A

Fast and Accurate Time-Series Clustering

John Paparrizos, Columbia University
Luis Gravano, Columbia University

The proliferation and ubiquity of temporal data across many disciplines has generated substantial interest in the analysis and mining of time series. Clustering is one of the most popular data mining methods, not only due to its exploratory power, but also because it is often a preprocessing step or subroutine for other techniques. In this paper, we present k-Shape and k-MS, two novel algorithms for time-series clustering. k-Shape and k-MS rely on a scalable iterative refinement procedure. As their distance measure, k-Shape and k-MS use SBD, a normalized version of the cross-correlation measure, in order to consider the shapes of time series while comparing them. Based on the properties of SBD, we develop two new methods, namely, ShapeExtraction (SE) and MultiShapesExtraction (MSE), to compute cluster centroids that are used in every iteration to update the assignment of time series to clusters. k-Shape relies on SE to compute a single centroid per cluster based on all time series in each cluster. In contrast, k-MS relies on MSE to compute multiple centroids per cluster, to account for the proximity and spatial distribution of time series in each cluster. To demonstrate the robustness of SBD, k-Shape, and k-MS, we perform an extensive experimental evaluation on 85 datasets against state-of-the-art distance measures and clustering methods for time series using rigorous statistical analysis. SBD, our efficient and parameter-free distance measure, achieves similar accuracy to Dynamic Time Warping (DTW), a highly accurate but computationally expensive distance measure that requires parameter tuning. For clustering, we compare k-Shape and k-MS against scalable and non-scalable partitional, hierarchical, spectral, density-based, and shapelet-based methods, with combinations of the most competitive distance measures. k-Shape outperforms all scalable methods in terms of accuracy. Furthermore, k-Shape also outperforms all non-scalable (and hence impractical) approaches, with one exception, namely, k-medoids with DTW, which achieves similar accuracy. However, unlike k-Shape, this approach requires tuning of its distance measure and is significantly slower than k-Shape. k-MS performs similarly to k-Shape in comparison to rival methods, but k-MS is significantly more accurate than k-Shape. Beyond clustering, we demonstrate the effectiveness of k-Shape to reduce the search space of one-nearest-neighbor classifiers for time series. Overall, SBD, k-Shape, and k-MS emerge as domain-independent, highly accurate, and efficient methods for time-series comparison and clustering, with broad applications.

CCS Concepts: •**Mathematics of computing → Time series analysis; Cluster analysis;** •**Information systems → Clustering; Nearest-neighbor search;**

Additional Key Words and Phrases: Time-series clustering, time-series classification, distance measures

ACM Reference Format:
John Paparrizos and Luis Gravano, Fast and Accurate Time-Series Clustering. *ACM Trans. Datab. Syst.* V, N, Article A (January YYYY), 45 pages.
DOI: 0000001.0000001

This article includes material from [Paparrizos and Gravano 2015] and significantly extends this earlier paper, as discussed in detail in Section 7. This research was supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract number D11PC20153. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government. This material is also based upon work supported by a generous gift from Microsoft Research.
Authors' addresses: John Paparrizos and Luis Gravano, Computer Science Department, Columbia University, 1214 Amsterdam Avenue, New York, NY 10027-7003; emails: {jopa, gravano}@cs.columbia.edu.

DOI: 0000001.0000001

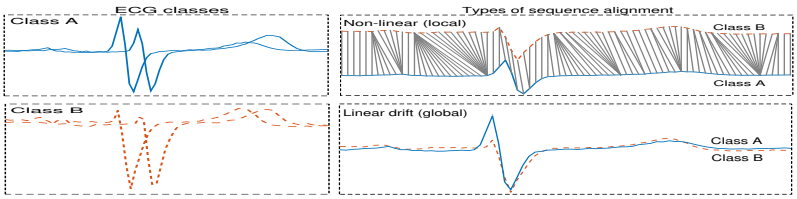


Fig. 1: ECG sequence examples and types of alignments for the two classes of the ECG-FiveDays dataset [Keogh et al. 2015].

1. INTRODUCTION

Temporal, or sequential, data mining deals with problems where data are naturally organized in sequences [Han et al. 2011]. We refer to such data sequences as time-series sequences if they contain explicit information about timing (e.g., as is the case in stock, audio, speech, and video data) or if an ordering on values can be inferred (e.g., as is the case in streams and handwriting data). Large volumes of time-series sequences appear in almost every discipline, including astronomy, biology, meteorology, medicine, finance, robotics, engineering, and others [Keogh et al. 2015; Bar-Joseph et al. 2002; Gavrilov et al. 2000; Goddard et al. 2003; Honda et al. 2002; Mantegna 1999; Ruiz et al. 2012; Uehara and Shimada 2002]. The ubiquity of time series has generated a substantial interest in querying [Agrawal et al. 1993; Ding et al. 2008; Kin-pong and Ada 1999; Korn et al. 1997; Lian et al. 2007; Papapetrou et al. 2011; Shou et al. 2005; Wang et al. 2014], indexing [Cai and Ng 2004; Chen et al. 2007a; Keogh 2006; Keogh et al. 2001; Keogh and Ratanamahatana 2005; Vlachos et al. 2006], classification [Hu et al. 2013; Mueen et al. 2011; Ratanamahatana and Keogh 2004; Ye and Keogh 2009], clustering [Keogh and Lin 2005; Megalooikonomou et al. 2005; Petitjean et al. 2011; Yang and Leskovec 2011; Zakaria et al. 2012], and modeling [Alon et al. 2003; Kalpakis et al. 2001; Xiong and Yeung 2002] of such data.

Among all techniques applied to time-series sequences, clustering is one of the most widely used as it does not rely on costly human supervision or time-consuming annotation of data. With clustering, we can identify and summarize interesting patterns and correlations in the underlying data [Halkidi et al. 2001]. In the last few decades, clustering of time-series sequences has received significant attention [Bagnall and Janacek 2004; Das et al. 1998; Gavrilov et al. 2000; Li et al. 1998; Oates 1999; Petitjean et al. 2011; Rakthanmanon et al. 2011; Yang and Leskovec 2011; Zakaria et al. 2012], not only as a powerful standalone exploratory method, but also as a preprocessing step or subroutine for other tasks.

Most time-series analysis techniques, including clustering, critically depend on the choice of distance measure. A key issue when comparing two time-series sequences is how to handle the variety of distortions, as we will discuss, that are characteristic of the sequences. To illustrate this point, consider the well-known ECGFiveDays dataset [Keogh et al. 2015], with ECG sequences recorded for the same patient on two different days. While the sequences seem similar overall, they exhibit patterns that belong in one of two distinct classes (see Figure 1): Class A is characterized by a sharp rise, a drop, and another gradual increase while Class B is characterized by a gradual increase, a drop, and another gradual increase. Ideally, a *shape-based* clustering method should generate a partition similar to the classes shown in Figure 1, where sequences exhibiting similar patterns are placed into the same cluster based on their *shape* similarity, regardless of differences in amplitude and phase. As the notion of shape cannot be precisely defined, dozens of distance measures have been proposed [Chen and Ng 2004; Chen et al. 2005; Chen et al. 2007b; Ding et al. 2008; Faloutsos et al. 1994; Frentzos et al. 2007; Morse and Patel 2007; Stefan et al. 2013; Vlachos et al. 2002; Wang et al. 2013] to offer invariances to multiple inherent distortions in the data. However, it has been shown that distance measures offering invariances to amplitude and phase

perform exceptionally well [Ding et al. 2008; Wang et al. 2013] and, hence, such distance measures are used for shape-based clustering [Meesrikamolkul et al. 2012; Niennattrakul and Ratanamahatana 2009; Petitjean et al. 2011; Yang and Leskovec 2011].

Due to these difficulties and the different needs for invariances from one domain to another, more attention has been given to the creation of new distance measures rather than to the creation of new clustering algorithms. It is generally believed that the choice of distance measure is more important than the clustering algorithm itself [Batista et al. 2013]. As a consequence, time-series clustering relies mostly on classic clustering methods, either by replacing the default distance measure with one that is more appropriate for time series, or by transforming time series into "flat" data so that existing clustering algorithms can be directly used [Warren Liao 2005]. However, the choice of clustering method can affect: (i) accuracy, as every method expresses homogeneity and separation of clusters differently; and (ii) efficiency, as the computational cost differs from one method to another. For example, spectral clustering [Filippone et al. 2008] or certain variants of hierarchical clustering [Kaufman and Rousseeuw 2009] are more appropriate to identify density-based clusters (i.e., areas of higher density than the remainder of the data) than partitional methods such as k-means [MacQueen 1967] or k-medoids [Kaufman and Rousseeuw 2009]. On the other hand, k-means is more efficient than hierarchical, spectral, or k-medoids methods.

Unfortunately, state-of-the-art approaches for shape-based clustering, which use partitional methods with distance measures that are scale- and shift-invariant, suffer from three main drawbacks: (i) these approaches cannot scale to large volumes of data as they depend on computationally expensive methods or distance measures [Meesrikamolkul et al. 2012; Niennattrakul and Ratanamahatana 2009; Petitjean et al. 2011; Yang and Leskovec 2011]; (ii) these approaches have been developed for particular domains [Yang and Leskovec 2011] or their effectiveness has only been shown for a limited number of datasets [Meesrikamolkul et al. 2012; Niennattrakul and Ratanamahatana 2009]; and (iii) these approaches are sensitive to outliers and noise as they do not take into consideration the proximity and spatial distribution of time series. Moreover, the most successful shape-based clustering methods handle phase invariance through a local, non-linear alignment of the sequence coordinates, even though a global, linear alignment is often adequate and in certain cases more accurate. For example, for the ECG dataset in Figure 1, an efficient linear drift can reveal the underlying differences in patterns of sequences of two classes, whereas an expensive non-linear alignment attempts to match every corresponding increase or drop of each sequence, making it difficult to distinguish the two classes (see Figure 1). Importantly, in contrast to linear alignment, the effectiveness of non-linear alignments might degrade for time series with large variability in their prefix and suffix coordinates [Silva et al. 2016]. Additionally, to the best of our knowledge, these approaches have never been extensively evaluated against each other, against other partitional methods, or against different approaches such as hierarchical or spectral methods. We present such an experimental evaluation, as discussed below.

In this paper, we propose k-Shape and k-MS, two novel algorithms for shape-based time-series clustering that are efficient and domain independent. k-Shape and k-MS are based on a scalable iterative refinement procedure similar to the one used by the k-means algorithm, but with significant differences. Specifically, k-Shape and k-MS use both a different distance measure and a different method for centroid computation from those of k-means. As argued above, k-Shape and k-MS attempt to preserve the shapes of time series while comparing them. Furthermore, k-MS also considers the proximity and spatial distribution of time series. To do so, k-Shape and k-MS require a distance measure that is invariant to scaling and shifting. Unlike other clustering methods [Meesrikamolkul et al. 2012; Petitjean et al. 2011; Yang and Leskovec 2011], we adapt the cross-correlation measure and we show: (i) how we can derive in a principled manner a time-series distance measure that is scale- and shift-invariant; and (ii) how this distance measure can be computed efficiently by exploiting intrinsic characteristics of Fourier transform algorithms. Based on the properties of this

normalized version of cross-correlation, namely, SBD, we develop ShapeExtraction (SE) and MultiShapesExtraction (MSE), two novel methods to compute cluster centroids, which are used in every iteration to update the assignment of time series to clusters. k-Shape relies on SE to compute a single centroid per cluster based on the entire set of time series in each cluster. In contrast, k-MS relies on MSE to compute multiple centroids per cluster in order to consider the proximity and spatial distribution of time series in each cluster.

To demonstrate the effectiveness of SBD, k-Shape, and k-MS we have conducted an extensive experimental evaluation over 85 datasets and compared the state-of-the-art distance measures and clustering approaches for time series using rigorous statistical analysis. We took steps to ensure the reproducibility of our results, including making available our source code as well as using public datasets. The current article substantially extends the analysis and experimental evaluation in [Paparrizos and Gravano 2015] and, importantly, confirms the earlier findings on a significantly larger set of datasets than the 48 datasets in our original paper. Specifically, our results show that SBD is competitive, outperforming Euclidean distance (ED) [Faloutsos et al. 1994], and achieving similar accuracy as constrained Dynamic Time Warping (cDTW) [Sakoe and Chiba 1978], one of the best performing distance measures [Wang et al. 2013], without requiring any tuning and performing significantly faster. Interestingly, our results further suggest that only the supervised tuning of cDTW leads to significant improvements in classification accuracy of cDTW over the unconstrained Dynamic Time Warping (DTW). In contrast, DTW outperforms cDTW in clustering accuracy, which contradicts the belief that cDTW generally improves the accuracy of DTW.

For time-series clustering, we show that the k-means algorithm with ED, in contrast to what has been reported in the literature, is a robust approach and that inadequate modifications of the distance measure and the centroid computation can reduce its performance. Similarly to the earlier findings in [Paparrizos and Gravano 2015], our new results in the expanded set of datasets suggest that partitional methods outperform hierarchical and spectral methods with the most competitive distance measures. Additionally, in this article, we compare partitional methods against density-based and shapelet-based clustering methods, which attempt to isolate and ignore outliers in time series, and show the superiority of partitional methods. These results indicate that the choice of algorithm, which is sometimes believed to be less important than that of distance measure, is as critical as the choice of distance measure. Similarly, we show that key characteristics of the clustering methods, such as the linkage criterion in hierarchical clustering, can significantly affect the accuracy of the clustering results, whereas the choice of distance measure is many times less important.

Our extensive experimental evaluation shows that k-Shape outperforms all scalable and non-scalable partitional, hierarchical, spectral, density-based, and shapelet-based methods in terms of accuracy, with the only exception of one existing approach that achieves similar accuracy results, namely, k-medoids with cDTW. However, there are problems with this approach that can be avoided with k-Shape: (i) the requirement of k-medoids to compute the dissimilarity matrix makes it unable to scale and particularly slow, two orders of magnitude slower than k-Shape; (ii) its distance measure requires tuning, either through automated methods that rely on labeling of instances or through the help of a domain expert; this requirement is problematic for clustering, which is an unsupervised task. In contrast, k-Shape uses a parameter-free and efficient distance measure.

k-MS behaves similarly to k-Shape in comparison to scalable and non-scalable rival methods but, importantly, k-MS is significantly more accurate than k-Shape on datasets with large variance in the proximity and spatial distribution of time series. Therefore, k-MS is suitable to cluster time series in the presence of outliers and noise.

Overall, k-Shape and k-MS are scalable — yet accurate — choices for time-series clustering that achieve state-of-the-art performance across different domains and are particularly effective for applications involving similar but out-of-phase sequences. In addition to our clustering results, and inspired by the work by [Petitjean et al. 2014; 2015], we also show

how k-Shape can also lead to efficient techniques for other related time-series tasks. Specifically, we show that we can rely on k-Shape as a subroutine to effectively reduce the search space for one-nearest-neighbor time-series classification algorithms.

Our article starts with a review of the state of the art for clustering time series, as well as with a precise definition of our problem of focus (Section 2). We continue as follows:

— We show how to derive SBD, a scale-, translate-, and shift-invariant distance measure, in a principled manner from cross-correlation and how to efficiently compute this measure by exploiting intrinsic characteristics of Fourier transform algorithms (Section 3.1).
— We present two novel methods to compute a single centroid or multiple centroids per cluster when the SBD distance measure is used (Section 3.2).
— We develop k-Shape and k-MS, two algorithms for time-series clustering (Section 3.3).
— We present NSC, a one-nearest-neighbor classification algorithm that relies on k-Shape as subroutine to reduce the search space of one-nearest-neighbor algorithms (Section 4).
— We perform an extensive experimental evaluation of our ideas (Sections 5 and 6).

Finally, we conclude with a discussion of related work (Section 7) and the implications of our work (Section 8). This article includes material from [Paparrizos and Gravano 2015] and significantly extends this earlier paper, as discussed in detail in Section 7. Additionally, a summary of the paper in [Paparrizos and Gravano 2015] appears in a SIGMOD Record "Research Highlights" special issue [Paparrizos and Gravano 2016].

2. PRELIMINARIES

In this section, we review the relevant theoretical background (Section 2.1). We discuss common distortions in time series (Section 2.2) and the most popular distance measures for such data (Section 2.3). Then, we summarize existing approaches for clustering time-series data (Section 2.4) and for centroid computation (Section 2.5). Finally, we define our clustering problem of focus (Section 2.6).

2.1. Theoretical Background

Hardness of clustering: Clustering is the general problem of partitioning n observations into k clusters, where a *cluster* is characterized with the notions of homogeneity — the similarity of observations within a cluster — and separation — the dissimilarity of observations from different clusters. Even though many clustering criteria to capture homogeneity and separation have been proposed [Hansen and Jaumard 1997], the minimum within-cluster sum of squared distances is most commonly used as it expresses both of them. Given a set of n observations $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$, where $\vec{x}_i \in \mathbb{R}^m$, and the number of clusters $k < n$, the objective is to partition X into k pairwise-disjoint clusters $P = \{p_1, \ldots, p_k\}$, such that the within-cluster sum of squared distances is minimized:

$$P^* = \arg\min_{P} \sum_{j=1}^{k} \sum_{\vec{x}_i \in p_j} dist(\vec{x}_i, \vec{c}_j)^2 \tag{1}$$

where $\vec{c}_j$ is the centroid of partition $p_j \in P$. In Euclidean space this is an NP-hard optimization problem for $k \geq 2$ [Aloise et al. 2009], even for number of dimensions $m = 2$ [Mahajan et al. 2009]. Because finding a global optimum is difficult, heuristics such as the k-means method [MacQueen 1967] are often used to find a local optimum. Specifically, k-means randomly assigns the data points into k clusters and then uses an iterative procedure that performs two steps in every iteration: (i) in the assignment step, every data point is assigned to the cluster of its nearest centroid, which is determined with the use of a distance function; (ii) in the refinement step, the centroids of the clusters are updated to reflect the changes in cluster memberships. The algorithm terminates either when there is no change in cluster memberships or when the maximum number of iterations is reached.

Steiner's sequence: In the refinement step, k-means computes new centroids to serve as representatives of the clusters. The centroid is defined as the data point that minimizes the sum of squared distances to all other data points and, hence, it depends on the distance measure used. Finding such a centroid is known as the Steiner's sequence problem [Petitjean and Gançarski 2012]: given a partition $p_j \in P$, the corresponding centroid $\vec{c}_j$ needs to fulfill:

$$\vec{c}_j = \arg\min_{\vec{w}} \sum_{\vec{x}_i \in p_j} dist(\vec{w}, \vec{x}_i)^2, \quad \vec{w} \in \mathbb{R}^m \tag{2}$$

When ED is used, the centroid can be computed with the arithmetic mean property [Dimitriadou et al. 2002]. In many cases where alignment of observations is required, this problem is referred to as the multiple sequence alignment problem, which is known to be NP-complete [Wang and Jiang 1994]. In the context of time series, DTW (see Section 2.3) is the most widely used measure to compare time-series sequences with alignment, and many heuristics have been proposed to find the average sequence under DTW (see Section 2.5).

2.2. Time-Series Invariances

Sequences are often distorted in some way and distance measures need to satisfy a number of invariances to compare sequences meaningfully. We now review common distortions and their invariances. For a more detailed review, see [Batista et al. 2013].

Scaling and translation invariances: In many cases, it is useful to recognize the similarity of sequences despite differences in amplitude and offset. In other words, transforming a sequence $\vec{x}$ as $\vec{x}' = a\vec{x} + b$, where a and b are constants, should not change $\vec{x}$'s similarity to other sequences. For example, these invariances might be useful to analyze seasonal variations in currency values on foreign exchange markets without being biased by inflation.

Shift invariance: When two sequences are similar but differ in phase (global alignment) or when there are regions of the sequences that are aligned and others are not (local alignment), we might still need to consider them similar. For example, heartbeats can be out of phase depending on when we start taking the measurements (global alignment) and handwritings of a phrase from different people will need alignment depending on the size of the letters and on the spaces between words (local alignment).

Uniform scaling invariance: Sequences that differ in length require either stretching of the shorter sequence or shrinking of the longer sequence to compare them effectively. This invariance is required for heartbeats with measurement periods of different duration.

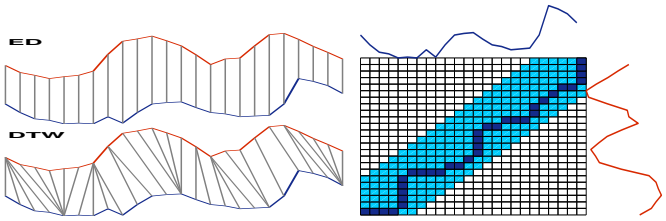
Occlusion invariance: When subsequences are missing, we can still compare the sequences by ignoring the subsequences that do not match well. This invariance is useful in handwritings if there is a typo or a letter is missing.

Complexity invariance: When sequences have similar shape but different complexities, we might want to make them have low or high similarity based on the application. For example, audio signals that were recorded indoors and outdoors might be considered similar, despite the fact that outdoor signals will be more noisy than indoor signals.

For many tasks, some or all of the above invariances are required when we compare time-series sequences. To satisfy the appropriate invariances, we could preprocess the data to eliminate the corresponding distortions before clustering. For example, by z-normalizing [Goldin and Kanellakis 1995] the data we can achieve the scaling and translation invariances. However, for invariances that cannot be trivially achieved with a preprocessing step, we can define sophisticated distance measures that offer distortion invariances. In the next section, we review the most common such distance measures.

2.3. Time-Series Distance Measures

The two state-of-the-art approaches for time-series comparison first z-normalize the sequences and then use a distance to determine their similarity, and possibly capture more invariances [Rakthanmanon et al. 2012]. The most widely used distance is ED [Faloutsos



(a) (b)

Fig. 2: Similarity computation: (a) alignment under ED (top) and DTW (bottom), (b) Sakoe-Chiba band with a warping window of 5 cells (light cells in band) and the warping path computed under cDTW (dark cells in band).

et al. 1994]. ED compares two time series $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{y} = (y_1, \ldots, y_m)$ as follows:

$$ED(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \tag{3}$$

Another popular distance measure is DTW [Sakoe and Chiba 1978]. DTW can be seen as an extension of ED that offers a local (non-linear) alignment. To achieve that, an m-by-m matrix M is constructed, with the ED between any two points of $\vec{x}$ and $\vec{y}$. A *warping path* $W = \{w_1, w_2, \ldots, w_r\}$, with $r \geq m$, is a contiguous set of matrix elements that defines a mapping between $\vec{x}$ and $\vec{y}$ under several constraints [Keogh and Ratanamahatana 2005]:

$$DTW(\vec{x}, \vec{y}) = \min \sqrt{\sum_{i=1}^{r} w_i} \tag{4}$$

This path can be computed on matrix M with dynamic programming as follows:

$$\gamma(i,j) = ED(i,j) + \min\{\gamma(i-1,j-1), \gamma(i-1,j), \gamma(i,j-1)\}.$$

It is common practice to constrain the warping path to visit only a subset of cells on matrix M. The shape of the subset matrix is called *band* and the width of the band is called *warping window*. The most frequently used band for cDTW is the Sakoe-Chiba band [Sakoe and Chiba 1978]. Figure 2a shows the difference in alignments of two sequences offered by ED and DTW, whereas Figure 2b presents the computation of the warping path (dark cells) for cDTW constrained by the Sakoe-Chiba band with width 5 cells (light cells).

Recently, Wang et al. [Wang et al. 2013] evaluated 9 distance measures and several variants thereof. They found that ED is the most efficient measure with a reasonably high accuracy, and that DTW and cDTW perform exceptionally well in comparison to other measures. cDTW is slightly better than DTW and significantly reduces the computation time. Several optimizations have been proposed to further speed up cDTW [Rakthanmanon et al. 2012]. Next, we review clustering methods that can utilize these distance measures.

2.4. Time-Series Clustering Algorithms

Several methods have been proposed to cluster time series. All approaches generally modify existing algorithms, either by replacing the default distance measures with a version that is more suitable for comparing time series (raw-based methods), or by transforming the sequences into "flat" data so that they can be directly used in classic algorithms (feature- and model-based methods) [Warren Liao 2005]. Raw-based approaches can easily leverage the vast literature on distance measures (see Section 2.3), which has shown that invariances offered by certain measures, such as DTW, are general and, hence, suitable for almost every domain [Ding et al. 2008]. In contrast, feature- and model-based approaches are usually domain-dependent and applications on different domains require that we modify the features or models. Because of these drawbacks of feature- and model-based methods, in this paper we follow a raw-based approach.

The four most popular raw-based methods are agglomerative hierarchical, spectral, density-based, and partitional clustering [Batista et al. 2013; Warren Liao 2005]. For hierarchical clustering, the most widely used "linkage" criteria are the single, average, and complete linkage variants [Kaufman and Rousseeuw 2009]. Spectral clustering [Ng et al. 2002] has recently started receiving attention [Batista et al. 2013] due to its success over other types of data [Filippone et al. 2008]. Similarly, density-based clustering [Ester et al. 1996] has gained popularity due to its ability to handle outliers in datasets [Begum et al. 2015]. Among partitional methods, k-means [MacQueen 1967] and k-medoids [Kaufman and Rousseeuw 2009] are the most representative examples. When partitional methods use distances that offer invariances to scaling, translation, and shifting, we consider them as shape-based approaches. From these methods, k-medoids is usually preferred [Warren Liao 2005]: unlike k-means, k-medoids computes the dissimilarity matrix of all data sequences and uses actual sequences as cluster centroids; in contrast, k-means requires the computation of artificial sequences as centroids, which hinders the easy adaptation of distance measures other than ED (see Section 2.1). However, from all these methods, only the k-means class of algorithms can scale linearly with the size of the datasets. Recently, k-means was modified to work with: (i) DTW [Petitjean et al. 2011] and (ii) a distance that offers pairwise scaling and shifting of sequences [Yang and Leskovec 2011]. Both of these modifications rely on new approaches to compute cluster centroids that we will review next.

2.5. Time-Series Averaging Techniques

The computation of an average sequence or, in the context of clustering, a centroid, is a difficult task and it critically depends on the distance measure used to compare time series. We now review the state-of-the-art methods for the computation of an average sequence.

With ED, the property of arithmetic mean is used to compute an average sequence (e.g., as is the case in the centroid computation of k-means). However, as DTW is more appropriate for many time-series tasks [Keogh and Ratanamahatana 2005; Rakthanmanon et al. 2012], several methods have been proposed to average sequences under DTW. Nonlinear alignment and averaging filters (NLAAF) [Gupta et al. 1996] uses a simple pairwise method where each coordinate of the average sequence is calculated as the center of the mapping produced by DTW. This method is applied sequentially to pairs of sequences until only one pair is left. Prioritized shape averaging (PSA) [Niennattrakul and Ratanamahatana 2009] uses a hierarchical method to average sequences. The coordinates of an average sequence are computed as the weighted center of the coordinates of two time-series sequences that were coupled by DTW. Initially, all sequences have weight one, and each average sequence produced in the nodes of the tree has a weight that corresponds to the number of sequences it averages. To avoid the high computation cost of previous approaches, Ranking Shape-based Template Matching Framework (RSTMF) [Meesrikamolkul et al. 2012] approximates an ordering of the time-series sequences by looking at the distances of sequences to all other cluster centroids, instead of computing the distances of all pairs of sequences.

Several drawbacks of these methods have led to the creation of a more robust technique called Dynamic Time Warping Barycenter Averaging (DBA) [Petitjean et al. 2011], which iteratively refines the coordinates of a sequence initially picked from the data. Each coordinate of the average sequence is updated with the use of barycenter of one or more coordinates of the other sequences that were associated with the use of DTW. Among all these methods, DBA seems to be the most efficient and accurate averaging approach when DTW is used [Petitjean et al. 2011]. Another averaging technique that is based on matrix decomposition was proposed as part of K-Spectral Centroid Clustering (KSC) [Yang and Leskovec 2011], to compute the centroid of a cluster when a distance measure for pairwise scaling and shifting is used. In our approach, which we will present in Section 3, we also rely on matrix decomposition to compute centroids.

2.6. Problem Definition

We address the problem of domain-independent, accurate, and scalable clustering of time series into k clusters, for a given value of the target number of clusters k.[1] Even though different domains might require different invariances to data distortions (see Section 2.2), we focus on distance measures that offer invariances to scaling and shifting, which are generally sufficient (see Section 2.3) [Ding et al. 2008]. Furthermore, to easily adopt such distance measures, we focus our analysis on raw-based clustering approaches, as we argued in Section 2.4. Next, we introduce k-Shape and k-MS, our novel time-series clustering algorithms.

3. K-SHAPE AND K-MS, TWO SHAPE-BASED CLUSTERING ALGORITHMS

Our objective is to develop domain-independent, accurate, and scalable algorithms for time-series clustering with a distance measure that is invariant to scaling and shifting. In this section, we first discuss our distance measure, namely, SBD, which is based on cross-correlation (Section 3.1). Based on SBD, we then present two methods to compute centroids of time-series clusters (Section 3.2). Finally, we propose k-Shape and k-MS, two novel clustering algorithms that rely on an iterative refinement procedure that scales linearly in the number of sequences and generates homogeneous and well-separated clusters (Section 3.3).

3.1. Time-Series Shape Similarity

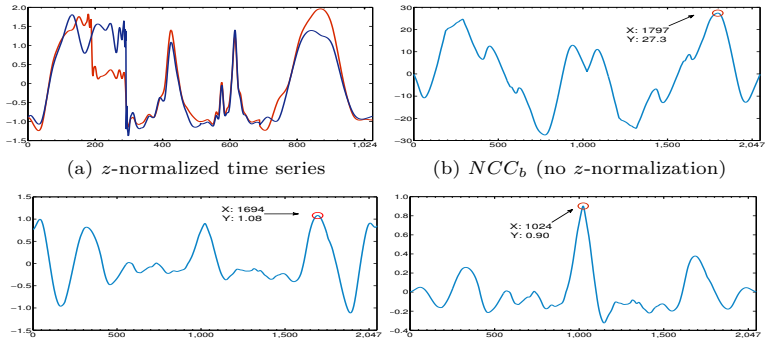
Capturing shape-based similarity requires measures that can handle distortions in amplitude and phase. Unfortunately, the best performing measures offering invariances to these distortions, such as DTW, are computationally expensive (see Section 2.3). To circumvent this efficiency limitation, we adapt a normalized version of cross-correlation.

Cross-correlation is a measure of similarity that compares points of time-lagged signals one-to-one. Cross-correlation is widely used in signal processing to compare sequences that differ in phase (global alignment). In contrast, DTW is a measure that is able to compare regions of sequences (local alignment). After DTW [Berndt and Clifford 1994], research on the problem of time-series comparison has mostly focused on elastic measures that compare one-to-many or one-to-none points [Chen and Ng 2004; Chen et al. 2005; Keogh and Ratanamahatana 2005; Morse and Patel 2007; Vlachos et al. 2002; Wang et al. 2013]. Therefore, the relative performance of elastic measures against cross-correlation remains largely unexplored. Different needs from one domain or application to another hinder the process of finding appropriate normalizations for the data and the cross-correlation measure. Moreover, inefficient implementations of cross-correlation can make it particularly slow. In the rest of this section, we show how to address these drawbacks. Specifically, we show how to choose normalizations that are domain-independent and efficient, and lead to a shape-based distance for comparing time series efficiently and effectively.

Cross-correlation: Cross-correlation is a measure with which we can determine the similarity of two sequences $\vec{x} = (x_1, \ldots, x_m)$ and $\vec{y} = (y_1, \ldots, y_m)$, even if they are not properly aligned.[2] To achieve shift-invariance, cross-correlation keeps $\vec{y}$ static and slides $\vec{x}$ over $\vec{y}$ to compute their inner product for each *shift* s of $\vec{x}$. We denote a shift of a sequence as follows:

$$\vec{x}_{(s)} = \begin{cases} (\overbrace{0, \ldots, 0}^{|s|}, x_1, x_2, \ldots, x_{m-s}), & s \geq 0 \\ (x_{1-s}, \ldots, x_{m-1}, x_m, \underbrace{0, \ldots, 0}_{|s|}), & s < 0 \end{cases} \tag{5}$$

[1] Although the estimation of k is difficult without a gold standard, we can do so by varying k and evaluating clustering quality with criteria to capture information intrinsic to the data [Kaufman and Rousseeuw 2009].
[2] For simplicity, we consider sequences of equal length even though cross-correlation can be computed on sequences of different length.



(a) z-normalized time series (b) NCC_b (no z-normalization)

(c) NCC_u with z-normalization (d) NCC_c with z-normalization

Fig. 3: Time-series and cross-correlation normalizations.

When all possible shifts $\vec{x}_{(s)}$ are considered, with $s \in [-m, m]$, we produce $CC_w(\vec{x}, \vec{y}) = (c_1, \ldots, c_w)$, the cross-correlation sequence with length $2m-1$, defined as follows:

$$CC_w(\vec{x}, \vec{y}) = R_{w-m}(\vec{x}, \vec{y}), \quad w \in \{1, 2, \ldots, 2m-1\} \tag{6}$$

where $R_{w-m}(\vec{x}, \vec{y})$ is computed, in turn, as:

$$R_k(\vec{x}, \vec{y}) = \begin{cases} \sum_{l=1}^{m-k} x_{l+k} \cdot y_l, & k \geq 0 \\ R_{-k}(\vec{y}, \vec{x}), & k < 0 \end{cases} \tag{7}$$

Our goal is to compute the position w at which $CC_w(\vec{x}, \vec{y})$ is maximized. Based on this value of w, the optimal shift of $\vec{x}$ with respect to $\vec{y}$ is then $\vec{x}_{(s)}$, where $s = m - w$.

Depending on the domain or the application, different normalizations for $CC_w(\vec{x}, \vec{y})$ might be required. The most common normalizations are the biased estimator, NCC_b, the unbiased estimator, NCC_u, and the coefficient normalization, NCC_c, defined as follows:

$$NCC_q(\vec{x}, \vec{y}) = \begin{cases} \frac{CC_w(\vec{x}, \vec{y})}{m}, & q = \text{“}b\text{”}\ (NCC_b) \\ \frac{CC_w(\vec{x}, \vec{y})}{m - |w-m|}, & q = \text{“}u\text{”}\ (NCC_u) \\ \frac{CC_w(\vec{x}, \vec{y})}{\sqrt{R_0(\vec{x}, \vec{x}) \cdot R_0(\vec{y}, \vec{y})}}, & q = \text{“}c\text{”}\ (NCC_c) \end{cases} \tag{8}$$

Beyond the cross-correlation normalizations, time series might also require normalization to remove inherent distortions. Figure 3 illustrates how the cross-correlation normalizations for two sequences $\vec{x}$ and $\vec{y}$ of length $m = 1024$ are affected by time-series normalizations. (Appendix A of [Paparrizos and Gravano 2015] elaborates on the classification accuracy of cross-correlation variants under other time-series normalizations.) Independently of the normalization applied to $CC_w(\vec{x}, \vec{y})$, the produced sequence will have length 2047. Initially, in Figure 3a, we remove differences in amplitude by z-normalizing $\vec{x}$ and $\vec{y}$ in order to show that they are aligned and, hence, no shifting is required. If $CC_w(\vec{x}, \vec{y})$ is maximized for $w \in [1025, 2047]$ (or $w \in [1, 1023]$), the $\vec{x}$ sequence should be shifted by $w - 1024$ to the left (or $1024 - w$ to the right). Otherwise, if $w = 1024$, $\vec{x}$ and $\vec{y}$ are properly aligned, which is what we expect in our example. Figure 3b shows that if we do not z-normalize $\vec{x}$ and $\vec{y}$, and we use the biased estimator, then NCC_b is maximized at $w = 1797$, which indicates a shifting of the $\vec{x}$ sequence to the left $1797 - 1024 = 773$ times. If we z-normalize $\vec{x}$ and $\vec{y}$, and use the unbiased estimator, then NCC_u is maximized at $w = 1694$, which indicates a shifting of the $\vec{x}$ sequence to the left $1694 - 1024 = 670$ times (Figure 3c). Finally, if we z-normalize $\vec{x}$ and $\vec{y}$, and use the coefficient normalization, then NCC_c is maximized at $w = 1024$, which indicates that no shifting is required (Figure 3d).

As illustrated by the example above, normalizations of the data and the cross-correlation measure can have a significant impact on the cross-correlation sequence produced, which

makes the creation of a distance measure a non-trivial task. Furthermore, as we have seen in Figure 3, cross-correlation sequences produced by pairwise comparisons of time series will differ in amplitude based on the normalizations. Thus, a normalization that produces values within a specified range should be used in order to meaningfully compare such sequences.

Shape-based distance (SBD): To devise a shape-based distance measure, and based on the previous discussion, we use the coefficient normalization that gives values between -1 and 1, regardless of the data normalization. Coefficient normalization divides the cross-correlation sequence by the geometric mean of autocorrelations of the individual sequences. After normalization of the sequence, we detect the position w where $NCC_c(\vec{x},\vec{y})$ is maximized and we derive the following distance measure:

$$SBD(\vec{x},\vec{y}) = 1 - \max_w \left(\frac{CC_w(\vec{x},\vec{y})}{\sqrt{R_0(\vec{x},\vec{x}) \cdot R_0(\vec{y},\vec{y})}} \right) \tag{9}$$

which takes values between 0 to 2, with 0 indicating perfect similarity for time series.

Up to now we have addressed shift invariance. For scaling invariance, we transform each sequence $\vec{x}$ into $\vec{x}' = \frac{\vec{x}-\mu}{\sigma}$, so that its mean μ is zero and its standard deviation σ is one.

Efficient computation of SBD: From Equation 6, the computation of $CC_w(\vec{x},\vec{y})$ for all values of w requires $\mathcal{O}(m^2)$ time, where m is the time-series length. The convolution theorem [Katznelson 2004] states that the convolution of two time series can be computed as the Inverse Discrete Fourier Transform (IDFT) of the product of the individual Discrete Fourier Transforms (DFT) of the time series, where DFT is:

$$\mathcal{F}(x_k) = \sum_{r=0}^{|\vec{x}|-1} x_r e^{\frac{-2jrk\pi}{|\vec{x}|}}, \quad k = 0, \ldots, |\vec{x}| - 1 \tag{10}$$

and IDFT is:

$$\mathcal{F}^{-1}(x_r) = \frac{1}{|\vec{x}|} \sum_{k=0}^{|\vec{x}|-1} \mathcal{F}(x_k) e^{\frac{2jrk\pi}{|\vec{x}|}}, \quad r = 0, \ldots, |\vec{x}| - 1 \tag{11}$$

where $j = \sqrt{-1}$. Cross-correlation is then computed as the convolution of two time series if one sequence is first reversed in time, $\vec{x}^{(t)} = \vec{x}^{(-t)}$ [Katznelson 2004], which equals taking the complex conjugate (represented by $*$) in the frequency domain. Thus, Equation 6 can be computed for every m as:

$$CC(\vec{x},\vec{y}) = \mathcal{F}^{-1}\{\mathcal{F}(\vec{x}) * \mathcal{F}(\vec{y})\} \tag{12}$$

However, DFT and IDFT still require $\mathcal{O}(m^2)$ time. By using a Fast Fourier Transform (FFT) algorithm [Cooley and Tukey 1965], the time reduces to $\mathcal{O}(m \cdot \log(m))$. Data and cross-correlation normalizations can also be efficiently computed; thus the overall time complexity of SBD remains $\mathcal{O}(m \cdot \log(m))$. Importantly, by exploiting intrinsic characteristics of FFT algorithms, we can further improve the performance of SBD. Specifically, recursive algorithms compute an FFT by dividing it into pieces of power-of-two size [Frigo and Johnson 2005]. Therefore, when $CC(\vec{x},\vec{y})$ is not an exact power of two we pad $\vec{x}$ and $\vec{y}$ with zeros to reach the next power-of-two length after $2m-1$. This important property of FFT algorithms, which is not often exploited in the literature (e.g., [Mueen et al. 2014; Zakaria et al. 2012]), leads to a significant improvement in the performance of SBD, as we show in Section 6.1. Algorithm 1 outlines how this efficient and parameter-free measure is computed in a few lines of code using modern mathematical software.

In this section, we showed effective cross-correlation and data normalizations to derive a shape-based distance measure. Importantly, we also discussed how cross-correlation can be efficiently computed. In our experimental evaluation (Sections 5 and 6), we will show that SBD is highly competitive, achieving similar results to cDTW while being significantly faster. We now turn to the critical problem of extracting cluster centroids, to represent the cluster data consistently with the shape-based distance measure described above.

Algorithm 1: $[dist, y'] = SBD(x, y)$

Input: Two z-normalized sequences x and y
Output: Dissimilarity $dist$ of x and y
Aligned sequence y' of y towards x

$length = 2^{nextpower2(2*length(x)-1)}$
$CC = IFFT\{FFT(x, length) * FFT(y, length)\}$ // Equation 12
$NCC_c = \frac{CC}{||x|| \, ||y||}$ // Equation 8
$[value, index] = max(NCC_c)$
$dist = 1 - value$ // Equation 9
$shift = index - length(x)$
if $shift \geq 0$ **then**
  $y' = [zeros(1, shift), y(1 : end - shift)]$ // Equation 5
else
  $y' = [y(1 - shift : end), zeros(1, -shift)]$ // Equation 5

3.2. Time-Series Shape Extraction Methods

Many tasks in time-series analysis rely on methods that effectively summarize a set of time series by a small number of sequences and, in some cases, by only one sequence. These summary sequences are often referred to as *average sequences* or, in the context of clustering, *centroids*. The extraction of meaningful centroids is a challenging task that critically depends on the choice of distance measure (see Section 2.1). We now show how to determine such centroids for time-series clustering for SBD, to capture shared characteristics of the underlying data. Specifically, we present ShapeExtraction, a method that extracts a single centroid to summarize an entire set of time series, and MultiShapesExtraction, a method that extracts multiple centroids to summarize an entire set of time series.

ShapeExtraction (SE): The easiest way to extract an average sequence from a set of sequences is to compute each coordinate of the average sequence as the arithmetic mean of the corresponding coordinates of all sequences. This approach is used by k-means, the most popular clustering method. In Figure 4, the solid line shows the centroid for "Class A" in the ECGFiveDays dataset of Figure 1: these centroids do not offer invariances to scaling and shifting and, therefore, such centroids do not capture effectively the class characteristics.

To avoid such problems, we cast the centroid computation as an optimization problem where the objective is to find the minimizer of the sum of squared distances to all other time series sequences (Equation 2). However, as cross-correlation intuitively captures the similarity — rather than the dissimilarity — of time series, we can express the computed sequence as the maximizer $\mu_k^\star$ of the squared similarities to all other time-series sequences. By rewriting Equation 2 as a maximization problem and from Equation 8, we obtain:

$$\begin{aligned} \vec{\mu_k}^\star &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \sum_{\vec{x_i} \in P_k} NCC_c(\vec{x_i}, \vec{\mu_k})^2 \\ &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \sum_{\vec{x_i} \in P_k} \left(\max_w \frac{CC_w(\vec{x_i}, \vec{\mu_k})}{\sqrt{R_0(\vec{x_i}, \vec{x_i}) \cdot R_0(\vec{\mu_k}, \vec{\mu_k})}} \right)^2 \end{aligned} \tag{13}$$

Equation 13 requires the computation of an optimal shift for every $\vec{x_i} \in P_k$. As this approach is used in the context of iterative clustering, we use the previously computed centroid as reference and align all sequences towards this reference sequence.[3]

This is a reasonable choice because the previous centroid will be very close to the new centroid. For this alignment, we use SBD, which identifies an optimal shift for every $\vec{x_i} \in P_k$. Subsequently, as sequences are already aligned towards a reference sequence before the centroid computation, we can also omit the denominator of Equation 13. Then, by

[3]For simplicity, we consider sequences of equal length. In cases where sequences are of different length, we first pad with zeros all sequences to reach the length of the longest sequence.

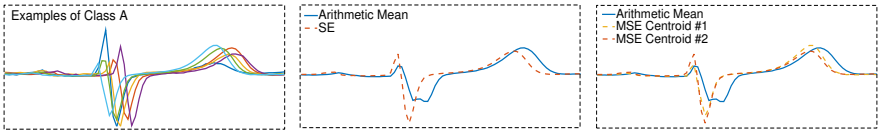


Fig. 4: Examples of "Class A" sequences of the ECGFiveDays dataset and centroids based on the arithmetic mean property (solid lines) and our SE and MSE methods (dashed lines).

Algorithm 2: $C = SE(X, R)$

Input: X is an n-by-m matrix with z-normalized time series.
R is a 1-by-m vector with the reference sequence against which time series of X are aligned.
Output: C is a 1-by-m vector with the centroid.

$X' \leftarrow [\,]$
for $i \leftarrow 1$ **to** n **do**
  $[dist, x'] \leftarrow SBD(R, X(i))$ // Algorithm 1
  $X' \leftarrow [X'; x']$
$S \leftarrow X'^T \cdot X'$ // S of Equation 15
$Q \leftarrow I - \frac{1}{m} \cdot O$ // Q of Equation 15
$M \leftarrow Q^T \cdot S \cdot Q$ // M of Equation 15
$C \leftarrow Eig(M, 1)$ // Extract first eigenvector

combining Equations 6 and 7, we obtain:

$$\vec{\mu_k}^\star = \underset{\vec{\mu_k}}{\operatorname{argmax}} \sum_{\vec{x_i} \in P_k} \left(\sum_{l \in [1,m]} x_{il} \cdot \mu_{kl} \right)^2$$

For simplicity, we express this equation with vectors and assume that the $\vec{x_i}$ sequences have already been z-normalized to handle the differences in amplitude:

$$\begin{aligned} \vec{\mu_k}^\star &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \sum_{\vec{x}_i \in P_k} (\vec{x}_i^T \cdot \vec{\mu}_k)^2 \\ &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \sum_{\vec{x_i} \in P_k} \vec{x_i}^T \cdot \vec{\mu_k} \cdot \vec{x_i}^T \cdot \vec{\mu_k} \\ &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \vec{\mu_k}^T \cdot \sum_{\vec{x_i} \in P_k} (\vec{x_i} \cdot \vec{x_i}^T) \cdot \vec{\mu_k} \end{aligned} \tag{14}$$

In the previous equation, only $\vec{\mu_k}$ is not z-normalized. To handle the centering (i.e., the subtraction of mean) of $\vec{\mu_k}$ we set $\vec{\mu_k} = \vec{\mu_k} \cdot Q$, where $Q = I - \frac{1}{m} O$, I is the identity matrix, and O is a matrix with all ones. Furthermore, to make $\vec{\mu_k}$ have unit norm, we divide Equation 14 by $\vec{\mu_k}^T \cdot \vec{\mu_k}$. (We omit the division with the standard deviation as such step will not alter the results.) Finally, by substituting S for $\sum_{\vec{x_i} \in P_k} (\vec{x_i} \cdot \vec{x_i}^T)$, we obtain:

$$\begin{aligned} \vec{\mu_k}^\star &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \frac{\vec{\mu_k}^T \cdot Q^T \cdot S \cdot Q \cdot \vec{\mu_k}}{\vec{\mu_k}^T \cdot \vec{\mu_k}} \\ &= \underset{\vec{\mu_k}}{\operatorname{argmax}} \frac{\vec{\mu_k}^T \cdot M \cdot \vec{\mu_k}}{\vec{\mu_k}^T \cdot \vec{\mu_k}} \end{aligned} \tag{15}$$

where $M = Q^T \cdot S \cdot Q$. Through the above transformations, we have reduced the optimization of Equation 13 to the optimization of Equation 15, which is a well-known problem called maximization of the Rayleigh Quotient [Golub and Van Loan 2012]. We can find the maximizer $\vec{\mu_k}^\star$ as the eigenvector that corresponds to the largest eigenvalue of matrix M.

Algorithm 2 shows how we can extract the most representative shape from the underlying data in a few lines of code. As a first step, SE aligns all sequences towards a reference sequence C. Then, SE computes matrix M (as shown in Equation 15) and, subsequently, extracts the primary eigenvector of M, which corresponds to the sequence with the maximum similarity to all others sequences. In Figure 4, we show the centroid of "Class A" in the ECGFiveDays dataset, extracted with SE and using a randomly selected sequence as reference sequence. SE method can more effectively capture the characteristics of "Class A" (Figure 1) than by using the arithmetic mean property (solid lines in Figure 4).

MultiShapesExtraction (MSE): Similarly to all averaging techniques described in Section 2.5, SE is based on two important assumptions: (i) a single centroid can effectively represent the characteristics of the underlying time series and (ii) all sequences contribute equally in the computation of the most representative sequence. However, in practice, time series are not uniformly distributed in space. Instead, some time series might appear in areas of higher density than the remainder of the time series. Therefore, a single centroid might not be sufficient to effectively capture characteristics of time series in clusters where time series are not uniformly distributed. Additionally, noisy sequences and outliers complicate the extraction of a representative sequence. Importantly, such noisy sequences and outliers do not always appear more distant to a reference sequence than the remainder of the sequences. Therefore, a weighted contribution of sequences in the computation of the most representative sequence is not sufficient.

To avoid the above limitations, we present MSE, a method to extract multiple centroids to summarize an entire set of time series. MSE relies on SE to extract each individual centroid but with two important differences. First, during the process of alignment of sequences towards a reference sequence, MSE computes the proximity of sequences to the reference sequence. Second, MSE divides the time series into evenly distributed "segments" by analyzing the proximity of sequences to a reference sequence. This division of time series into segments leads to the computation of multiple centroids, with each centroid capturing characteristics of segments with different proximity to the reference sequence. With this process, MSE solves the limitations of SE discussed above: (i) dense areas that might contain the majority of time series are now represented by multiple centroids and (ii) noisy sequences and outliers only influence the centroids for the segments where the sequences belong.

Algorithm 3 shows how we can extract multiple representative shapes from the underlying data in a few lines of code. As a first step, MSE aligns all sequences towards a reference sequence and computes the proximity of those sequences to the reference sequence (lines 3-6). MSE analyzes the proximity information and computes L quantiles that divide the data set into $L+1$ evenly distributed segments. Vector V contains distance values that correspond to L evenly spaced cumulative probabilities $\frac{1}{L+1}, \frac{2}{L+1}, \ldots, \frac{L}{L+1}$. Then, MSE computes centroids as follows. Each centroid c_j, with $j = 1, \ldots, L-1$, covers sequences with distances less than V(mod(j, L)), where mod denotes the modulo operation; and the last centroid, namely, centroid c_L, covers all sequences. Finally, similarly to SE, MSE computes matrix M (as shown in Equation 15) for each subset of sequences and, subsequently, extracts the primary eigenvector of M, which corresponds to the sequence with the maximum similarity to all other sequences in the subset of sequences. In Figure 4, we show two centroids of "Class A" in ECGFiveDays, extracted with MSE and using a randomly selected sequence as reference sequence. MSE can more effectively capture the characteristics of "Class A" (Figure 1) than by using either SE or the arithmetic mean property (solid lines in Figure 4).

We now show how SE and MSE are used in time-series clustering algorithms.

3.3. Shape-based Time-Series Clustering Methods

In this section, we present k-Shape and k-MS, our novel algorithms for time-series clustering. We first describe k-Shape, which relies on the SBD distance measure (Section 3.1) and SE

Algorithm 3: $C = MSE(X, R, L)$

Input: X is an n-by-m matrix with z-normalized time series.
R is a 1-by-m vector with the reference sequence against which time series of X are aligned.
L is the desired number of centroids per cluster.
Output: C is an L-by-m matrix with the L centroids.

```
1  Dists ← [ ]
2  X' ← [ ]
3  for i ← 1 to n do
4      [dist, x'] ← SBD(R, X(i))                                  // Algorithm 1
5      X' ← [X'; x']
6      Dists ← [Dists; dist]
7  V ← quantile(Dists, L)
8  for LIndex ← 1 to L do
9      NewX' ← []
10     for i ← 1 to n do
11         if mod(LIndex, L) = 0 then
12             NewX' ← [NewX'; X(i)']
13         else
14             if Dists(i) ≤ V(mod(LIndex, L)) then
15                 NewX' ← [NewX'; X(i)']
16     S ← NewX'^T · NewX'                                        // S of Equation 15
17     Q ← I − (1/m) · O                                          // Q of Equation 15
18     M ← Q^T · S · Q                                            // M of Equation 15
19     C(LIndex) ← Eig(M, 1)                                      // Extract first eigenvector
```

for centroid computation (Section 3.2) to efficiently produce clusters of time series.

k-Shape Clustering Algorithm: k-Shape is a partitional clustering method that is based on an iterative refinement procedure similar to the one used in k-means. Through this iterative procedure, k-Shape minimizes the sum of squared distances (Equation 1) and manages to: (i) produce homogeneous and well-separated clusters, and (ii) scale linearly with the number of time series. k-Shape is a nontrivial instantiation of k-means that compares sequences efficiently and computes centroids effectively under the scaling, translation, and shift invariances. As we will see, the choice of distance measure and centroid computation method make k-Shape the only scalable method that significantly outperforms k-means.

In every iteration, k-Shape performs two steps: (i) in the assignment step, k-Shape updates the cluster memberships by comparing each time series with all computed centroids and by assigning each time series to the cluster of the closest centroid; (ii) in the refinement step, the cluster centroids are updated using the SE method to reflect the changes in cluster memberships in the previous step. k-Shape repeats these two steps until either no change in cluster membership occurs or the maximum number of iterations allowed is reached.

k-Shape (see Algorithm 4) expects as input the time series set X and the number of clusters k that we want to produce. Initially, we randomly assign the time series in X to clusters. Then, we compute each cluster centroid using Algorithm 2 (lines 5-10). Once the centroids are computed, we refine the memberships of the clusters by using SBD (Algorithm 1) in lines 11-17. We repeat this procedure until the algorithm converges or reaches the maximum number of iterations (usually a small number, such as 100). The output of the algorithm is the assignment of sequences to clusters and the centroids for each cluster.

Having described k-Shape, we now present k-MS, which relies on the SBD distance measure (Section 3.1) and MSE for centroid computation (Section 3.2).

k-MS Clustering Algorithm: k-MS relies on a similar iterative refinement procedure to the one used for k-Shape. In every iteration, k-MS performs two steps: (i) in the assignment step, k-MS compares each time series to all computed centroids per cluster, assigns each time series to the closest centroid, and updates cluster memberships based on the memberships of centroids that belong to the same cluster; (ii) in the refinement step, the cluster

Algorithm 4: $[IDX, C] = k\text{-}Shape(X, k)$

Input: X is an n-by-m matrix containing n time series of length m that are initially z-normalized.
k is the number of clusters to produce.
Output: IDX is an n-by-1 vector with the assignment of n time series to k clusters (initialized randomly).
C is a k-by-m matrix with k centroids of length m (initialized as vectors with all zeros).

```
1  iter ← 0
2  IDX' ← [ ]
3  while IDX! = IDX' and iter < 100 do
4      IDX' ← IDX
       // Refinement step
5      for j ← 1 to k do
6          X' ← [ ]
7          for i ← 1 to n do
8              if IDX(i) = j then
9                  X' ← [X'; X(i)]
10         C(j) ← SE(X', C(j))                                  // Algorithm 2
       // Assignment step
11     for i ← 1 to n do
12         mindist ← ∞
13         for j ← 1 to k do
14             [dist, x'] ← SBD(C(j), X(i))                     // Algorithm 1
15             if dist < mindist then
16                 mindist ← dist
17                 IDX(i) ← j
18     iter ← iter + 1
```

centroids are updated using the MSE method to reflect the changes in cluster memberships in the previous step. k-MS repeats these two steps until either no change in cluster membership occurs or the maximum number of iterations allowed is reached.

k-MS (see Algorithm 5) expects three input parameters: (i) the time series set X; (ii) the number of clusters k; and (iii) the number of centroids L per cluster that we want to produce. In contrast to k-Shape, k-MS needs to handle multiple centroids in each cluster after the refinement and the assignment steps. In particular, in the refinement step, k-MS extracts multiple centroids for each cluster and, subsequently, k-MS appends these centroids in appropriate positions in the global list of centroids (lines 15-16). This step is critical for k-MS and allows the computation of centroids in each cluster with the same reference sequence. In the assignment step, k-MS performs an additional step not present in k-Shape: after the assignment of each time series to a centroid, k-MS merges the memberships of centroids that belong to the same cluster (lines 24-25).

Complexity of k-Shape and k-MS: As we claimed earlier, k-Shape and k-MS scale linearly with the number of time series. To see why, we will analyze the computational complexity of Algorithms 4 and 5, where n is the number of time series, k is the number of clusters, L is the number of centroids per cluster, and m is the length of the time series. In the assignment step, k-Shape computes the dissimilarity of n time series to k centroids by using SBD, which requires $\mathcal{O}(m \cdot \log(m))$ time. Thus, the time complexity of this step is $\mathcal{O}(n \cdot k \cdot m \cdot \log(m))$. k-MS computes the dissimilarity of n time series to $k \cdot L$ centroids and, therefore, the time complexity of this step is $\mathcal{O}(n \cdot k \cdot L \cdot m \cdot \log(m))$. In the refinement step, for every cluster, k-Shape computes matrix M, which requires $\mathcal{O}(m^2)$ time, and performs an eigenvalue decomposition on M, which requires $\mathcal{O}(m^3)$ time. Thus, the complexity of this step is $\mathcal{O}(\max\{n \cdot m^2, k \cdot m^3\})$. k-MS performs the same procedure L times per cluster and, therefore, the complexity of this step for k-MS is $\mathcal{O}(\max\{n \cdot m^2, k \cdot L \cdot m^3\})$. Overall, k-Shape requires $\mathcal{O}(\max\{n \cdot k \cdot m \cdot \log(m), n \cdot m^2, k \cdot m^3\})$ time per iteration and k-MS requires $\mathcal{O}(\max\{n \cdot k \cdot L \cdot m \cdot \log(m), n \cdot m^2, k \cdot L \cdot m^3\})$. We see that both algorithms have a linear dependence in the number of time series, and the majority of the

Algorithm 5: $[IDX, C] = k\text{-}MS(X, k, L)$

Input: X is an n-by-m matrix containing n time series of length m that are initially z-normalized.
k is the number of clusters to produce.
L is the number of centroids per cluster.
Output: IDX is an n-by-1 vector with the assignment of n time series to k clusters (initialized randomly).
C is a $k \cdot L$-by-m matrix containing $k \cdot L$ centroids of length m (initialized as vectors with all zeros).

```
1  iter ← 0
2  IDX' ← [ ]
3  [IDXtemp, Ctmp] ← k − Shape(X, k)                                  // Algorithm 4
4  for j ← 1 to k do
5      for w ← 1 to L do
6          C(w + (j − 1) · L) ← Ctmp(j)
7  while IDX! = IDX' and iter < 100 do
8      IDX' ← IDX
       // Refinement step
9      for j ← 1 to k do
10         X' ← [ ]
11         for i ← 1 to n do
12             if IDX(i) = j then
13                 X' ← [X'; X(i)]
14         Ctmp ← MSE(X', C(j · L)), L)                               // Algorithm 3
15         for w ← 1 to L do
16             C(w + (j − 1) · L) ← Ctmp(w)
       // Assignment step
17     for i ← 1 to n do
18         mindist ← ∞
19         for j ← 1 to k · L do
20             [dist, x'] ← SBD(C(j), X(i))                           // Algorithm 1
21             if dist < mindist then
22                 mindist ← dist
23                 IDX(i) ← j
24     for i ← 1 to n do
25         IDX(i) ← ceil(IDX(i)/L)
26     iter ← iter + 1
```

computation cost depends on the length of the time series. However, this length is usually much smaller than the number of time series (i.e., $m \ll n$) and, hence, the dependence on m is not a bottleneck. (Appendix B of [Paparrizos and Gravano 2015] examines the scalability of k-Shape.) Importantly, when m is very large (i.e., $m \gg n$), dimensionality reduction approaches can be used to sufficiently reduce the length of the sequences [Lin et al. 2004].

4. EXPLOITING CLUSTERING FOR FASTER ONE-NEAREST-NEIGHBOR CLASSIFICATION

So far, we have focused on how to perform shape-based clustering of time series effectively. However, as noted earlier, clustering is not only useful as a powerful standalone method, but also as a preprocessing or subroutine for other related tasks over time series. In this section, we show how k-Shape, our simplest shape-based clustering method, can effectively reduce the search space of one-nearest-neighbor algorithms and lead to fast and accurate techniques for the important problem of classification of time series.

One-nearest-neighbor, or 1-NN, classifiers address the time-series classification problem by assigning each time series to the category or class of the "nearest" time series according to a distance measure. 1-NN classifiers are popular because they do not rely on tunable parameters, and also because they can be used in conjunction with an appropriate choice of distance measure, out of the vast array of options (see Section 2.3 for examples of distance measures). Furthermore, 1-NN classifiers perform exceptionally well for time-series classification when used in conjunction with competitive distance measures for time series, as shown in [Bagnall and Lines 2014; Lines and Bagnall 2014; 2015; Bagnall et al. 2016].

Algorithm 6: $Accuracy = NNC(Training, Test, TrainingLabels, TestLabels, LeaveOneOut)$

Input: $Training$ is a k-by-m matrix containing k time series of length m.
$Test$ is an n-by-m matrix containing n time series of length m.
$TrainingLabels$ is a 1-by-k vector with the class labels of k time series in $Training$.
$TestLabels$ is a 1-by-n vector with the class labels of n time series in $Test$.
$LeaveOneOut$ is a Boolean indicating if this is a leave-one-out classification (true) or one-nearest-neighbor classification (false).
Output: $Accuracy$ is the classification accuracy.

```
1  Accuracy ← 0
2  for i ← 1 to n do
3      best_dist ← Inf
4      for j ← 1 to k do
5          if LeaveOneOut = true then
6              if i != j then
7                  [dist, ~] ← SBD(Training(j), Test(i))          // Algorithm 1
8          else
9              [dist, ~] ← SBD(Training(j), Test(i))              // Algorithm 1
10         if dist < best_dist then
11             class ← TrainingLabels(j)
12             best_dist ← dist
13     if TestLabels(i) = class then
14         Accuracy ← Accuracy + 1
15 Accuracy ← Accuracy/n
```

Reduction of dimensionality, auxiliary index structures, and lower bounding of distance measures have been previously exploited to reduce the computational time of 1-NN classifiers for time series [Agrawal et al. 1993; Faloutsos et al. 1994; Rakthanmanon et al. 2012]. Alternative methods for 1-NN classification, known as k-Nearest Centroid Classifiers (k-NCC), extract for each "query" the k nearest neighbors from each class in the search space, and then summarize each of the sets of k neighbors with a centroid that is computed as some variant of a mean vector. k-NCC methods determine the class of the query using factors such as the proximity and the spatial distribution of the computed centroids relative to the query [Mitani and Hamamoto 2000; 2006; Chaudhuri 1996; Sánchez et al. 1997; Gou et al. 2012]. However, k-NCC methods are usually more expensive than 1-NN classifiers because of the additional processing that is required beyond nearest-neighbor search. Interestingly, these approaches reduce to 1-NN when $k = 1$. Recently, [Petitjean et al. 2014; 2015] demonstrated the potential of representing each class in the search space with a small number of centroids computed via clustering. However, the clustering methods that they use are particularly inefficient, as we show in Section 6, and they significantly increase the cost to estimate an appropriate number of centroids to represent each class in the search space.

To address this issue, and inspired by [Petitjean et al. 2014; 2015], we now introduce the Nearest Shape Classifier (NSC), a 1-NN classifier that relies on k-Shape to effectively summarize time series in its search space (see Algorithm 7). In contrast to k-NCC methods, NSC uses cluster centroids to summarize the entire search space and does not recompute the centroids for every query. The estimation of the number of centroids relies solely on analysis over the search space of the 1-NN classifier. Specifically, NSC first determines how accurately it can estimate the class for queries taken from the search space (line 1). This accuracy serves as a baseline estimation of the best achieved accuracy when the entire search space is used. Then, NSC varies the number of centroids required to summarize each class in the search space (line 7). NSC computes centroids for each class in the search space by considering the following four cases: (i) when one centroid is requested and the class contains one time series, NSC uses this time series as centroid (lines 11-14); (ii) when one centroid is requested and the class contains more than one time series, NSC uses Algorithm 2 in Section 3.2 to summarize the time series (lines 15-20); (iii) when two or more centroids are

Algorithm 7: $Accuracy = NSC(Training, Test, TrainingLabels, TestLabels, InstancesLabels)$

Input: $Training$ is a k-by-m matrix containing k time series of length m.
$Test$ is an n-by-m matrix containing n time series of length m.
$TrainingLabels$ is a 1-by-k vector with the class labels of the k time series in $Training$.
$TestLabels$ is a 1-by-n vector with the class labels of the n time series in $Test$.
$InstancesLabels$ is a 1-by-w vector with each cell pointing to a list with IDs of instances in $Training$ belonging to each of the w classes.
Output: $Accuracy$ is a scalar value containing the classification accuracy on $Test$.

```
1  Best_Accuracy ← NNC(Training, Training, TrainingLabels, TrainingLabels, true)        // Algorithm 6
2  Best_NumberCentroids ← k
3  Best_Centroids ← Training
4  Best_CentroidsLabels ← TrainingLabels
5  Centroids ← []
6  CentroidsLabels ← []
7  for i ← 1 to max(length(InstancesLabels([1 : w]))) − 1 do
8      for class ← 1 to w do
9          IDs ← InstancesLabels(class)
10         [nrows, ncolumns] ← size(Training(IDs))
11         if i = 1 and nrows = 1 then
12             tmp_centroids ← Training(IDs)
13             Centroids ← [Centroids; tmp_centroids]
14             CentroidsLabels ← [CentroidsLabels; class]
15         else if i = 1 and nrows > 1 then
16             index ← rand(IDs)
17             ref_centroid ← Training(index)
18             tmp_centroids ← SE(Training(IDs), ref_centroid)                          // Algorithm 2
19             Centroids ← [Centroids; tmp_centroids]
20             CentroidsLabels ← [CentroidsLabels; class]
21         else if i > 1 and nrows > i then
22             [∼, tmp_centroids] ← k − Shape(Training(IDs), i)                         // Algorithm 4
23             Centroids ← [Centroids; tmp_centroids]
24             for p ← 1 to nrows do
25                 CentroidsLabels ← [CentroidsLabels; class]
26         else if i > 1 and nrows ≤ i then
27             Centroids ← [Centroids; Training(IDs)]
28             for p ← 1 to nrows do
29                 CentroidsLabels ← [CentroidsLabels; class]
30     AccuracyCentroids ← NNC(Centroids, Training, CentroidsLabels, TrainingLabels, true)  // Algorithm 6
31     if AccuracyCentroids > Best_Accuracy then
32         Best_Accuracy ← AccuracyOnCentroids
33         Best_NumberCentroids ← size(Centroids, 1)
34         Best_Centroids ← Centroids
35         Best_CentroidsLabels ← CentroidsLabels
36     Centroids ← []
37     CentroidsLabels ← []
38 Accuracy ← NNC(Best_Centroids, Test, Best_CentroidsLabels, TestLabels, false)      // Algorithm 6
```

requested and the class contains more time series than the requested centroids, NSC uses Algorithm 4 of Section 3.3 to cluster and summarize the time series (lines 21-25); and (iv) when two or more centroids are requested and the class contains fewer time series than the requested centroids, NSC uses the time series as centroids and does not perform clustering, to avoid producing singleton clusters with just one time series in them (lines 26-29). For each number of centroids, NSC evaluates how accurately it can estimate the class of the queries in the search space when the search space is summarized by that number of centroids. The process terminates if this accuracy outperforms the baseline accuracy computed earlier (in line 1). In the end, NSC determines the class of a query as the class of the nearest centroid.

The above procedure permits the estimation of the number of centroids and the computation of centroids that capture internal characteristics of every class of time series accurately, as we will see in Section 6.7. NSC requires as input the instances and the labels of the training and test sets along with a breakdown of training instances per class. As output, NSC

computes the 1-NN classification accuracy (Algorithm 6) on the test set but using as search space the estimated number of centroids per class over the training set.

5. EXPERIMENTAL SETTINGS

We now describe the settings for the evaluation of SBD, k-Shape, k-MS, and NSC.

Datasets: We use the largest public collection of class-labeled time-series datasets, namely, the UCR collection [Keogh et al. 2015]. It consists of 85 datasets, both synthetic and real, which span several different domains. Each dataset contains from 40 to 16,637 sequences. The sequences in each dataset have equal length, but from one dataset to another the sequence length varies from 24 to 2,709. These datasets are annotated and every sequence can belong to only one class. In the context of clustering, and for the sake of convenience, the class label for a sequence is often interpreted as identifying the cluster where the sequence belongs. Furthermore, the datasets are already z-normalized and split into training and test sets. As we will see, we use this split of the datasets for the distance measure evaluation; we also use the training sets for tuning some of the baselines.

Platform: We ran our experiments on a cluster of 61 servers with identical configuration: Dual Intel Xeon E5-2650 (8-core with 2-way SMT) processor with clock speed at 2.6 GHz and up to 256 GB RAM. We utilized on average 20 servers continuously for a period of two months in order to perform all experiments included in this paper. Each server runs Red Hat Enterprise Linux 6.6 (64-bit) and Matlab R2014a (64-bit).

Implementation: We implemented all approaches under the same framework, in Matlab, for a consistent evaluation in terms of both accuracy and efficiency. For repeatability purposes, we make all datasets and source code available.[4]

Baselines: We evaluate SBD, our distance measure; k-Shape and k-MS, our clustering approaches; and NSC, our one-nearest neighbor method. Specifically, we compare SBD against the strongest state-of-the-art distance measures for time series (see Section 2.3):

— **ED:** a simple, efficient — yet accurate — distance measure [Faloutsos et al. 1994]
— **DTW:** one of the best performing — but expensive — distance measure for time series [Sakoe and Chiba 1978]
— **cDTW:** the constrained version of DTW, with improved accuracy and efficiency [Sakoe and Chiba 1978]

We compare k-Shape and k-MS against the three strongest types of scalable and non-scalable clustering methods, namely, partitional, hierarchical, and spectral methods (see Section 2.4 for a detailed discussion), combined with the most competitive distance measures. As scalable methods, we consider the classic k-means algorithm with ED (k-AVG+ED) [MacQueen 1967] and the following variants of k-means:

— **k-AVG+Dist:** k-means with DTW and SBD as distance measures and the arithmetic mean of time series coordinates for centroid computation
— **k-DBA:** k-means with DTW as distance measure and the DBA method for centroid computation [Petitjean et al. 2011]
— **KSC:** k-means with a distance measure offering pairwise scaling and shifting of time series and computation of the spectral norm of a matrix (i.e., matrix decomposition) for centroid computation [Yang and Leskovec 2011]

As non-scalable methods, among partitional methods we consider the Partitioning Around Medoids (PAM) implementation of the k-medoids algorithm [Kaufman and Rousseeuw 2009]. Among hierarchical methods, we use agglomerative hierarchical clustering with single, average, and complete linkage criteria [Kaufman and Rousseeuw 2009]. Among spectral methods, we consider the popular normalized spectral clustering method [Ng et al. 2002]. These non-scalable approaches require a large number of distance calculations to compute the full dissimilarity matrix and, hence, they become unacceptably inefficient with high-cost

[4] http://www.cs.columbia.edu/~jopa/kshape.html

Name	Clustering Algorithm	Distance Measure
PAM+ED	Partitioning Around Medoids	ED
PAM+cDTW	Partitioning Around Medoids	cDTW[5]
PAM+SBD	Partitioning Around Medoids	SBD
H-S+ED	Hierarchical with *single* linkage	ED
H-A+ED	Hierarchical with *average* linkage	ED
H-C+ED	Hierarchical with *complete* linkage	ED
H-S+cDTW	Hierarchical with *single* linkage	cDTW[5]
H-A+cDTW	Hierarchical with *average* linkage	cDTW[5]
H-C+cDTW	Hierarchical with *complete* linkage	cDTW[5]
H-S+SBD	Hierarchical with *single* linkage	SBD
H-A+SBD	Hierarchical with *average* linkage	SBD
H-C+SBD	Hierarchical with *complete* linkage	SBD
S+ED	Normalized Spectral Clustering	ED
S+cDTW	Normalized Spectral Clustering	cDTW[5]
S+SBD	Normalized Spectral Clustering	SBD
DBSCAN$^{0.3}$+ED	Density-based Spatial Clustering	ED
DBSCAN^{Best}+ED	Density-based Spatial Clustering	ED
DBSCAN$^{0.3}$+SBD	Density-based Spatial Clustering	SBD
DBSCAN^{Best}+SBD	Density-based Spatial Clustering	SBD
DBSCAN$^{0.3}$+cDTW	Density-based Spatial Clustering	cDTW[5]
DBSCAN^{Best}+cDTW	Density-based Spatial Clustering	cDTW[5]
TADPole$^{0.3}$	Time-series Anytime Density Peaks Clustering	cDTW[5]
TADPole^{Best}	Time-series Anytime Density Peaks Clustering	cDTW[5]
U-Shapelets$^{0.5}$	Unsupervised-Shapelets Clustering	-
U-Shapelets^{Best}	Unsupervised-Shapelets Clustering	-

Table I: Combinations of PAM, hierarchical, spectral, density-based, shapelet-based methods with ED, cDTW, and SBD for our evaluation.

distance measures. For this reason, we distribute the computation of the full dissimilarity matrices and, therefore, we do not report runtime results for these methods.

Beyond these scalable and non-scalable methods, we further evaluate k-Shape and k-MS against the strongest types of density-based methods, namely, DBSCAN [Ester et al. 1996] and TADPole [Begum et al. 2015]. Density-based methods have started to receive attention for time-series clustering to identify outliers in the data. DBSCAN can be combined with any distance measure, while TADPole, a variant of the recently proposed Density Peaks (DP) clustering method [Rodriguez and Laio 2014], uses the cDTW distance. DBSCAN and TADPole are also non-scalable methods. Specifically, DBSCAN requires the computation of the full dissimilarity matrix to operate.[5] In contrast, TADPole significantly prunes the dissimilarity matrix for cDTW, but requires the computation of two additional dissimilarity matrices based on ED. TADPole improves in runtime performance relative to DP [Rodriguez and Laio 2014], but remains non-scalable and particularly inefficient relative to k-means variants. Importantly, as we discuss later, both algorithms require the computation of the dissimilarity matrix for cDTW in order to accurately estimate input parameters, a step that is often omitted from runtime experiments. Therefore, both density-based methods are inefficient and, hence, we do not report runtime results for them.

We also compare k-Shape and k-MS against U-Shapelets [Zakaria et al. 2012]. Unlike previous approaches, U-Shapelets exploits patterns in subsequences of time series to isolate outliers. Unfortunately, U-Shapelets cannot handle large datasets, so we use a variant that has been shown to be faster than the original technique without a significant loss in accuracy [Begum et al. 2015]. Table I summarizes the non-scalable clustering combinations used in our evaluation. Overall, we compared k-Shape and k-MS against 30 clustering approaches.

[5]DBSCAN can use auxiliary index structures to accelerate neighborhood queries only for metric distance functions, such as ED.

We compare NSC against data editing and condensing algorithms that are applied in the full dissimilarity matrix of the training instances of a 1-NN classifier and determine the order in which instances can be removed to guarantee the performance of a 1-NN classifier. Specifically, we evaluate NSC against three popular variants of such *reduction techniques*, namely, RT1, RT2, and RT3 [Wilson and Martinez 1997; 2000]. Additionally, we also compare NSC against a simple baseline, namely, Random, that randomly chooses instances for removal from the training instances. Finally, we compare the performance of NSC with three competitive distance measures (Sections 2.3 and 3.1), namely, ED (NSC+ED), SBD (NSC+SBD), and cDTW^5 (NSC+cDTW) against 1-NN classifiers.

Parameter settings: Among the distance measures discussed above, only cDTW requires setting a parameter, to constrain its warping window. We consider two cases from the literature: (i) cDTW^{opt}: we compute the optimal window by performing a leave-one-out classification step over the training set of each dataset; (ii) cDTW^w: we use as window 5%, for cDTW^5, and 10%, for cDTW^{10}, of the length of the time series of each dataset; this second case is more realistic for an unsupervised setting such as clustering.[6] For the 1-NN classification computation, we also consider the state-of-the-art lower bounding approach LB$_{Keogh}$ [Keogh and Ratanamahatana 2005], which prunes time series that could not be matched when DTW and cDTW are used. We denote lower bounding with the LB subscript (e.g., cDTW$^{10}_{LB}$). For clustering, all partitional and spectral algorithms that we compare receive the target number of clusters k as input, which is equal to the number of classes for each dataset. The only exception is the partitional k-MS method, which requires setting an additional parameter, namely, the number of centroids per cluster. For k-MS, we set the number of centroids per cluster, L, to $L = 5$ across all datasets. For hierarchical clustering, we compute a threshold that cuts the produced dendrogram at the minimum height such that k clusters are formed. We set the maximum number of iterations for k-Shape, all variants of k-means, PAM, and spectral methods to 100. Finally, in every iteration of k-Shape, k-DBA, and KSC, we use the centroids of the previous run as reference sequences to refine the centroids of the current run once.

In contrast to partitional, hierarchical, and spectral methods, density-based methods require as input two parameters. Specifically, DBSCAN requires as input a cutoff threshold, e, and the minimum number of points to form a density region, *minPts*. To estimate e for each distance measure, we fix the value of *minPts* (we consider values 3, 4, 5, 10, and 20), compute the full dissimilarity matrix, and keep, for each time series, the largest distance value such that exactly *minPts* time series are within that distance from it. We sort these distance values in descending order and compute the knee point of the curve of distance values. The process of determining this knee point is similar to the process of determining the number of clusters in clustering methods [Salvador and Chan 2004]. We consider all bisections of the curve and, then, by walking along the curve one bisection point at a time, we fit two straight lines, one to all the points to the left of the bisection point and one to all the points to the right of the bisection point. The knee point of the curve corresponds to the bisection point with the minimum sum of errors for the two fitted lines. For all distance measures, $minPts = 3$ performed the best on average across all datasets and we denote this variant as DBSCAN^3. However, the best $minPts$ value for a specific dataset might be different, so we also determine the best $minPts$ parameter for each dataset after performing a post-hoc analysis on the Rand Index results (see below). We denote the resulting version of DBSCAN with dataset-specific values of $minPts$ as DBSCAN^{Best}.

The second density-based method that we consider, TADPole, also requires setting the value of two parameters: (i) the number k of clusters and (ii) a cutoff threshold e. Considering that DBSCAN and TADPole operate similarly, and the fact that the authors of TADPole

[6]For non-scalable methods, we use the cDTW^5 variant of cDTW for its efficiency, as noted in Table I, even though cDTW^{10}, cDTW^{opt}, and DTW perform similarly. We explicitly note when DTW is used.

[Begum et al. 2015] do not provide details on the estimation of these parameters, we use the same procedure as with DBSCAN. TADPole also performs the best on average across all datasets when $minPts = 3$. We refer to this version of the algorithm as TADPole[3]. To determine the best $minPts$ parameter for each dataset, we perform a post-hoc analysis on the Rand Index results. We refer to this version of the algorithm as TADPole^{Best}.

U-Shapelets requires setting the value of just one parameter, namely, the length of subsequences of the entire time series. We consider values of 10%, 20%, 30%, 40%, and 50% of the time series length of each dataset. The best performing length for U-Shapelets across all datasets is 50% and we denote the resulting technique as U-Shapelets$^{0.5}$. Similarly to DBSCAN and TADPole, we also consider the best length parameter for each dataset after performing a post-hoc analysis. We denote this approach as U-Shapelets^{Best}. We note that U-Shapelets^{Best}, DBSCAN^{Best}, and TADPole^{Best} are not truly options for fully unsupervised clustering, given that their parameters are the result of post-hoc analysis with labeled data. However, we consider these options for completeness.

Finally, we use cDTW[5] to compute the dissimilarity matrices for RT1, RT2, and RT3, as well as to measure the accuracy of 1-NN classifiers for Random, RT1, RT2, and RT3.

Metrics: We compare the approaches on both runtime and accuracy. For runtime, we compute CPU time utilization and measure the time ratios for our comparisons for each dataset. We summarize the runtime performance by reporting, for each method, the number of datasets with time ratios of up to one order of magnitude, between one but no higher than two orders of magnitude, and at least two orders of magnitude in comparison to another method. Following [Ding et al. 2008], we use the 1-NN classifier, which is a simple and parameter-free classifier, to evaluate distance measures. We report the classification accuracy (i.e., number of correctly classified instances over all instances) by performing 1-NN classification over the training and test sets of each dataset. Because the 1-NN classifier is deterministic, we make this computation once. Following [Batista et al. 2013], we also visualize the *actual accuracy gain* and the *expected accuracy gain* for pairs of distance measures to evaluate if we can predict in advance which distance measure is more useful for which datasets. The actual accuracy gain for each dataset is defined as the ratio of the 1-NN classification accuracies of the two distance measures over the test portion of the dataset. The expected accuracy gain for each dataset is defined as the ratio of the leave-one-out classification accuracies of the two distance measures over the training portion of the dataset. In both ratios, the denominator is the accuracy of the baseline distance measure whereas the numerator is the accuracy of the competing distance measure. The resulting plot illustrates the actual accuracy gains against the expected accuracy gains over all datasets and consists of four regions. Each region corresponds to the true-positive (TP), true-negative (TN), false-positive (FP), and false-negative (FN) cases of the well-known contingency table for comparing pairs of classifiers. For datasets in the TP region, we can predict (i.e., by evaluation on the training portions) that the competing distance measure will outperform the baseline distance measure. For datasets in the TN region, we can predict that the baseline distance measure will outperform the competing distance measure. For datasets in the FN region, we inaccurately predict that the baseline distance measure will outperform the competing distance measure. Similarly, for datasets in the FP region, we inaccurately predict that the competing distance measure will outperform the baseline distance measure.

We use the Rand Index [Rand 1971] to evaluate clustering accuracy over the fused training and test sets of each dataset. This metric is related to the classification accuracy and is defined as $R = \frac{TP+TN}{TP+TN+FP+FN}$, where TP is the number of time series pairs that belong to the same class and are assigned to the same cluster, TN is the number of time series pairs that belong to different classes and are assigned to different clusters, FP is the number of time series pairs that belong to different classes but are assigned to the same cluster, and

FN is the number of time series pairs that belong to the same class but are assigned to different clusters. As hierarchical algorithms are deterministic, we report the Rand Index over one run. However, for partitional methods, we report the average Rand Index over 10 runs and for spectral methods the average Rand Index over 100 runs; in every run we use a different random initialization.

Statistical analysis: Following [Batista et al. 2013; Giusti and Batista 2013], we analyze the results of every pairwise comparison of algorithms over multiple datasets using the Wilcoxon test [Wilcoxon 1945] with a 99% confidence level. According to [Demšar 2006], the Wilcoxon test is less affected by outliers than is the t-test [Rice 2006], as Wilcoxon does not consider absolute commensurability of differences. Moreover, using pairwise tests to reason about multiple algorithms is not fully satisfactory because sometimes the null hypotheses are rejected due to random chance. Therefore, we also use the Friedman test [Friedman 1937] followed by the post-hoc Nemenyi test [Nemenyi 1963] for comparison of multiple algorithms over multiple datasets, as suggested in [Demšar 2006]. The Friedman and Nemenyi tests require more evidence to detect statistical significance than the Wilcoxon test [Giusti and Batista 2013] (i.e., the larger the number of methods, the larger the number of datasets required) and, hence, as we already use all 85 datasets for the Wilcoxon test, we report statistical significant results with a 95% confidence level.

6. EXPERIMENTAL RESULTS

In this section, we discuss our experiments. Firstly, we evaluate SBD against the state-of-the-art distance measures (Section 6.1) and corroborate our performance results over optimized implementations for the state-of-the-art distance measures (Section 6.2). Secondly, we compare k-Shape and k-MS against scalable (Section 6.3) and non-scalable (Section 6.4) clustering approaches. Thirdly, we evaluate k-Shape and k-MS against approaches that handle outliers in datasets (Section 6.5) and approaches that consider only subsequences of the entire time series in an attempt to handle abnormal behaviors in time series (Section 6.6). Then, we evaluate NSC against data editing algorithms that reduce the search space of 1-NN classifiers (Section 6.7). Finally, we summarize our findings (Section 6.8).

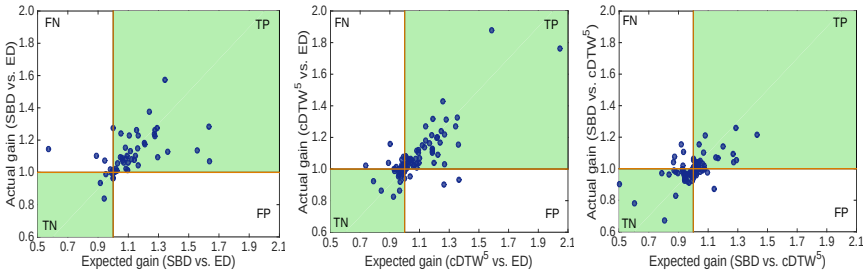
6.1. Evaluation of SBD

Comparison against ED: To understand if SBD (Section 3.1) is an effective measure for time-series comparison, we evaluate it against the state-of-the-art distance measures, using their 1-NN classification accuracy across 85 datasets (Section 5). Table II reports the performance of the state-of-the-art measures against the baseline ED. All distance measures, including SBD, outperform ED with statistical significance. In particular, SBD performs at least as well as ED in 77 out of 85 datasets in contrast to DTW, cDTW^5, and cDTW^{10}, which perform at least as well as ED in 56, 63, and 60 datasets, respectively. Only cDTW^{Opt}, with its optimal warping window constraint, reaches the performance of SBD (i.e., cDTW^{Opt} performs at least as well as ED in 92% of the datasets).

However, to better understand the usefulness of each distance measure, we further analyze if we can predict in advance the accuracy gain by using one distance measure over another distance measure (i.e., we want to predict the ratio of the classification accuracy of one distance measure over that of another distance measure). Figure 5a shows that for the large majority of the datasets we can accurately predict in advance if SBD will outperform ED or not (please refer to the TP and TN regions of the figure). For only 6 datasets we incorrectly predict that ED will outperform SBD (please refer to the FN region of the figure). Importantly, 3 out of the 6 datasets are very close to the (1, 1) region, which highlights the difficulties for identifying differences over those datasets. We note that none of the datasets belongs to the worst-case FP region, an important characteristic that also appears in the evaluation of the supervised cDTW^{Opt} distance. In contrast, for cDTW^5 (Figure 5b), 3 datasets appear in the FP region, 10 in the FN region, and the rest of the datasets are

Distance Measure	>	=	<	Better	[0,10x)	[10x,100x)	[100x,+∞)
DTW	53	3	29	✓	0	9	76
DTW$_{LB}$					0	14	71
cDTW^{opt}	54	24	7	✓	22	28	35
cDTW$^{opt}_{LB}$					40	29	16
cDTW^{5}	58	5	22	✓	5	39	41
cDTW$^{5}_{LB}$					23	41	21
cDTW^{10}	56	4	25	✓	2	32	51
cDTW$^{10}_{LB}$					8	46	31
SBD$_{NoFFT}$	54	23	8	✓	14	50	21
SBD$_{NoPow2}$					57	28	0
SBD					79	6	0

Table II: Comparison of distance measures. Columns ">," "=," and "<" denote the number of datasets over which a distance measure is better, equal, or worse, respectively, in comparison to ED. "Better" indicates that a distance measure outperforms ED with statistical significance. Columns "[0, 10x)," "[10x, 100x)," and "[100x, +∞)" denote the number of datasets over which a distance measure is slower up to 10x, between 10x but no higher than 100x, and at least 100x, respectively, in comparison to ED.



(a) SBD vs. ED (b) cDTW^5 vs. ED (c) SBD vs. cDTW^5

Fig. 5: Comparison of accuracy gain for ED, SBD, and cDTW^5 over 85 datasets.

scattered in the TP and TN regions. As a result, SBD, a parameter-free and unsupervised distance measure, shows similar merit to the supervised cDTW^{Opt} distance.

To further elaborate on the importance of optimally constraining the warping window of DTW, we note that only cDTW^{Opt} is significantly better than DTW. In particular, cDTW^{opt} performs at least as well as DTW in 61 datasets, cDTW^5 performs at least as well as DTW in 57 datasets, and cDTW^{10} performs at least as well as DTW in 67 datasets. For cDTW^5, the Wilcoxon test suggests no statistical significance over DTW whereas for cDTW^{10} the improvement is statistically significant but with a p-value close to the confidence level, which indicates that deeper statistical analysis is required.

Figure 6 shows the average rank across datasets of cDTW^{opt}, cDTW^5, cDTW^{10}, and DTW. cDTW^{opt} is the top measure, with an average rank of 2.1, meaning that cDTW^{opt} performed best in the majority of the datasets. The Friedman test rejects the null hypothesis that all measures behave similarly, and, hence, we proceed with a post-hoc Nemenyi test, to evaluate the significance of the differences in the ranks. The wiggly line in the figure connects all measures that do not perform statistically differently according to the Nemenyi test. We observe two clusters: one where the ranks of cDTW^{opt}, cDTW^5, and cDTW^{10} do not present a significant difference, and one where the ranks of cDTW^5, cDTW^{10}, and DTW do not present a significant difference. As a consequence, only cDTW^{opt} appears to perform significantly better than DTW. This conclusion contradicts the general belief that constraining DTW's warping window significantly improves accuracy, which was also confirmed by our previous analysis on a subset of 48 datasets [Paparrizos and Gravano 2015].

Comparison against DTW and cDTW: SBD performs at least as well as DTW in 50

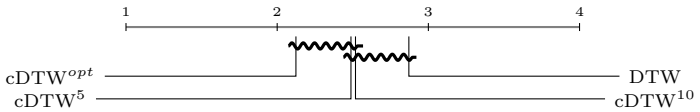


Fig. 6: Ranking of distance measures based on the average of their ranks across datasets. The wiggly line connects all measures that do not perform statistically differently according to the Nemenyi test.

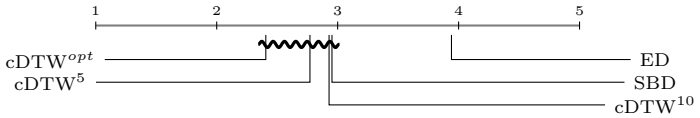


Fig. 7: Ranking of distance measures based on the average of their ranks across datasets. The wiggly line connects all measures that do not perform statistically differently according to the Nemenyi test.

datasets, but the statistical test reveals no evidence that either measure is better than the other. Considering the cDTW versions, we observe that SBD performs similarly to or better than cDTW^{opt}, cDTW^5, and cDTW^{10} in 41, 37, and 40 datasets, respectively. Figure 5c shows that for the large majority of datasets we can accurately predict in advance when either SBD or cDTW^5 should be used (we omit figures for cDTW^{10} and cDTW^{Opt} as they exhibit similar trends). Out of 10 datasets in the FP region, only 2 cases are problematic, while the remaining 8 datasets are very close to the (1, 1) region, and hence it is challenging to predict which distance measure is best. Interestingly, there is no significant difference among SBD, cDTW^5, and cDTW^{10}, but cDTW^{opt} is significantly better than SBD. Importantly, the p-values for Wilcoxon between cDTW^{opt} and SBD are close to the confidence level, which indicates that deeper statistical analysis is required, as we discuss next.

Statistical analysis: To better understand the performance of SBD in comparison with cDTW^{opt}, cDTW^5, and cDTW^{10}, we evaluate the significance of their differences in accuracy when considered all together. Figure 7 shows the average rank across datasets of each distance measure. cDTW^{opt} is the top measure, with an average rank of 2.4, meaning that cDTW^{opt} performed best in the majority of the datasets. The Friedman test rejects the null hypothesis that all measures behave similarly, and, hence, we proceed with a post-hoc Nemenyi test, to evaluate the significance of the differences in the ranks. We observe that the ranks of cDTW^{opt}, cDTW^5, cDTW^{10}, and SBD do not present a significant difference, and ED, which is ranked last, is significantly worse than the others. In conclusion, SBD is a very competitive distance measure that significantly outperforms ED and achieves similar results to both cDTW and DTW. Moreover, SBD is the most robust variant of the cross-correlation measure (see Appendix A of [Paparrizos and Gravano 2015]).

Efficiency: We now show that SBD is not only competitive in terms of accuracy, but also highly efficient. We also demonstrate that implementation choices for SBD can significantly impact its speed. The last row of Table II shows the number of datasets over which each SBD variation is slower than ED. The optimized version of SBD, denoted simply as SBD, is the fastest version, performing up to 10x slower than ED in 93% of the datasets and 10x but less than 100x slower in the remaining 7% of the datasets. When we use SBD still with FFT but without the power-of-two-length optimization discussed in Section 3.1, the resulting measure, SBD$_{NoPow2}$, is up to 10x slower than ED in 67% of the datasets and 10x but less than 100x slower in 33% of the datasets. Furthermore, SBD$_{NoFFT}$, the version of SBD without FFT, performs up to 10x, 10x but no higher than 100x, and at least 100x slower than ED in 16%, 59%, and 25% of the datasets, respectively. Table II also shows that DTW and cDTW variants are substantially slower than SBD. Specifically, DTW is up to 100x slower in 40% of the datasets and at least 100x in 60% of the datasets in comparison to SBD. Similarly, cDTW^{opt}, cDTW^5, and cDTW^{10} are more than 10x slower in comparison

Distance Measure	Language	[0,10x)	[10x,100x)	[100x,+∞)	Reference
$cDTW_{LB}^{opt}$	Java	25	45	9	**[Schäfer 2015]**
	Java	19	52	8	**[Wang et al. 2013]**
	Matlab	34	29	16	**This work**
DTW_{LB}	Java	1	12	66	**[Schäfer 2015]**
	Java	0	14	65	**[Wang et al. 2013]**
	Matlab	0	13	66	**This work**

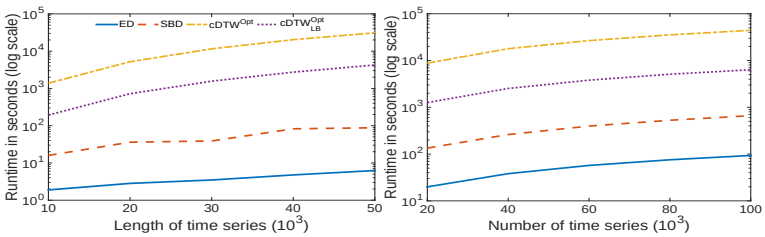
Table III: Comparison of implementations for $cDTW^{Opt}$ and DTW. Column "Language" indicates the programming language. Columns "[0, 10x)," "[10x, 100x)," and "[100x, $+\infty$)" denote the number of datasets over which a distance measure is slower up to 10x, between 10x but no higher than 100x, and at least 100x, respectively, in comparison to ED. "Reference" indicates the work that describes the implementation details.

to SBD in 51%, 65%, and 79% of the datasets, respectively. Even when we speed up the search of 1-NN classification computation, by pruning time series impossible for a match using LB_{Keogh}, SBD is still faster. In particular, DTW_{LB} is up to 100x slower in 46% of the datasets and more than 100x in 54% of the datasets in comparison to SBD. $cDTW_{LB}^{opt}$, $cDTW_{LB}^{5}$, and $cDTW_{LB}^{10}$ are more than 10x slower in comparison to SBD in 24%, 35%, and 54% of the datasets, respectively. Next, we corroborate our performance findings over different implementations of the rival distance measures and perform experiments to better understand the scalability of distance measures.

6.2. Robustness of Runtime Results Across Implementations

Considering that the choice of programming language as well as optimizations in implementation may affect the performance of distance measures, we now compare our Matlab implementation of cDTW and DTW variants across two state-of-the-art Java implementations from the literature. Specifically, we compare our performance results for $cDTW_{LB}^{opt}$ and DTW_{LB} with the results as reported in [Schäfer 2015], which follows the optimizations described in [Rakthanmanon et al. 2012], and with results we obtained in our experimental settings with an implementation that follows the optimizations described in [Wang et al. 2013]. Table III compares the performance of the three implementations over 79 datasets that are common between [Schäfer 2015] and our work. DTW_{LB} performs similarly across all implementations. In particular, DTW_{LB} is up to 100x slower than ED in approximately 17% of the datasets and at least 100x slower than ED in approximately 83% of the datasets. Lower bounding methods prune only a small fraction of the DTW computations (on average about 25%) and, hence, there is minor variability in performance across implementations.

In contrast to DTW_{LB}, for $cDTW_{LB}^{opt}$ there is some variability in the results across implementations. However, for a vast majority of datasets the results are consistent across implementations. Specifically, $cDTW_{LB}^{opt}$ is up to 100x slower than ED in approximately 89% of the datasets and at least 100x slower than ED in approximately 11% of the datasets for the two Java implementations. The implementation of [Schäfer 2015], which exploits multiple lower bounding measures in a cascade, is more efficient than [Wang et al. 2013], which uses only one lower bounding method. Our version of $cDTW_{LB}^{opt}$ is up to 100x slower than ED in 80% of the datasets and at least 100x slower than ED in approximately 20% of the datasets. In comparison to the Java implementations, we observe that the performance of $cDTW_{LB}^{opt}$ appears to decrease in a small number of datasets with long time series because, in such datasets, Matlab's underlying optimizations in linear algebra operations benefit ED. Therefore, for approximately 10% of the datasets, our version of $cDTW_{LB}^{opt}$ appears slower than the Java implementations but, importantly, for approximately 18% of the datasets (i.e., in datasets with short time series), $cDTW_{LB}^{opt}$ appears faster than the Java



(a) Varying m for n=500 (b) Varying n for m=4K

Fig. 8: Runtime of ED, SBD, cDTW^{Opt}, and cDTW$^{Opt}_{LB}$ as a function of (a) the number of time series n and (b) the time series length m.

implementations. Overall, we can conclude that the performance results of our implementation of DTW and cDTW variants are consistent with what has been reported previously in the literature, with some small variability in the results of cDTW due to underlying optimizations in programming languages and choices of lower bounding methods.

Even though our performance results are consistent across implementations, we believe that several factors hinder the process of understanding the runtime performance when solely the UCR datasets are used. In particular, the large variations in the number and length of time series across datasets do not help to provide a clear picture of how distance measures scale as a function of such variable numbers. Moreover, the choice of different parameters to constrain DTW's warping window can significantly affect the runtime performance of the distance measure and the pruning power of lower bounding approaches.

To better understand the runtime performance and scalability of distance measures, we perform an additional experiment using the synthetic CBF dataset [Saito 1994]. This dataset enables experiments with varying values of (a) the number of time series n and (b) the time series length m, without changing any of its general properties. We report the CPU runtime[7] over values for n and m that are orders of magnitude larger than those for the majority of the datasets in the UCR archive [Keogh et al. 2015]. Specifically, in Figure 8a we vary m from 10,000 to 50,000 while we set $n = 500$. We observe that SBD is an order of magnitude slower (8-18x) than ED with varying lengths while cDTW$^{Opt}_{LB}$ is two orders of magnitude slower (102-679x) and cDTW^{Opt} is two to three orders of magnitude slower (732-4969x) than ED. SBD remains an order of magnitude faster (12x-48x) than cDTW$^{Opt}_{LB}$ and one to two orders of magnitude faster (87-354x) than cDTW^{Opt}. Similarly, in Figure 8b we vary n from 20,000 to 100,000 while we set $m = 4,000$. We observe that all measures scale linearly with the number of time series when m is static and SBD remains less than an order of magnitude slower (6.9x) than ED while cDTW$^{Opt}_{LB}$ is an order of magnitude slower (62x) and cDTW^{Opt} is two orders of magnitude slower (464x) than ED.

6.3. k-Shape and k-MS Against Other Scalable Methods

Comparison against k-AVG+ED: Having shown the robustness of SBD, we now compare k-Shape and k-MS against scalable time-series clustering algorithms. Table IV reports the performance of variants of k-means against k-AVG+ED, using their Rand Index on the 85 datasets (see Section 5). From all these variants of k-means, only k-Shape and k-MS outperform k-AVG+ED with statistical significance, as we show next. In most cases, replacing ED in k-means with other distances not only does not improve accuracy significantly, but in certain cases results in substantially lower performance. For example, k-AVG+SBD achieves higher accuracy than k-AVG+ED in 56% of the datasets, but the differences in ac-

[7]For this experiment, we distributed the computation across 8 processes. The code was extracted and compiled into Matlab's C/C++ MEX files.

Algorithm	>	=	<	Better	Worse	[0,10x)	[10x,100x)	[100x,+∞)
k-AVG+SBD	48	5	32	✗	✗	53	32	0
k-AVG+DTW	24	2	59	✗	✓	1	11	73
KSC	32	2	51	✗	✓	1	3	81
k-DBA	38	1	46	✗	✗	0	0	85
k-Shape+DTW	32	2	51	✗	✗	0	7	78
k-Shape	57	6	22	✓	✗	17	55	13
k-MS	64	3	18	✓	✗	3	46	36

Table IV: Comparison of k-means variants against k-AVG+ED. Columns ">," "=," and "<" denote the number of datasets over which a method is better, equal, or worse, respectively, in comparison to k-AVG+ED. "Better" indicates that a method outperforms k-AVG+ED with statistical significance whereas "Worse" indicates that k-AVG+ED outperforms a method with statistical significance. Columns "[0, 10x)," "[10x, 100x)," and "[100x, $+\infty$)" denote the number of datasets over which a method is slower up to 10x, between 10x but no higher than 100x, and at least 100x, respectively, in comparison to k-AVG+ED.

curacy are not statistically significant. Interestingly, when DTW is combined with k-means, in k-AVG+DTW, the performance is significantly worse than with k-AVG+ED.

Even in cases where both the distance measure and the centroid computation method of k-means are modified, the performance does not improve significantly in comparison to k-AVG+ED. k-DBA outperforms k-AVG+DTW with statistical significance in 67 out of 85 datasets. Both of these approaches use DTW as distance measure, but k-DBA also modifies its centroid computation method. This modification significantly improves the performance of k-DBA over that of k-AVG+DTW, with an average improvement in Rand Index of 19%. Despite this improvement, k-DBA still does not perform better than k-AVG+ED in a statistical significant manner.[8] Another algorithm that modifies both the distance measure and the centroid computation method of k-means is KSC. In contrast to k-DBA, KSC is significantly worse than k-AVG+ED. We note that this finding contradicts our previous analysis on a subset of 48 datasets [Paparrizos and Gravano 2015] where k-AVG+ED was shown to perform better in 54% of the datasets, but not in a statistically significant manner. Our current analysis is based on almost twice as many datasets as our analysis in [Paparrizos and Gravano 2015], which provides substantially more evidence to the Wilcoxon test. (The p-value for Wilcoxon was very close to the confidence level in [Paparrizos and Gravano 2015], which resulted in considering the difference in accuracy of k-AVG+ED and KSC negligible.)

Comparison of k-MS against k-Shape: Having shown that inadequate modifications of the distance and the centroid computation method of k-means do not improve accuracy, we now evaluate k-Shape and k-MS, the only k-means variants with significant improvement in accuracy over k-AVG+ED. Specifically, k-Shape performs better than k-AVG+ED in 67% of the datasets and k-MS performs better than k-AVG+ED in 75% of the datasets. Importantly, in both of these comparisons, Wilcoxon indicates the superiority of k-Shape and k-MS against k-AVG+ED. Comparing now the performance of k-Shape and k-MS, our results show that k-MS performs better, equally, and worse in 59, 4, and 22 datasets, respectively, in comparison to k-Shape. Importantly, Wilcoxon indicates that the difference in accuracy of k-MS and k-Shape is statistically significant. Therefore, we can conclude that k-Shape and k-MS are the only k-means variants that significantly outperform k-AVG+ED and, importantly, k-MS is the only variant that significantly outperforms k-Shape.

Comparison against k-DBA and KSC: Both k-DBA and KSC are similar to k-Shape and k-MS in that they all modify both the centroid computation method and the distance measure of k-means. Therefore, to understand the impact of these modifications, we com-

[8]Even when multiple refinements of k-DBA's centroids are performed per iteration, k-DBA still does not outperform k-AVG+ED. In particular, performing five refinements per iteration improves the Rand Index by 4% but runtime increases by 30% according to experiments over 48 datasets [Paparrizos and Gravano 2015]. (These 48 datasets are a subset of the 85 datasets in our experiments.)

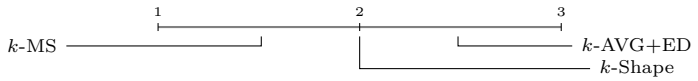


Fig. 9: Ranking of the best performing variants of k-means, namely, k-MS and k-Shape, based on the average of their ranks across datasets.

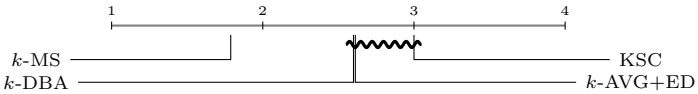


Fig. 10: Ranking of k-means variants based on the average of their ranks across datasets. The wiggly line connects all techniques that do not perform statistically differently according to the Nemenyi test.

pare them against our k-Shape and k-MS algorithms. k-Shape performs equally or better in 59 datasets in comparison to KSC and performs equally or better in 58 datasets in comparison to k-DBA. In both of these comparisons, the statistical test indicates the superiority of k-Shape. Similarly, k-MS performs equally or better in 62 datasets in comparison to KSC and performs equally or better in 61 datasets in comparison to k-DBA. In all these comparisons, the statistical test indicates the superiority of k-Shape and k-MS.

Statistical analysis: In addition to the pairwise comparisons performed with the Wilcoxon test, we further evaluate the significance of the differences in algorithm performance when considered all together. Figure 9 shows the average rank across datasets of k-MS, k-Shape, and k-AVG+ED, the best performing scalable clustering methods. k-MS is the top method, with an average rank of 1.51, meaning that k-MS achieved better rank in the majority of the datasets. The Friedman test rejects the null hypothesis that all algorithms behave similarly, and, hence, we proceed with a post-hoc Nemenyi test, to evaluate the significance of the differences in the ranks. We observe that the difference in ranks of each method are statistically significant: k-Shape is significantly better than k-AVG+ED and, in turn, k-MS is significantly better than k-Shape. Figure 10 shows the average rank across datasets of k-MS and k-means variants that modify both the centroid computation method and the distance measure of k-means. As before, k-MS is ranked first, with an average rank of 1.78. We observe that the ranks of KSC, k-DBA, and k-AVG+ED do not present a statistically significant difference, whereas k-MS, which is ranked first, is significantly better than the others. We note that according to the Wilcoxon test, as mentioned earlier, KSC is significantly worse than k-AVG+ED. However, when KSC, k-DBA, k-AVG+ED, and k-MS are considered collectively, KSC performs similarly to k-DBA and k-AVG+ED. (We observe similar findings for k-Shape in our analysis over 85 datasets. We omit figures due to space limitations. In [Paparrizos and Gravano 2015], we provide such comparisons on a subset of 48 datasets.) To conclude, modifying k-means with inappropriate distance measures or centroid computation methods might lead to unexpected results. The same holds for k-Shape, where the use of DTW, in k-Shape+DTW, results in a significant drop in performance (i.e., k-Shape outperforms k-Shape+DTW with statistical significance in 60 out of 85 datasets).

Efficiency: k-Shape and k-MS are the only algorithms that outperform all k-means variants, including the simple, yet robust, k-AVG+ED. We now investigate whether the superiority of k-Shape and k-MS in terms of accuracy has an associated penalty in efficiency. Table IV shows the number of datasets over which each method is slower than k-AVG+ED. k-Shape is the fastest method that significantly outperforms k-AVG+ED. k-MS is slower than k-Shape due to the additional cost required to compute more centroids. However, k-MS remains only up to 10x slower than k-Shape in 93% of the datasets. Importantly, k-MS remains significantly faster than all k-means variants that modify both the distance and the centroid computation method of k-means. Specifically, KSC performs up to 10x, 10x but less than 100x, and at least 100x slower than k-Shape, in 20%, 78%, and 2% of the datasets, respectively. KSC performs up to 10x slower than k-MS in 60% of the datasets, and 10x

Algorithm	>	=	<	Better	Worse
H-S+ED	17	1	67	✗	✓
H-S+cDTW	21	0	64	✗	✓
H-S+SBD	20	1	64	✗	✓
H-A+ED	16	0	69	✗	✓
H-A+cDTW	22	0	63	✗	✓
H-A+SBD	23	0	62	✗	✓
H-C+ED	28	0	57	✗	✓
H-C+cDTW	33	0	52	✗	✓
H-C+SBD	35	0	50	✗	✓
S+ED	27	1	57	✗	✓
S+cDTW	36	1	48	✗	✓
S+SBD	61	0	24	✓	✗
PAM+ED	43	1	41	✗	✗
PAM+cDTW	55	2	28	✓	✗
PAM+SBD	58	2	25	✓	✗

Table V: Comparison of hierarchical, spectral, and k-medoids variants against k-AVG+ED. Columns ">," "=," and "<" denote the number of datasets over which an algorithm is better, equal, or worse, respectively, in comparison to k-AVG+ED. "Better" indicates that an algorithm outperforms k-AVG+ED with statistical significance whereas "Worse" indicates that k-AVG+ED outperforms an algorithm with statistical significance.

but less than 100x slower than k-MS in 40%. k-DBA performs at least 100x slower than k-Shape in 74% of the datasets and at least 100x slower than k-MS in 44% of the datasets.

6.4. k-Shape and k-MS Against Non-Scalable Methods

Comparison against k-AVG+ED: Up to now, we have focused our evaluation on scalable clustering algorithms. In order to show the robustness of k-Shape and k-MS in terms of accuracy beyond scalable approaches, we now ignore scalability and compare k-Shape and k-MS against clustering methods that scale quadratically with the number of time series, namely, hierarchical, spectral, and k-medoids methods. Table V reports the performance of non-scalable methods against k-AVG+ED. Among all existing state-of-the-art methods that use ED or cDTW as distance measures, only partitional methods perform similarly to or better than k-AVG+ED. In particular, PAM+cDTW is the only method that outperforms k-AVG+ED with statistical significance. PAM+ED achieves better performance in 51% of the datasets in comparison to k-AVG+ED; however, this difference is not statistically significant. Moreover, PAM+cDTW performs better in 51 datasets, equal in 2 datasets, and worse in 32 datasets relative to PAM+ED. For this comparison, the statistical significance test indicates the superiority of PAM+cDTW. We note that constraining the warping window for DTW does not affect significantly the accuracy of clustering methods. Figure 11 shows the ranking of PAM+DTW, PAM+cDTW^5, PAM+cDTW^{10}, and k-AVG+ED across all datasets. PAM+DTW, PAM+cDTW^5, and PAM+cDTW^{10} do not present significant difference in accuracy, but PAM+DTW is ranked first. This is contradictory to the findings for the classification task where DTW was ranked last (see Figure 6).

All combinations of hierarchical clustering, with all different linkage criteria, perform poorly in comparison to k-AVG+ED. Interestingly, k-AVG+ED outperforms all of them with statistical significance. We observe that the major difference in performance among hierarchical methods is attributed to the linkage criterion and not to the distance measure, as we noted in [Paparrizos and Gravano 2015]. This highlights the importance of the clustering method, in addition to the distance measure for time-series clustering. Similarly to hierarchical methods, spectral methods also perform poorly against k-AVG+ED. S+cDTW performs better in more datasets than S+ED in comparison to k-AVG+ED: comparing S+cDTW with S+ED, S+cDTW achieves similar or better accuracy in 41 datasets, but this difference is not significant. Importantly, k-AVG+ED is significantly better than both S+ED and S+cDTW. Therefore, for hierarchical and spectral methods, these distance mea-

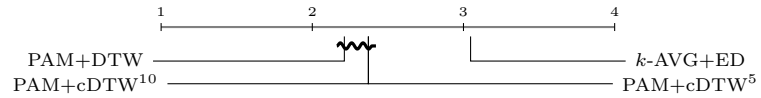


Fig. 11: Ranking of partitional methods that outperform k-AVG+ED based on the average of their ranks across datasets. The wiggly line connects all techniques that do not perform statistically differently according to the Nemenyi test.

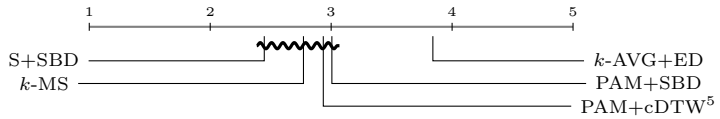


Fig. 12: Ranking of methods that outperform k-AVG+ED based on the average of their ranks across datasets. The wiggly line connects all techniques that do not perform statistically differently according to the Nemenyi test.

sures have a small impact on their performance.

k-Shape and k-MS against H-C+cDTW, S+cDTW, and PAM+cDTW: Among all methods using cDTW as distance measure that we evaluated, only PAM+cDTW outperforms k-AVG+ED with statistical significance. Similarly to k-AVG+ED, k-Shape and k-MS also outperform H-C+cDTW and S+cDTW with statistical significance. Therefore, we compare this approach with k-Shape and k-MS. PAM+cDTW performs equally or better in 45 datasets in comparison to k-Shape and equally or better in 41 datasets in comparison to k-MS, but these differences are not significant.

For completeness, we also evaluate SBD with hierarchical, spectral, and k-medoids methods. For hierarchical methods, H-C+SBD is better than H-A+SBD, and, in turn, H-A+SBD is better than H-S+SBD, all with statistical significance. S+SBD, in contrast to S+ED and S+cDTW, outperforms k-AVG+ED in 61 out of 85 datasets, with statistical significance. We note that S+SBD is the method that outperforms k-AVG+ED in the largest number of datasets (72%) than any other method evaluated in this work. S+SBD is also significantly better than S+ED and S+cDTW, but S+SBD does not outperform k-Shape or k-MS. Similarly, PAM+SBD performs better in 58, equal in 2, and worse in 25 datasets in comparison to k-AVG+ED. The statistical test suggests that PAM+SBD is significantly better than k-AVG+ED but not better than k-Shape or k-MS.

Statistical analysis: We evaluate the significance of the differences in algorithm performance for all algorithms that significantly outperform k-AVG+ED. (We omit k-Shape to simplify our analysis. Our analysis on 85 datasets indicate similar behavior as on a subset of 48 datasets [Paparrizos and Gravano 2015].) Figure 12 shows that k-MS, PAM+SBD, PAM+cDTW, and S+SBD do not present a significant difference in accuracy, whereas k-AVG+ED, which is ranked last, is significantly worse than the others.

From our extensive evaluation of existing clustering approaches for time series that use ED, cDTW, or DTW as distance measures, PAM+cDTW is the only approach that achieves similar — but not better — results compared to k-Shape and k-MS. In contrast to k-Shape and k-MS, PAM+cDTW has two drawbacks that make it an unrealistic choice for time-series clustering: (i) its distance measure, cDTW, requires tuning to improve its performance and reduce the computation cost; and (ii) the computation of the dissimilarity matrix that PAM+cDTW requires as input makes it unable to scale in both time and space. For example, the matrix computation alone is already two orders of magnitude slower than the computation required by k-Shape. Thus, k-Shape and k-MS emerge as domain-independent, highly accurate, and scalable approaches for time-series clustering.

Algorithm	>	=	<	Better	Worse
DBSCAN^3+ED	25	0	60	✗	✓
DBSCAN^{Best}+ED	32	0	53	✗	✓
DBSCAN^3+SBD	26	0	59	✗	✓
DBSCAN^{Best}+SBD	33	0	52	✗	✓
DBSCAN^3+cDTW	30	0	55	✗	✓
DBSCAN^{Best}+cDTW	37	0	48	✗	✓
TADPole^3	35	1	49	✗	✗
TADPole^{Best}	41	1	43	✗	✗
U-Shapelets$^{0.5}$	31	0	54	✗	✓
U-Shapelets^{Best}	44	0	41	✗	✗

Table VI: Comparison of variants of density-based and shapelet-based methods against k-AVG+ED. Columns ">," "=," and "<" denote the number of datasets over which an algorithm is better, equal, or worse, respectively, in comparison to k-AVG+ED. "Better" indicates that an algorithm outperforms k-AVG+ED with statistical significance whereas "Worse" indicates that k-AVG+ED outperforms an algorithm with statistical significance. "Rand Index" denotes the accuracy achieved in the 85 datasets.

6.5. k-Shape and k-MS Against Outlier-Aware Methods

Comparison against k-AVG+ED: Up to now, we have focused our evaluation on state-of-the-art scalable and non-scalable clustering methods that do not exploit any particular mechanism to handle outliers in the datasets. Table VI reports the performance of outlier-aware methods (see Section 5) against k-AVG+ED, using their Rand Index on the 85 datasets (see Section 5). From all these approaches, none outperforms k-AVG+ED and, importantly, k-AVG+ED significantly outperforms all variants of DBSCAN. In particular, k-AVG+ED achieves better performance in 71%, 69%, and 65% of the datasets in comparison to DBSCAN^3+ED, DBSCAN^3+SBD, and DBSCAN^3+cDTW, respectively. DBSCAN^3+cDTW is the strongest of the combinations of DBSCAN with other distance measures. Specifically, DBSCAN^3+cDTW performs at least as well as DBSCAN^3+ED in 50 datasets and at least as well as DBSCAN^3+SBD in 45 datasets. However, the Wilcoxon test suggests that these differences in accuracy are not statistically significant.

Similarly to DBSCAN, TADPole does not outperform k-AVG+ED. Specifically, k-AVG+ED performs at least as well as TADPole^3 in 50 datasets, but this difference is not significant. Comparing DBSCAN and TADPole, we observe that TADPole^3 significantly outperforms all DBSCAN variants. In particular, TADPole^3 outperforms DBSCAN^3+cDTW, DBSCAN^3+SBD, and DBSCAN^3+ED in 54, 56, and 55 datasets, respectively.

As discussed in Section 5, both DBSCAN and TADPole require as input two parameters. For fairness in the comparison, we now also report experimental results for the DBSCAN and TADPole variants when they receive as input the best performing parameters for each dataset, as described in Section 5. DBSCAN^{Best}+cDTW performs at least as well as DBSCAN^{Best}+ED and DBSCAN^{Best}+SBD in 46 datasets, but these differences are not statistically significant. Interestingly, k-AVG+ED performs at least as well as DBSCAN^{Best}+cDTW in 48 datasets, at least as well as DBSCAN^{Best}+SBD in 52 datasets, and at least as well as DBSCAN^{Best}+ED in 53 datasets, and all these differences in accuracy are statistically significant. In contrast, k-AVG+ED performs equally or better in 44 in comparison to TADPole^{Best}, but this difference in accuracy is not statistically significant.

Comparison against k-Shape and k-MS: Among all density-based methods we evaluated, none outperformed k-AVG+ED with statistical significance. Therefore, even though k-Shape and k-MS are significantly better than k-AVG+ED, we evaluate k-Shape and k-MS against all density-based methods. Similarly to k-AVG+ED, k-Shape and k-MS also significantly outperform DBSCAN^3+ED, DBSCAN^3+SBD, and DBSCAN^3+cDTW. In particular, k-Shape and k-MS perform at least as well as DBSCAN^3+cDTW in 74% of the datasets. In contrast to k-AVG+ED, k-Shape significantly outperforms TADPole^3 in 70% of the datasets and k-MS significantly outperforms TADPole^3 in 74% of the datasets.

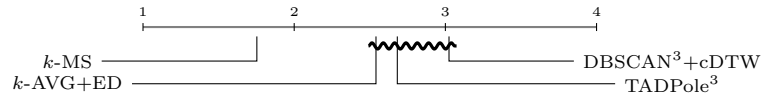


Fig. 13: Ranking of TADPole^3, DBSCAN^3, k-AVG+ED, and k-MS based on the average of their ranks across datasets. The wiggly line connects all techniques that do not perform statistically differently according to the Nemenyi test.

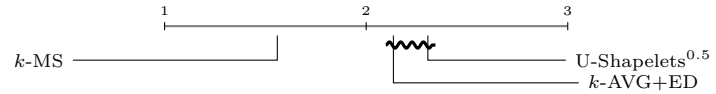


Fig. 14: Ranking of U-Shapelets$^{0.5}$, k-AVG+ED, and k-MS based on the average of their ranks across datasets. The wiggly line connects all techniques that do not perform statistically differently according to the Nemenyi test.

Statistical Analysis: Up to now, we evaluated density-based methods pairwise. We now evaluate the significance of the differences in algorithm performance for the best performing density-based methods when considered collectively. The Friedman test rejects the null hypothesis that all algorithms behave similarly, and, hence, as before, we proceed with a post-hoc Nemenyi test. Figure 13 shows that TADPole^3 and DBSCAN^3+cDTW do not exhibit a significant difference in accuracy. However, in contrast to TADPole^3, DBSCAN^3+cDTW is significantly worse than k-AVG+ED. k-MS, which is ranked first, is significantly better than both density-based methods. (We omit similar results for k-Shape.)

6.6. k-Shape and k-MS Against a Shapelet-Based Method

Comparison against k-AVG+ED: To conclude our analysis, we now evaluate U-Shapelets, a shapelet-based method that exploits patterns in subsequences of the entire time series (see Section 5). The last two rows of Table VI report the performance of U-Shapelets variants against k-AVG+ED, using their Rand Index on the 85 datasets (see Section 5). These two variants do not outperform k-AVG+ED and, importantly, k-AVG+ED significantly outperforms U-Shapelets$^{0.5}$. In particular, k-AVG+ED performs better in 54 and worse in 31 datasets in comparison to U-Shapelets$^{0.5}$. For completeness, we note that U-Shapelets^{Best} significantly improves the performance of U-Shapelets. In particular, U-Shapelets^{Best} performs better in 57 and equal in 28 datasets in comparison to U-Shapelets$^{0.5}$. U-Shapelets^{Best} performs better in 44 and worse in 41 datasets in comparison to k-AVG+ED but the difference in accuracy is not statistically significant.

Comparison against DBSCAN^3+cDTW and TADPole^3: We now evaluate the performance of U-Shapelets$^{0.5}$ against density-based methods. U-Shapelets$^{0.5}$ performs better in 52, equal in 1, and worse in 32 datasets in comparison to the best performing DBSCAN variant, namely, DBSCAN^3+cDTW. The Wilcoxon text suggests that this difference in accuracy is not statistically significant, but with a p-value close to the confidence level. Similarly, TADPole^3 performs at least as well as U-Shapelets$^{0.5}$ in 47 datasets, but this difference in accuracy is not statistically significant.

Comparison against k-Shape and k-MS: Among all shapelet-based methods we evaluated, none outperformed k-AVG+ED. Therefore, even though k-Shape and k-MS are significantly better than k-AVG+ED, we evaluate k-Shape and k-MS against the best performing shapelet-based method: both k-Shape and k-MS outperform U-Shapelets$^{0.5}$ in 57 out of 85 datasets. The Wilcoxon test suggests that this difference in accuracy is significant.

Statistical Analysis: Having shown the superiority of k-Shape and k-MS against U-Shapelets$^{0.5}$, we now conclude our analysis with an evaluation of the significance of the differences in algorithm performance when considered collectively. The Friedman test rejects the null hypothesis that all algorithms behave similarly, and, hence, as before, we proceed with a post-hoc Nemenyi test. Figure 14 shows that k-AVG+ED and U-Shapelets$^{0.5}$ do not

Algorithm	k=1	k=2	k=3	k=4
Random	2.919	2.713	3.081	2.978
RT1	3.603	3.706	3.331	3.404
RT2	3.132	3.132	3.147	3.140
RT3	3.684	3.926	3.824	3.853
NSC	1.662	1.522	1.618	1.625
$\mathbf{R}_{Best}$-$\mathbf{R}_{NSC}$	1.257	1.191	1.463	1.353

Table VII: Comparison of Random, RT1, RT2, RT3, and NSC. Columns "k=1," "k=2," "k=3," and "k=4" denote the average ranking, in terms of accuracy, of each algorithm over 75 datasets when k centroids represent each class. The last row represents the difference in ranking of the best performing rival method and NSC. Differences higher than 0.55 are considered statistically significant according to the Nemenyi test.

present a significant difference, whereas k-MS, which is ranked first, is significantly better than k-AVG+ED and U-Shapelets$^{0.5}$. (We omit similar results for k-Shape.)

6.7. Evaluation of NSC

Up to now, we have demonstrated the robustness of k-Shape and k-MS as standalone clustering methods for time series. In this section, we evaluate the performance of k-Shape, our simplest—and competitive—clustering method, as a subroutine to effectively reduce the search space of one-nearest-neighbor algorithms.

We perform three experiments to assess the performance of NSC against rival methods. First, following [Petitjean et al. 2014; 2015], we evaluate performance when a limited number of centroids are allowed per class. Second, we evaluate the impact of several key characteristics of the methods. Third, we compare the performance of 1-NN classifiers against three variants of NSC, namely, NSC+ED, NSC+SBD, and NSC+cDTW. The first two experiments are over the 75 datasets—out of the 85 datasets—for which all algorithms terminated after running continuously for one week, as discussed in Section 5. In contrast, the third experiment, which is based solely on the efficient NSC, is over the full set of 85 datasets.

Table VII reports the average rank, in terms of accuracy, of Random, RT1, RT2, RT3, and NSC when k centroids represent each class in the training set.[9] NSC is the top method for all values of k that we considered, meaning that NSC achieved higher accuracy in the majority of the datasets. Specifically, NSC significantly outperforms each of the rival methods in at least 80% of the datasets for all values of k, according to Wilcoxon. To understand the performance of NSC in comparison with Random, RT1, RT2, and RT3, we also evaluate the significance of their differences in accuracy when considered all together. The Friedman test rejects the null hypothesis that all algorithms behave similarly for all values of k and, hence, we proceed with a post-hoc Nemenyi test. The last row in Table VII shows the difference in ranks of NSC and the best performing method. According to the Nemeyi test, differences higher than 0.55 are significant. Therefore, for all different values of k, the difference in the rank of NSC against all rival methods is statistically significant.

Having shown the superiority in terms of accuracy of NSC against other methods, we now explore the impact on four important characteristics of NSC, RT1, RT2, and RT3 in their attempt to outperform their corresponding 1-NN classifier.[10] In Table VIII, we first evaluate the methods based on the number of instances required to outperform their corresponding 1-NN classifier. NSC is the top method, meaning that NSC required fewer centroids than the other methods to outperform its corresponding 1-NN classifier in the majority of datasets. The Friedman test rejects the null hypothesis that all algorithms behave similarly and,

[9]We vary k from 1 to 4 in order to include datasets with a small number of instances per class.
[10]We exclude the Random method from this experiment as Random could not outperform 1-NN classifier with cDTW^5 in the majority of the datasets.

Algorithm	Average Rank	Initialization Cost			Query Time Improvement				Amortization Cost		
		[0,10x)	[10x,100x)	[100x,+∞)	[0,25)	[25,50)	[50,75)	[75,100]	[0,10x)	[10x,100x)	[100x,+∞)
RT1	2.838	40	30	5	48	10	7	10	9	17	48
RT2	2.561				43	9	10	13			
RT3	2.696				45	7	12	11			
NSC	1.905	49	26	0	21	8	11	35	45	16	14

Table VIII: Comparison of RT1, RT2, RT3, and NSC. Column "Average Rank" denotes the average rank in terms of the number of centroids required to outperform the corresponding 1-NN accuracy. Column "Initialization Cost" denotes the number of datasets over which an algorithm requires up to 10x (column "[0, 10x)"), between 10x but no higher than 100x (column "[10x, 100x)"), and at least 100x (column "[100x, +∞)") more preprocessing than the version of the Random method with the corresponding distance. Column "Query Time Improvement" indicates the number of datasets over which an algorithm improves the query response time up to 25% (column "[0, 25)"), from 25% to 50% (column "[25, 50)"), from 50% to 75% (column "[50, 75)"), and from 75% to 100% (column "[75, 100)") relative to the corresponding 1-NN. Column "Amortization Cost" denotes the number of datasets over which an algorithm requires to increase the number of queries up to 10x (column "[0, 10x)"), between 10x but no higher than 100x (column "[10x, 100x)"), and at least 100x (column "[100x, +∞)") relative to the corresponding 1-NN to amortize the initialization cost.

hence, we proceed as before with a post-hoc Nemenyi test. According to Nemenyi, the difference in the rank of NSC against all rival methods is statistically significant.

We then investigate the associated initialization cost for each method to outperform the corresponding 1-NN classifiers. Considering that 1-NN classifiers require no preprocessing of the instances in the training set (i.e., the initialization cost of 1-NN classifiers is negligible), we measure the initialization cost of each method against the corresponding Random method. NSC performs up to 10x slower than Random in 65% of the datasets, while in the remaining 35% of the datasets NSC performs between 10x but no higher than 100x slower in comparison to Random. In contrast, RT1, RT2, and RT3 perform up to 10x, between 10x but no higher than 100x, and at least 100x slower than Random in 53%, 40%, and 7% of the datasets, respectively.[11] Therefore, we can conclude that NSC is faster at preprocessing the instances in the training set than all other reduction techniques.

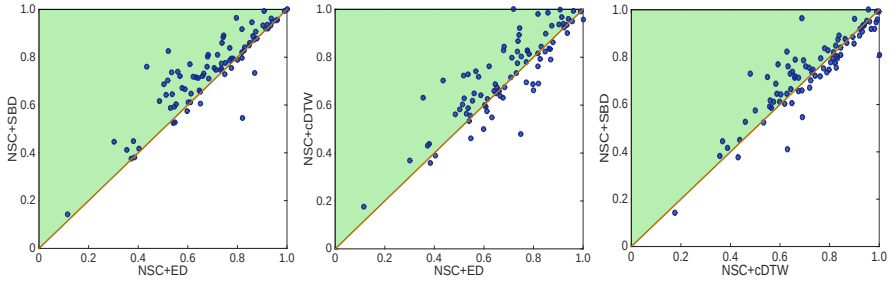
Considering that all methods significantly reduce the number of instances in the training set in comparison with 1-NN classifiers, we now evaluate the associated improvement in query runtime in comparison with the corresponding 1-NN classifiers. Specifically, NSC reduces the query response time by at least 50% in 61% of the datasets. In contrast, the best reduction techniques, namely, RT1 and RT2, reduce the query response time by at least 50% in 31% of the datasets. Therefore, NSC reduces the query response time by at least 50% in approximately twice as many datasets as do the reduction techniques.

Finally, we report the number of queries each method has to process to amortize the initialization cost in comparison with the associated 1-NN classifiers. NSC increases the number of queries up to 10x in 60% of the datasets and by at least 10x in 40% of the datasets in order to amortize the initialization cost. In contrast, the reduction techniques increase the number of queries up to 10x in 13% of the datasets and by at least 10x in 67% of the datasets. Therefore, we can conclude that NSC is a competitive method to reduce the instances required for training the 1-NN classifiers. NSC is significantly more accurate than rival methods, requires fewer centroids to effectively summarize the search space, and yields important reductions in query response time in comparison with 1-NN classifiers.

[11] Considering that all reduction techniques are statistically indistinguishable in terms of accuracy and number of instances required to outperform 1-NN classifier with cDTW[5], in Table VIII we only report the "Initialization Cost" and the "Amortization Cost" of the best performing reduction technique, namely, RT2, in order to simplify our analysis.

Algorithm	>	=	<	Better	Worse
NSC+cDTW	53	2	30	✓	✗
NSC+SBD	67	3	15	✓	✗

Table IX: Comparison of NSC+cDTW and NSC+SBD against NSC+ED. Columns ">," "=," and "<" denote the number of datasets over which an algorithm is better, equal, or worse, respectively, in comparison to NSC+ED. "Better" indicates that an algorithm outperforms NSC+ED with statistical significance whereas "Worse" indicates that NSC+ED outperforms an algorithm with statistical significance.



(a) NSC+SBD vs. NSC+ED (b) NSC+cDTW vs. NSC+ED (c) NSC+SBD vs. NSC+cDTW

Fig. 15: Comparison of NSC+SBD against NSC+ED (a), NSC+cDTW against NSC+ED (b), and NSC+SB against NSC+cDTW (c) over 85 datasets. Circles above the diagonal indicate datasets over which the method in the vertical axis has better accuracy than the method in the horizontal axis.

Having shown the superiority of NSC against other data editing techniques, we now evaluate the effectiveness of NSC when combined with competitive distance measures. Table IX reports the performance of NSC+SBD and NSC+cDTW against NSC+ED on 85 datasets. Both NSC+SBD and NSC+cDTW significantly outperform NSC+ED. In particular, NSC+SBD performs at least as well as NSC+ED in 82% of the datasets and NSC+cDTW performs at least as well as NSC+ED in 65% of the datasets. Figures 15a and 15b illustrate the superiority of NSC+SBD and NSC+cDTW against NSC+ED, respectively. In contrast, NSC+SBD performs better in 45, equal in 1, and worse in 39 datasets in comparison to NSC+cDTW. Wilcoxon suggests that this difference in accuracy is not statistically significant. Figure 15c shows the performance of NSC+SBD against NSC+cDTW. In addition to the pairwise comparisons performed with Wilcoxon, we further evaluate the significance of the differences in algorithm performance when considered collectively. Figure 16 shows the average rank across datasets of each NSC variant. NSC+SBD is the top method, with an average rank of 1.66, meaning that NSC+SBD achieved better rank than NSC+ED and NSC+cDTW in the majority of the datasets. The Friedman test rejects the null hypothesis that all algorithms behave similarly and, hence, we proceed with a post-hoc Nemenyi test, to evaluate the significance of the differences in the ranks. We observe that the ranks between NSC+SBD and NSC+cDTW do not present any significant difference, whereas NSC+ED, which is ranked last, is significantly worse than the others.

Considering now the performance of NSC and 1-NN variants for the same distances collectively, we observe no significant difference between NSC+SBD and 1-NN+SBD, and no significant difference between NSC+ED and 1-NN+ED. However, NSC+cDTW is significantly worse than 1-NN+cDTW, which indicates that k-Shape is not an appropriate clustering method to compute centroids when cDTW is used. Instead, other methods, such as PAM+cDTW or k-DBA, would be preferable in conjunction with cDTW.

In conclusion, k-Shape is an effective method to summarize time series, and 1-NN classifiers can benefit from using k-Shape as a subroutine to significantly reduce their search space. Importantly, these findings have implications in settings where time-series classifica-

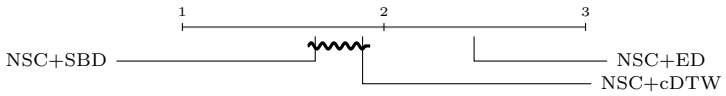


Fig. 16: Ranking of nearest shape classifier methods based on the average of their ranks across datasets. The wiggly line connects all techniques that do not perform statistically differently according to the Nemenyi test.

tion methods operate under limited computational resources [Petitjean et al. 2014; 2015] or within a tight time budget [Ding et al. 2015; Pelkonen et al. 2015].

6.8. Summary of Results

In short, our experimental evaluation suggests that: (1) cross-correlation measures (e.g., SBD), which are not widely adopted as time-series distance measures, are as competitive as state-of-the-art measures, such as cDTW and DTW, but significantly faster; (2) only the optimally tuned cDTW measure significantly outperforms the DTW measure when 1-NN classification is used to evaluate distance measures, in contrast to the belief that cDTW generally improves the accuracy of DTW; (3) for time-series clustering, the DTW measure often achieves better performance than the cDTW measure, which contradicts the findings under 1-NN classification; (4) the k-means algorithm with ED, in contrast to what has been reported in the literature, is a robust approach for time-series clustering, but inadequate modifications of its distance measure and centroid computation can reduce its performance; (5) the choice of clustering method, such as partitional versus density-based methods, which was believed to be less important than that of distance measure, is as important as the choice of distance measure; (6) the intrinsic characteristics of clustering algorithms, such as the linkage criterion in hierarchical clustering, are more important than the choice of distance measures; (7) the attempts to isolate and ignore abnormal behaviors in time series can significantly hurt the performance of outlier-aware clustering methods; (8) k-Shape and k-MS are highly accurate and efficient methods for time-series clustering; and (9) exploiting clustering methods, such as k-Shape, as subroutines to reduce the search space of 1-NN algorithms can lead to fast and accurate classification methods for time series.

7. RELATED WORK

This article substantially extends our previous paper [Paparrizos and Gravano 2015] on efficient and accurate clustering of time series. The earlier paper presented the SBD distance measure (Section 3.1), the k-Shape clustering algorithm (Section 3.3), and an experimental evaluation over 48 datasets. We can summarize the contributions of the current article as follows: (i) we present a new shape-based clustering algorithm, k-MS, that takes into consideration the proximity and spatial distribution of time series; (ii) we substantially expand the analysis and experimental evaluation in [Paparrizos and Gravano 2015]; and (iii) we demonstrate that we can rely on our approaches to improve 1-NN time-series classification.

In particular, we present k-MS, a novel time-series clustering method (see Section 3). k-MS relies on SBD to consider the shapes of time series while comparing them and on the same iterative refinement procedure used in k-Shape to allow linear scalability with the number of time series. However, k-MS is based on a different method to compute cluster centroids than the one used for k-Shape. Specifically, k-Shape relies on the SE method to compute a single centroid per cluster assuming that a single centroid can effectively represent the characteristics of the underlying time series. In practice, time series are not uniformly distributed in space and noisy sequences and outliers complicate the extraction of a representative sequence. To avoid these important limitations, k-MS relies on MSE to compute multiple centroids per cluster in order to consider the proximity and spatial distribution of time series on each cluster. Our experimental results suggest that k-MS

behaves similarly to k-Shape in comparison to rival methods we consider in our work but, importantly, k-MS is significantly more accurate than k-Shape (see Sections 6.3 and 6.4).

In addition to our new clustering algorithm, this article expands the evaluation of our techniques along several dimensions. First, all of our results are now over the new version of the UCR archive of time-series datasets [Keogh et al. 2015]. This archive now includes 85 time-series datasets, a substantial increase over the earlier 48 datasets for our evaluation in [Paparrizos and Gravano 2015]. The larger number of datasets has allowed us to both confirm our earlier findings in an even more robust setting, as well as produce additional findings, all supported with a rigorous statistical significance analysis. Second, we also significantly expanded the families of alternative time-series clustering methods in our evaluation. We now include density-based methods and shapelet-based methods (see Sections 6.5 and 6.6). In contrast to all other clustering approaches considered in our earlier work, these clustering methods detect abnormal behaviors in the dataset at hand and attempt to isolate and ignore outliers in time series. Our results suggest that k-Shape and k-MS are significantly better than these outlier-aware methods (see Sections 6.5 and 6.6).

Furthermore, we show that we can accurately predict in advance if SBD will outperform other state-of-the-art distance measures on each dataset (see Section 6.1 and Figure 5). Additionally, we corroborate our performance results over optimized implementations for the state-of-the-art distance measures and we perform a large-scale runtime experiment of SBD as a function of the number and the length of the time series (see Section 6.2, Table III, and Figure 8). Our results show that there is merit in considering SBD as an alternative distance measure for time-series comparison. SBD combines the efficiency and scalability of ED with the accuracy of cDTW, without the need to tune parameters.

Finally, we also show how we can use k-Shape to improve other related time-series tasks. Motivated by recent successful efforts that use centroid computation methods [Gou et al. 2012] and clustering methods [Petitjean et al. 2014; 2015] to improve the performance of nearest-neighbor classifiers, we present NSC, a 1-NN classification method that relies on k-Shape as a subroutine to effectively reduce the search space for 1-NN classification algorithms (see Section 4). We evaluate NSC against data editing algorithms along several key characteristics and we show that we can significantly reduce the search space for 1-NN classifiers without loss in accuracy (see Section 6.7). Such 1-NN classifiers are useful for real-world settings where the classification of time series is required under limited computational resources or time constraints.

As additional related work, Section 2 provided an in-depth discussion of the state of the art for time-series clustering, which we will not repeat for brevity. Specifically, Section 2.1 summarized the relevant theoretical background, Section 2.2 reviewed common distortions in time series, and Section 2.3 discussed the most popular state-of-the-art distance measures for time series. (We refer the reader to [Ding et al. 2008; Wang et al. 2013] for a thorough review of the alternative time-series distance measures.) Section 2.4 highlighted existing state-of-the-art approaches for time-series clustering. (We refer the reader to [Warren Liao 2005] for a more detailed view of these approaches.) Finally, Section 2.5 discussed the methods for centroid computation that are part of many time-series clustering algorithms.

As argued throughout the paper, we focus on shape-based clustering of time series. Beyond shape-based clustering algorithms, statistical-based approaches use measures that summarize characteristics [Wang et al. 2006], coefficients of models [Kalpakis et al. 2001], or portions of time series (i.e., shapelets) [Zakaria et al. 2012] as descriptive features to cluster time series. Unfortunately, statistical-based approaches frequently require the non-trivial tuning of multiple parameters, which often leads to ad-hoc decisions, or their effectiveness has been established only for isolated domains and not in a domain-independent manner. Instead, shape-based approaches are general and leverage the vast literature on distance measures. In Section 6.6, we showed that both k-Shape and k-AVG+ED significantly outperform the U-Shapelets method [Zakaria et al. 2012].

The best performing shape-based approaches from the literature are partitional methods combined with scale- and shift-invariant distance measures. Among partitional methods, k-medoids [Kaufman and Rousseeuw 2009] is the most popular method as it enables the easy adoption of any shape-based distance measure [Ding et al. 2008; Wang et al. 2013]. However, k-medoids requires the computation of the full dissimilarity matrix among all time series, which makes it particularly slow and unable to scale. Recent alternative approaches [Gupta et al. 1996; Meesrikamolkul et al. 2012; Niennattrakul and Ratanamahatana 2009; Petitjean et al. 2011; Yang and Leskovec 2011] have focused on k-means [MacQueen 1967], which is scalable but requires the modification of the centroid computation method when the distance measure is altered, in order to support the same properties (e.g., scaling, translation, and shifting). Because DTW is the most prominent shape-based distance measure [Ding et al. 2008; Wang et al. 2013], the majority of the k-means approaches have proposed new centroid computation methods to be used in conjunction with DTW [Gupta et al. 1996; Meesrikamolkul et al. 2012; Niennattrakul and Ratanamahatana 2009; Petitjean et al. 2011]. k-DBA has been shown to be the most robust of these approaches [Petitjean et al. 2011]. Another approach worth mentioning is KSC [Yang and Leskovec 2011], which focuses on a different shape-based distance measure that offers simultaneously pairwise scaling and shifting of time series. Unfortunately, the effectiveness of such pairwise scaling and shifting, and, hence, KSC, has not been established beyond a limited number of datasets. In Sections 6.3 and 6.4, we evaluated k-Shape against shape-based approaches, including combinations of k-means and k-medoids with the most competitive distance measures, as well as k-DBA and KSC. Apart from shape-based approaches, we also compared k-Shape against hierarchical and spectral methods (see Section 6.4), density-based methods (see Section 6.5), and shapelet-based methods (see Section 6.6).

For completeness, we note that [Golay et al. 1998] used the arithmetic mean property for centroid computation and cross-correlation in distance measures with parameters to regulate the fuzziness of fuzzy clustering of fMRI data. (In Section 6, we showed that k-AVG+SBD is not competitive for our non-fuzzy setting.) [Goutte et al. 1999] used cross-correlation to transform fMRI data into features for clustering. [Baragona 2000] evaluated several weighting schemes of cross-correlation along with a genetic algorithm to avoid inappropriate initialization for the k-min clustering criterion [Sahni and Gonzalez 1976]. [Hőppner and Klawonn 2009] proposed a fuzzy c-means [Bezdek 2013] clustering method that uses a weighting scheme to penalize alignment of time series in larger lags when cross-correlation is used, a weighted arithmetic mean for centroid computation, and a mechanism to handle outliers given a user-defined threshold. Finally, cross-correlation was used to speed up subsequence matching [Mueen et al. 2014] and the computation of shapelets [Mueen et al. 2011], as well as for stream mining, pattern extraction, and time-series monitoring [Papadimitriou et al. 2006; Sakurai et al. 2005; Wu et al. 2010; Zhu and Shasha 2002].

8. CONCLUSIONS

In this article, we presented k-Shape and k-MS, two novel scalable partitional algorithms that preserve the shapes of time series. k-Shape and k-MS compare time series efficiently and compute centroids effectively under the scaling and shift invariances. k-Shape and k-MS use as their distance measure SBD, a normalized version of cross-correlation. To compute cluster centroids based on SBD, k-Shape extracts a single centroid per cluster. In contrast, k-MS extracts multiple centroids per cluster. Our extensive evaluation showed the robustness of SBD, k-Shape, and k-MS against state-of-the-art distance measures and clustering approaches for time series. In particular, SBD is an efficient and parameter-free distance measure that achieves similar results to the most accurate but computationally expensive distance measures that require parameter tuning. k-Shape and k-MS outperform all state-of-the-art scalable and non-scalable partitional, hierarchical, spectral, density-based, and shapelet-based clustering approaches, with only one method achieving similar performance.

Interestingly, this method is significantly slower than both k-Shape and k-MS and, importantly, its distance measure requires tuning. Beyond clustering, we demonstrated the effectiveness of k-Shape to reduce the search space of one-nearest-neighbor classifiers for time series. Overall, SBD, k-Shape, and k-MS emerge as domain-independent, accurate, and scalable approaches for time-series comparison and clustering.

We have identified many interesting directions for future work. For example, k-Shape and k-MS operate over a single time-series representation and cannot handle multiple representations. Considering that several transformations (e.g., smoothing) can reduce noise and eliminate outliers in time series, an extension of k-Shape and k-MS to leverage characteristics from multiple views could significantly improve their accuracy. Another direction is to explore the usefulness of k-Shape and k-MS as subroutines of other methods, just as we did for 1-NN classifiers. For example, anomaly detection in time-series streams occasionally relies on effective clustering of time-series subsequences to identify abnormal behaviors.

ACKNOWLEDGMENTS

We thank Or Biran, Christos Faloutsos, Eamonn Keogh, Kathy McKeown, Taesun Moon, François Petitjean, and Kapil Thadani for invaluable discussions and feedback. We also thank Ken Ross for sharing computing resources for our experiments. John Paparrizos is an Onassis Foundation Scholar.

REFERENCES

Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. 1993. Efficient Similarity Search In Sequence Databases. In *FODO*. 69–84.

Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. 2009. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning* 75, 2 (2009), 245–248.

Jonathan Alon, Stan Sclaroff, George Kollios, and Vladimir Pavlovic. 2003. Discovering clusters in motion time-series data. In *CVPR*. 375–381.

Anthony Bagnall, Aaron Bostrom, James Large, and Jason Lines. 2016. The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version. *arXiv preprint arXiv:1602.01711* (2016).

Anthony Bagnall and Jason Lines. 2014. An experimental evaluation of nearest neighbour time series classification. *arXiv preprint arXiv:1406.4757* (2014).

Anthony J Bagnall and Gareth J Janacek. 2004. Clustering time series from ARMA models with clipped data. In *KDD*. 49–58.

Ziv Bar-Joseph, Georg Gerber, David K Gifford, Tommi S Jaakkola, and Itamar Simon. 2002. A new approach to analyzing gene expression time series data. In *RECOMB*. 39–48.

Roberto Baragona. 2000. Genetic Algorithms And Cross-Correlation Clustering Of Time Series. (2000).

Gustavo EAPA Batista, Eamonn J Keogh, Oben Moses Tataw, and Vinícius MA de Souza. 2013. CID: An efficient complexity-invariant distance for time series. *Data Mining and Knowledge Discovery* (2013), 1–36.

Nurjahan Begum, Liudmila Ulanova, Jun Wang, and Eamonn Keogh. 2015. Accelerating Dynamic Time Warping Clustering with a Novel Admissible Pruning Strategy. In *SIGKDD*. 49–58.

Donald J Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series.. In *AAAI Workshop on KDD*. 359–370.

James C Bezdek. 2013. *Pattern recognition with fuzzy objective function algorithms*. Springer Science.

Yuhan Cai and Raymond Ng. 2004. Indexing spatio-temporal trajectories with Chebyshev polynomials. In *SIGMOD*. 599–610.

BB Chaudhuri. 1996. A new definition of neighborhood of a point in multi-dimensional space. *Pattern Recognition Letters* 17, 1 (1996), 11–17.

Lei Chen and Raymond Ng. 2004. On the marriage of Lp-norms and edit distance. In *VLDB*. 792–803.

Lei Chen, M Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *SIGMOD*. 491–502.

Qiuxia Chen, Lei Chen, Xiang Lian, Yunhao Liu, and Jeffrey Xu Yu. 2007a. Indexable PLA for efficient similarity search. In *VLDB*. 435–446.

Yueguo Chen, Mario A Nascimento, Beng Chin Ooi, and Anthony KH Tung. 2007b. Spade: On shape-based pattern detection in streaming time series. In *ICDE*. 786–795.

James W Cooley and John W Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.* 19, 90 (1965), 297–301.

Gautam Das, King-Ip Lin, Heikki Mannila, Gopal Renganathan, and Padhraic Smyth. 1998. Rule Discovery from Time Series. In *KDD*. 16–22.

Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research* 7 (2006), 1–30.

Evgenia Dimitriadou, Andreas Weingessel, and Kurt Hornik. 2002. A combination scheme for fuzzy clustering. *International Journal of Pattern Recognition and Artificial Intelligence* 16, 07 (2002), 901–912.

Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn Keogh. 2008. Querying and mining of time series data: Experimental comparison of representations and distance measures. *PVLDB* 1, 2 (2008), 1542–1552.

Rui Ding, Qiang Wang, Yingnong Dang, Qiang Fu, Haidong Zhang, and Dongmei Zhang. 2015. Yading: Fast clustering of large-scale time series data. *PVLDB* 8, 5 (2015), 473–484.

Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *SIGKDD*. 226–231.

Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. 1994. Fast Subsequence Matching in Time-series Databases. In *SIGMOD*. 419–429.

Maurizio Filippone, Francesco Camastra, Francesco Masulli, and Stefano Rovetta. 2008. A survey of kernel and spectral methods for clustering. *Pattern Recognition* 41, 1 (2008), 176–190.

Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. 2007. Index-based most similar trajectory search. In *ICDE*. 816–825.

Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Amer. Statist. Assoc.* 32 (1937), 675–701.

Matteo Frigo and Steven G Johnson. 2005. The design and implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

Martin Gavrilov, Dragomir Anguelov, Piotr Indyk, and Rajeev Motwani. 2000. Mining the stock market: Which measure is best?. In *KDD*. 487–496.

Rafael Giusti and Gustavo EAPA Batista. 2013. An Empirical Comparison of Dissimilarity Measures for Time Series Classification. In *BRACIS*. 82–88.

Steve Goddard, Sherri K Harms, Stephen E Reichenbach, Tsegaye Tadesse, and William J Waltman. 2003. Geospatial decision support for drought risk management. *Commun. ACM* 46, 1 (2003), 35–37.

Xavier Golay, Spyros Kollias, Gautier Stoll, Dieter Meier, Anton Valavanis, and Peter Boesiger. 1998. A new correlation-based fuzzy logic clustering algorithm for fMRI. *Magnetic Resonance in Medicine* 40, 2 (1998), 249–260.

Dina Q Goldin and Paris C Kanellakis. 1995. On similarity queries for time-series data: Constraint specification and implementation. In *CP*. 137–153.

Gene H Golub and Charles F Van Loan. 2012. *Matrix computations*. Vol. 3. JHU Press.

Jianping Gou, Zhang Yi, Lan Du, and Taisong Xiong. 2012. A local mean-based k-nearest centroid neighbor classifier. *The computer journal* 55, 9 (2012), 1058–1071.

Cyril Goutte, Peter Toft, Egill Rostrup, Finn Å Nielsen, and Lars Kai Hansen. 1999. On clustering fMRI time series. *NeuroImage* 9, 3 (1999), 298–310.

Lalit Gupta, Dennis L Molfese, Ravi Tammana, and Panagiotis G Simos. 1996. Nonlinear alignment and averaging for estimating the evoked potential. *IEEE Transactions on Biomedical Engineering* 43, 4 (1996), 348–356.

Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. 2001. On clustering validation techniques. *Journal of Intelligent Information Systems* 17, 2-3 (2001), 107–145.

Jiawei Han, Micheline Kamber, and Jian Pei. 2011. *Data Mining: Concepts and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc.

Pierre Hansen and Brigitte Jaumard. 1997. Cluster analysis and mathematical programming. *Mathematical Programming* 79, 1-3 (1997), 191–215.

Rie Honda, Shuai Wang, Tokio Kikuchi, and Osamu Konishi. 2002. Mining of moving objects from time-series images and its application to satellite weather imagery. *Journal of Intelligent Information Systems* 19, 1 (2002), 79–93.

Frank Hőppner and Frank Klawonn. 2009. Compensation of translational displacement in time series clustering using cross correlation. In *Advances in Intelligent Data Analysis VIII*. Springer, 71–82.

Bing Hu, Yanping Chen, and Eamonn Keogh. 2013. Time Series Classification under More Realistic Assumptions. In *SDM*. 578–586.

Konstantinos Kalpakis, Dhiral Gada, and Vasundhara Puttagunta. 2001. Distance measures for effective clustering of ARIMA time-series. In *ICDM*. 273–280.

Yitzhak Katznelson. 2004. *An introduction to harmonic analysis*. Cambridge University Press.

Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: An introduction to cluster analysis*. Vol. 344. John Wiley & Sons.

Eamonn Keogh. 2006. A decade of progress in indexing and mining large time series databases. In *VLDB*. 1268–1268.

Eamonn Keogh, Kaushik Chakrabarti, Michael Pazzani, and Sharad Mehrotra. 2001. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *SIGMOD*. 151–162.

Eamonn Keogh and Jessica Lin. 2005. Clustering of time-series subsequences is meaningless: Implications for previous and future research. *Knowledge and Information Systems* 8, 2 (2005), 154–177.

Eamonn Keogh and Chotirat Ann Ratanamahatana. 2005. Exact indexing of dynamic time warping. *Knowledge and Information Systems* 7, 3 (2005), 358–386.

Eamonn Keogh, Xiaopeng Xi, Li Wei, and Chotirat Ann Ratanamahatana. Accessed October 2015. The UCR Time Series Classification/Clustering Homepage:. *www.cs.ucr.edu/~eamonn /time_series_data* (Accessed October 2015).

Chan Kin-pong and Fu Ada. 1999. Efficient Time Series Matching by Wavelets. In *ICDE*. 126–133.

Flip Korn, H. V. Jagadish, and Christos Faloutsos. 1997. Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences. In *SIGMOD*. 289–300.

Chung-Sheng Li, Philip S Yu, and Vittorio Castelli. 1998. MALM: A framework for mining sequence database at multiple abstraction levels. In *CIKM*. 267–272.

Xiang Lian, Lei Chen, Jeffrey Xu Yu, Guoren Wang, and Ge Yu. 2007. Similarity match over high speed time-series streams. In *ICDE*. 1086–1095.

Jessica Lin, Michail Vlachos, Eamonn Keogh, and Dimitrios Gunopulos. 2004. Iterative incremental clustering of time series. In *EDBT*. 106–122.

Jason Lines and Anthony Bagnall. 2014. Ensembles of Elastic Distance Measures for Time Series Classification.. In *SDM*. 524–532.

Jason Lines and Anthony Bagnall. 2015. Time series classification with ensembles of elastic distance measures. *Data Mining and Knowledge Discovery* 29, 3 (2015), 565–592.

James MacQueen. 1967. Some methods for classification and analysis of multivariate observations. In *BSMSP*. 281–297.

Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. 2009. The planar k-means problem is NP-hard. In *WALCOM*. 274–285.

Rosario N Mantegna. 1999. Hierarchical structure in financial markets. *The European Physical Journal B-Condensed Matter and Complex Systems* 11, 1 (1999), 193–197.

Warissara Meesrikamolkul, Vit Niennattrakul, and Chotirat Ann Ratanamahatana. 2012. Shape-Based clustering for time series data. In *PAKDD*. 530–541.

Vasileios Megalooikonomou, Qiang Wang, Guo Li, and Christos Faloutsos. 2005. A multiresolution symbolic representation of time series. In *ICDE*. 668–679.

Yoshihiro Mitani and Yoshihiko Hamamoto. 2000. Classifier design based on the use of nearest neighbor samples. In *ICPR*, Vol. 2. 769–772.

Yoshihiro Mitani and Yoshihiko Hamamoto. 2006. A local mean-based nonparametric classifier. *Pattern Recognition Letters* 27, 10 (2006), 1151–1159.

Michael D Morse and Jignesh M Patel. 2007. An efficient and accurate method for evaluating time series similarity. In *SIGMOD*. 569–580.

Abdullah Mueen, Hossein Hamooni, and Trilce Estrada. 2014. Time Series Join on Subsequence Correlation. In *ICDM*. 450–459.

Abdullah Mueen, Eamonn Keogh, and Neal Young. 2011. Logical-shapelets: an expressive primitive for time series classification. In *KDD*. 1154–1162.

Peter Nemenyi. 1963. *Distribution-free Multiple Comparisons*. Ph.D. Dissertation. Princeton University.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *NIPS*. 849–856.

Vit Niennattrakul and Chotirat Ann Ratanamahatana. 2009. Shape averaging under time warping. In *ECTI-CON*. 626–629.

Tim Oates. 1999. Identifying distinctive subsequences in multivariate time series by clustering. In *KDD*. 322–326.

Spiros Papadimitriou, Jimeng Sun, and Philip S Yu. 2006. Local correlation tracking in time series. In *ICDM*. 456–465.

Panagiotis Papapetrou, Vassilis Athitsos, Michalis Potamias, George Kollios, and Dimitrios Gunopulos. 2011. Embedding-based subsequence matching in time-series databases. *TODS* 36, 3 (2011), 17.

John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *SIGMOD*. 1855–1870.

John Paparrizos and Luis Gravano. 2016. k-Shape: Efficient and Accurate Clustering of Time Series. *ACM SIGMOD Record* 45, 1 (2016), 69–76.

Tuomas Pelkonen, Scott Franklin, Justin Teller, Paul Cavallaro, Qi Huang, Justin Meza, and Kaushik Veeraraghavan. 2015. Gorilla: a fast, scalable, in-memory time series database. *PVLDB* 8, 12 (2015), 1816–1827.

François Petitjean, Germain Forestier, Geoffrey I Webb, Ann E Nicholson, Yanping Chen, and Eamonn Keogh. 2014. Dynamic time warping averaging of time series allows faster and more accurate classification. In *ICDM*. 470–479.

François Petitjean, Germain Forestier, Geoffrey I Webb, Ann E Nicholson, Yanping Chen, and Eamonn Keogh. 2015. Faster and more accurate classification of time series by exploiting a novel dynamic time warping averaging algorithm. *Knowledge and Information Systems* (2015), 1–26.

François Petitjean and Pierre Gançarski. 2012. Summarizing a set of time series by averaging: From Steiner sequence to compact multiple alignment. *Theoretical Computer Science* 414, 1 (2012), 76–91.

François Petitjean, Alain Ketterlin, and Pierre Gançarski. 2011. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition* 44, 3 (2011), 678–693.

Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*. 262–270.

Thanawin Rakthanmanon, Eamonn J Keogh, Stefano Lonardi, and Scott Evans. 2011. Time series epenthesis: Clustering time series streams requires ignoring some data. In *ICDM*. 547–556.

William M Rand. 1971. Objective criteria for the evaluation of clustering methods. *J. Amer. Statist. Assoc.* 66, 336 (1971), 846–850.

Chotirat Ann Ratanamahatana and Eamonn Keogh. 2004. Making time-series classification more accurate using learned constraints. In *SDM*. 11–22.

John Rice. 2006. *Mathematical statistics and data analysis*. Cengage Learning.

Alex Rodriguez and Alessandro Laio. 2014. Clustering by fast search and find of density peaks. *Science* 344, 6191 (2014), 1492–1496.

Eduardo J Ruiz, Vagelis Hristidis, Carlos Castillo, Aristides Gionis, and Alejandro Jaimes. 2012. Correlating financial time series with micro-blogging activity. In *WSDM*. 513–522.

Sartaj Sahni and Teofilo Gonzalez. 1976. P-complete approximation problems. *J. ACM* 23, 3 (1976), 555–565.

Naoki Saito. 1994. *Local Feature Extraction and Its Applications Using a Library of Bases*. Ph.D. Dissertation. Yale University.

Hiroaki Sakoe and Seibi Chiba. 1978. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing* 26, 1 (1978), 43–49.

Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. 2005. Braid: Stream mining through group lag correlations. In *SIGMOD*. 599–610.

Stan Salvador and Philip Chan. 2004. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *ICTAI*. 576–584.

José Salvador Sánchez, Filiberto Pla, and Francesc J Ferri. 1997. On the use of neighbourhood-based non-parametric classifiers. *Pattern Recognition Letters* 18, 11 (1997), 1179–1186.

Patrick Schäfer. 2015. Scalable time series classification. *Data Mining and Knowledge Discovery* (2015), 1–26.

Yutao Shou, Nikos Mamoulis, and David Cheung. 2005. Fast and exact warping of time series using adaptive segmental approximations. *Machine Learning* 58, 2-3 (2005), 231–267.

Diego F Silva, Gustavo EAPA Batista, and Eamonn Keogh. 2016. Prefix and Suffix Invariant Dynamic Time Warping. (2016).

Antoniu Stefan, Vassilis Athitsos, and Goutam Das. 2013. The move-split-merge metric for time series. *Knowledge and Data Engineering, IEEE Transactions on* 25, 6 (2013), 1425–1438.

Kuniaki Uehara and Mitsuomi Shimada. 2002. Extraction of primitive motion and discovery of association rules from human motion data. In *Progress in Discovery Science*. 338–348.

Michail Vlachos, Marios Hadjieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. 2006. Indexing multidimensional time-series. *The VLDB Journal* 15, 1 (2006), 1–20.

Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In *ICDE*. 673–684.

Hao Wang, Yong-fu Cai, Yin Yang, Shiming Zhang, and Nikos Mamoulis. 2014. Durable queries over historical time series. *Knowledge and Data Engineering, IEEE Transactions on* 26, 3 (2014), 595–607.

Lusheng Wang and Tao Jiang. 1994. On the complexity of multiple sequence alignment. *Journal of Computational Biology* 1, 4 (1994), 337–348.

Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn Keogh. 2013. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery* 26, 2 (2013), 275–309.

Xiaozhe Wang, Kate Smith, and Rob Hyndman. 2006. Characteristic-based clustering for time series data. *Data Mining and Knowledge Discovery* 13, 3 (2006), 335–364.

T Warren Liao. 2005. Clustering of time series data - a survey. *Pattern Recognition* 38, 11 (2005), 1857–1874.

Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin* (1945), 80–83.

D Randall Wilson and Tony R Martinez. 1997. Instance pruning techniques. In *ICML*, Vol. 97. 403–411.

D Randall Wilson and Tony R Martinez. 2000. Reduction techniques for instance-based learning algorithms. *Machine learning* 38, 3 (2000), 257–286.

Di Wu, Yiping Ke, Jeffrey Xu Yu, S Yu Philip, and Lei Chen. 2010. Detecting leaders from correlated time series. In *DASFAA*. 352–367.

Yimin Xiong and Dit-Yan Yeung. 2002. Mixtures of ARMA models for model-based time series clustering. In *ICDM*. 717–720.

Jaewon Yang and Jure Leskovec. 2011. Patterns of temporal variation in online media. In *WSDM*. 177–186.

Lexiang Ye and Eamonn Keogh. 2009. Time series shapelets: A new primitive for data mining. In *KDD*. 947–956.

Jamaluddin Zakaria, Abdullah Mueen, and Eamonn Keogh. 2012. Clustering time series using unsupervised-shapelets. In *ICDM*. 785–794.

Yunyue Zhu and Dennis Shasha. 2002. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*. 358–369.