM-TWE across all metrics, for AMI and NMI, EUPW-MSM ranks as the second-best ensemble scheme, only behind SV. For ARI, EUPW performs worse than three ensemble schemes: NMF, SV, and IV. Interestingly, for ARI, EUPW-Euclidean outperforms both EUPW-MSM and EUPW-TWE. In fact, EUPW-TWE ranks as the worst-performing ensemble scheme for ARI, which is surprising given that it was the most consistent performer for the combined test-train split. Lastly, for CLACC, EUPW is outperformed by two ensemble schemes: IV and MCLA. Notably, both EUPW-MSM and EUPW-TWE outperform EUPW-Euclidean for CLACC.

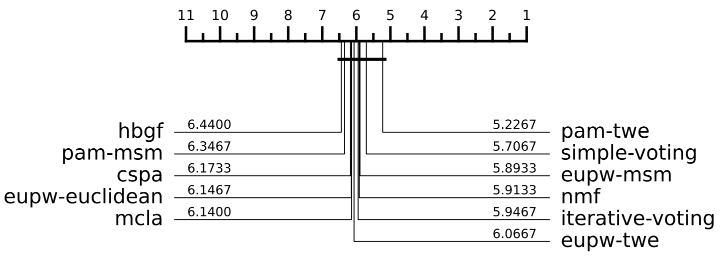


Fig. 9.18 AMI

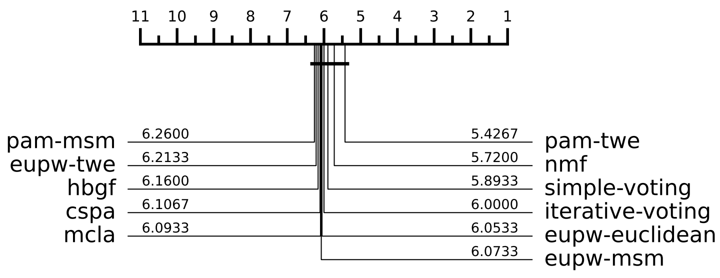


Fig. 9.19 ARI

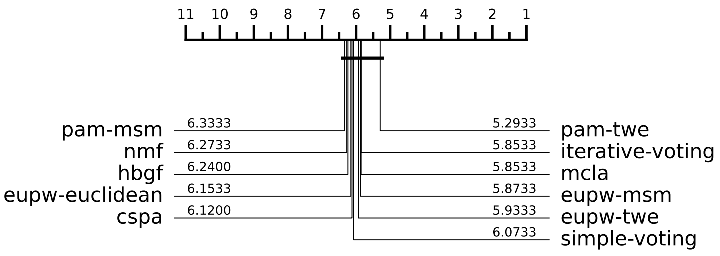


Fig. 9.20 CLACC

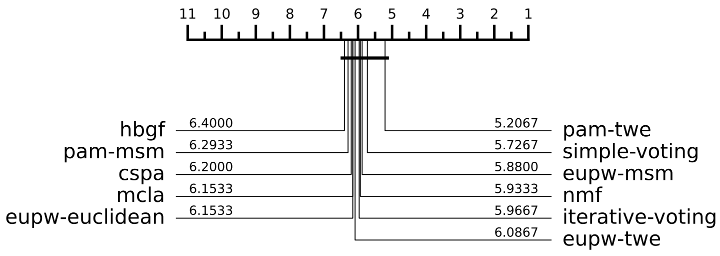


Fig. 9.21 NMI

Fig. 9.22 CD diagrams of EUPW compared to other ensemble schemes over 75 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.47

When analysing the results by time series domain for ARI in Table 9.8, we observe that for the Spectro, Simulated, Motion, and ECG domains, the ensemble schemes perform significantly better than PAM-TWE and PAM-MSM. For instance, in the Spectro domain, PAM-MSM averages a rank of 7.688 and PAM-TWE averages a rank of 7.062, whereas the best-performing ensemble scheme, CSPA, averages a rank of 3.312. A similar trend is evident in the Simulated and ECG domains.

It is also noteworthy that for the ECG domain, CSPA and HBGF achieve average ranks of over 10.00. If CSPA had not performed so poorly for ECG data, it might have outperformed both PAM-TWE and PAM-MSM overall. However, EUPW-MSM has a significantly lower rank than any other ensemble scheme in the ECG domain, demonstrating a particular strength of the EUPW scheme.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| cspa | 6.875 | **3.312** | 6.125 | 4.500 | 6.583 | 6.600 | 10.000 |
| eupw-euclidean | 5.800 | 7.125 | 6.175 | 6.625 | 7.667 | **4.400** | 4.833 |
| eupw-msm | 6.350 | 6.688 | 6.550 | 5.375 | 7.583 | 4.550 | **3.333** |
| eupw-twe | 6.025 | 6.312 | 7.300 | 5.375 | 6.250 | 5.450 | 4.667 |
| hbgf | 7.425 | 3.812 | 5.225 | 4.500 | 5.417 | 7.900 | 10.333 |
| iterative-voting | 6.050 | 7.250 | 6.525 | **4.188** | 7.250 | 4.750 | 5.333 |
| mcla | 5.500 | 6.125 | 6.450 | 7.500 | 5.917 | 6.100 | 4.167 |
| nmf | 6.275 | 3.938 | 5.750 | 5.312 | 4.833 | 6.600 | 6.500 |
| pam-msm | **4.675** | 7.688 | 5.300 | 9.375 | 4.833 | 8.350 | 7.000 |
| pam-twe | 5.200 | 7.062 | **4.550** | 7.438 | **4.000** | 5.600 | 5.333 |
| simple-voting | 5.825 | 6.688 | 6.050 | 5.812 | 5.667 | 5.700 | 4.500 |

Table 9.8 Average ARI rank performance on problems split by problem domain over 75 datasets from the UCR archive using the test-train split.

### 9.6.3 Conclusion: EUPW Compared to Other Ensemble Schemes

We have presented results for six different ensemble schemes compared to EUPW and the two best-performing PAM variants. For the combined test-train split, EUPW-TWE was the only ensemble scheme to consistently outperform PAM-TWE across all evaluation metrics. However, CSPA and HBGF were also identified as strong alternative ensemble schemes. While EUPW-TWE was outperformed by CSPA on average for ARI and CLACC, EUPW-TWE demonstrated higher consistency in performance across different time series domains.

Our experiments for the test-train split highlighted that not only is EUPW unsuitable for use on the test-train split, but none of the tested ensemble schemes are suitable. PAM-TWE consistently outperformed all ensemble schemes across all evaluation metrics. However, EUPW remained one of the more reliable ensemble schemes, particularly excelling in domains such as ECG, where other ensemble methods failed to perform well.

Overall, EUPW stands out as a strong and consistent ensemble scheme. For the combined test-train split, it is the only ensemble scheme to outperform the best-performing PAM variant across all evaluation metrics. However, it is evident that not just EUPW, but all ensemble schemes require further refinement to improve their performance on the test-train split.

## 9.7 EUPW with Other Unsupervised Evaluation Metrics

For our evaluation, we initially chose to use the ECHI evaluation metric with EUPW. We have demonstrated that EUPW performs better with an elastic version of CHI (ECHI) than with the traditional CHI. However, as previously mentioned, EUPW can be used with any unsupervised evaluation metric. To further explore how the choice of evaluation metric impacts EUPW's performance, we conduct an experiment using the Elastic Davies-Bouldin Index (EDBI) with EUPW (EUPW-EDBI) and compare the performance to EUPW with ECHI (EUPW-ECHI).

### 9.7.1 Combined Test-Train Split

Figure 9.27 shows the CD diagrams for EUPW using ECHI compared to EUPW using EDBI. Across all evaluation metrics, EUPW-EDBI-TWE, EUPW-EDBI-MSM, and EUPW-ECHI-TWE outperform the best-performing PAM variants. For AMI and NMI, EUPW-EDBI outperforms the best EUPW-ECHI ensemble. However, for CLACC and ARI, the best EUPW-ECHI outperforms the best EUPW-EDBI. Overall, the difference in rank between these methods is minimal.

Table 9.9 shows the average scores for each clusterer across all evaluation metrics. EUPW-ECHI achieves higher average scores than EUPW-EDBI for ARI, AMI, CLACC, and NMI, but both are outperformed in RI by PAM-TWE. We also compared EUPW-ECHI and EUPW-EDBI directly for both TWE and MSM, as shown in Figure 9.28. For TWE, both ensemble schemes perform similarly, with 35 ties. However, the median score of EUPW-ECHI-TWE is notably higher

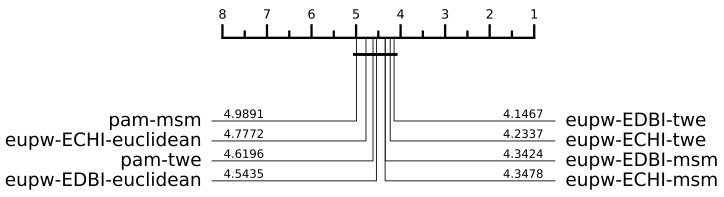


Fig. 9.23 AMI

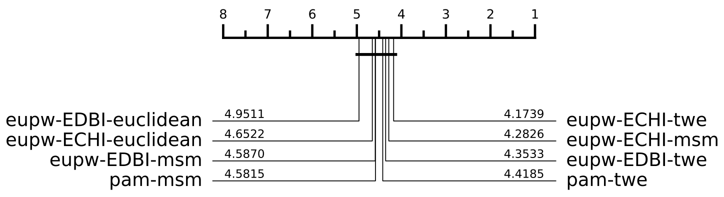


Fig. 9.24 ARI

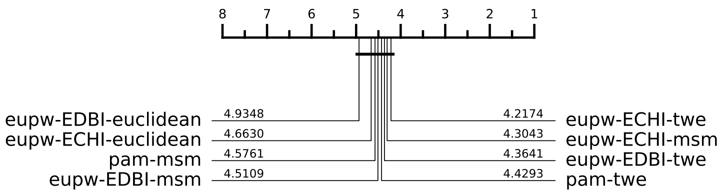


Fig. 9.25 CLACC

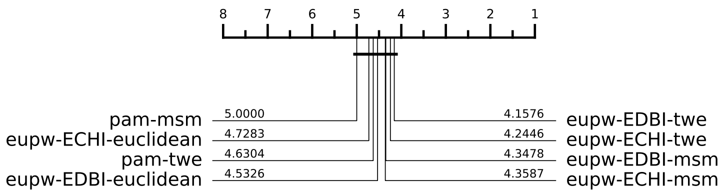


Fig. 9.26 NMI

Fig. 9.27 CD diagrams of EUPW using ECHI and EDBI compared over 92 datasets from the UCR archive using the combine test-train split. Missing datasets are outlined in Table A.48.

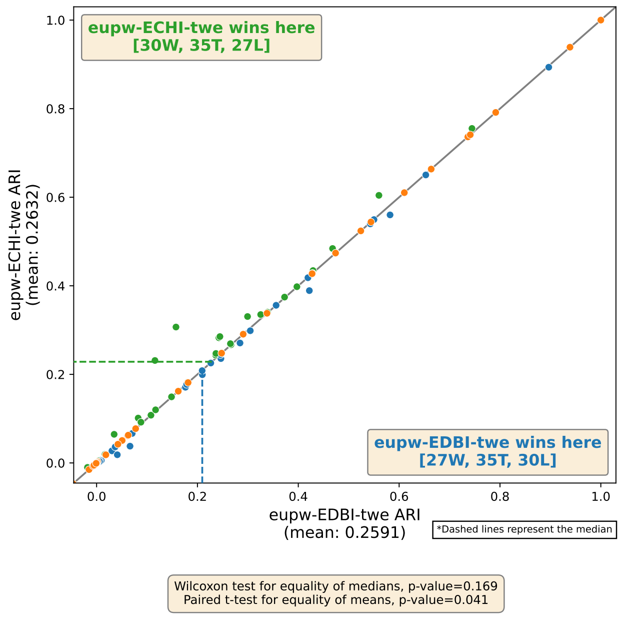
than EUPW-EDBI-TWE, suggesting that EUPW-ECHI has much more consistent performance than EUPW-EDBI on some datasets. A similar trend is observed for MSM, where EUPW-ECHI-MSM has a higher median and achieves 15 more wins than EUPW-EDBI-MSM.

We also compared EUPW-EDBI to six other cluster ensemble schemes. Figure 9.33 presents the CD diagram for this comparison. Unlike EUPW-ECHI, EUPW-EDBI does not suffer as much from issues related to empty clusters, allowing it to produce results for a larger number of datasets. However, some datasets remain missing, particularly for the SV scheme, which likely experiences the same issue due to using the same label alignment algorithm. Consequently, 105 datasets were included in this analysis.

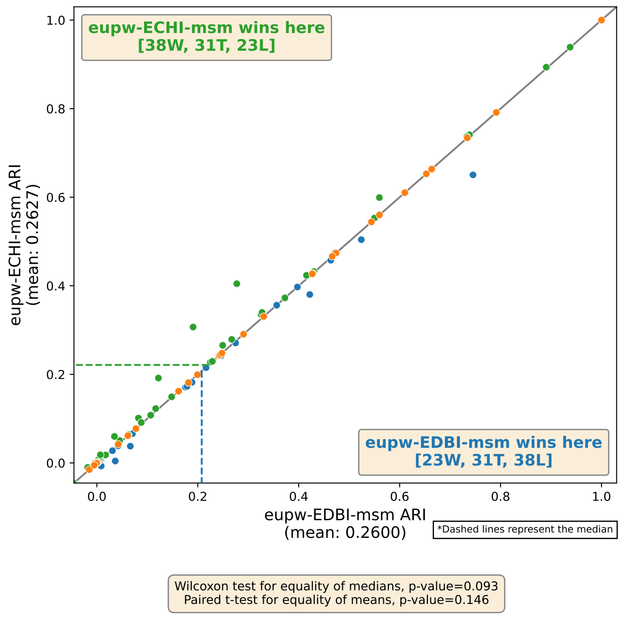
The CD diagram in Figure 9.33 reveals a similar rank order for the ensemble schemes. For AMI and NMI, EUPW-EDBI-TWE is the only ensemble scheme to outperform PAM-TWE. However, for CLACC and ARI, both PAM-TWE and PAM-

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| eupw-ECHI-euclidean | 0.244 | 0.276 | 0.584 | 0.283 | 0.676 |
| eupw-ECHI-msm | 0.263 | **0.291** | 0.598 | **0.298** | 0.685 |
| eupw-ECHI-twe | **0.263** | 0.291 | **0.599** | 0.298 | 0.685 |
| eupw-EDBI-euclidean | 0.248 | 0.281 | 0.588 | 0.287 | 0.677 |
| eupw-EDBI-msm | 0.260 | 0.289 | 0.597 | 0.296 | 0.683 |
| eupw-EDBI-twe | 0.259 | 0.289 | 0.597 | 0.295 | 0.682 |
| pam-msm | 0.253 | 0.284 | 0.590 | 0.291 | 0.684 |
| pam-twe | 0.262 | 0.291 | 0.597 | 0.298 | **0.688** |

Table 9.9 Summary of average score across multiple evaluation metrics over 92 datasets from the UCR archive using the combined test-train split.



(a) EUPW-ECHI-TWE compared to EUPW-EDBI-TWE



(b) EUPW-ECHI-MSM compared to EUPW-EDBI-MSM

Fig. 9.28 Comparison of EUPW-ECHI and EUPW-EDBI over 92 datasets from the UCR archive using the combined test-train split.

MSM outperform all other cluster ensemble schemes, a contrast to the previous experiment where CSPA and HBGF outperformed the best PAM variants for these metrics.

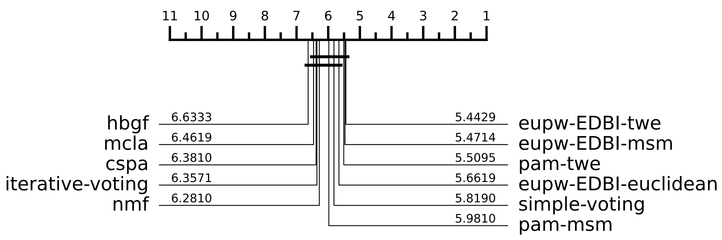


Fig. 9.29 AMI

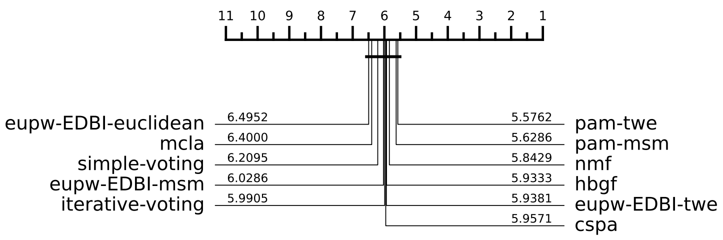


Fig. 9.30 ARI

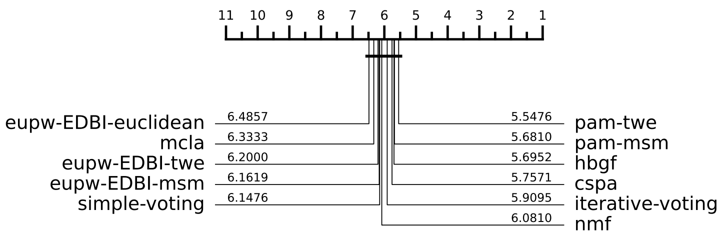


Fig. 9.31 CLACC

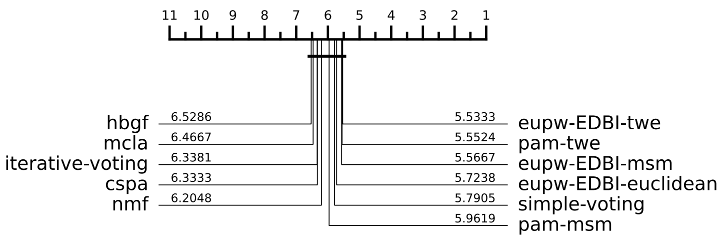


Fig. 9.32 NMI

Fig. 9.33 CD diagrams of EUPW-EDBI compared to other ensemble schemes over 105 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.50.

### 9.7.2 Test-Train Split

Figure 9.38 presents the CD for EUPW-EDBI over the test-train split. Neither EUPW-EDBI nor EUPW-ECHI outperform the best-performing PAM variant. However, EUPW-EDBI-TWE consistently outperforms the best-performing EUPW-ECHI, suggesting that it may be better suited for the test-train split. Still, none of the EUPW variants show statistically significant differences in performance.

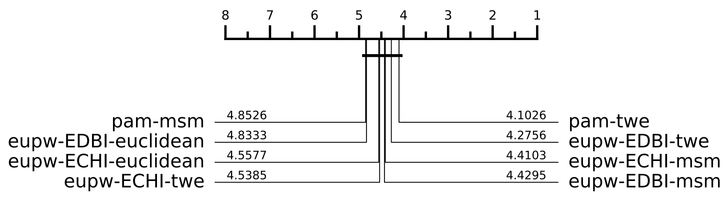


Fig. 9.34 AMI

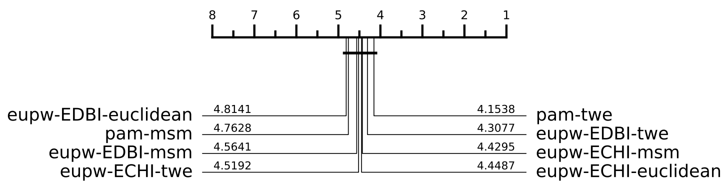


Fig. 9.35 ARI

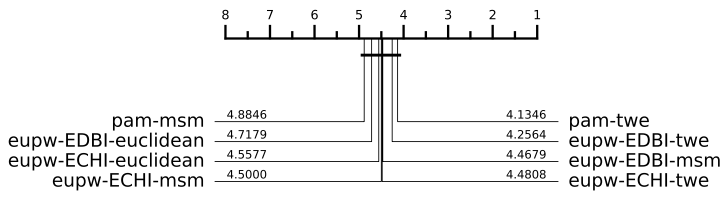


Fig. 9.36 CLACC

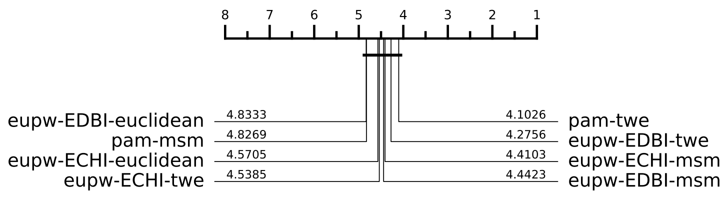


Fig. 9.37 NMI

Fig. 9.38 CD diagrams of EUPW using ECHi and EDBI compared over 78 datasets from the UCR archive using the test train split. Missing datasets are outlined in Table A.49.

### 9.7.3 Conclusion: EUPW with Other Unsupervised Evaluation Metrics

We have evaluated EUPW using two different unsupervised elastic evaluation metrics: ECHI and EDBI. We found that their performance is similar across both the combined test-train split and the test-train split. However, we showed that EUPW-ECHI has more consistent performance than EUPW-EDBI for the combined test-train split. For this reason, we recommend using ECHI as the default unsupervised metric for the final EUPW scheme.

## 9.8 The Elastic Clustering Ensemble (ECE)

We have identified a set of base clusterers: PAM-DTW, PAM-MSM, PAM-TWE, PAM-ERP, PAM-WDTW, PAM-ADTW, PAM-WDDTW, and PAM-soft-DTW, along with an ensemble scheme, EUPW. When combined, we have shown the ensemble performs better than any individual PAM variant. Specifically, from our previous evaluations, we identified the EUPW-TWE ensemble scheme as the most consistently performing scheme among the 12 different ensemble schemes we considered.

Therefore, we now formally define a new ensemble clusterer that utilises the eight base PAM clusterers and the EUPW-TWE ensemble scheme, which we call the Elastic Clustering Ensemble (ECE).

We now present the results for ECE compared to the state-of-the-art clusterers we have identified throughout this thesis.

### 9.8.1 Combined Test-Train Split

Figure 9.43 presents the CD diagrams comparing ECE to the state-of-the-art TSCL approaches and the baseline clusterers. For all evaluation metrics, ECE outperforms the other clusterers considered. However, for each evaluation metric, ECE remains in the same top clique as *k*-means-ba-TWE, PAM-MSM, *k*-means-ba-DTW, *k*-means-ba-MSM, and PAM-TWE. This improvement in performance over the Elastic Barycentre Average clusterers and the PAM clusterers highlights the success of the ensemble scheme.

Table 9.10 presents the average scores for each evaluation metric for each clusterer. ECE performs best in ARI and CLACC, while PAM-TWE performs best in AMI, NMI, and RI. As discussed previously, although PAM-TWE achieves high average scores, it has lower median performance, making it far less consistent than ECE. This is further highlighted in Table 9.11, which shows the average ARI rank by problem domain. Although the ECE clusterer ranks highest in only one domain (Motion), it consistently ranks within the top five across all domains.

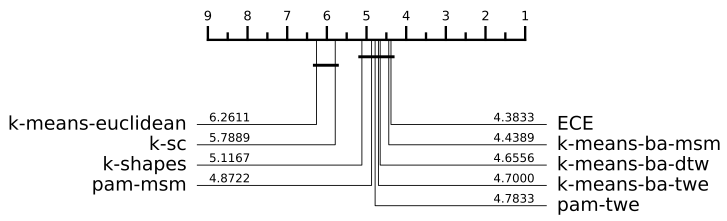


Fig. 9.39 AMI

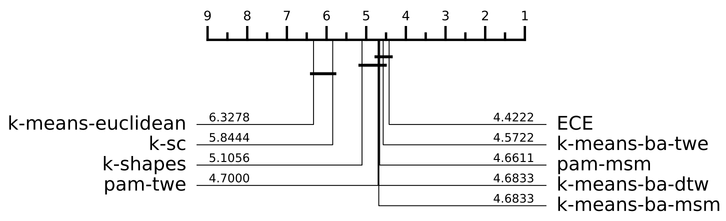


Fig. 9.40 ARI

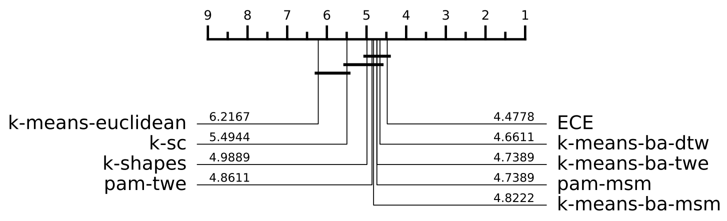


Fig. 9.41 CLACC

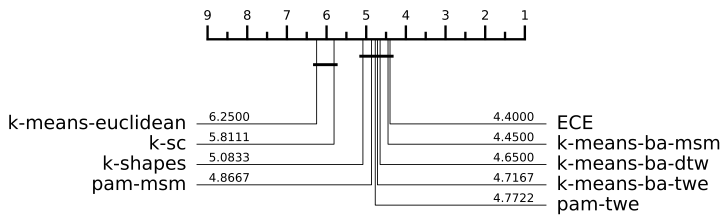


Fig. 9.42 NMI

Fig. 9.43 CD diagrams of ECE compared to the state-of-the-art clusterers and the baseline clusterers over 90 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.44.

In Chapter 4, we identified *k*-means-soft-DBA as the best-performing TSCL approach by a significant margin. However, due to its very high computational runtime, we were unable to obtain a complete set of results, with 27 datasets failing to finish within our seven-day runtime limit. To ensure a comprehensive evaluation, we first compared ECE to other clusterers over as many datasets as possible, excluding *k*-means-soft-DBA. Now, we reintroduce *k*-means-soft-DBA into the analysis, reducing the number of datasets included in the comparison.

Figure 9.48 shows the CD diagram comparing ECE to state-of-the-art clusterers, including *k*-means-soft-DBA. For all evaluation metrics, *k*-means-soft-DBA remains the best-performing clusterer. However, ECE shows no statistically significant difference in performance across any of the evaluation metrics.

Finally, we conduct an evaluation to verify ECE outperforms each of the individual PAM clusterers that comprise it, as shown in Figure 9.53. For AMI, ARI, and

| | ARI | AMI | CLAcc | NMI | RI |
|---|---|---|---|---|---|
| ECE | **0.2589** | 0.286 | **0.5985** | 0.292 | 0.681 |
| k-means-ba-dtw | 0.246 | 0.277 | 0.591 | 0.284 | 0.678 |
| k-means-ba-msm | 0.235 | 0.273 | 0.581 | 0.280 | 0.673 |
| k-means-ba-twe | 0.255 | 0.285 | 0.593 | 0.292 | 0.681 |
| k-means-euclidean | 0.186 | 0.222 | 0.535 | 0.229 | 0.655 |
| k-sc | 0.196 | 0.224 | 0.552 | 0.231 | 0.622 |
| k-shapes | 0.229 | 0.268 | 0.579 | 0.274 | 0.673 |
| pam-msm | 0.249 | 0.279 | 0.590 | 0.286 | 0.680 |
| pam-twe | 0.2587 | **0.287** | 0.5984 | **0.293** | **0.685** |

Table 9.10 Summary of average score across multiple evaluation metrics over 90 datasets from the UCR archive using the combined test-train split.

| | Image | Spectro | Sensor | Simulated | Device | Motion | ECG |
|---|---|---|---|---|---|---|---|
| ECE | 4.587 | 5.136 | 4.522 | 3.500 | 4.750 | **3.769** | 4.250 |
| k-means-ba-dtw | 4.891 | **3.455** | 5.935 | **2.625** | **2.500** | 4.885 | 7.500 |
| k-means-ba-msm | 4.543 | 5.000 | **4.370** | 5.125 | 5.000 | 4.577 | 5.250 |
| k-means-ba-twe | **4.022** | 5.591 | 4.609 | 5.000 | 4.375 | 4.923 | 3.125 |
| k-means-euclidean | 6.891 | 4.409 | 5.696 | 7.562 | 7.625 | 6.000 | 8.000 |
| k-sc | 6.283 | 3.955 | 5.174 | 7.938 | 7.500 | 5.769 | 5.125 |
| k-shapes | 4.739 | 4.636 | 5.348 | 5.125 | 6.250 | 5.077 | 4.875 |
| pam-msm | 4.065 | 6.227 | 4.500 | 5.125 | 3.875 | 5.077 | 4.000 |
| pam-twe | 4.978 | 6.591 | 4.848 | 3.000 | 3.125 | 4.923 | **2.875** |

Table 9.11 Average ARI rank performance on problems split by problem domain over 90 datasets from the UCR archive using the cobined test-train split.

NMI, ECE achieves a lower rank than the next best-performing PAM variant. However, for CLACC, PAM-soft-DTW slightly outperforms ECE, indicating a potential weakness in the ensemble. Overall, Figure 9.53 demonstrates that ECE improves upon PAM clustering, although the improvement is not statistically significant.

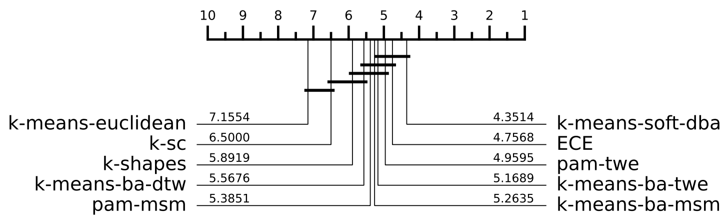


Fig. 9.44 AMI

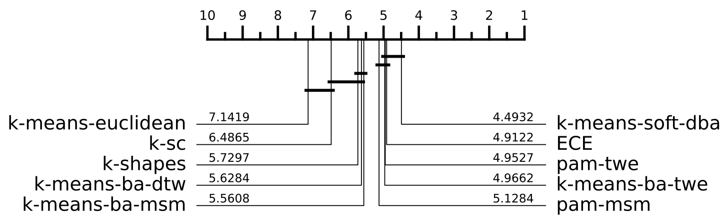


Fig. 9.45 ARI

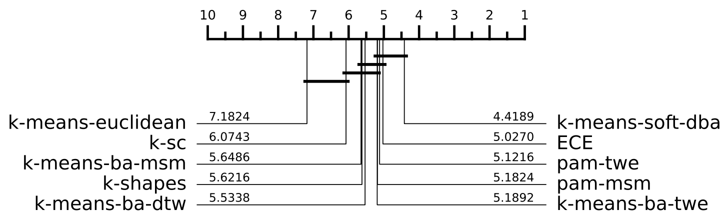


Fig. 9.46 CLACC

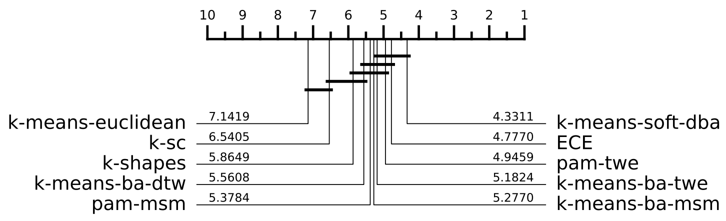


Fig. 9.47 NMI

Fig. 9.48 CD diagrams of ECE compared to the baseline clusterers, state-of-the-art and soft-DBA over 74 datasets from the UCR archive using the combine test train split. Missing datasets are outlined in Table A.51.

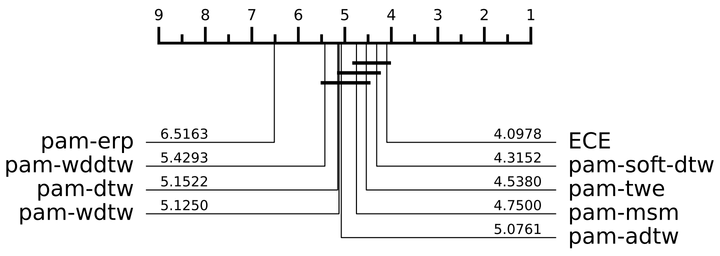


Fig. 9.49 AMI

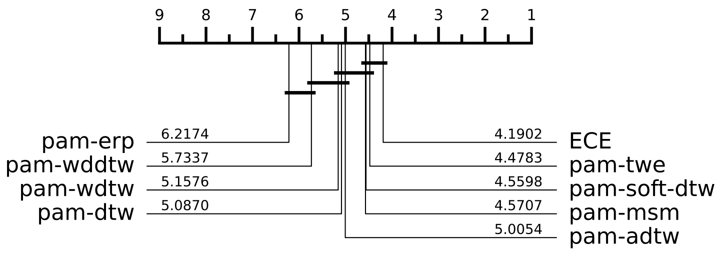


Fig. 9.50 ARI

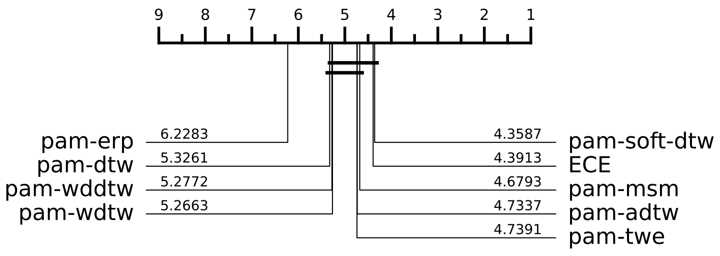


Fig. 9.51 CLACC

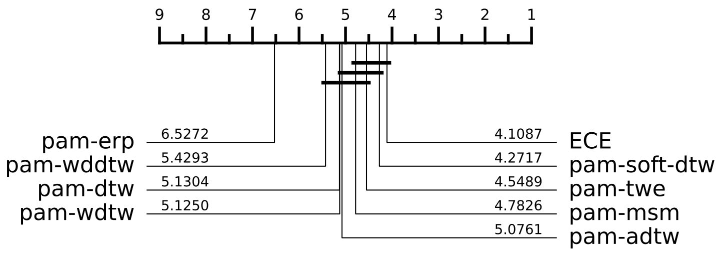


Fig. 9.52 NMI

Fig. 9.53 CD diagrams of ECE compared to each PAM clusterer over 92 datasets from the UCR archive using the combined test-train split. Missing datasets are outlined in Table A.48.

### 9.8.2 Test-Train Split

Figure 9.58 presents the CD diagrams comparing the ECE clusterer to the baseline and state-of-the-art clusterers for the test-train split. As previously noted, none of the cluster ensemble schemes performed well on the test-train split, including EUPW-ECHI. These findings remain true when comparing ECE to the state-of-the-art: ECE is consistently outperformed by both PAM-TWE and PAM-MSM, though it consistently appears in the top clique. Overall, for the test-train split, we recommend using PAM-TWE or PAM-MSM directly, as these individual components outperform the ensemble version.

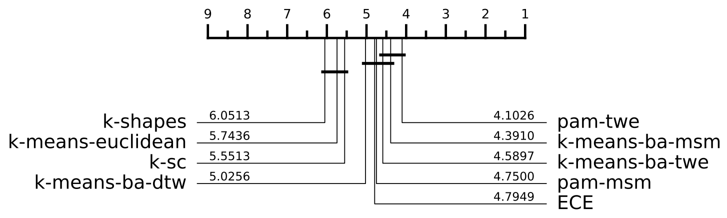


Fig. 9.54 AMI

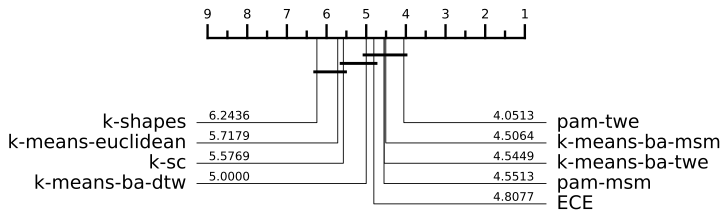


Fig. 9.55 ARI

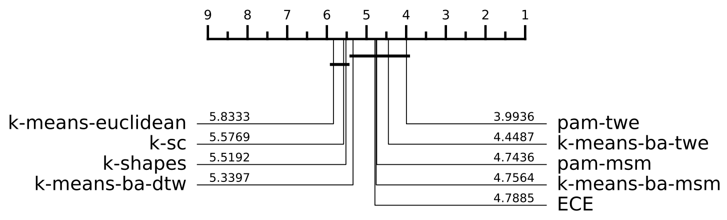


Fig. 9.56 CLACC

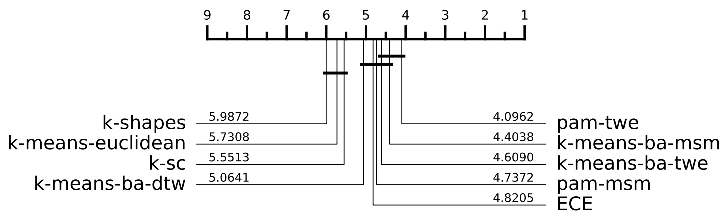


Fig. 9.57 NMI

Fig. 9.58 CD diagrams of ECE compared to the state-of-the-art clusterers and the baseline clusterers over 78 datasets from the UCR archive using the test-train split. Missing datasets are outlined in Table A.45.

## 9.9 Conclusion

In this chapter, we introduced the ECE clusterer, which utilises eight elastic distance PAM models outlined in Chapter 6. To ensemble these PAM models, we proposed the Elastic Unsupervised Proportional Weighting (EUPW) ensemble scheme, using elastic, unsupervised evaluation metrics to achieve better and more consistent ensemble predictions than six other considered ensemble schemes from the literature.

We have demonstrated that the ECE clusterer outperforms the best PAM variants that compose it and also outperforms the Elastic Barycentre Average clusterers. While ECE does not surpass $k$-means-soft-DBA in any of our considered evaluation metric, it further narrows the performance gap, making it one of the top-performing clustering methods identified in this thesis.

# Chapter 10

# Conclusion

In this thesis, we have conducted the most comprehensive review of elastic distances for TSCL to date, examining 12 different elastic distances, nine of which had not previously been explored for TSCL. We proposed new methods that advance the state-of-the-art in TSCL performance. Our key contributions include the development of a robust, standardised Lloyd's-based clustering model, a novel Elastic Barycentre Averaging technique, KESBA—a state-of-the-art, versatile, and highly scalable clusterer designed for real-world TSCL applications—and a new ensemble scheme (EUPW) to create the ECE, a state-of-the-art PAM-based ensemble clusterer. The work presented in this thesis establishes a new benchmark for TSCL research, offering scalable and highly effective approaches to clustering time series data.

## 10.1 Discussion of Contributions

We began by addressing the inconsistent configurations of Lloyd's-based clustering algorithms in the TSCL literature. By standardising Lloyd's algorithm, we established a unified framework for comparing five popular Lloyd's-based TSCL algorithms, which serve as our baseline of comparison. This standardisation enables us to more accurately attribute differences in clustering performance to specific model enhancements, rather than variations in Lloyd's configuration.

Using this standardised Lloyd's model, we conducted a comprehensive evaluation of 12 elastic distances with the *k*-means clusterer. Our findings revealed that several elastic distances, along with the Euclidean distance, outperform Dynamic Time Warping (DTW), which has long been considered the state-of-the-art. We further investigated the causes of DTW's poor performance, identifying pathological warping—exacerbated by the use of the arithmetic mean—as the key issue. Additionally, our extensive analysis highlighted the traits of the best-performing elastic distances, revealing that those with an explicit warping penalty performed best. Notably, ADTW, MSM, shape-DTW, TWE, and soft-DTW achieved state-of-the-art results relative to the previously established baseline.

Building on our *k*-means evaluation, we applied the same methodology to four different *k*-medoids clusterers. We found that similar elastic distances performed best for *k*-medoids as they did for *k*-means. PAM emerged as the top-performing model, followed by CLARANS and alternate *k*-medoids, while CLARA lagged behind. Additionally, we conducted a detailed comparison between *k*-means and alternate *k*-medoids, showing that *k*-medoids outperformed *k*-means across most elastic distances. This analysis highlighted the critical role of the medoids computation in achieving superior clustering performance. Notably, PAM-ADTW, PAM-MSM, PAM-TWE, and PAM-soft-DTW surpassed the current state-of-the-art, establishing a new baseline for TSCL performance.

Our *k*-medoids experiments highlighted the importance of incorporating the elastic distances into the centroid computation. While the medoids significantly outperformed the arithmetic mean, we noted that for DTW, which has a bespoke averaging technique (DBA), the average performed better than the medoids. This observation led us to develop the Elastic Barycentre Average.

The Elastic Barycentre Average is a generalisation of DBA that can be applied to any elastic distance capable of computing a full alignment path through a cost matrix. We incorporated the Elastic Barycentre Average into eight different elastic distance *k*-means clusterers and observed substantial improvements in clustering performance across all distances compared to using the arithmetic mean or medoids. Notably, the best-performing elastic distances, such as MSM and TWE, when combined with the Elastic Barycentre Average, not only outperformed but significantly surpassed the existing state-of-the-art.

The state-of-the-art performance achieved by PAM-TWE, PAM-MSM, *k*-means-soft-DBA, *k*-means-ba-MSM, and *k*-means-ba-TWE comes at a substantial computational cost, making these algorithms impractical for many real-world datasets. To address this limitation, we developed KESBA—a versatile, highly scalable clustering algorithm tailored for large-scale datasets. KESBA provides practitioners with a flexible set of parameters to balance computational efficiency and clustering performance. At the heart of KESBA is the novel Random Subset Elastic Stochastic Subgradient Barycentre Average, an elastic distance averaging algorithm, along with several optimisations to our Lloyd's baseline configuration. We demonstrate that KESBA not only achieves state-of-the-art performance with significantly faster runtime but also offers practitioners the flexibility to adjust parameters based on their runtime requirements.

Finally, we introduced ECE, an ensemble of eight elastic PAM models developed using a novel Elastic Unsupervised Proportional Weighting ensemble scheme. The ECE outperformed six widely used ensemble schemes, all utilising the same eight PAM base clusterers. Moreover, ECE surpassed the performance of each individual PAM clusterer within the ensemble. ECE achieves state-of-the-art clustering performance and ranks among the best clustering models considered in this thesis.

We have made all the code for our clusterers and experiments available through the open-source projects *aeon* and *tsml-eval*. This allows practitioners to reproduce all the work presented in this thesis and further build upon and extend the algorithms proposed.

## 10.2 Future Work and Extensions

The potential applications of elastic distances in TSCL are vast, and in this thesis, we have primarily focused on partition-based TSCL algorithms. While we hypothesise that our findings will extend to other TSCL approaches, such as density-based and hierarchical-based algorithms, our future work will aim to benchmark these methods similarly to our evaluation of partition-based approaches. Our ultimate goal is to produce a comprehensive TSCL "bakeoff" paper, akin to the TSC bakeoff [8].

By introducing elastic distances beyond DTW and developing a generalised Elastic Barycentre Averaging function that can be applied to any elastic distance capable of computing a complete alignment path, we hope to encourage the adoption of superior elastic distances—such as MSM, TWE, soft-DTW, and shape-DTW—both in existing and new TSCL models to advance state-of-the-art methods. Additionally, for models where computational runtime is a concern, the Random Subset Elastic Stochastic Subgradient Barycentre Average can be utilised with any elastic distance, offering a flexible balance between computational efficiency and clustering performance.

We also aim to study the impact of combining various elastic distances with the Elastic Barycentre Average on existing unsupervised evaluation metrics, such as Davies-Bouldin, Silhouette Coefficient, and Calinski-Harabasz. These metrics all rely on a distance measure and an averaging technique. In Chapter 9, we indirectly demonstrated that using the elastic distance and Elastic Barycentre Average in unsupervised evaluation metrics significantly enhances their effectiveness. However, we plan to empirically validate these finding, as it could greatly improve real-world TSCL parameter tuning and evaluation.

Although KESBA is already a fast and versatile TSCL algorithm, we believe it can be made even faster. TWE, the best-performing elastic distance in our KESBA evaluation, satisfies the triangular inequality, which opens the possibility of using alternative *k*-means algorithms like Elkan's algorithm [28]. This approach could significantly reduce the number of distance computations required, thereby further improving KESBA's runtime.

Lastly, throughout this thesis, we noted the strong performance of soft-DBA. While soft-DTW already has a bespoke averaging technique, we experimented with soft-DTW using the Elastic Barycentre Average. Although *k*-means-ba-soft-DTW performed well, it did not surpass *k*-means-ba-MSM or *k*-means-ba-TWE. Therefore, we hypothesise that developing a "soft" version of TWE and MSM, to allow the development of Soft MSM and TWE Barycentre Averaging, could potentially outperform the *k*-means-soft-DBA clusterer.

# References

[1] Ahmed, M., Seraj, R., and Islam, S. M. S. (2020). The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 9(8).

[2] Almahamid, F. and Grolinger, K. (2022). Agglomerative hierarchical clustering with dynamic time warping for household load curve clustering.

[3] Alqahtani, A., Ali, M., Xie, X., and Jones, M. W. (2021). Deep time-series clustering: A review. *Electronics*, 10(23).

[4] Alqurashi, T. and Wang, W. (2019). Clustering ensemble method. *International Journal of Machine Learning and Cybernetics*, 10(6):1227–1246.

[5] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J. (1999). Optics: ordering points to identify the clustering structure. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, SIGMOD '99, page 49–60, New York, NY, USA. Association for Computing Machinery.

[6] Arthur, D. and Vassilvitskii, S. (2007). k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA. Society for Industrial and Applied Mathematics.

[7] Bagirov, A., Karmitsa, N., and Mkel, M. M. (2014). *Introduction to Nonsmooth Optimization: Theory, Practice and Software*. Springer Publishing Company, Incorporated.

[8] Bagnall, A., Lines, J., Bostrom, A., Large, J., and Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *DAMI*, 31(3):606–660.

[9] Begum, N., Ulanova, L., Dau, H. A., Wang, J., and Keogh, E. J. (2016). A general framework for density based time series clustering exploiting a novel admissible pruning strategy. *ArXiv*, abs/1612.00637.

[10] Bellman, R. and Kalaba, R. (1959). On adaptive control processes. *IRE Transactions on Automatic Control*, 4(2):1–9.

[11] Benavoli, A., Corani, G., and Mangili, F. (2016). Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, 17(5):1–10.

[12] Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, page 359–370. AAAI Press.

[13] Blázquez-García, A., Conde, A., Mori, U., and Lozano, J. A. (2021). A review on outlier/anomaly detection in time series data. *ACM Comput. Surv.*, 54(3).

[14] Bonner, R. E. (1964). On some clustering techniques. *IBM Journal of Research and Development*, 8(1):22–32.

[15] Caliński, T. and Harabasz, J. (1974). A dendrite method for cluster analysis. *Communications in Statistics*, 3(1):1–27.

[16] Campello, R. J. G. B., Moulavi, D., and Sander, J. (2013). Density-based clustering based on hierarchical density estimates. In Pei, J., Tseng, V. S., Cao, L., Motoda, H., and Xu, G., editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg. Springer Berlin Heidelberg.

[17] Celebi, M. E., Kingravi, H. A., and Vela, P. A. (2013). A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Systems with Applications*, 40(1):200–210.

[18] Chen, L. and Ng, R. (2004). On the marriage of lp-norms and edit distance. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, page 792–803. VLDB Endowment.

[19] Chen, L., Özsu, M. T., and Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, page 491–502, New York, NY, USA. Association for Computing Machinery.

[20] Comaniciu, D. and Meer, P. (2002). Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:603–619.

[21] Cuturi, M. and Blondel, M. (2017). Soft-dtw: a differentiable loss function for time-series. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 894–903. JMLR.org.

[22] Datta, S., Karmakar, C. K., and Palaniswami, M. (2020). Averaging methods using dynamic time warping for time series classification. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 2794–2798.

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

[24] Dau, H. A., Silva, D. F., Petitjean, F., Forestier, G., Bagnall, A., Mueen, A., and Keogh, E. (2018b). Optimizing dynamic time warping's window width for time series data mining applications. *Data Mining and Knowledge Discovery*, 32(4):1074–1120.

[25] Davies, D. L. and Bouldin, D. W. (1979). A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227.

[26] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30.

[27] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. (2008). Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552.

[28] Elkan, C. (2003). Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 147–153. AAAI Press.

[29] Ester, M., Kriegel, H.-P., Sander, J., and 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, page 226–231. AAAI Press.

[30] Esteves, R. M. and Rong, C. (2011). Using mahout for clustering wikipedia's latest articles: A comparison between k-means and fuzzy c-means in the cloud. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, page 565–569, USA. IEEE Computer Society.

[31] Estivill-Castro, V. (2002). Why so many clustering algorithms: A position paper. *SIGKDD Explor. Newsl.*, 4(1):65–75.

[32] Ezugwu, A. E., Ikotun, A. M., Oyelade, O. O., Abualigah, L., Agushaka, J. O., Eke, C. I., and Akinyelu, A. A. (2022). A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743.

[33] Fatima, S. S. W. and Rahimi, A. (2024). A review of time-series forecasting algorithms for industrial manufacturing systems. *Machines*, 12(6).

[34] Fern, X. Z. and Brodley, C. E. (2004). Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, ICML '04, page 36, New York, NY, USA. Association for Computing Machinery.

[35] Forgy, E. W. (1965). Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics*, 21:768–769.

[36] Fraley, C. and Raftery, A. E. (1998). How Many Clusters? Which Clustering Method? Answers Via Model-Based Cluster Analysis. *The Computer Journal*, 41(8):578–588.

[37] Fränti, P. and Sieranoja, S. (2019). How much can k-means be improved by using better initialization and repeats? *Pattern Recognition*, 93:95–112.

[38] Fréchet, M. (1948). Les éléments aléatoires de nature quelconque dans un espace distancié. *Annales de l'institut Henri Poincaré*, 10(4):215–310.

[39] García, S. and Herrera, F. (2008). An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9(89):2677–2694.

[40] Golub, G. H. and Van Loan, C. F. (2012). *Matrix Computations*, volume 3. JHU Press.

[41] Guha, S., Rastogi, R., and Shim, K. (1998). Cure: an efficient clustering algorithm for large databases. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, page 73–84, New York, NY, USA. Association for Computing Machinery.

[42] Gutiérrez, A. P., Delfa, J. L. V., and López, M. V. (2023). Time series clustering using trend, seasonal and autoregressive components to identify maximum temperature patterns in the iberian peninsula. *Environmental and Ecological Statistics*, 30(3):421–442.

[43] Hautamaki, V., Nykanen, P., and Franti, P. (2008). Time-series clustering by approximate prototypes. In *2008 19th International Conference on Pattern Recognition*, pages 1–4.

[44] Herrmann, M. and Webb, G. I. (2023). Amercing: An intuitive and effective constraint for dynamic time warping. *Pattern Recognition*, 137:109333.

[45] Hirschberg, D. S. (1977). Algorithms for the longest common subsequence problem. *J. ACM*, 24(4):664–675.

[46] Holder, C., Guijo-Rubio, D., and Bagnall, A. (2023). Clustering time series with k-medoids based algorithms. In Ifrim, G., Tavenard, R., Bagnall, A., Schaefer, P., Malinowski, S., Guyet, T., and Lemaire, V., editors, *Advanced Analytics and Learning on Temporal Data*, pages 39–55, Cham. Springer Nature Switzerland.

[47] Holder, C., Middlehurst, M., and Bagnall, A. (2024). A review and evaluation of elastic distance functions for time series clustering. *Knowledge and Information Systems*, 66(2):765–809.

[48] Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., and Heming, J. (2023). K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. *Information Sciences*, 622:178–210.

[49] Itakura, F. (1975). Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):67–72.

[50] Jain, A. K. and Dubes, R. C. (1988). *Algorithms for clustering data.* Prentice-Hall, Inc., USA.

[51] Javed, A., Lee, B. S., and Rizzo, D. M. (2020). A benchmark study on time series clustering. *Machine Learning with Applications*, 1:100001.

[52] Javed, A., Rizzo, D. M., Lee, B. S., and Gramling, R. (2024). Somtimes: self organizing maps for time series clustering and its application to serious illness conversations. *Data Mining and Knowledge Discovery*, 38(3):813–839.

[53] Jeong, Y., Jeong, M., and Omitaomu, O. (2011). Weighted dynamic time warping for time series classification. *Pattern Recognition*, 44:2231–2240.

[54] Jorge, M.-B. and Cuevas, R. (2024). Time series clustering with random convolutional kernels. *Data Mining and Knowledge Discovery*, 38(4):1862–1888.

[55] Jr., J. H. W. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301):236–244.

[56] Karypis, G., Han, E.-H., and Kumar, V. (1999). Chameleon: hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75.

[57] Karypis, G. and Kumar, V. (1998). Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129.

[58] Kaufman, L. and Rousseeuw, P. (1990). Finding groups in data: An introduction to cluster analysis. *Wiley, New York. ISBN 0-471-87876-6.*

[59] Keogh, E. and Kasetty, S. (2003). On the need for time series data mining benchmarks: A survey and empirical demonstration. *Data Mining and Knowledge Discovery*, 7(4):349–371.

[60] Keogh, E. J. and Pazzani, M. J. (2001). Derivative dynamic time warping. In *SDM*.

[61] Kobylin, O. and Lyashenko, V. (2020). Time series clustering based on the k-means algorithm. *Journal La Multiapp*, 1(3):1–7.

[62] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

[63] Kumar, U., Legendre, C. P., Lee, J.-C., Zhao, L., and Chao, B. F. (2022). On analyzing gnss displacement field variability of taiwan: Hierarchical agglomerative clustering based on dynamic time warping technique. *Computers & Geosciences*, 169:105243.

[64] Lafabregue, B., Weber, J., Gançarski, P., and Forestier, G. (2022). End-to-end deep representation learning for time series clustering: a comparative study. *Data Mining and Knowledge Discovery*, 36(1):29–81.

[65] Lam, D. and Wunsch, D. C. (2014). Chapter 20 - clustering. In Diniz, P. S., Suykens, J. A., Chellappa, R., and Theodoridis, S., editors, *Academic Press Library in Signal Processing: Volume 1*, volume 1 of *Academic Press Library in Signal Processing*, pages 1115–1149. Elsevier.

[66] Le Quy Nhon, V. and Anh, D. T. (2012). A birch-based clustering method for large time series databases. In Cao, L., Huang, J. Z., Bailey, J., Koh, Y. S., and Luo, J., editors, *New Frontiers in Applied Data Mining*, pages 148–159, Berlin, Heidelberg. Springer Berlin Heidelberg.

[67] Lee, C. and Van Der Schaar, M. (2020). Temporal phenotyping using deep predictive clustering of disease progression. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

[68] Leonard Kaufman, P. J. R. (1990a). *Clustering Large Applications (Program "CLARA")*, chapter 3, pages 126–163. John Wiley and Sons, Ltd.

[69] Leonard Kaufman, P. J. R. (1990b). *Partitioning Around Medoids (Program PAM)*, chapter 2, pages 68–125. John Wiley and Sons, Ltd.

[70] Li, T., Ding, C., and Jordan, M. I. (2007). Solving consensus and semi-supervised clustering problems using nonnegative matrix factorization. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 577–582.

[71] Li, X., Lin, J., and Zhao, L. (2019). Linear time complexity time series clustering with symbolic pattern forest. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2930–2936. International Joint Conferences on Artificial Intelligence Organization.

[72] Li, X., Xi, W., and Lin, J. (2024). Randomnet: clustering time series using untrained deep neural networks. *Data Mining and Knowledge Discovery*.

[73] Li, Z., Yang, Y., Liu, J., Zhou, X., and Lu, H. (2021). Unsupervised feature selection using nonnegative spectral analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 26(1):1026–1032.

[74] Lines, J. and Bagnall, A. (2015). Time series classification with ensembles of elastic distance measures. *"dami"*, 29:565–592.

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

[76] Lucas, B., Shifaz, A., Pelletier, C., O'neill, L., Zaidi, N., Goethals, B., Petitjean, F., and Webb, G. I. (2019). Proximity forest: an effective and

scalable distance-based classifier for time series. *Data Min. Knowl. Discov.*, 33(3):607–635.

[77] Ma, Q., Zheng, J., Li, S., and Cottrell, G. W. (2019). Learning representations for time series clustering. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[78] MacQueen, J. B. (1967). *Some methods for classification and analysis of multivariate observations*, pages ::::281–297. University of California Press.

[79] Marteau, P.-F. (2009). Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318.

[80] McDowell, I. C., Manandhar, D., Vockley, C. M., Schmid, A. K., Reddy, T. E., and Engelhardt, B. E. (2018). Clustering gene expression time series data using an infinite gaussian process mixture model. *PLOS Computational Biology*, 14(1):1–27.

[81] McLachlan, G. J. and Krishnan, T. (2007). *The EM algorithm and extensions*. John Wiley & Sons.

[82] Middlehurst, M., Ismail-Fawaz, A., Guillaume, A., Holder, C., Rubio, D. G., Bulatova, G., Tsaprounis, L., Mentel, L., Walter, M., Schäfer, P., and Bagnall, A. (2024a). aeon: a python toolkit for learning from time series.

[83] Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., and Bagnall, A. (2021). Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(11):3211–3243.

[84] Middlehurst, M., Schäfer, P., and Bagnall, A. (2024b). Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*.

[85] Mori, U., Mendiburu, A., and Lozano, J. A. (2016). Similarity measure selection for clustering time series databases. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):181–195.

[86] Ng, R. and Han, J. (2002). CLARANS: A method for clustering objects for spatial data mining. *Knowledge and Data Engineering, IEEE Transactions on*, 14:1003– 1016.

[87] Nguyen, N. and Caruana, R. (2007). Consensus clusterings. In *Seventh IEEE International Conference on Data Mining (ICDM 2007)*, pages 607–612.

[88] Niennattrakul, V. and Ratanamahatana, C. A. (2007). On clustering multimedia time series data using k-means and dynamic time warping. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*, pages 733–738.

[89] Paparrizos, J. and Gravano, L. (2016). k-shape: Efficient and accurate clustering of time series. *SIGMOD Rec.*, 45(1):69–76.

[90] Paparrizos, J. and Gravano, L. (2017). Fast and accurate time-series clustering. *ACM Trans. Database Syst.*, 42(2).

[91] Paparrizos, J. and Reddy, S. P. T. (2023). Odyssey: An engine enabling the time-series clustering journey. *Proc. VLDB Endow.*, 16(12):4066–4069.

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

[93] Petitjean, F., Inglada, J., and Gancarski, P. (2012). Satellite image time series analysis under time warping. *IEEE Transactions on Geoscience and Remote Sensing*, 50(8):3081–3095.

[94] Petitjean, F., Ketterlin, A., and Gançarski, P. (2011). A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693.

[95] Qian, M. and Zhai, C. (2013). Robust unsupervised feature selection. In *IJCAI 2013 - Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, IJCAI International Joint Conference on Artificial Intelligence, pages 1621–1627. Copyright: Copyright 2014 Elsevier B.V., All rights reserved.; 23rd International Joint Conference on Artificial Intelligence, IJCAI 2013 ; Conference date: 03-08-2013 Through 09-08-2013.

[96] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and 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).

[97] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850.

[98] Ransom, S. M., Eikenberry, S. S., and Middleditch, J. (2002). Fourier techniques for very long astrophysical time-series analysis. *The Astronomical Journal*, 124(3):1788.

[99] Ratanamahatana, C. A. and Keogh, E. (2004). *Making Time-series Classification More Accurate Using Learned Constraints*, pages 11–22.

[100] Rodriguez, A. and Laio, A. (2014). Clustering by fast search and find of density peaks. *Science*, 344(6191):1492–1496.

[101] Ruiz, L., Pegalajar, M., Arcucci, R., and Molina-Solana, M. (2020). A time-series clustering methodology for knowledge extraction in energy consumption data. *Expert Systems with Applications*, 160:113731.

[102] Räsänen, T. and Kolehmainen, M. (2009). Feature-based clustering for electricity use time series data. volume 5495, pages 401–412.

[103] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49.

[104] Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., Er, M. J., Ding, W., and Lin, C.-T. (2017). A review of clustering techniques and developments. *Neurocomputing*, 267:664–681.

[105] Schmidl, S., Wenig, P., and Papenbrock, T. (2022). Anomaly detection in time series: a comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797.

[106] Schubert, E. and Rousseeuw, P. J. (2019). Faster k-medoids clustering: Improving the pam, clara, and clarans algorithms. In Amato, G., Gennaro, C., Oria, V., and Radovanović, M., editors, *Similarity Search and Applications*, pages 171–187, Cham. Springer International Publishing.

[107] Schubert, E. and Rousseeuw, P. J. (2021). Fast and eager k-medoids clustering: O(k) runtime improvement of the pam, clara, and clarans algorithms. *Information Systems*, 101:101804.

[108] Schultz, D. and Jain, B. (2018). Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. *Pattern Recognition*, 74:340–358.

[109] Sezer, O. B., Gudelek, M. U., and Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning : A systematic literature review: 2005–2019. *Applied Soft Computing*, 90:106181.

[110] Shi, L., Du, L., and Shen, Y.-D. (2014). Robust spectral learning for unsupervised feature selection. In *2014 IEEE International Conference on Data Mining*, pages 977–982.

[111] Shifaz, A., Pelletier, C., Petitjean, F., and Webb, G. (2023). Elastic similarity and distance measures for multivariate time series. *Knowledge and Information Systems*, 65(6).

[112] Shiudkar, K. and Takmare, S. (2017). Review of existing methods in k-means clustering algorithm. *International Research Journal Engineering Technology*, 4(2):1213–1216.

[113] Silva, M. and Henriques, R. (2020). Exploring time-series motifs through dtw-som. pages 1–8.

[114] Stefan, A., Athitsos, V., and Das, G. (2013). The move-split-merge metric for time series. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1425–1438.

[115] Strehl, A. and Ghosh, J. (2003). Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3(null):583–617.

[116] Su, T. and Dy, J. (2004). A deterministic method for initializing k-means clustering. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 784–786.

[117] Tan, C. W., Bergmeir, C., Petitjean, F., and Webb, G. I. (2021). Time series extrinsic regression. *Data Mining and Knowledge Discovery*, 35(3):1032–1060.

[118] Tavenard, R., Faouzi, J., Vandewiele, G., Divo, F., Androz, G., Holtz, C., Payne, M., Yurchak, R., Rußwurm, M., Kolar, K., and Woods, E. (2020). Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6.

[119] Tibshirani, R., Walther, G., and Hastie, T. (2002). Estimating the Number of Clusters in a Data Set Via the Gap Statistic. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 63(2):411–423.

[120] Tiwari, M., Zhang, M. J., Mayclin, J., Thrun, S., Piech, C., and Shomorony, I. (2020). Banditpam: Almost linear time k-medoids clustering via multi-armed bandits. In *Advances in Neural Information Processing Systems*, pages 368–374.

[121] Trirat, P., Shin, Y., Kang, J., Nam, Y., Na, J., Bae, M., Kim, J., Kim, B., and Lee, J.-G. (2024). Universal time-series representation learning: A survey. *ArXiv*, abs/2401.03717.

[122] Truong, C., Oudre, L., and Vayatis, N. (2020). Selective review of offline change point detection methods. *Signal Processing*, 167:107299.

[123] Umatani, R., Imai, T., Kawamoto, K., and Kunimasa, S. (2023). Time series clustering with an em algorithm for mixtures of linear gaussian state space models. *Pattern Recognition*, 138:109375.

[124] Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586 – 600. Cited by: 2070.

[125] Vinh, N. X., Epps, J., and Bailey, J. (2010). Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854.

[126] Vlachos, M., Kollios, G., and Gunopulos, D. (2002). Discovering similar multidimensional trajectories. In *Proceedings 18th International Conference on Data Engineering*, pages 673–684.

[127] Wenig, P., Höfgen, M., and Papenbrock, T. (2024). Jet: Fast estimation of hierarchical time series clustering. *Engineering Proceedings*, 68(1).

[128] Yang, J. and 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, page 177–186, New York, NY, USA. Association for Computing Machinery.

[129] Yang, Y., Shen, H. T., Ma, Z., Huang, Z., and Zhou, X. (2011). $l_{2,1}$-norm regularized discriminative feature selection for unsupervised learning. In *IJCAI International Joint Conference on Artificial Intelligence*, pages 1589–1594.

[130] Zhang, Q., Wu, J., Zhang, P., Long, G., and Zhang, C. (2019). Salient subsequence learning for time series clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2193–2207.

[131] Zhang, T., Ramakrishnan, R., and Livny, M. (1996). Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, SIGMOD '96, page 103–114, New York, NY, USA. Association for Computing Machinery.

[132] Zhao, J. and Itti, L. (2018). shapedtw: Shape dynamic time warping. *Pattern Recognition*, 74:171–184.

[133] Zolhavarieh, S., Aghabozorgi, S., and Teh, Y. W. (2014). A review of subsequence time series clustering. *TheScientificWorldJournal*, 2014:312521. Epub 2014 Jul 21.

# Appendix A

# Excluded Datasets for Models

This appendix details the datasets excluded from each experiment's analysis. Datasets were excluded for one of the following reasons:

- The algorithm was too computationally expensive to complete within our seven-day runtime limit.
- The algorithm began to diverge, resulting in random clustering and causing the models to become stuck in infinite loops. Consequently, we had to terminate the models' execution before any results could be produced.

To show the missing datasets for each experiments we produce missing dataset tables. Each table in this chapter is labelled with the corresponding experiment name and lists the excluded datasets along with the reasons for their exclusion. Datasets not mentioned in the tables are assumed to have been included in the analysis. To conserve space, only the missing datasets are listed.

Table A.1 provides an example for a example experiment. Datasets are listed in the rows, while clusterer names are listed in the columns. An "x" in a cell indicates that the dataset was missing, whereas a "✓" indicates that the dataset was included in the analysis. The table caption references the experiment and briefly outline the reasons for the missing datasets.

Table A.1 Example of Missing Datasets for Clusterers. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **Clusterer A** | **Clusterer B** | **Clusterer C** | **Clusterer D** |
|---|---|---|---|---|
| Dataset 1 | ✓ | ✓ | x | ✓ |
| Dataset 2 | x | ✓ | ✓ | ✓ |
| Dataset 3 | ✓ | x | ✓ | ✓ |
| Dataset 4 | ✓ | ✓ | ✓ | x |
| Dataset 5 | ✓ | ✓ | x | ✓ |

Table A.2 Baseline Lloyd's with *k*-means-soft-DBA using the combined test-train split experiment missing datasets. A total of 27 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-euclidean** | **k-means-soft-dba** | **k-sc** | **k-shapes** |
|---|---|---|---|---|---|
| CinCECGTorso | ✓ | ✓ | x | ✓ | ✓ |
| EOGHorizontalSignal | ✓ | ✓ | x | ✓ | ✓ |
| EOGVerticalSignal | ✓ | ✓ | x | ✓ | ✓ |
| EthanolLevel | ✓ | ✓ | x | ✓ | ✓ |
| FordA | ✓ | ✓ | x | ✓ | ✓ |
| FordB | ✓ | ✓ | x | ✓ | ✓ |
| HandOutlines | x | ✓ | x | ✓ | ✓ |
| InlineSkate | ✓ | ✓ | x | ✓ | ✓ |
| LargeKitchenAppliances | ✓ | ✓ | x | ✓ | ✓ |
| Mallat | ✓ | ✓ | x | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | x | ✓ | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | x | ✓ | ✓ |
| Phoneme | x | ✓ | x | ✓ | ✓ |
| PigAirwayPressure | ✓ | ✓ | x | ✓ | ✓ |
| PigArtPressure | ✓ | ✓ | x | ✓ | ✓ |
| PigCVP | ✓ | ✓ | x | ✓ | ✓ |
| RefrigerationDevices | ✓ | ✓ | x | ✓ | ✓ |
| ScreenType | ✓ | ✓ | x | ✓ | ✓ |
| SemgHandGenderCh2 | ✓ | ✓ | x | ✓ | ✓ |
| SemgHandMovementCh2 | ✓ | ✓ | x | ✓ | ✓ |
| SemgHandSubjectCh2 | ✓ | ✓ | x | ✓ | ✓ |
| StarLightCurves | x | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryAll | x | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryX | ✓ | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | x | x | ✓ |
| **Total Missing** | **5** | **0** | **27** | **1** | **0** |

Table A.3 Baseline Lloyd's with *k*-means-soft-DBA using the test-train split experiment missing datasets. A total of 8 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-euclidean** | **k-means-soft-dba** | **k-sc** | **k-shapes** |
|---|---|---|---|---|---|
| FordA | ✓ | ✓ | x | ✓ | ✓ |
| FordB | ✓ | ✓ | x | ✓ | ✓ |
| HandOutlines | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | x | ✓ | ✓ |
| SemgHandMovementCh2 | ✓ | ✓ | x | ✓ | ✓ |
| SemgHandSubjectCh2 | ✓ | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryAll | ✓ | ✓ | x | ✓ | ✓ |
| **Total Missing** | **0** | **0** | **8** | **0** | **0** |

Table A.4 Baseline Lloyd's using the combined test-train split experiment missing datasets. A total of 6 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-euclidean** | **k-sc** | **k-shapes** |
|---|---|---|---|---|
| HandOutlines | x | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | ✓ | ✓ |
| Phoneme | x | ✓ | ✓ | ✓ |
| StarLightCurves | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryAll | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | x | ✓ |
| **Total Missing** | **5** | **0** | **1** | **0** |

Table A.5 k-means-elastic-distances-initial-experiments combined test-train split experiment missing datasets. A total of 32 datasets are excluded. Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained. Models that obtained results for all datasets are excluded to conserve space.

| **Dataset** | **k-means-adtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-edr** | **k-means-lcss** | **k-means-msm** | **k-means-shape-dtw** | **k-means-soft-dtw** | **k-means-twe** |
|---|---|---|---|---|---|---|---|---|---|
| Adiac | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | x | ✓ |
| CinCECGTorso | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ |
| Coffee | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| DistalPhalanxTW | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| EOGHorizontalSignal | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| EOGVerticalSignal | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| EthanolLevel | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| FiftyWords | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ |
| Fish | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| InlineSkate | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| Mallat | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| Meat | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| MiddlePhalanxTW | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ | ✓ | ✓ | x | x | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | ✓ | ✓ | ✓ | ✓ | x | x | ✓ |
| OliveOil | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | x | x | x | x | ✓ | x | x | x |
| PigAirwayPressure | ✓ | x | x | x | ✓ | ✓ | x | x | ✓ |
| PigArtPressure | x | x | x | x | x | x | x | x | x |
| PigCVP | x | x | x | x | x | x | x | x | x |
| ProximalPhalanxOutlineAgeGroup | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| ProximalPhalanxTW | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | x | ✓ |
| SemgHandMovementCh2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| SemgHandSubjectCh2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| ShapesAll | ✓ | ✓ | x | x | ✓ | ✓ | x | x | ✓ |
| StarLightCurves | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryAll | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ |
| Wine | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| WordSynonyms | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| **Total Missing** | **3** | **6** | **5** | **7** | **13** | **2** | **22** | **12** | **3** |

Table A.6 k-means-elastic-distances-initial-experiments test-train split experiment missing datasets. A total of 24 datasets are excluded. Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained. Models that obtained results for all datasets are excluded to conserve space.

| **Dataset** | **k-means-adtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-edr** | **k-means-lcss** | **k-means-msm** | **k-means-shape-dtw** | **k-means-soft-dtw** |
|---|---|---|---|---|---|---|---|---|
| Adiac | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | x |
| Coffee | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| DiatomSizeReduction | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| DistalPhalanxTW | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| EOGHorizontalSignal | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| EOGVerticalSignal | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| EthanolLevel | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| FiftyWords | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x |
| Fish | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| Meat | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ |
| MiddlePhalanxTW | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ | x | ✓ | x | x |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x |
| OliveOil | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | x |
| Phoneme | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x |
| PigAirwayPressure | ✓ | x | x | ✓ | ✓ | ✓ | x | x |
| PigArtPressure | x | x | x | x | ✓ | ✓ | x | x |
| PigCVP | ✓ | ✓ | x | x | ✓ | ✓ | x | x |
| ProximalPhalanxOutlineAgeGroup | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| ProximalPhalanxTW | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | x |
| SemgHandMovementCh2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| ShapesAll | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x |
| UWaveGestureLibraryAll | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| Wine | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| **Total Missing** | **1** | **3** | **3** | **5** | **12** | **1** | **13** | **11** |

Table A.7 k-means-elastic-distances-initial-experiments-no-lcss-soft-shape combined test-train split experiment excluding methods with no missing datasets. A total of 9 datasets are excluded. Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained. Models that obtained results for all datasets are excluded to conserve space.

| **Dataset** | **k-means-adtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-edr** | **k-means-msm** | **k-means-twe** |
|---|---|---|---|---|---|---|
| Meat | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| OliveOil | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| Phoneme | x | x | x | x | ✓ | x |
| PigAirwayPressure | ✓ | x | x | x | ✓ | ✓ |
| PigArtPressure | x | x | x | x | x | x |
| PigCVP | x | x | x | x | x | x |
| ShapesAll | ✓ | ✓ | x | x | ✓ | ✓ |
| **Total Missing** | **3** | **6** | **5** | **7** | **2** | **3** |

Table A.8 k-means-elastic-distances-initial-experiments-with-baseline combined test-train split experiment excluding methods with no missing datasets. A total of 14 datasets are excluded. Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained. Specifically for *k*-means-ba-dtw and *k*-sc datasets are excluded due to runtime exceeding our seven day limit. Models that obtained results for all datasets are excluded to conserve space.

| **Dataset** | **k-means-adtw** | **k-means-ba-dtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-edr** | **k-means-msm** | **k-means-twe** | **k-sc** |
|---|---|---|---|---|---|---|---|---|
| HandOutlines | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Meat | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| OliveOil | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| Phoneme | x | x | x | x | x | ✓ | x | ✓ |
| PigAirwayPressure | ✓ | ✓ | x | x | x | ✓ | ✓ | ✓ |
| PigArtPressure | x | ✓ | x | x | x | ✓ | x | ✓ |
| PigCVP | x | ✓ | x | x | x | ✓ | x | ✓ |
| ShapesAll | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ |
| StarLightCurves | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryAll | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x |
| **Total Missing** | **3** | **5** | **6** | **5** | **7** | **2** | **3** | **1** |

Table A.9 k-means-elastic-distances-initial-experiments-with-baseline test-train split experiment excluding methods with no missing datasets. A total of 7 datasets are excluded. Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained. Models that obtained results for all datasets are excluded to conserve space.

| **Dataset** | **k-means-adtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-edr** | **k-means-msm** | **k-means-twe** |
|---|---|---|---|---|---|---|
| Adiac | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| Meat | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| OliveOil | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| PigAirwayPressure | ✓ | x | x | ✓ | ✓ | ✓ |
| PigArtPressure | x | x | x | x | ✓ | x |
| PigCVP | ✓ | ✓ | x | x | ✓ | ✓ |
| **Total Missing** | **1** | **3** | **3** | **5** | **1** | **1** |

Table A.10 k-means-elastic-distances-window-tuning combined test-train split experiment missing datasets. A total of 11 datasets are excluded .Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained.

| **Dataset** | **k-means-adtw** | **k-means-ba-dtw** | **k-means-ba-dtw-20-window** | **k-means-ddtw** | **k-means-ddtw-20-window** | **k-means-dtw** | **k-means-dtw-20-window** | **k-means-euclid** | **k-means-msm** |
|---|---|---|---|---|---|---|---|---|---|
| HandOutlines | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | x | ✓ | x | x | x | x | ✓ | ✓ |
| PigAirwayPressure | ✓ | ✓ | ✓ | x | x | x | x | ✓ | ✓ |
| PigArtPressure | x | ✓ | ✓ | x | x | x | x | ✓ | x |
| PigCVP | x | ✓ | ✓ | x | x | x | x | ✓ | x |
| ShapesAll | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| StarLightCurves | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryAll | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **3** | **5** | **1** | **6** | **5** | **5** | **4** | **0** | **2** |

Table A.11 k-means-elastic-distances-window-tuning test-train split experiment missing datasets. A total of 4 datasets are excluded. Datasets are missing due to the repeated creation of empty cluster meaning results could not be obtained.

| **Dataset** | **k-means-adtw** | **k-means-ba-dtw** | **k-means-ba-dtw-20-window** | **k-means-ddtw** | **k-means-ddtw-20-window** | **k-means-dtw** | **k-means-dtw-20-window** | **k-means-euclid** | **k-means-msm** |
|---|---|---|---|---|---|---|---|---|---|
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigAirwayPressure | ✓ | ✓ | ✓ | x | ✓ | x | ✓ | ✓ | ✓ |
| PigArtPressure | x | ✓ | ✓ | x | ✓ | x | ✓ | ✓ | x |
| PigCVP | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| **Total Missing** | **1** | **0** | **0** | **3** | **0** | **3** | **0** | **0** | **1** |

Table A.12 Alternate *k*-medoids all distances combined test-train split experiment missing datasets. A total of 36 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-edr** | **alternate-lcss** | **alternate-shape-dtw** | **alternate-soft-dtw** |
|---|---|---|---|---|
| ACSF1 | x | x | ✓ | x |
| Adiac | x | x | ✓ | x |
| Beef | ✓ | x | ✓ | ✓ |
| Car | ✓ | x | ✓ | ✓ |
| Chinatown | ✓ | x | ✓ | ✓ |
| Coffee | ✓ | x | ✓ | ✓ |
| DiatomSizeReduction | x | x | ✓ | ✓ |
| DistalPhalanxOutlineAgeGroup | ✓ | x | ✓ | ✓ |
| DistalPhalanxTW | ✓ | x | ✓ | x |
| EthanolLevel | ✓ | x | ✓ | ✓ |
| FiftyWords | ✓ | x | ✓ | ✓ |
| Fish | ✓ | x | ✓ | ✓ |
| FordA | ✓ | ✓ | x | ✓ |
| HandOutlines | ✓ | ✓ | x | x |
| Herring | ✓ | x | ✓ | ✓ |
| Mallat | ✓ | x | ✓ | ✓ |
| Meat | x | x | ✓ | x |
| MiddlePhalanxOutlineAgeGroup | ✓ | x | ✓ | ✓ |
| MiddlePhalanxOutlineCorrect | ✓ | x | ✓ | ✓ |
| MiddlePhalanxTW | x | x | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | ✓ | x | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | x | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | ✓ | ✓ |
| OliveOil | x | x | ✓ | x |
| PigAirwayPressure | x | x | ✓ | ✓ |
| Plane | ✓ | x | ✓ | ✓ |
| ProximalPhalanxOutlineAgeGroup | ✓ | x | ✓ | x |
| ProximalPhalanxOutlineCorrect | ✓ | x | ✓ | ✓ |
| ProximalPhalanxTW | x | x | ✓ | x |
| ShapesAll | ✓ | x | ✓ | ✓ |
| StarLightCurves | ✓ | ✓ | x | x |
| Symbols | ✓ | x | ✓ | ✓ |
| Trace | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryAll | ✓ | ✓ | x | x |
| Wine | x | x | ✓ | ✓ |
| **Total Missing** | **9** | **30** | **6** | **10** |

Table A.13 Alternate *k*-medoids without LCSS, EDR and soft-DTW combined test-train split experiment missing datasets. A total of 6 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-shape-dtw** |
|---|---|
| FordA | x |
| HandOutlines | x |
| MixedShapesRegularTrain | x |
| MixedShapesSmallTrain | x |
| StarLightCurves | x |
| UWaveGestureLibraryAll | x |
| **Total Missing** | **6** |

Table A.14 Alternate *k*-medoids without LCSS, EDR and soft-DTW with baseline clusterers combined test-train split experiment missing datasets. A total of 8 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-shape-dtw** | **k-means-ba-dtw** | **k-sc** |
|---|---|---|---|
| FordA | x | ✓ | ✓ |
| HandOutlines | x | x | ✓ |
| MixedShapesRegularTrain | x | x | ✓ |
| MixedShapesSmallTrain | x | ✓ | ✓ |
| Phoneme | ✓ | x | ✓ |
| StarLightCurves | x | x | ✓ |
| UWaveGestureLibraryAll | x | x | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | x |
| **Total Missing** | **6** | **5** | **1** |

Table A.15 Alternate *k*-medoids with all 12 elastic distances test-train split experiment missing datasets. A total of 10 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-adtw** | **alternate-euclidean** | **alternate-shape-dtw** | **alternate-soft-dtw** |
|---|---|---|---|---|
| ACSF1 | ✓ | ✓ | ✓ | x |
| Adiac | ✓ | ✓ | ✓ | x |
| HandOutlines | ✓ | ✓ | x | ✓ |
| MiddlePhalanxOutlineAgeGroup | ✓ | ✓ | ✓ | x |
| MiddlePhalanxTW | ✓ | ✓ | ✓ | x |
| OliveOil | ✓ | ✓ | ✓ | x |
| ProximalPhalanxTW | ✓ | ✓ | ✓ | x |
| Wine | ✓ | ✓ | ✓ | x |
| Worms | x | x | x | ✓ |
| WormsTwoClass | x | x | x | ✓ |
| **Total Missing** | **2** | **2** | **3** | **7** |

Table A.16 Alternate *k*-medoids with all 12 elastic distance and the baseline clusterers test-train split experiment missing datasets. A total of 10 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-adtw** | **alternate-euclidean** | **alternate-shape-dtw** | **alternate-soft-dtw** |
|---|---|---|---|---|
| ACSF1 | ✓ | ✓ | ✓ | x |
| Adiac | ✓ | ✓ | ✓ | x |
| HandOutlines | ✓ | ✓ | x | ✓ |
| MiddlePhalanxOutlineAgeGroup | ✓ | ✓ | ✓ | x |
| MiddlePhalanxTW | ✓ | ✓ | ✓ | x |
| OliveOil | ✓ | ✓ | ✓ | x |
| ProximalPhalanxTW | ✓ | ✓ | ✓ | x |
| Wine | ✓ | ✓ | ✓ | x |
| Worms | x | x | x | ✓ |
| WormsTwoClass | x | x | x | ✓ |
| **Total Missing** | **2** | **2** | **3** | **7** |

Table A.17 Alternate *k*-medoids compared to *k*-means over the combined test-train split across experiment missing datasets. A total of 7 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-adtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-msm** | **k-means-twe** |
|---|---|---|---|---|---|
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | ✓ | ✓ | ✓ |
| Phoneme | x | x | x | ✓ | x |
| PigAirwayPressure | ✓ | x | x | ✓ | ✓ |
| PigArtPressure | x | x | x | x | x |
| PigCVP | x | x | x | x | x |
| ShapesAll | ✓ | ✓ | x | ✓ | ✓ |
| **Total Missing** | **3** | **6** | **5** | **2** | **3** |

Table A.18 Alternate *k*-medoids compared to *k*-means across test-train split experiment missing datasets. A total of 6 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-adtw** | **alternate-euclidean** | **k-means-adtw** | **k-means-ddtw** | **k-means-dtw** | **k-means-msm** | **k-means-twe** |
|---|---|---|---|---|---|---|---|
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| PigAirwayPressure | ✓ | ✓ | ✓ | x | x | ✓ | ✓ |
| PigArtPressure | ✓ | ✓ | x | x | x | x | x |
| PigCVP | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| Worms | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| WormsTwoClass | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **2** | **2** | **1** | **3** | **3** | **1** | **1** |

Table A.19 PAM with all distances combined test-train split experiment missing datasets. A total of 11 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **pam-adtw** | **pam-ddtw** | **pam-dtw** | **pam-edr** | **pam-erp** | **pam-euclidean** | **pam-lcss** | **pam-msm** | **pam-shape-dtw** | **pam-soft-dtw** | **pam-twe** | **pam-wddtw** | **pam-wdtw** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CinCECGTorso | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| Crop | x | x | x | x | x | x | x | x | x | x | x | x | x |
| FordA | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ |
| Mallat | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ |
| StarLightCurves | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | x | ✓ | ✓ |
| UWaveGestureLibraryAll | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ | ✓ |
| **Total Missing** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **11** | **6** | **2** | **1** | **1** |

Table A.20 PAM with 12 elastic distances with baseline clusterers on the combined test-train split experiment missing datasets. A total of 13 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-sc** | **k-shapes** | **pam-adtw** | **pam-ddtw** | **pam-dtw** | **pam-edr** | **pam-erp** | **pam-euclidean** | **pam-lcss** | **pam-msm** | **pam-twe** | **pam-wddtw** | **pam-wdtw** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CinCECGTorso | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Crop | ✓ | ✓ | ✓ | x | x | x | x | x | x | x | x | x | x | x |
| FordA | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Mallat | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryAll | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **5** | **1** | **0** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **2** | **1** | **1** |

Table A.21 PAM with 10 elastic distances exclude PAM-soft-DTW and PAM-shape-DTW with baseline clusterers on the combined test-train split experiment missing datasets. A total of 7 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-sc** | **pam-adtw** | **pam-ddtw** | **pam-dtw** | **pam-edr** | **pam-erp** | **pam-euclidean** | **pam-lcss** | **pam-msm** | **pam-twe** | **pam-wddtw** | **pam-wdtw** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Crop | ✓ | ✓ | x | x | x | x | x | x | x | x | x | x | x |
| HandOutlines | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryAll | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **5** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **1** | **2** | **1** | **1** |

Table A.22 PAM with 12 elastic distances with baseline clusterers on the test-train split experiment missing datasets. A total of 1 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **pam-shape-dtw** |
|---|---|
| HandOutlines | x |
| **Total Missing** | **1** |

Table A.23 CLARANS with 12 elastic distances on the combined test-train split experiment missing datasets. A total of 6 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **clarans-shape-dtw** | **clarans-soft-dtw** |
|---|---|---|
| HandOutlines | x | ✓ |
| MixedShapesRegularTrain | x | ✓ |
| NonInvasiveFetalECGThorax1 | x | ✓ |
| NonInvasiveFetalECGThorax2 | x | ✓ |
| StarLightCurves | x | x |
| UWaveGestureLibraryAll | x | x |
| **Total Missing** | **6** | **2** |

Table A.24 CLARA with 12 elastic distances with baseline clusterers on the test-train split experiment missing datasets. A total of 6 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-sc** |
|---|---|---|
| HandOutlines | x | ✓ |
| MixedShapesRegularTrain | x | ✓ |
| Phoneme | x | ✓ |
| StarLightCurves | x | ✓ |
| UWaveGestureLibraryAll | x | ✓ |
| UWaveGestureLibraryZ | ✓ | x |
| **Total Missing** | **5** | **1** |

Table A.25 CLARANS with 12 elastic distances with baseline clusterers on the combined test-train split experiment missing datasets. A total of 8 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **clarans-shape-dtw** | **clarans-soft-dtw** | **k-means-ba-dtw** | **k-sc** |
|---|---|---|---|---|
| HandOutlines | x | ✓ | x | ✓ |
| MixedShapesRegularTrain | x | ✓ | x | ✓ |
| NonInvasiveFetalECGThorax1 | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | x | ✓ | ✓ | ✓ |
| Phoneme | ✓ | ✓ | x | ✓ |
| StarLightCurves | x | x | x | ✓ |
| UWaveGestureLibraryAll | x | x | x | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | ✓ | x |
| **Total Missing** | **6** | **2** | **5** | **1** |

Table A.26 Different *k*-medoids clusterers comparison over the combined test-train split experiment missing datasets. A total of 22 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days. To conserve space we exclude PAM from this table and refer to Table A.19 for PAMs missing datasets.

| **Dataset** | **alternate-edr** | **alternate-shape-dtw** | **alternate-soft-dtw** | **clarans-shape-dtw** | **clarans-soft-dtw** |
|---|---|---|---|---|---|
| ACSF1 | x | ✓ | x | ✓ | ✓ |
| Adiac | x | ✓ | x | ✓ | ✓ |
| CinCECGTorso | ✓ | ✓ | ✓ | ✓ | ✓ |
| Crop | ✓ | ✓ | ✓ | ✓ | ✓ |
| DiatomSizeReduction | x | ✓ | ✓ | ✓ | ✓ |
| DistalPhalanxTW | ✓ | ✓ | x | ✓ | ✓ |
| FordA | ✓ | x | ✓ | ✓ | ✓ |
| HandOutlines | ✓ | x | x | x | ✓ |
| Mallat | ✓ | ✓ | ✓ | ✓ | ✓ |
| Meat | x | ✓ | x | ✓ | ✓ |
| MiddlePhalanxTW | x | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | x | ✓ | x | ✓ |
| MixedShapesSmallTrain | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | ✓ | x | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | ✓ | x | ✓ |
| OliveOil | x | ✓ | x | ✓ | ✓ |
| PigAirwayPressure | x | ✓ | ✓ | ✓ | ✓ |
| ProximalPhalanxOutlineAgeGroup | ✓ | ✓ | x | ✓ | ✓ |
| ProximalPhalanxTW | x | ✓ | x | ✓ | ✓ |
| StarLightCurves | ✓ | x | x | x | x |
| UWaveGestureLibraryAll | ✓ | x | x | x | x |
| Wine | x | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **9** | **6** | **10** | **6** | **2** |

Table A.27 Different *k*-medoids clusterers comparison over the test-train split experiment missing datasets. A total of 10 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-adtw** | **alternate-euclidean** | **alternate-shape-dtw** | **alternate-soft-dtw** | **pam-shape-dtw** |
|---|---|---|---|---|---|
| ACSF1 | ✓ | ✓ | ✓ | x | ✓ |
| Adiac | ✓ | ✓ | ✓ | x | ✓ |
| HandOutlines | ✓ | ✓ | x | ✓ | x |
| MiddlePhalanxOutlineAgeGroup | ✓ | ✓ | ✓ | x | ✓ |
| MiddlePhalanxTW | ✓ | ✓ | ✓ | x | ✓ |
| OliveOil | ✓ | ✓ | ✓ | x | ✓ |
| ProximalPhalanxTW | ✓ | ✓ | ✓ | x | ✓ |
| Wine | ✓ | ✓ | ✓ | x | ✓ |
| Worms | x | x | x | ✓ | ✓ |
| WormsTwoClass | x | x | x | ✓ | ✓ |
| **Total Missing** | **2** | **2** | **3** | **7** | **1** |

Table A.28 Best *k*-medoids clusterers comparison over combined test-train split experiment missing datasets. A total of 20 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days. To conserve space we exclude PAM from this table and refer to Table A.19 for PAMs missing datasets.

| **Dataset** | **alternate-shape-dtw** | **alternate-soft-dtw** | **clarans-shape-dtw** | **clarans-soft-dtw** | **k-means-ba-dtw** | **k-sc** |
|---|---|---|---|---|---|---|
| ACSF1 | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| Adiac | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| CinCECGTorso | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Crop | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| DistalPhalanxTW | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| FordA | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | x | x | x | ✓ | x | ✓ |
| Mallat | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Meat | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | x | ✓ | x | ✓ |
| MixedShapesSmallTrain | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| OliveOil | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| Phoneme | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| ProximalPhalanxOutlineAgeGroup | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| ProximalPhalanxTW | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | x | x | x | x | ✓ |
| UWaveGestureLibraryAll | x | x | x | x | x | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | ✓ | ✓ | ✓ | x |
| **Total Missing** | **6** | **10** | **6** | **2** | **5** | **1** |

Table A.29 Best *k*-medoids clusterers comparison over test-train split experiment missing datasets. A total of 10 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-adtw** | **alternate-shape-dtw** | **alternate-soft-dtw** | **pam-shape-dtw** |
|---|---|---|---|---|
| ACSF1 | ✓ | ✓ | x | ✓ |
| Adiac | ✓ | ✓ | x | ✓ |
| HandOutlines | ✓ | x | ✓ | x |
| MiddlePhalanxOutlineAgeGroup | ✓ | ✓ | x | ✓ |
| MiddlePhalanxTW | ✓ | ✓ | x | ✓ |
| OliveOil | ✓ | ✓ | x | ✓ |
| ProximalPhalanxTW | ✓ | ✓ | x | ✓ |
| Wine | ✓ | ✓ | x | ✓ |
| Worms | x | x | ✓ | ✓ |
| WormsTwoClass | x | x | ✓ | ✓ |
| **Total Missing** | **2** | **3** | **7** | **1** |

Table A.30 Elastic Barycentre $k$-means using 8 different elastic distances over the combined test-train split experiment missing datasets. A total of 29 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-ba-shape-dtw** | **k-means-ba-soft-dtw** | **k-means-ba-twe** |
|---|---|---|---|---|
| Adiac | ✓ | ✓ | x | ✓ |
| CinCECGTorso | ✓ | x | x | ✓ |
| EOGHorizontalSignal | ✓ | x | ✓ | ✓ |
| EOGVerticalSignal | ✓ | x | ✓ | ✓ |
| EthanolLevel | ✓ | x | ✓ | ✓ |
| FordA | ✓ | x | x | ✓ |
| FordB | ✓ | x | x | ✓ |
| HandOutlines | x | x | x | ✓ |
| InlineSkate | ✓ | x | ✓ | ✓ |
| Mallat | ✓ | x | ✓ | ✓ |
| MixedShapesRegularTrain | x | x | x | ✓ |
| MixedShapesSmallTrain | ✓ | x | x | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | x | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | x | ✓ |
| Phoneme | x | x | x | x |
| PigAirwayPressure | ✓ | x | x | ✓ |
| PigArtPressure | ✓ | x | x | ✓ |
| PigCVP | ✓ | x | x | ✓ |
| ProximalPhalanxTW | ✓ | ✓ | x | ✓ |
| SemgHandGenderCh2 | ✓ | x | ✓ | ✓ |
| SemgHandMovementCh2 | ✓ | x | x | ✓ |
| SemgHandSubjectCh2 | ✓ | x | ✓ | ✓ |
| ShapesAll | ✓ | x | ✓ | ✓ |
| StarLightCurves | x | x | x | ✓ |
| UWaveGestureLibraryAll | x | x | x | ✓ |
| UWaveGestureLibraryX | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryY | ✓ | x | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | x | ✓ | ✓ |
| Yoga | ✓ | x | ✓ | ✓ |
| **Total Missing** | **5** | **27** | **17** | **1** |

Table A.31 Elastic Barycentre *k*-means using 6 different elastic distances over the combined test-train split experiment missing datasets. A total of 5 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-ba-twe** |
|---|---|---|
| HandOutlines | x | ✓ |
| MixedShapesRegularTrain | x | ✓ |
| Phoneme | x | x |
| StarLightCurves | x | ✓ |
| UWaveGestureLibraryAll | x | ✓ |
| **Total Missing** | **5** | **1** |

Table A.32 Barycentre averaging using 6 different elastic distances with the baseline clusterers over the combined test-train split experiment missing datasets. A total of 7 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-ba-twe** | **k-sc** | **pam-adtw** | **pam-dtw** | **pam-erp** | **pam-msm** | **pam-twe** | **pam-wdtw** |
|---|---|---|---|---|---|---|---|---|---|
| Crop | ✓ | ✓ | ✓ | x | x | x | x | x | x |
| HandOutlines | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| UWaveGestureLibraryAll | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **5** | **1** | **1** | **1** | **1** | **1** | **1** | **2** | **1** |

Table A.33 Elastic Barycentre *k*-means using 6 different elastic distances with the baseline clusterers and soft-DBA over the combined test-train split experiment missing datasets. A total of 33 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days. To conserve space we exclude PAM from this table and refer to Table A.19 for PAMs missing datasets.

| **Dataset** | **k-means-ba-dtw** | **k-means-ba-shape-dtw** | **k-means-ba-soft-dtw** | **k-means-ba-twe** | **k-means-soft-dba** | **k-sc** |
|---|---|---|---|---|---|---|
| Adiac | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| CinCECGTorso | ✓ | x | x | ✓ | x | ✓ |
| Crop | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| EOGHorizontalSignal | ✓ | x | ✓ | ✓ | x | ✓ |
| EOGVerticalSignal | ✓ | x | ✓ | ✓ | x | ✓ |
| EthanolLevel | ✓ | x | ✓ | ✓ | x | ✓ |
| FordA | ✓ | x | x | ✓ | x | ✓ |
| FordB | ✓ | x | x | ✓ | x | ✓ |
| HandOutlines | x | x | x | ✓ | x | ✓ |
| InlineSkate | ✓ | x | ✓ | ✓ | x | ✓ |

Table A.34 Elastic Barycentre *k*-means using 8 different elastic distances over the test-train split experiment missing datasets. A total of 14 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-shape-dtw** | **k-means-ba-soft-dtw** |
|---|---|---|
| Adiac | ✓ | x |
| EOGHorizontalSignal | x | ✓ |
| EOGVerticalSignal | x | ✓ |
| EthanolLevel | x | ✓ |
| FordA | x | x |
| FordB | x | x |
| HandOutlines | x | ✓ |
| NonInvasiveFetalECGThorax1 | x | x |
| NonInvasiveFetalECGThorax2 | x | x |
| ProximalPhalanxTW | ✓ | x |
| SemgHandMovementCh2 | x | ✓ |
| SemgHandSubjectCh2 | x | ✓ |
| StarLightCurves | x | ✓ |
| UWaveGestureLibraryAll | x | ✓ |
| **Total Missing** | **12** | **6** |

Table A.35 Elastic Barycentre *k*-means using 8 different elastic distances with the baseline clusterers and soft-DBA over the test-train split experiment missing datasets. A total of 14 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-shape-dtw** | **k-means-ba-soft-dtw** | **k-means-soft-dba** | **pam-shape-dtw** |
|---|---|---|---|---|
| Adiac | ✓ | x | ✓ | ✓ |
| EOGHorizontalSignal | x | ✓ | ✓ | ✓ |
| EOGVerticalSignal | x | ✓ | ✓ | ✓ |
| EthanolLevel | x | ✓ | ✓ | ✓ |
| FordA | x | x | x | ✓ |
| FordB | x | x | x | ✓ |
| HandOutlines | x | ✓ | x | x |
| NonInvasiveFetalECGThorax1 | x | x | x | ✓ |
| NonInvasiveFetalECGThorax2 | x | x | x | ✓ |
| ProximalPhalanxTW | ✓ | x | ✓ | ✓ |
| SemgHandMovementCh2 | x | ✓ | x | ✓ |
| SemgHandSubjectCh2 | x | ✓ | x | ✓ |
| StarLightCurves | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryAll | x | ✓ | x | ✓ |
| **Total Missing** | **12** | **6** | **8** | **1** |

Table A.36 Lloyd's-based clusterers comparison with Elastic Barycentre Average test-train split experiment missing datasets. A total of 5 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **alternate-adtw** | **k-means-adtw** | **k-means-dtw** | **k-means-msm** | **k-means-twe** |
|---|---|---|---|---|---|
| PigAirwayPressure | ✓ | ✓ | x | ✓ | ✓ |
| PigArtPressure | ✓ | x | x | x | x |
| PigCVP | ✓ | ✓ | x | ✓ | ✓ |
| Worms | x | ✓ | ✓ | ✓ | ✓ |
| WormsTwoClass | x | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **2** | **1** | **3** | **1** | **1** |

Table A.37 KESBA baseline experiment using the combined test-train split experiment missing datasets. A total of 7 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-ba-twe** | **k-sc** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|
| Crop | ✓ | ✓ | ✓ | x | x |
| HandOutlines | x | ✓ | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | x | ✓ | ✓ | ✓ |
| StarLightCurves | x | ✓ | ✓ | ✓ | x |
| UWaveGestureLibraryAll | x | ✓ | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | x | ✓ | ✓ |
| **Total Missing** | **5** | **1** | **1** | **1** | **2** |

Table A.38 KESBA baseline with soft-DBA experiment using the combined test-train split experiment missing datasets. A total of 28 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-ba-dtw** | **k-means-ba-twe** | **k-means-soft-dba** | **k-sc** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|
| CinCECGTorso | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| Crop | ✓ | ✓ | ✓ | ✓ | x | x |
| EOGHorizontalSignal | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| EOGVerticalSignal | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| EthanolLevel | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| FordA | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| FordB | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| HandOutlines | x | ✓ | x | ✓ | ✓ | ✓ |
| InlineSkate | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| LargeKitchenAppliances | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| Mallat | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | x | ✓ | x | ✓ | ✓ | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| Phoneme | x | x | x | ✓ | ✓ | ✓ |
| PigAirwayPressure | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| PigArtPressure | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| PigCVP | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| RefrigerationDevices | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| ScreenType | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| SemgHandGenderCh2 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| SemgHandMovementCh2 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| SemgHandSubjectCh2 | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| StarLightCurves | x | ✓ | x | ✓ | ✓ | x |
| UWaveGestureLibraryAll | x | ✓ | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryX | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | x | x | ✓ | ✓ |
| **Total Missing** | **5** | **1** | **27** | **1** | **1** | **2** |

Table A.39 KESBA baseline experiment using the test-train split experiment missing datasets. A total of 8 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-means-soft-dba** |
|---|---|
| FordA | x |
| FordB | x |
| HandOutlines | x |
| NonInvasiveFetalECGThorax1 | x |
| NonInvasiveFetalECGThorax2 | x |
| SemgHandMovementCh2 | x |
| SemgHandSubjectCh2 | x |
| UWaveGestureLibraryAll | x |
| **Total Missing** | **8** |

Table A.40 KESBA with different initlisation strategies over the combined test-train split experiment missing datasets. A total of 24 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **k-sc** | **kesba-forgy-restarts-msm** | **kesba-forgy-restarts-twe** | **kesba-full-twe** |
|---|---|---|---|---|
| CinCECGTorso | ✓ | x | x | ✓ |
| EOGHorizontalSignal | ✓ | x | x | ✓ |
| EOGVerticalSignal | ✓ | x | x | ✓ |
| EthanolLevel | ✓ | x | x | ✓ |
| HandOutlines | ✓ | x | x | ✓ |
| Haptics | ✓ | x | ✓ | ✓ |
| InlineSkate | ✓ | x | x | ✓ |
| Mallat | ✓ | x | x | ✓ |
| MixedShapesRegularTrain | ✓ | x | x | ✓ |
| MixedShapesSmallTrain | ✓ | x | x | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | x | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | x | ✓ |
| Phoneme | ✓ | x | x | x |
| PigAirwayPressure | ✓ | x | ✓ | ✓ |
| PigArtPressure | ✓ | x | x | ✓ |
| PigCVP | ✓ | x | x | ✓ |
| SemgHandGenderCh2 | ✓ | x | x | ✓ |
| SemgHandMovementCh2 | ✓ | x | x | ✓ |
| SemgHandSubjectCh2 | ✓ | x | x | ✓ |
| ShapesAll | ✓ | x | x | ✓ |
| StarLightCurves | ✓ | x | x | ✓ |
| UWaveGestureLibraryAll | ✓ | x | x | ✓ |
| UWaveGestureLibraryX | ✓ | ✓ | x | ✓ |
| UWaveGestureLibraryZ | x | ✓ | ✓ | ✓ |
| **Total Missing** | **1** | **22** | **21** | **1** |

Table A.41 KESBA with different ba subset sizes over the combined test-train split experiment missing datasets. A total of 5 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **kesba-average-10-twe** | **kesba-average-20-twe** | **kesba-average-30-twe** | **kesba-average-40-twe** | **kesba-average-50-twe** | **kesba-full-twe** |
|---|---|---|---|---|---|---|
| NonInvasiveFetalECGThorax1 | x | x | x | x | x | ✓ |
| NonInvasiveFetalECGThorax2 | x | x | x | x | x | ✓ |
| Phoneme | x | x | x | x | x | x |
| PigArtPressure | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| UWaveGestureLibraryAll | x | x | x | x | x | ✓ |
| **Total Missing** | **4** | **4** | **4** | **4** | **5** | **1** |

Table A.42 KESBA runtime comparison experiment combined test-train split experiment missing datasets. A total of 6 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **kesba-average-10-twe** | **kesba-average-20-twe** | **kesba-average-30-twe** | **kesba-average-40-twe** | **kesba-average-50-twe** | **kesba-both-50-twe** |
|---|---|---|---|---|---|---|
| MiddlePhalanxOutlineAgeGroup | ✓ | ✓ | ✓ | ✓ | ✓ | x |
| NonInvasiveFetalECGThorax1 | x | x | x | x | x | ✓ |
| NonInvasiveFetalECGThorax2 | x | x | x | x | x | ✓ |
| Phoneme | x | x | x | x | x | x |
| PigArtPressure | ✓ | ✓ | ✓ | ✓ | x | ✓ |
| UWaveGestureLibraryAll | x | x | x | x | x | x |
| **Total Missing** | **4** | **4** | **4** | **4** | **5** | **3** |

Table A.43 KESBA both experiment combined test-train split experiment missing datasets. A total of 3 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **kesba-both-20-twe** | **kesba-both-30-twe** | **kesba-both-40-twe** | **kesba-both-50-twe** | **kesba-full-twe** |
|---|---|---|---|---|---|
| NonInvasiveFetalECGThorax2 | ✓ | ✓ | ✓ | x | ✓ |
| Phoneme | ✓ | ✓ | x | x | x |
| UWaveGestureLibraryAll | x | x | x | x | ✓ |
| **Total Missing** | **1** | **1** | **2** | **3** | **1** |

Table A.44 Baseline EUPW experiment combined test-train split experiment missing datasets. A total of 22 datasets are excluded. Datasets are missing due to empty clusters forming.

| **Dataset** | **eupw-euclidean** | **eupw-msm** | **eupw-twe** | **k-means-ba-dtw** | **k-sc** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|---|
| ACSF1 | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| Adiac | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| CricketY | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| Crop | x | x | x | ✓ | ✓ | x | x |
| DiatomSizeReduction | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| FiftyWords | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| Fish | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | x | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | x | x | x | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | x | x | x | ✓ | ✓ | ✓ | ✓ |
| OliveOil | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| Phoneme | ✓ | x | x | x | ✓ | ✓ | ✓ |
| PigAirwayPressure | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| PigArtPressure | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| PigCVP | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| ShapesAll | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | x | x | x | ✓ | ✓ | x |
| UWaveGestureLibraryAll | x | x | x | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | ✓ | ✓ | x | ✓ | ✓ |
| WordSynonyms | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| Worms | ✓ | x | x | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **6** | **19** | **19** | **5** | **1** | **1** | **2** |

Table A.45 Baseline EUPW experiment test-train split experiment missing datasets. A total of 34 datasets are excluded. Datasets are missing due to empty clusters forming.

| **Dataset** | **eupw-msm** | **eupw-twe** |
|---|---|---|
| ACSF1 | x | x |
| Adiac | x | x |
| ArrowHead | x | x |
| Beef | x | x |
| Car | x | x |
| CricketY | x | x |
| DiatomSizeReduction | x | x |
| ECGFiveDays | x | x |
| FaceFour | x | x |
| FacesUCR | x | x |
| FiftyWords | x | x |
| GunPoint | x | x |
| Haptics | x | x |
| HouseTwenty | x | x |
| InsectEPGRegularTrain | x | x |
| InsectEPGSmallTrain | x | x |
| Lightning7 | x | x |
| MiddlePhalanxTW | x | x |
| NonInvasiveFetalECGThorax1 | x | x |
| NonInvasiveFetalECGThorax2 | x | x |
| Phoneme | x | x |
| PigAirwayPressure | x | x |
| PigArtPressure | x | x |
| PigCVP | x | x |
| Rock | x | x |
| SemgHandMovementCh2 | x | x |
| ShapesAll | x | x |
| SwedishLeaf | x | x |
| Symbols | x | x |
| UWaveGestureLibraryX | x | x |
| UWaveGestureLibraryY | x | x |
| WordSynonyms | x | x |
| Worms | x | x |
| WormsTwoClass | x | x |
| **Total Missing** | **34** | **34** |

Table A.46 EUPW compared to other ensembles schemes experiment over the combined test-train split experiment missing datasets. A total of 21 datasets are excluded. Datasets are missing due to empty clusters forming. To conserve space we exclude PAM from this table and refer to Table A.19 for PAMs missing datasets.

| **Dataset** | **cspa** | **eupw-euclidean** | **eupw-msm** | **eupw-twe** | **hbgf** | **iterative-voting** | **mcla** | **nmf** | **si vo** |
|---|---|---|---|---|---|---|---|---|---|
| ACSF1 | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Adiac | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| CricketY | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Crop | x | x | x | x | x | x | x | x | x |
| DiatomSizeReduction | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| ElectricDevices | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| FiftyWords | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fish | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | ✓ | x | ✓ | ✓ | x | x | x | x | x |
| NonInvasiveFetalECGThorax1 | ✓ | x | x | x | x | x | x | x | x |
| NonInvasiveFetalECGThorax2 | ✓ | x | x | x | x | x | x | x | x |
| OliveOil | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigAirwayPressure | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigArtPressure | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigCVP | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| ShapesAll | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | x | x | x | x | x | x | x | x |
| UWaveGestureLibraryAll | x | x | x | x | x | x | x | x | x |
| WordSynonyms | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Worms | ✓ | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **4** | **6** | **19** | **19** | **6** | **6** | **6** | **6** | **6** |

Table A.47 EUPW compared to other ensembles schemes experiment over the test-train split experiment missing datasets. A total of 37 datasets are excluded. Datasets are missing due to empty clusters forming.

| **Dataset** | **cspa** | **eupw-msm** | **eupw-twe** | **iterative-voting** | **simple-voting** |
|---|---|---|---|---|---|
| ACSF1 | ✓ | x | x | ✓ | ✓ |
| Adiac | ✓ | x | x | ✓ | ✓ |
| ArrowHead | ✓ | x | x | ✓ | ✓ |
| Beef | ✓ | x | x | ✓ | ✓ |
| Car | ✓ | x | x | ✓ | ✓ |
| CricketY | ✓ | x | x | ✓ | ✓ |
| Crop | x | ✓ | ✓ | ✓ | ✓ |
| DiatomSizeReduction | ✓ | x | x | ✓ | ✓ |
| ECGFiveDays | ✓ | x | x | ✓ | ✓ |
| ElectricDevices | x | ✓ | ✓ | ✓ | ✓ |
| FaceFour | ✓ | x | x | ✓ | ✓ |
| FacesUCR | ✓ | x | x | ✓ | ✓ |
| FiftyWords | ✓ | x | x | ✓ | ✓ |
| GunPoint | ✓ | x | x | ✓ | ✓ |
| Haptics | ✓ | x | x | ✓ | ✓ |
| HouseTwenty | ✓ | x | x | ✓ | ✓ |
| InsectEPGRegularTrain | ✓ | x | x | ✓ | ✓ |
| InsectEPGSmallTrain | ✓ | x | x | ✓ | ✓ |
| Lightning7 | ✓ | x | x | ✓ | ✓ |
| MiddlePhalanxTW | ✓ | x | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | ✓ | x | x | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | ✓ | x | x | ✓ | ✓ |
| OliveOil | ✓ | ✓ | ✓ | x | x |
| Phoneme | ✓ | x | x | ✓ | ✓ |
| PigAirwayPressure | ✓ | x | x | x | ✓ |
| PigArtPressure | ✓ | x | x | ✓ | ✓ |
| PigCVP | ✓ | x | x | x | ✓ |
| Rock | ✓ | x | x | ✓ | ✓ |
| SemgHandMovementCh2 | ✓ | x | x | ✓ | ✓ |
| ShapesAll | ✓ | x | x | ✓ | ✓ |
| SwedishLeaf | ✓ | x | x | ✓ | ✓ |
| Symbols | ✓ | x | x | ✓ | ✓ |
| UWaveGestureLibraryX | ✓ | x | x | ✓ | ✓ |
| UWaveGestureLibraryY | ✓ | x | x | ✓ | ✓ |
| WordSynonyms | ✓ | x | x | ✓ | ✓ |
| Worms | ✓ | x | x | ✓ | ✓ |
| WormsTwoClass | ✓ | x | x | ✓ | ✓ |
| **Total Missing** | **2** | **34** | **34** | **3** | **1** |

Table A.48 EUPW-ECHI compared to EUPW-EDBI over the combined test-train split experiment missing datasets. A total of 20 datasets are excluded. Datasets are missing due to empty clusters forming.

| **Dataset** | **eupw-ECHI-euclidean** | **eupw-ECHI-msm** | **eupw-ECHI-twe** | **eupw-EDBI-euclidean** | **eupw-EDBI-msm** | **eupw-EDBI-twe** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|---|---|
| ACSF1 | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Adiac | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| CricketY | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Crop | x | x | x | x | x | x | x | x |
| DiatomSizeReduction | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| FiftyWords | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fish | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | x | ✓ | ✓ | x | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | x | x | x | x | ✓ | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | x | x | x | x | ✓ | ✓ | ✓ | ✓ |
| OliveOil | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigAirwayPressure | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigArtPressure | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| PigCVP | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| ShapesAll | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | x | x | x | x | x | ✓ | x |
| UWaveGestureLibraryAll | x | x | x | x | x | x | ✓ | ✓ |
| WordSynonyms | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| Worms | ✓ | x | x | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **6** | **19** | **19** | **6** | **3** | **3** | **1** | **2** |

Table A.49 EUPW-ECHI compared to EUPW-EBI over the test-train split experiment missing datasets. A total of 34 datasets are excluded. Datasets are missing due to empty clusters forming.

| **Dataset** | **eupw-ECHI-msm** | **eupw-ECHI-twe** |
|---|---|---|
| ACSF1 | x | x |
| Adiac | x | x |
| ArrowHead | x | x |
| Beef | x | x |
| Car | x | x |
| CricketY | x | x |
| DiatomSizeReduction | x | x |
| ECGFiveDays | x | x |
| FaceFour | x | x |
| FacesUCR | x | x |
| FiftyWords | x | x |
| GunPoint | x | x |
| Haptics | x | x |
| HouseTwenty | x | x |
| InsectEPGRegularTrain | x | x |
| InsectEPGSmallTrain | x | x |
| Lightning7 | x | x |
| MiddlePhalanxTW | x | x |
| NonInvasiveFetalECGThorax1 | x | x |
| NonInvasiveFetalECGThorax2 | x | x |
| Phoneme | x | x |
| PigAirwayPressure | x | x |
| PigArtPressure | x | x |
| PigCVP | x | x |
| Rock | x | x |
| SemgHandMovementCh2 | x | x |
| ShapesAll | x | x |
| SwedishLeaf | x | x |
| Symbols | x | x |
| UWaveGestureLibraryX | x | x |
| UWaveGestureLibraryY | x | x |
| WordSynonyms | x | x |
| Worms | x | x |
| WormsTwoClass | x | x |
| **Total Missing** | **34** | **34** |

Table A.50 EUPW using EDBI compared to other ensemble schemes over the combined test-train split experiment missing datasets. A total of 7 datasets are excluded. Datasets are missing due to empty clusters forming. To conserve space we exclude PAM from this table and refer to Table A.19 for PAMs missing datasets.

| **Dataset** | **cspa** | **eupw-EDBI-euclidean** | **eupw-EDBI-msm** | **eupw-EDBI-twe** | **hbgf** | **iterative-voting** | **mcla** | **nmf** | **simple-voting** |
|---|---|---|---|---|---|---|---|---|---|
| Crop | x | x | x | x | x | x | x | x | x |
| ElectricDevices | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| HandOutlines | ✓ | x | ✓ | ✓ | x | x | x | x | x |
| NonInvasiveFetalECGThorax1 | ✓ | x | ✓ | ✓ | x | x | x | x | x |
| NonInvasiveFetalECGThorax2 | ✓ | x | ✓ | ✓ | x | x | x | x | x |
| StarLightCurves | x | x | x | x | x | x | x | x | x |
| UWaveGestureLibraryAll | x | x | x | x | x | x | x | x | x |
| **Total Missing** | **4** | **6** | **3** | **3** | **6** | **6** | **6** | **6** | **6** |

Table A.51 ECE compared with the baseline clusterers and the state-of-the-art and soft-DBA over the combined test-train split experiment missing datasets. A total of 38 datasets are excluded. Datasets are missing due to computational runtime exceeding seven days.

| **Dataset** | **ECE** | **k-means-ba-dtw** | **k-means-ba-twe** | **k-means-soft-dba** | **k-sc** | **pam-msm** | **pam-twe** |
|---|---|---|---|---|---|---|---|
| ACSF1 | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Adiac | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| CinCECGTorso | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| CricketY | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Crop | x | ✓ | ✓ | ✓ | ✓ | x | x |
| DiatomSizeReduction | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| EOGHorizontalSignal | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| EOGVerticalSignal | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| EthanolLevel | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| FiftyWords | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Fish | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| FordA | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| FordB | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| HandOutlines | ✓ | x | ✓ | x | ✓ | ✓ | ✓ |
| InlineSkate | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| LargeKitchenAppliances | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| Mallat | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| MixedShapesRegularTrain | ✓ | x | ✓ | x | ✓ | ✓ | ✓ |
| MixedShapesSmallTrain | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax1 | x | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| NonInvasiveFetalECGThorax2 | x | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| OliveOil | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Phoneme | x | x | x | x | ✓ | ✓ | ✓ |
| PigAirwayPressure | x | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| PigArtPressure | x | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| PigCVP | x | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| RefrigerationDevices | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| ScreenType | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| SemgHandGenderCh2 | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| SemgHandMovementCh2 | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| SemgHandSubjectCh2 | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| ShapesAll | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| StarLightCurves | x | x | ✓ | x | ✓ | ✓ | x |
| UWaveGestureLibraryAll | x | x | ✓ | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryX | ✓ | ✓ | ✓ | x | ✓ | ✓ | ✓ |
| UWaveGestureLibraryZ | ✓ | ✓ | ✓ | x | x | ✓ | ✓ |
| WordSynonyms | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Worms | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Total Missing** | **19** | **5** | **1** | **27** | **1** | **1** | **2** |