Data Min Knowl Disc
DOI 10.1007/s10618-015-0411-4

Accelerating the discovery of unsupervised-shapelets

Jesin Zakaria · Abdullah Mueen · Eamonn Keogh · Neal Young

Received: 11 September 2013 / Accepted: 26 February 2015

Abstract Over the past decade, time series clustering has become an increasingly important research topic in data mining community. Most existing methods for time series clustering rely on distances calculated from the *entire* raw data using the Euclidean distance or Dynamic Time Warping distance as the distance measure. However, the presence of significant noise, dropouts, or extraneous data can greatly limit the accuracy of clustering in this domain. Moreover, for most real world problems, we cannot expect objects from the same class to be equal in length. As a consequence, most work on time series clustering only considers the clustering of individual time series "behaviors," e.g., individual heart beats or individual gait cycles, and contrives the time series in some way to make them all equal in length. However, automatically formatting the data in such a way is often a harder problem than the clustering itself. In this work, we show that by using only some local patterns and deliberately ignoring the rest of the data, we can mitigate the above problems and cluster time series of different lengths, e.g., cluster one heartbeat with multiple heartbeats. To achieve this, we exploit and extend a recently introduced concept in time series data mining called *shapelets*. Unlike existing work, our work demonstrates the unintuitive fact that shapelets can be

Responsible editor: Ian Davidson

J. Zakaria (✉) · A. Mueen · E. Keogh · N. Young
Department of Computer Science & Engineering, University of California, Riverside, Riverside, CA 92521, USA
e-mail: jzaka001@ucr.edu

A. Mueen
e-mail: mueen@cs.ucr.edu

E. Keogh
e-mail: eamonn@cs.ucr.edu

N. Young
e-mail: neal@cs.ucr.edu

Published online: 07 May 2015

learned from *unlabeled* time series. We show, with extensive empirical evaluation in diverse domains, that our method is more accurate than existing methods. Moreover, in addition to accurate clustering results, we show that our work also has the potential to give insight into the domains to which it is applied. While a brute-force algorithm to discover shapelets in an unsupervised way could be untenably slow, we introduce two novel optimization procedures to significantly speed up the unsupervised-shapelet discovery process and allow it to be cast as an anytime algorithm.

Keywords Time series · Clustering · Unsupervised-shapelet

1 Introduction

Time series occur in many real world domains and thus analysis of time series has become an important topic within many fields of research, including aerospace, finance, business, meteorology, medical science, motion capture, animal behavior, etc. (Hirano and Tsumoto 2006; Hu et al. 2011; Ruiz et al. 2012; Zakaria et al. 2012a, b; Zhang and Sawchuk 2012). However, most research on time series analysis is limited by the need for costly labeled data. This has led to an increase of interest in clustering time series data, which, by definition, does not require access to labeled data (Garilov et al. 2000; Hirano and Tsumoto 2006; Jiang et al. 2004; Zhang et al. 2005).

A decade old empirical comparison by Keogh and Kasetty (2002) reveals the somewhat surprising fact that the simple Euclidean distance metric is highly competitive with other, more sophisticated, distance measures, and more recent work confirms this (Ding et al. 2008). However, the time series must be of equal length for the Euclidean distance to be defined. Dynamic time warping (DTW) can both solve this problem and handle the difficulty of clustering time series containing out-of-phase similarities as shown in Keogh et al. (2003), Ratanamahatna and Keogh (2004), and Ding et al. (2008). In this work, however, we argue that the apparent utility of Euclidean distance or DTW for clustering may come from an over dependence on the UCR time series archive (Keogh et al. 2011) for testing clustering algorithms (Keogh et al. 2003; Möerchen 2003; Zhang et al. 2005). The problem is that the data in this archive has already been hand-edited to have equal length and (approximate) alignment. However, the task of contriving the data in this format is often harder than the task of giving labels to the data, i.e., the clustering itself.

As a concrete example, consider the famous Gun-Point dataset, which has been used in hundreds of studies for both clustering and classification, in every case reporting near perfect accuracy (Ding et al. 2008). This dataset was contrived to have perfect alignment/length by audible cues that both signaled the actor and started/stopped the video recording. Figure 1 shows two examples of data from the archive (*just* the parts highlighted in red/bold); however, by examining the original archive, we are able to show the data with the 3 s proceeding/trailing data used in the UCR archive.

Our central argument is that this less "clean" example is a much more realistic format in which the data is likely to appear in any real world problem worthy of our attention. Note, however, that if we attempt to cluster the data with the Euclidean distance (e.g. the dendrogram in Fig. 1), we get a very poor result. In Fig. 2, we show

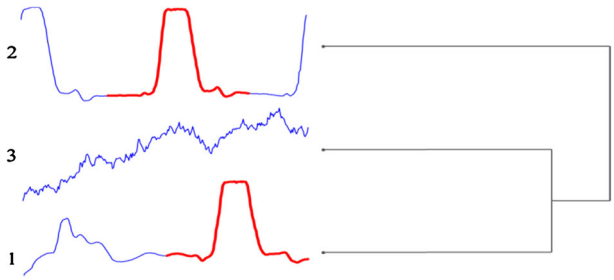

Fig. 1 A Euclidean distance clustering of two exemplars from the "raw" Gun-Point dataset, together with a random walk sequence. The hundreds of papers that have used the Gun-Point dataset have only considered the human edited version, corresponding to *just* the *red/bold* data (Color figure online)

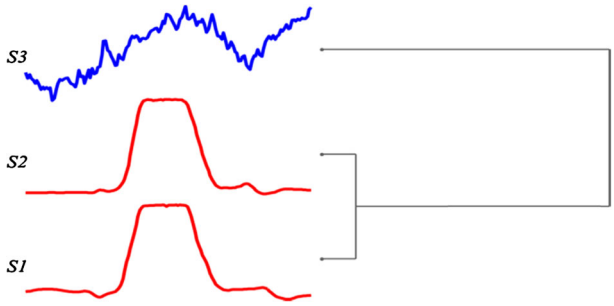

Fig. 2 Clustering Gun-Point after ignoring some data (Color figure online)

the result of clustering these time series if we are allowed to ignore some of the data. Here, we use the section from random walk sequence that has minimum distance to the Gun-Point gestures.

As we can see from Fig. 2, the situation has improved drastically. For the moment, we gloss over the question of how we knew which sections to ignore. However, as we shall see, we can do this without any human intervention, and introducing this ability is the core contribution of this work.

We believe that this observation, that for most datasets we must ignore some (potentially the *vast majority of*) data, is a critical insight in allowing time series clustering in realistic problem settings.

We are aware of the "chicken-and-egg" nature of our claim. We must ignore some data to allow a good clustering, and a good clustering is the obvious way to reveal the best data to keep (and ignore). However, we will show that time series shapelets (Ye and Keogh 2009) can be adapted to resolve this paradox.

The idea of time series shapelets was introduced by Ye and Keogh (2009) as a primitive for time series data mining. Time series shapelets are small, local patterns in a time series that are highly predictive of a class. Since then, numerous researchers have shown the utility of shapelets for classifying time series data (Hartmann et al. 2010; Lin et al. 2012; Mueen et al. 2011; Xing et al. 2011; Zakaria et al. 2012a).

In this work, we show that shapelets can also be highly competitive in clustering time series data. Since we do not know the labels of the time series in the dataset,

this begs the question, "How can we discover shapelets from a dataset without having any knowledge of the class labels?" We propose a new form of shapelet that we call unsupervised-shapelet (or *u-shapelet*) and demonstrate its utility for clustering time series data. While the brute-force algorithm for discovering u-shapelets is untenably slow for most problems, we introduce two novel admissible optimizations to speed up the u-shapelet discovery process. Moreover, we further demonstrate that our optimizations enhance the *diminishing returns* property of the search, thus allowing us to cast the u-shapelet discovery process as an *anytime algorithm* (Zilberstein 1996).

The rest of the paper is organized as follows: In Sect. 2 we define the necessary notation; in Sect. 3, we discuss previous work on clustering time series; Sect. 4 explains our method for obtaining shapelets in the absence of class labels; Sect. 5 presents our acceleration methods; Sect. 6 shows empirical evidence of our algorithm's utility on many datasets from diverse domains; lastly, Sect. 7 concludes and gives direction of future research.

2 Definitions and notations

We present the definitions of key terms that we use in this work. For our problem, each object in the dataset is a time series, which may be of different lengths.

Definition 1 (*Time Series*), a time series $T = t_1, t_2, \ldots, t_n$ is an ordered set of real values. The total number of real values is equal to the length of the time series. A dataset $D = \{T1, T2, \ldots TN\}$ is a collection of N such time series.

In this work, we demonstrate that if we can find small subsequences of few time series that best represent their clusters (e.g. red/bold subsequences of Fig. 1), then those subsequences may give us a better clustering result than using the entire time series.

Definition 2 *(Subsequence)*, a subsequence $S_{i,l}$, where $1 \leq l \leq n$ and $1 \leq i \leq l$, is a set of l continuous real values from a time series, T, that starts at position i.

For a time series of length n, there can be, in total, $\frac{\mathrm{n(n+1)}}{2}$ subsequences of all possible lengths. If there are N time series in the dataset with length n, then there will be, in total, $\mathrm{N} \times \frac{\mathrm{n(n+1)}}{2}$ subsequences.

Previous research efforts on shapelet discovery (Mueen et al. 2011; Ye and Keogh 2009) had to consider *all* the $\mathrm{N} \times \frac{\mathrm{n(n+1)}}{2}$ subsequences to find optimal shapelets for building the classifier. However, as we shall see, we need to explore only a small subset of these subsequences to find the optimal u-shapelets. We will formalize the definition of u-shapelets shortly.

We can compute the distance between two time series of equal length by simply calculating the Euclidean distance between them. To make our distance measure invariant to scale and offset, we need to *z-normalize* the time series before computing their Euclidean distance. As demonstrated in Keogh and Kasetty (2002) and Hu et al. (2013), even tiny differences in scale and offset rapidly swamp any similarity in shape.

In order to compare or rank candidate u-shapelets, we need to consider the utility (i.e. discriminating power) of subsequences of different lengths, as in most cases

we expect the distance between shorter subsequences will be less than the distance between longer subsequences. Thus, we normalize the distance by the length of the subsequence. We call this the *length-normalized Euclidean distance*. The length-normalized Euclidean distance between two time series TX and TY can be computed using $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(TX_i - TY_i)^2}$, which takes time linear to the length of time series.

We can reduce the amortized time complexity of this calculation from linear to constant by caching the results of some calculations. The five results to cache are $\sum_{i=1}^{n} TX_i$, $\sum_{i=1}^{n} TY_i$, $\sum_{i=1}^{n} TX_i^2$, $\sum_{i=1}^{n} TY_i^2$, and $\sum_{i=1}^{n}(TX_i \times TY_i)$. This method has been adopted and described in Mueen et al. (2011). For more description, we refer the interested reader to that work (Mueen et al. 2011). Using these numbers, we can compute the mean μ (1) and standard deviation (std) σ (2).

$$\mu_{TX} = \frac{1}{n}\sum_{i=1}^{n} TX_i \tag{1}$$

$$\sigma_{TX} = \frac{1}{n}\sum_{i=1}^{n} TX_i^2 - \mu_{TX^2} \tag{2}$$

The positive correlation (3) and the length-normalized Euclidean distance (4) can be computed as follows (Mueen et al. 2011),

$$C(TX, TY) = \frac{\sum_{i=1}^{n}(TX_i \times TY_i) - n\mu_{TX}\mu_{TY}}{n\sigma_{TX}\sigma_{TY}} \tag{3}$$

$$dist(TX, TY) = \sqrt{2\,(1 - C\,(TX, TY))} \tag{4}$$

We are now in a position to explain how we calculate the distance between a (typically short) subsequence S and a (typically much longer) time series T. We need to "slide" S against T to find the alignment that minimizes the distance.

Definition 3 The *subsequence distance* between a subsequence S of length m and a time series T of length n is the distance between S and the subsequence of T that has minimum distance. We denote it as $sdist(S, T)$.

$$sdist(S, T) = \min_{1 \le i \le n-m} dist(S, T_{i,m}); \quad [1 \le m \le n] \tag{5}$$

Note that we generally expect $m \ll n$, and that *both* $sdist(S, T)$ and $sdist(T, S)$ are only defined if $n = m$. This subsequence distance is essentially the nearest neighbor distance of S to a time series T.

In previous works, a definition similar to *sdist* was used to support a definition of *shapelets;* small subsequences that separate time series into two or more distinct groups by their "nearness" or "distance" to that subsequence (Ye and Keogh 2009), Mueen et al. (2011). Since we cannot use the class labels of the time series to discover the shapelets, we call our definition of informative subsequences *unsupervised-shapelets* or *u-shapelets* to differentiate them from the classic shapelets which assume access to class labels (Mueen et al. 2011; Ye and Keogh 2009).

Definition 4 An*unsupervised-shapelet* $\acute{S}$ is a subsequence of a time series T for which the *sdists* between $\acute{S}$ and the time series from a group D_A are much smaller than the *sdists* between $\acute{S}$ and rest of the time series D_B in the dataset D (6).

$$sdist(\acute{S}, D_A) \ll sdist(\acute{S}, D_B) \tag{6}$$

We will expound on how we formalize "*much smaller*" below. The reader may wonder how we identify D_A and D_B without class labels. In Sect. 4.2, we have formalized an algorithm to discover u-shapelets from a dataset. The algorithm will rigorously clarify the concept of D_A and D_B, so we defer a detailed discussion about it until then.

Note that, unlike the trivial example shown in Fig. 2, many problems may require more than one u-shapelet. Thus, we need to generalize our representation to allow for multiple u-shapelets. We call the matrix that contains the *sdists* vectors between u-shapelets and each time series in the dataset a *Distance map*.

Definition 5 A *Distance map* contains the *sdists* between each of the u-shapelets and all the time series in the dataset. If we have m u-shapelets for a dataset of N time series, the size of the distance map is $[N \times m]$ where each column is a distance vector of a u-shapelet.

For any dataset, once we have this *distance map* of the u-shapelets' distances, we can simply pass it into an off-the-shelf clustering algorithm such as k-means. Thus, the focus of our work is in algorithms for obtaining high quality distance maps. Note that the *distance map* representation is somewhat similar to the *vector space model*, which is a cornerstone of most text mining algorithms (Salton et al. 1975).

2.1 A discrete analogue of U-shapelet

In order to enhance the reader's understanding of u-shapelets and to lay the groundwork for the intuition about our proposed *distance map* building method, we present a very simple example from a *discrete* (rather than *real-valued*) sequence domain. This example is a close analogue of the task at hand. The use of the discrete data simply allows us to explain our ideas in a domain where the reader's intuitions are highly developed and without the need of resorting to the indirection of figures.

Consider the following collection of phrases, which is an analogue of a dataset of time series:

San *Jose*; *Earth* ***Day***; ***San*** *Francisco*; *Memorial* ***Day***; *Fink Nottle*; *Labor* ***Day***; *Bingo Little*

If we are asked to cluster this set of phrases into three groups, then, as intuitive humans, we would almost certainly split the data based on the presence/absence of the two substrings "***San***" and "***Day***."

However, doing this algorithmically is a little more challenging. Since the phrases differ in length, we cannot apply *Hamming distance*, which is analogous to *Euclidean distance*, to compute the distance vector of the full phrases. The problem of length

Table 1 Distance map of "Day" and "San"

	San Jose	Earth Day	San Fran..	Mem.. Day	Fink Nottle	Labor Day	Bingo Little
San	0	2	0	2	2	2	2
Day	2	0	2	0	3	0	3

difference is solved by using *Edit distance*, which is the discrete version of *DTW*, but the Edit distance produces a clustering that looks essentially random, as it needs to "explain" the irreverent sections e.g., *Earth, Francisco, Memorial,* etc.

Instead, we can try to find *representative substrings* of the above phrases. By visual inspection, the obvious candidates are "***San***" and "***Day***." We compute the *Hamming* distances of these words to their *nearest substrings* in all phrases and produce the distance map shown in Table 1.

Using this distance map, we can cluster the phrases perfectly by using k-means on the columns. Moreover, examining the key-phrases (u-shapelet analogues) chosen for clustering gives us some intuition about the domain. For example, we may notice that the second, fourth, and sixth phrases are some kind of "***day***."

The above example is rather simple and the reader may wonder if this method works only if some u-shapelets have zero distance to the nearest neighbors of their own group. The answer is *no* this method can handle more complex datasets. Consider the following set of sentences,

Abraham ***Lincoln*** *lived here for many years*	(English)
She is looking for Ibrahim	(Arabic)
You can find Abrahan in that house	(Portuguese)
Michael is singing a song for her	(English)
She bought a gift for Michaël	(Dutch)
She can teach Michales chess	(Hebrew)

The Hamming distances from the key-phrase *Abraham* to its nearest neighbors is [0, 2, 1, 7, 7, 7] respectively, and this distance vector is *only* entry necessary in a distance map that can cluster the sentences correctly into two groups. Note, however, that the choice of *Abraham* is somewhat arbitrary, as the data can also be correctly clustered by using either the similar key-phases *Ibrahim*/*Abrahan* or by any of the three variants of *Michael.*

We conclude this section by summarizing what we have learned from this analogy. First of all, to cluster data, we are generally-somewhat paradoxically-better off *ignoring* large sections of the data. Secondly, while sometimes a single key element (e.g. *Abraham*) may be enough to separate the data into meaningful clusters, in some cases we may need two or more (e.g. ***Day*** and ***San***) elements to separate the data.

3 Related work

The literature on clustering time series is vast; we refer the reader to Liao (2005) which offers a useful survey. Much of the work can be broadly classified into two categories:

- *Shape*-based clustering, which attempts to cluster the data based on the shape of the *entire* time series (Garilov et al. 2000), using Euclidean distance, DTW, or one of a host of other distance measures (Ding et al. 2008). Note that our method contrasts with these ideas in that we explicitly allow our representation to *ignore* much of the data.
- *Statistical*-based clustering, which attempts to cluster the data based on statistical features extracted from the time series. These features include common measures such as mean, standard deviation, skewness, etc., in addition to more exotic features such as coefficients of ARIMA models (Kalpakis et al. 2001), fractal measures (Wang et al. 2006), etc.

One problem with the latter approaches is that they typically require a great many parameters and (at least *apparently*) ad-hoc choices. The problem with the former approaches is that the distance measures consider all data points, even though (as hinted in Figs. 1, 2) we may be better off ignoring much of the data. Note that there are a handful of methods that combine both ideas by using statistics/features which are *derived* from time series shapes (Ding et al. 2008; Hirano and Tsumoto 2006; Zhang and Sawchuk 2012).

The work closest to ours in spirit is that of Rakthanmanon et al., which shows the utility of ignoring some data for clustering within a single time series stream (Rakthanmanon et al. 2011).

4 Our algorithm

We are finally in a position to present a more direct insight and formal description of our algorithm.

4.1 An intuition of our clustering algorithm

Consider the *Trace* dataset from the UCR archive (Keogh et al. 2011). It contains 50 instances from each of four classes, all with a length of 275. In Fig. 3, we present ten random instances from each class. Note that while the global patterns within each class are the same, they are not aligned perfectly. As a result, if we use entire time series for k-means clustering with the Euclidean distance as distance measure, we will get poor results (we formally measure the quality of the results using the *Rand index* in the experimental section below, for the moment it is suffice to note the results are *poor*).

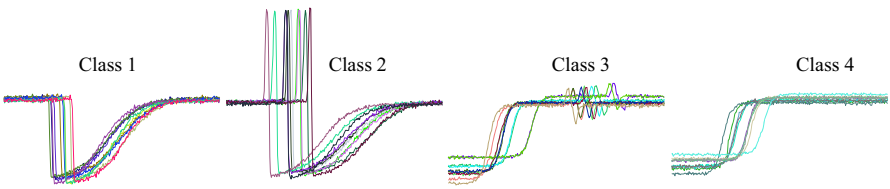

Fig. 3 Sample time series from four classes of Trace dataset (Color figure online)

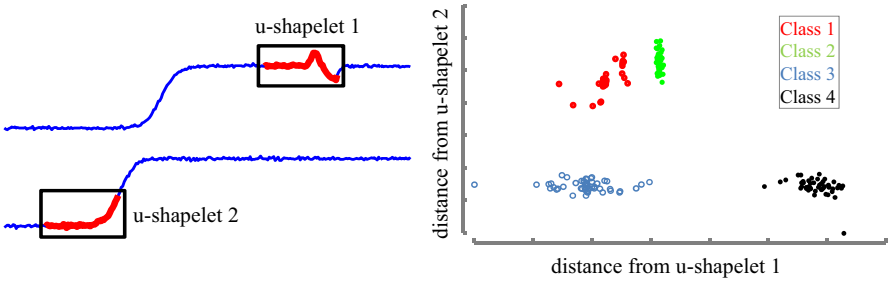

Fig. 4 (*left*) Two *u-shapelets* (marked with *red*) used for clustering *Trace* dataset. (*right*) A plot of *distance map* of the *u-shapelets*. An animated video based on this figure is at Jesin's Webpage (2013) (Color figure online)

However, suppose that we use subsequences that are prominent in one class but not in other classes as u-shapelets (e.g. the red subsequences in Fig. 4 (left)). We can then use their *distance map* to separate all the time series. In Fig. 4 (right), we plot the *distance map* of the two u-shapelets in two-dimensional space. In this plot, the X-axis represents the *sdists* of the first u-shapelet while the Y-axis encodes the *sdists* of the second u-shapelet. From this plot, it is clear that by using the *distance map*, we could get a perfect clustering.

The choice of u-shapelets here is not arbitrary. For example, if we slide u-shapelet 2 to the right, the cloud of blue points (class 3) will rise up and intermingle with the clouds of red and green points (classes 1 and 2). In a sense, this would be like trying to use a stop-word to cluster text documents (Kosala and Blockeel 2000). Stop-words simply do not have any such discriminating power. At Jesin's Webpage (2013) there is an animation that shows how the clouds of points defuse and intermingle as we change the location of the candidate u-shapelets.

Note that in this example we are "cheating" in so much as that in our (visual) evaluation we know the class labels. However, in the next section, we will introduce an algorithm that we can use to discover the u-shapelets *without* using class labels.

4.2 A formal description of our algorithm

The high-level idea of our algorithm is that it searches for a u-shapelet which can separate and remove a subset of time series from the rest of the dataset, then iteratively repeats this search among the remaining data until no data remains to be separated.

As hinted at before, an ideal u-shapelet $\acute{S}$ has the ability to divide a dataset $\boldsymbol{D}$ into two groups of time series, D_A and D_B. D_A consists of the time series that have subsequences similar to $\acute{S}$, while D_B contains the rest of the time series in $\boldsymbol{D}$. Simply stated, we expect the mean value of $sdist(\acute{S}, D_A)$ to be much smaller than the mean value of $sdist(\acute{S}, D_B)$. Since we ultimately use a distance map that contains distance vectors to cluster the dataset, the larger the *gap* between these two means of these distances vectors, the better.

We use the algorithm in Table 2 to extract u-shapelets. In essence, this algorithm can be seen as a greedy search algorithm which attempts to maximize the separation *gap* between two subsets of $\boldsymbol{D}$. This separation measure is formally encoded in the

Table 2 An algorithm to extract U-shapelets

```
Algorithm: Extract U-Shapelets(D, sLen)
Require: D: dataset; sLen: u-shapelet length
Ensure: Ś: set of u-shapelets
1:  Ś ← []                          // set of u-shapelets, initially empty
2:  ts ← D(1,:)                     // a time series of the dataset
3:  while true
4:    cnt ← 0                       // count of candidate u-shapelets from ts
5:    ŝ ← []                        // set of subsequences, initially empty
6:    for sl ← sLen(1) to sLen(end) do   // each u-shapelet length
7:      for i ← 1 to |ts|-sl+1 do      // each subsequence from ts
8:        ŝ(cnt++) ← ts(i:i+sl-1)      // a subsequence of length sl
9:        [gap(cnt++), dt(cnt++)] ← computeGap(ŝ(cnt++), D)
10:   index1 ← max(gap)           // find maximum gap score
11:   Ś ← ŝ(index1)               // add the u-shapelet with max gap score
12:   dis ← computeDistance(ŝ(index1), D)
13:   D_A ← find(dis < dt)        // points to the left of dt
14:   if length(D_A) ==1,
15:      break;
16:   else
17:     index2 ← max(dis), ts ← D(index2, :) // ts to extract..
18:     θ ← mean(disD_A)+std(disD_A)          // ..next u-shapelet
19:     Ď ← find(dis < θ) , D ← D-Ď          // remove ts with
20:   end                                    // distance less than θ
21:   return Ś                      // set of u-shapelets
```

following equation:

$$gap = \mu_B - \sigma_B - (\mu_A + \sigma_A) \tag{7}$$

Here, μ_A and μ_B represent $mean(sdist(\acute{S}, D_A))$ and $mean(sdist(\acute{S}, D_B))$, respectively, while σ_A and σ_B represent $std(\text{sdist}(\acute{S}, D_A))$ and $std(sdist(\acute{S}, D_B))$, respectively.

In the nested **for** loop of lines 6–9, we consider all subsequences of the time series as candidate u-shapelets and compute their distance vectors. We can represent a distance vector as a schematic line, which we call an *orderline*. In line 10, we search these *orderlines* for the location that maximizes the *gap* function introduced in (7). We refer to this point as *dt*. Points to the left of *dt* represent $sdist(\acute{S}, D_A)$, while points to the right correspond to $sdist(\acute{S}, D_B)$.

Figure 5 presents the *orderline* of the words "*Abraham*" and "*Lincoln*" from the discrete toy example of Sect. 2.1. Several tentative locations of *dt* are shown with gray/light lines and are annotated by their gap scores. However, the locations of the actual *dt* are shown with blue/bold lines.

Note that the maximum gap score for "*Lincoln*" is larger than the maximum gap score for "*Abraham.*" Nevertheless, we would rather use "*Abraham*" as the key phrase. This is because we want a u-shapelet to have discriminative power. If all but one time

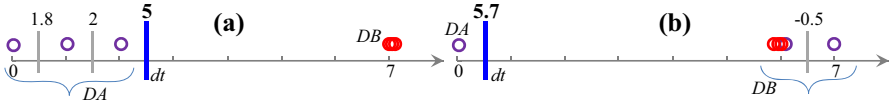

Fig. 5 Orderline for (*left*) "*Abraham*," (*right*) "*Lincoln*" (Color figure online)

Table 3 An algorithm to compute gap

```
Algorithm: computeGap(ŝ, D)
Require: ŝ: candidate u-shapelet; D: dataset
Ensure: maxGap: maximum gap score; dt
1:  dis ← computeDistance(ŝ, D)   //distance vector of ŝ
2:  dis ← sort(dis)        //sort distance vector in ascending order
3:  maxGap ← 0, dt ← 0
4:  for l ← 1 to |dis|-1 do,         // all possible location of dt
5:    d ← (dis(l) + dis(l+1))/2
6:    DA ← find(dis<d)                  // points to the left of dt
7:    DB ← find(dis>d)                  // points to the right of dt
8:    r ← |DA|/|DB|                     // ratio of |DA| and |DB|
9:    if  1/k<r <(1-1/k),               // k: number of clusters
10:     mA ← mean(dis(DA)), mB ← mean(dis(DB))
11:     sA ← std(dis(DA)), sB ← std(dis(DB))
12:     gap ← mB-sB-(mA+.sA)
13:     if gap > maxGap, { maxGap ← gap; dt ← d }
14: return maxGap, dt                  //max gap score and dt for ŝ
```

series belong to either D_B or D_A, then we do not have discriminating ability, but rather have a single outlier or universal pattern, both of which are undesirable. In order to exclude such pathological candidate u-shapelets, we check if the ratio of D_A and D_B is within a certain range (8).

$$\left(\frac{1}{k}\right) < |D_A|/|D_B| < \left(1 - \frac{1}{k}\right), k : \textit{total number of clusters} \tag{8}$$

The algorithm shown in Table 3 is called as a subroutine in line 9 of Table 2 to compute the maximum gap score (7) and *dt* of a candidate u-shapelet. This subroutine takes a candidate u-shapelet and the dataset as input. In line 1, the algorithm computes the distance vector of the candidate u-shapelet. The **for** loop is then used to compute the *gap* for every possible location of *dt*. Note that for a distance vector with N values, there are just N − 1 possible locations to check. The first **if** block is used to check the condition of (8), while the second **if** block is used to find the maximum gap.

The algorithm in Table 4 is used to compute the distance vectors. It takes a subsequence and the dataset as input and computes the *sdists* between the subsequence

Table 4 An algorithm to compute distance vector

```
Algorithm: computeDistance(ŝ, D)
Require: ŝ: subsequence, D: dataset
Ensure: dis: distances from all the time series in the dataset
1:  dis ← []                          // set of distances, initially empty
2:  ŝ ← zNorm(ŝ)                      // z-normalize ŝ
3:  for i ← 1 to |D|, do
4:    ts ← D(i); dis(i) ← INF //D(i) is the i^th time series in D
5:    for j ← 1 to |ts|-|ŝ|+1 do,    // every start position of ts
6:      z ← zNorm(ts_{j,|ŝ|})
7:      d ← euclideanDistance(z, ŝ)
8:      dis(i) ← min(d, dis(i))
9:  return dis/sqrt(|ŝ|)              // set of distances
```

and each time series in the dataset. In order to return the length normalized Euclidean distance we divide the *sdists* by the square root of the subsequence length (cf. line 9).

Once we know the gap scores for all the subsequences of a time series, we add the subsequence with maximum gap score in the set of u-shapelets (cf. lines 10, 11 of Table 2).

Given that we have selected a u-shapelet, we do not want subsequences similar to it to be selected as u-shapelets in subsequent iterations. Thus, we remove the time series (cf. line 19 of Table 2) that have subsequences similar to the u-shapelet from the dataset and use only the remaining dataset to search for the next u-shapelet.

To illustrate with our toy example phrases of Sect. 2.1, once we identify the key-phrase "***San***" as a high quality key-phrase (i.e. "u-shapelet"), we remove "***San*** *Jose*" and "***San*** *Francisco*" from the set of phrases and search for the second key-phrase within just:

Earth ***Day***, *Memorial* ***Day***, *Fink Nottle, Labor* ***Day***, *Bingo Little*

When we identify the second key-word "***Day***," we remove the relevant phrases and search only within:

Fink Nottle, *Bingo Little*

…and so on, until the terminating condition in line 14 is true.

For our real-valued time series, we use the threshold θ to exclude the time series that have *sdist*s less than θ from the dataset (cf. line 17–18).

The reader may wonder why we use θ instead of just directly removing D_A from D. In simple terms, the use of θ is more *selective* than use of D_A.

Imagine that we add the phrases "*Hoover Dam*" and "*Alamo Dam*" to the above set of phrases. If we observe the orderline of "***Day***" in Fig. 6, we find "*Hoover Dam*" and "*Alamo Dam*" are included in D_A. Using θ, we are more selective and only remove the phrases that contain the word "*Day*," a tighter cluster.

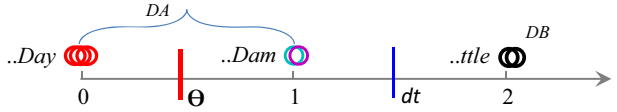

Fig. 6 Orderline for "Day". Θ is shown with *red/thick line* and *dt* is shown with *blue/thin line* (Color figure online)

Table 5 An algorithm to cluster time series

```
Algorithm: clusterData(D, Ś, k)
Require: D: dataset; Ś: set of u-shapelets; k: number of clusters
Ensure: cls: cluster label for each time series in the dataset
1:  DIS ← []      //distance map that contains all the distance vectors
2:  cls(0) ← c    // assign same cluster label to all time series
3:  for cnt ← 1 to length(Ś)
4:    ŝ ← Ś             // a u-shapelet
5:    dis ← computeDistance(ŝ, D)     //distance vector of a u-shapelet
6:    DIS ← [DIS dis], sumDis ← INF
7:    for i ← 1 to n
8:      [IDX, SUMD] ← k-means(DIS, k)
9:      if sum(SUMD) <sumDis // SUMD: distances to the cluster
10:       sumDis ← sum(SUMD) // centers, i.e. the objective function
11:       cls(cnt) ← IDX          // IDX:cluster labels by k-means
12:   CRI(cnt) ← 1-RI(cls(cnt-1), cls(cnt))
                                  //RI:Rand index, CRI: change in RI
13: a ← min(CRI)                  // a:index when CRI is minimum
14: return cls(a)                 //class labels when CRI is minimum
```

The extraction algorithm terminates when the search reveals that for the best u-shapelet choice, the size of D_A is just one. In practice, this means the algorithm tends to run beyond the point where the u-shapelets are useful, and we must do some post-pruning, an idea we discuss in detail in Sect. 4.3.

Note that our algorithm only requires the users to provide a single parameter: the desired length, or range of lengths, for the u-shapelets. Moreover, they can even abrogate this responsibility-at the expense of speed-by searching in the full range of [2:*inf*].

While the distance map created by our algorithm can be used by essentially any clustering algorithm, for concreteness in Table 5, we show how we use the ubiquitous *k*-means.

Because the result of *k*-means clustering may differ from run to run, we call *k*-means multiple times (cf., **for** loop of lines 7–11). In the **if** block of lines 9–11, we compute the objective function and keep the clustering that minimizes it.

4.3 Stopping criteria

For any non-trivial time series dataset, the algorithm in Table 2 will almost certainly extract too many u-shapelets. Thus, the last step of our approach involves pruning away

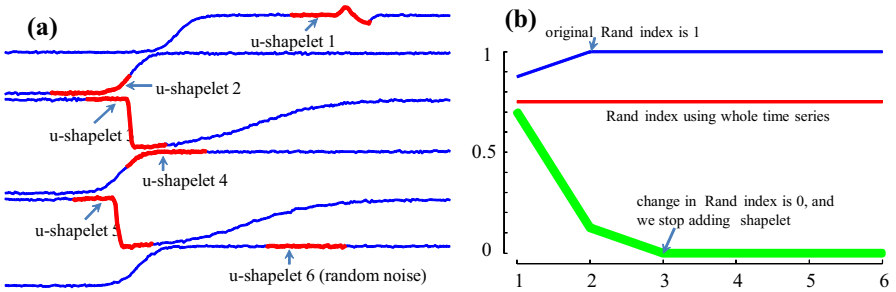

Fig. 7 (*left*) The six u-shapelets returned by our algorithm on the Trace dataset. (*right*) The *CRI* (*green/bold*) predicts that the best number of u-shapelets to use is two. By peeking at the ground truth labels (*blue/light*), we can see that the choice does produce a *perfect* clustering (Color figure online)

some u-shapelets (much like decision-tree post-pruning algorithms, Muata 2007). Our task is made easier by the fact that the order in which our algorithm selects the u-shapelets is "best first," thus our task reduces to finding a stopping criterion.

Many iterative improvement algorithms (e.g. k-means, EM) have stopping criteria based on the observation that the model does not change much after a certain point. We have adopted a similar idea for our algorithm. In essence, our idea is that, **for** i equals 1 to the number of u-shapelets extracted, we treat the ith clustering as though it were correct and measure the Change in Rand index (*CRI*) that the $i^{\text{th}} + 1$ clustering produces. The value of i which minimizes this *CRI* (i.e., the clustering is most "stable") is the final clustering returned. Ties are broken by choosing the smaller set.

The clustering algorithm in Table 5 takes the dataset, the set of u-shapelets returned by algorithm 1, and the user-specified number of clusters as input. The *distance map* is initially empty. Inside **for** loop of lines 3–11, the algorithm computes the distance vector of each u-shapelet and adds the distance vector to the distance map. For each addition of a distance vector to the distance map, we pass the new distance map into k-means. The k-means algorithm returns a cluster label for all the time series in the dataset. In line 12, we use the labels of the current step and the labels of the previous step to compute the change in Rand index. Finally, in line 14, we return the cluster labels for which the *CRI* is minimum.

In Fig. 7 (left), we present all six u-shapelets returned by the Trace dataset experiment shown in Fig. 3. The first two u-shapelets are sufficient to cluster the dataset (cf. Fig. 4 (right)) and the remaining four u-shapelets are spurious. The red curve in Fig. 7 (right) shows the value of *CRI* as u-shapelets are added. This curve predicts that we should return *two* u-shapelets, but how good is this prediction? In this case we happen to have ground truth class labels for this dataset, we can do a post-hoc analysis and add a measurement of the *true* clustering quality to Fig. 7 (right) by showing the *Rand index* with respect to the ground truth. As we can see, at least in this case, our method *does* return the minimal set that can produce a perfect clustering.

5 Speeding up the discovery of unsupervised shapelets

The reader will have now gained an appreciation of the utility of u-shapelets for clustering time series (Zakaria et al. 2012a). However, the reader may also notice that

the algorithm in the previous section searches for a u-shapelet only in a subset of the dataset. As searching for an unsupervised shapelet in the entire dataset is a computationally expensive process, thus far we contented ourselves with the u-shapelets we discover from searching a small subset of data. In particular, we searched *just* the first object in the dataset. Thus, the algorithm in the previous section (cf. Table 2) depends on the ordering of the time series within the dataset, and its performance can be degraded if the first time series happens to be noisy or simply atypical.

In this section we introduce two novel techniques that result in a faster search, allowing an *exhaustive* algorithm for discovering u-shapelets. By "exhaustive" we simply mean that the entire dataset is searched.

Our experimental results show that for most datasets our new u-shapelet discovery algorithm terminates thirty times faster than the brute-force algorithm. It is this speed that allows us to expand the search space to an *exhaustive* search of the dataset in most cases, making our algorithm order independent and more robust to the occasional noisy/atypical object.

In addition to the above, we further show that we can cast the u-shapelet discovery algorithm as an *anytime algorithm* (Zilberstein 1996). Obviously, like any iterative improvement search algorithm, it is trivial to consider a u-shapelet search as an anytime algorithm simply by adding the ability to "break out" of the loops at the user's direction. However, as we shall show, we can intelligently change the search order from the normal left-to-right, top-to-bottom (cf. Table 2) to an ordering that maximizes the *diminishing returns* property of anytime algorithms. As we shall show, in most cases, we can obtain 95 % of the utility of u-shapelets after examining only 5 % of the data.

Both our speedup technique and rapid anytime convergence technique rely upon the consideration of the *complexity* of time series subsequences (we will define *complexity* below, cf. Sect. 5.1). In brief, we demonstrate that the utility of a candidate u-shapelet is highly related to its complexity, and we can leverage this fact to optimize the search order. In addition, we introduce a novel lower bound based on complexity that can admissibly prune off unpromising candidates.

In Sect. 5.1, we describe the concept of complexity that we exploit; Sect. 5.2 contains the description of the relationship of complexity with u-shapelets; and finally we describe our complexity-based speed up techniques in Sect. 5.3.

5.1 Complexity and complexity difference

The term *complexity* is notoriously difficult to define, but in the context of time series can be intuitively thought of as reflecting the number of peaks/valleys/features in the data.

The complexity of a time series can be estimated by a number of different approaches, such as Kolmogorov complexity (Li and Vitanyi 1997), many variants of entropy (Andino et al. 2000; Aziz and Arif 2006), the number of zero crossings, etc. Recently, Batista et al. proposed a parameter-free method to measure the complexity of two z-normalized time series (Batista et al. 2011). The authors use their measure to develop a correction factor for Euclidean distance between objects of different complexities, showing that this produces a significant reduction in the classification error

rate in most cases. We refer the reader to Batista et al. (2011) for more details. In this work, we adopt this complexity measure, but use it for a completely different purpose. We present the formula to measure the complexity from Batista et al. (2011) in (9).

$$C(Q) = \sqrt{\sum_{i=1}^{n-1} (q_i - q_{i+1})^2} \tag{9}$$

This complexity measure is trivial to implement; for example, in Matlab it requires just the statement: `C(Q) = sqrt(sum(diff(Q).^2))`. The intuition behind the measurement is to see it as the total length of the time series *in the y-axis*. Equivalently, a more physical intuition is to imagine that if we could hold the first and last points of the time series and "stretch," it becomes a straight line; the length of this line is the complexity (Batista et al. 2011).

The time taken to compute the complexity of a time series of length n is O(n). However, recall that we must z-normalize each subsequence (cf. Definition 2). We can compute the complexity at the same time with essentially zero overhead; thus, we can get the time series complexity for free. Moreover, we only need O(1) space to save the complexity of a candidate u-shapelet.

For this work, we have developed a novel one-dimensional lower bound of the Euclidean distance between two time series using only the complexities of those time series. Our claim is that, if we divide the difference in complexity between two z-normalized time series by two, the resulting value is always less than or equal to their Euclidean distance. We present our claim in Theorem 1,

Theorem 1 *$CD(P, Q)/2 \leq ED(P, Q)$; where, P and Q are two time series of length n; $CD(P, Q) = abs(C(P) - C(Q))$, and $ED(P, Q) = \sqrt{\sum_{i=1}^{n} (P_i - Q_i)^2}$.*

In order to enhance the flow of the paper, we defer the proof of Theorem 1 to Appendix 1. In the remaining subsections, we show how this complexity-based lower bound is exploited to speed up our u-shapelet discovery process.

5.2 Relationship between complexity and shapelet-score

We have discovered that there is a significant relationship between a time series shapelet-score and its complexity. The plot in Fig. 8 illustrates one example of this relationship on a particular dataset. Note that in line 9 of the extract u-shapelets algorithm in Table 2, we compute gap to measure the goodness of a candidate u-shapelet. We denote this gap as shapelet-score in the rest of the work.

Here we plot the *complexity* versus *shapelet-score* of 10,000 candidate u-shapelets from Two-Pattern dataset. We observe that subsequences with high complexity have low probability to be a u-shapelet. This observation holds for other datasets, although the exact shape of the point cloud can vary. For the example in Fig. 8, we note that, at least for this dataset, any subsequence with a complexity greater than six clearly has no chance to be a u-shapelet. Moreover, if we start our u-shapelet search in the ascending order of the complexities of subsequences (i.e., from *left-to-right* in Fig. 8),

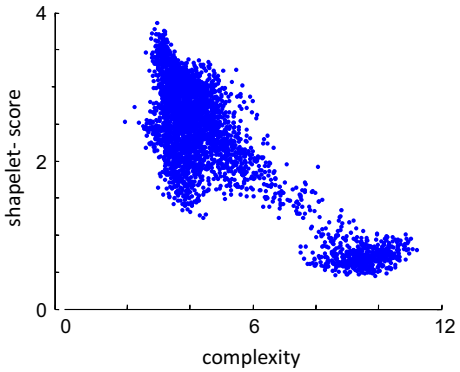

Fig. 8 The relationship between complexity and shapelet-score. Here the correlation is −0.30. Note that computing the complexity of a single time series takes time dependent on the length of that *time series* only, and can be done essentially for free during z-normalization. In contrast, computing the shapelet-score of a single time series takes time dependent on the size of the *dataset*, and may take several orders of magnitude longer than the complexity calculation. Thus we have discovered a very cheap approximate proxy for our measure of interest, which we will exploit in several ways (Color figure online)

we will find the u-shapelet at the very early stage of the search, which in this particular example is after examining just 3.2 % of the data.

In order to demonstrate the utility of the complexity measure in accelerating the u-shapelet discovery process, let us first consider the time required to find a u-shapelet using brute-force search. If there are N time series in a dataset, and each time series is n points long, then there are $N \times (n - m + 1)$ candidate u-shapelets of size m. Thus, brute-force search takes $O(N \times (n - m + 1))$ Euclidean distance computations to compute the shapelet-score of one candidate u-shapelet and $O((N \times (n - m + 1))^2)$ Euclidean distance computations to discover a u-shapelet. This is a very computationally expensive process. Let us denote the total number of candidate u-shapelets as S, and the total amount of time to discover a u-shapelet as $O(S^2)$.

Note that it takes only O(S) time to compute the complexity of all candidate u-shapelets and O(S) space to save those complexities. Thus, the computation of complexities is inconsequential compared to the number of computations required in the brute-force algorithm. We can exploit these complexities to order the search of candidate u-shapelets, producing a more rapid convergence to high scoring candidates, the desirable "diminishing returns" property of anytime algorithms. Moreover, as we shall show in the next section, if we find a high-scoring u-shapelet `best-so-far` in the early stages of our search, we can use its shapelet-score to admissibly prune off many Euclidean distance computations.

5.3 Speeding up the discovery of U-shapelets

Our proposed method comprises of two novel optimization procedures which can speed up the u-shapelet discovery independently of each other. Moreover, if we combine the two optimization procedures, it results in a super linear speedup. The first procedure optimizes the order of choosing the candidate u-shapelets and tends to pro-

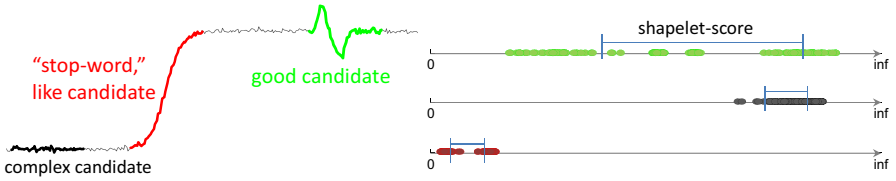

Fig. 9 Example of stop-word, complex candidate, and good candidate with their corresponding orderlines. The complex candidates become noisier after z-normalization (Color figure online)

duce a very good `best-so-far` early in the search. The second procedure prunes off Euclidean distance computations for candidate u-shapelets and can prune most aggressively when the `best-so-far` has a larger value. Thus, both methods work best in each other's presence.

We begin by presenting some notation before describing the speed up techniques in detail.

We note that there are essentially three types of candidate u-shapelets that we will encounter and have to evaluate during search.

1. **Stop word** candidate u-shapelets. These have a very small nearest neighbor distance to almost every time series in the dataset. These candidates are like "stop-words" in text processing (Silva and Ribeiro 2003). That is to say, they are so ubiquitous that they have little or no discriminating power and are thus useless for clustering.
2. **Complex** candidate u-shapelets. These candidate u-shapelets are noisy and have a large nearest neighbor distance to almost every time series in the dataset. These candidates are like *hapax legomena* in text processing (Silva and Ribeiro 2003); that is to say, they are so rare that they have little or no discriminating power.
3. **Good** candidate u-shapelets. These candidate u-shapelets have a small nearest neighbor distance from *some* time series but a large nearest neighbor distance from some other time series (in essence, Definition 4)

In Fig. 9, using a single time series from the trace dataset, we present examples of these three types of candidates along with their orderlines.

By definition, the third group contains the most promising candidates. Thus, if we can find a method that helps us to identify the good candidate u-shapelets very early in our search, then we can design a fast anytime algorithm for u-shapelet discovery. Moreover, if the shapelet-score is high at the beginning of the search, we can exploit the high shapelet-score to prune off a huge number of Euclidean distance computations (cf. Sect. 5.3.2) and result in a faster *exact* algorithm.

Thus, in the first step we determine the order in which we will consider the candidate u-shapelets to compute their shapelet-score. We call this step *outer loop ordering*. In the second step, inside the inner loop for each candidate u-shapelet, we use the best shapelet-score that we have thus far to prune as many Euclidean distance computations as possible. We refer to our new algorithm as *O*uter *L*oop Ordering with Inner *L*oop *O*ptimization (OLLO) and present the algorithm in Table 6.

Table 6 OLLO: An algorithm to speed up U-shapelet discovery

```
Algorithm: speeding up u-shapelet discovery
  1. Optimize outer loop ordering using complexity
  2. Prune off the Euclidean distance computations that are
unpromising in inner loop
```

5.3.1 Algorithm for optimizing outer loop ordering

If we could "magically" begin our u-shapelet search with the candidate u-shapelet that has the highest shapelet-score, that would clearly be the best possible outer loop ordering (the ordering of the remaining candidates would be irrelevant). However, this is clearly not possible as it introduces a "*chicken and egg*" paradox. Therefore, we will exploit the relationship between complexity and shapelet-score that we presented in the last two subsections to design our *Outer Loop Ordering* algorithm.

In order to optimize the convergence rate of the outer loop, we compute an *approximate orderline* for each candidate u-shapelet. The candidate u-shapelets that have a high spread in their approximate orderline have high probability to be a good u-shapelet, and are thus worth examining early in the search. We will refer to this spread as *approximate shapelet-score*. To calculate this term from the *approximate orderline*, we must first compute the *approximate nearest neighbor distance* of the candidate u-shapelet to a time series in the dataset.

Definition 6 (*Approximate nearest neighbor distance*) It is the Euclidean distance between a candidate u-shapelet and a subsequence in a time series which has minimum complexity difference with the candidate u-shapelet.

If the length of a time series is n and the u-shapelet length is m, then to find the *true* nearest neighbor of a candidate u-shapelet in that time series, we need to perform $n \times m$ Euclidean distance computations. However, we propose finding the *approximate* nearest neighbor using only a single Euclidean distance computation. If we *randomly* choose a subsequence to compare to the candidate u-shapelet, we would generally expect it to be a poor match. However, we propose to compare with the subsequence of the time series that has minimum complexity difference with that candidate. The overhead for this is inconsequential, as the subsequence complexities are calculated once and cached in a sorted list; however, this tends to produce a much closer approximate nearest neighbor on average. The set of approximate distances from the candidate u-shapelet to every time series in the dataset is called an Approximate Orderline:

Definition 7 (*Approximate orderline*) An orderline that contains all the *approximate nearest neighbor distances* of a candidate u-shapelet to all the time series of a dataset.

If there are N time series in a dataset, then it takes $O(N)$ time to compute the approximate orderline for a candidate u-shapelet, which is trivial compared to the total amount of Euclidean distance computations required to obtain an *exact orderline*. An

Table 7 An algorithm for finding the optimal order of the candidate U-shapelets

```
Algorithm: Complexity-Based Outer Loop Ordering

For each candidate u-shapelet
  1. Find approximate nearest neighbor distance from each time
series
  2. Compute approximate shapelet-score
  3. Sort the candidate u-shapelets in the descending order of
their approximate shapelet-score
```

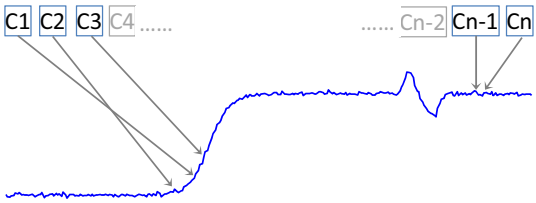

Fig. 10 The complexities of a time series in sorted order with the pointers to the subsequences (Color figure online)

exact orderline contains the exact nearest neighbor distances and it requires $O(N \times (n - m + 1))$ time to compute the exact orderline. The approximate orderline allows the calculation of an *Approximate shapelet-score*:

Definition 8 (*Approximate shapelet-score*) The difference between the maximum and minimum point in the approximate order line.

The approximate shapelet-score is neither an upper nor a lower bound to the true shapelet-score. However, the two *are* correlated, and thus, we can use the former as a cheap-to-compute proxy for the latter when creating a search order.

We search candidate u-shapelets in the descending order of their approximate shapelet-score. We summarize the steps for optimizing the outer loop in Table 7. We refer to this candidate u-shapelets ordering as the *complexity-based outer loop ordering*.

We note here that in order to speed up the search of an approximate nearest neighbor, we must maintain a data structure. During the computation of the complexities of the subsequences of a time series, we store them in sorted order. Thus, it takes just $log(n)$ time to find an approximate nearest neighbor of a candidate u-shapelet in a time series. In Fig. 10, we illustrate the data structure for one time series. Here, $C1, C2, C3, \ldots Cn$ represent the complexities of the subsequences in sorted order, and the pointers show the positions of the subsequences with those complexities.

Before we move on to the next section, we will illustrate the improvement in convergence rate we obtain using our optimization method for the outer loop.

Consider the Trace dataset, which contains 200 time series of length 275. Let the length of u-shapelet be fifty. Thus, there are 45,200 candidate u-shapelets in the dataset. Using brute-force search, it would require 45,199 Euclidean distance computations to obtain the shapelet-score for one candidate u-shapelet. However, using our complexity-based outer loop ordering, we find the best u-shapelet after just 2,086

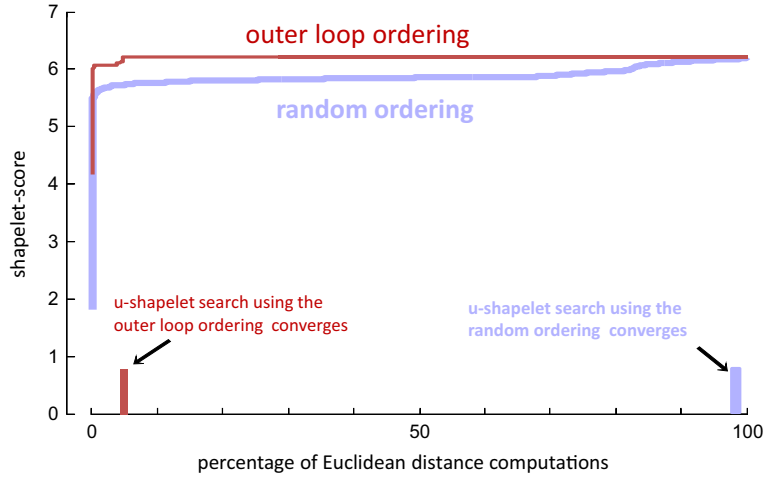

Fig. 11 Comparison of outer loop ordering with random ordering. Complexity-based outer loop ordering is 21.6 times faster than random ordering (Color figure online)

Euclidean distance computations. We compared our method with random ordering, the only obvious strawman. Fig. 11 shows the convergence plot for both approaches by comparing the shapelet-score vs. percentage of Euclidean distance computations. Here our outer loop ordering method converged to the final answer **21.6** times faster than random ordering.

5.3.2 Pruning Euclidean distance computations in inner loop

The complexity-based outer loop ordering produces the desirable fast convergence/diminishing returns property as required by anytime algorithms; however, the absolute time to termination has not changed. Here we consider an *inner loop* technique to address this.

In order to speed up the computations in the inner loop, we leverage off the shapelet-score we get from the outer loop ordering. We keep track of the shapelet-score that has maximum value and call it `best-so-far`. For the candidate u-shapelet in hand, we compute an *inner loop approximate orderline*, which is different from the approximate orderline we described in the previous section.

Definition 9 (*Inner loop approximate orderline*) An orderline that contains the lower bounds and upper bounds of the nearest neighbor distances of a candidate u-shapelet to each time series of a dataset.

This definition begs the question, how do we obtain the lower bounds and upper bounds of the nearest neighbor distances? Since the complexity difference between two time series can give us a lower bound of their Euclidean distance (cf. Theorem 1), we can say that the complexity difference of a candidate u-shapelet and a sub-sequence of a time series that has minimum value can give us a *lower* bound of the nearest neighbor distance.

Definition 10 (*Lower bound of a nearest neighbor distance*) The complexity difference between a candidate u-shapelet and a subsequence from a time series that has minimum value divided by two is a *lower bound of the nearest neighbor distance* of the candidate u-shapelet to that time series.

Moreover, we can use the Euclidean distance between the candidate u-shapelet and the subsequence that has minimum complexity difference as an *upper* bound of the nearest neighbor distance.

Definition 11 (*Upper bound of a nearest neighbor distance*) The Euclidean distance between a candidate u-shapelet and *any* subsequence from a time series is an upper bound of the nearest neighbor distance to *all* subsequences.

From an inner loop approximate orderline, we compute an upper bound of a candidate's shapelet-score and denote it as *best-possible-score* for the candidate. This idea is worth explicitly rephrasing to avoid confusion. We are using an order line that contains upper (and lower) bounds of *Euclidean distance* to compute an upper bound of a candidate's *shapelet-score*.

If this best-possible-score of a candidate is *less than* the `best-so-far` we have, we can abandon that candidate because its shapelet-score can never be greater than the `best-so-far` and thus it cannot become a u-shapelet. On the other hand, if the best-possible-score is *greater than* the `best-so-far`, then we keep updating the inner loop approximate orderline until the best-possible-score is less than the `best-so-far` or we have computed the exact orderline. We summarize these steps in Table 8.

Each time we update an inner loop approximate orderline, we update the upper bounds and lower bounds. In order to get the nearest neighbor distance, we need to update the upper bound and the lower bound until the lower bound is less than the upper bound *or* we have searched all the subsequences of the time series. In Table 9, we present an algorithm for updating the bounds.

In lines five to eleven, we update the lower bounds and upper bounds. We update the lower bound with the next minimum complexity difference. If the lower bound is less than the upper bound, then we compute the Euclidean distance between the candidate u-shapelet and the subsequence with next minimum complexity difference. If the Euclidean distance is less than the current upper bound, then we update the upper bound with this Euclidean distance. Once there is a change in upper bound, we return to line eight of Table 8 and check if the best-possible-score is less than the `best-so-far`.

If there are N time series in a dataset, then for each candidate u-shapelet, we have N points in the exact orderline, each of which represents a nearest neighbor distance. Similarly, we should have N points in an inner loop approximate orderline. As we have a lower bound and an upper bound of a nearest neighbor distance, this opens the question of how to use these to compute the upper bound of the shapelet-score.

Before we answer the question, let us review the attributes of a u-shapelet. As noted in Sect. 2, a u-shapelet separates all its nearest neighbor distances into two distant groups (D_A and D_B). Moreover, a u-shapelet needs to have discriminative power, i.e. each of the group should contain a certain amount of nearest neighbor

Table 8 An algorithm to Prune Euclidean distance computations in inner loop

```
Algorithm: Prune off Euclidean distance computations in inner loop
Require: best-so-far; a candidate u-shapelet
Ensure: best-so-far
1.  Compute inner loop approximate orderline for the candidate u-shapelet
2.  Compute best-possible-score
3.  while true
4.    if best-possible-score < best-so-far
5.      return;     //as this candidate is unpromising
6.    end
7.    Update inner loop approximate orderline
8.    Compute best-possible-score
9.    if inner loop approximate orderline is equal to the exact orderline
10.     exact-score ← best-possible-score
11.     break;
12.   end
13. end
14. if exact-score > best-so-far
15.   best-so-far ← best-possible-score
16. end
```

Table 9 An algorithm to update an inner loop approximate orderline

```
Algorithm: update inner loop approximate orderline
Require: candidate u-shapelet
1.  for each time series
2.    if we have searched all the subsequences or lower bound > upper bound
3.      continue;
4.    end
5.    lower bound ← next minimum complexity difference / 2
6.    if lower bound < upper bound
7.      d ← Euclidean distance between candidate u-shapelet and the subsequence with next smallest complexity difference
8.      if d < upper bound
9.        upper bound ← d
10.       break;
11.     end
12.   end
13. end
```

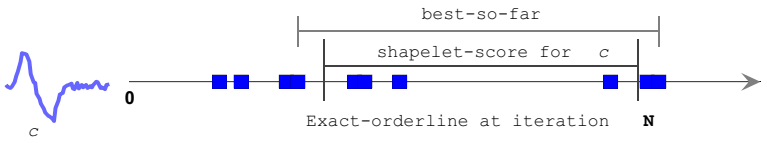

Fig. 12 Exact orderline of a candidate u-shapelet (Color figure online)

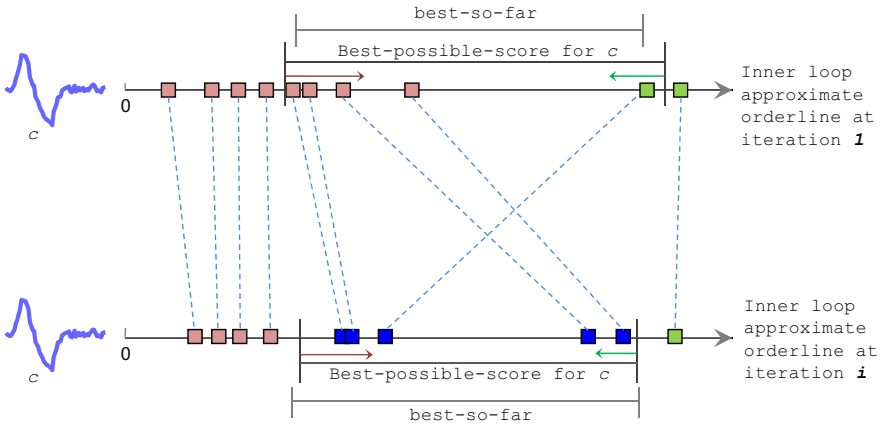

Fig. 13 A candidate u-shapelet and its approximate orderline inner loop at the (*top*) first iteration and (*bottom*) ith iteration. After the first iteration, the best-possible-score is *greater* than the `best-so-far` and we are forced to keep iterating. However, after the ith iteration the best-possible-score is *less* than the `best-so-far` and we abandon the remaining Euclidean distance computations (figure best viewed in *color*). *Key*: Points known by upper bound values only are in *green*; they can only move *left* if updated. Points known by lower bound values only are in *pink*; they can only move *right* if updated. Points known by their true values only are in *blue* (Color figure online)

distances. How we obtain the ranges was discussed in Sect. 2. Let us consider the lower range as $R1$ and the upper range as $R2$. Clearly, $R2$ is greater than $R1$. The readers will note that during iteration of the *while loop* in Table 8, we have T lower bounds and T upper bounds, where T is the total number of time series in the dataset. If we sort all the upper bounds and include $R2$ top upper bounds in the inner loop approximate orderline and add lower bounds for the remaining points, we can obtain the best-possible-score from that inner loop approximate orderline.

In order to illustrate the utility of inner loop optimization, we present a synthetic example. Assume we have ten time series in a dataset and there are N candidate u-shapelets. Therefore, to get the shapelet-score of one candidate u-shapelet, we need N Euclidean distance computations. In Fig. 12, we present a candidate u-shapelet and its exact orderline which requires N Euclidean distance computations.

Using our inner loop optimization, we can reduce the number of Euclidean distance computations from N to some smaller value. In Fig. 13, we present the inner loop approximate orderlines for the above example.

Our first inner loop approximate orderline requires ten Euclidean distance computations. Let, for the above example, our lower range $R1$ is two and our upper range

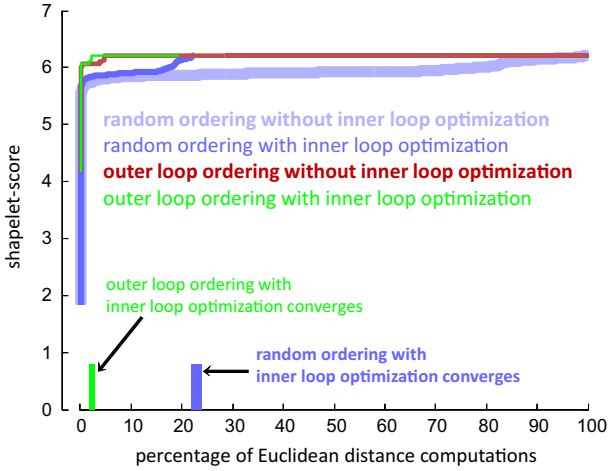

Fig. 14 U-shapelet discovery using outer loop ordering and inner loop optimization. Note that this plot is an updated version of Fig. 11 (Color figure online)

$R2$ is eight. Therefore, we put the top two upper bounds (marked with green) in the inner loop approximate orderline and put eight lower bounds for the rest of the points (marked with pink). If the best-possible-score that we get after the first iteration is greater than the `best-so-far`, then we keep updating the inner loop approximate orderline until the best-possible-score is less than the `best-so-far`. Note that the green points can *only* move to the left (cf. the **if** block in lines eight to eleven in Table 9), and the pink points can *only* move to the right (cf. line 5 in Table 9). Thus, the best-possible-score can only decrease in the next iteration. Note also that the green points can "jump over" with the pink points.

In order to illustrate the utility of inner loop optimization, we combine the inner loop optimization with the plots in Fig. 11. That is to say, we combine inner loop optimization with both complexity-based outer loop ordering and random ordering and measure their performance. This time we find that our method converges to the right answer **nine** times faster than the random ordering (cf. Fig. 14). This demonstrates that inner loop optimization helps to speed up u-shapelet discovery independent of outer loop optimization, and if we combine inner loop optimization with outer loop optimization, the speed up is even better.

We conclude this section with some results from two other datasets (Starlight-curve and Mallat from UCR archive, Keogh et al. 2011). In both cases, our two methods both terminate much earlier and converge much faster (Fig. 15).

6 Experimental evaluation

We have created a webpage (Jesin's Webpage 2013), where we have further results in addition to all datasets and code in order to ensure reproducibility.

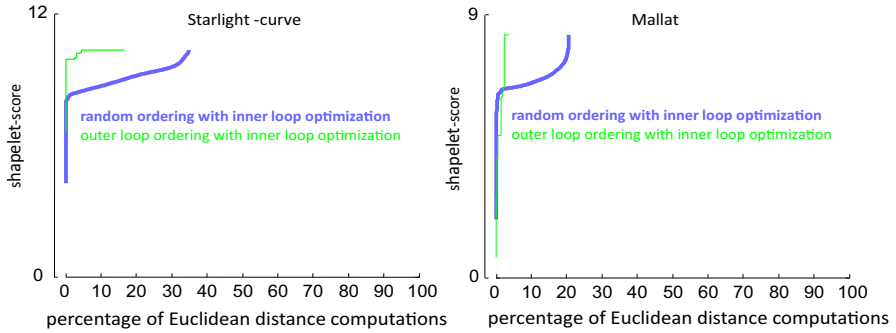

Fig. 15 Speed-up result on *Starlight-curve* and Mallat. The 100% marker on the X-axis is where the brute-force algorithm (i.e. Table 2) terminates (not shown for clarity) (Color figure online)

6.1 Evaluation metrics and experimental setup

There are many different measures for evaluating the quality of time series clustering, including Jaccard score, Rand index, Folkes and Mallow index, etc. (Halkidi et al. 2001). For brevity and clarity of presentation, we only consider the Rand index (Rand 1971). This appears to be the most commonly used clustering quality measure, and many of the other measures can be seen as minor variants of it.

To compute the Rand index, we need the cluster labels *cls1*, returned by a clustering algorithm, and the set of *ground truth* class labels *cls2*. Let A be the number of object pairs that are placed in the same cluster in *cls1* and *cls2*, B be the number of object pairs in different clusters in *cls1* and *cls2*, C be the number of object pairs in the same cluster in *cls1* but not in *cls2*, and D be the number of object pairs in different clusters in *cls1* but in same cluster in cls2. We can then compute the Rand index as follows (Rand 1971):

$$Rand\ index = (A + B)/(A + B + C + D)$$

Values close to one indicate a high quality clustering.

For fairness, we do not present results on execution times for the various algorithms, as we have optimized rival methods only for *quality* not for *speed*. For example, it has been shown that for DTW, the quality of implementation can make at least a two-order magnitude difference (Ding et al. 2008). Moreover, where possible, we take the results directly from other authors' papers. This is possible for *quality*, but essentially meaningless for *timing* results. Note that in our real world case studies presented below (geology, cardiology, electrical demand, etc.), the time to *cluster* the data is an inconsequential fraction of the time to *gather* the data.

Unless otherwise stated, we use k-means as the underlying clustering algorithm. We give the algorithm the objectively correct value of k where known and report the best of twenty runs. Here, "best" means the run that minimized the objective function, *before* we see the class labels and compute the Rand index. We do all this in order to allow meaningful comparisons between various methods, as we have "factored out" anything extraneous that might influence the results. However, note that our algorithm (and

most of the others) also allows spectral clustering, hierarchical clustering, etc. Except where otherwise stated, we use the u-shapelet algorithm as introduced in Zakaria et al. (2012a). The algorithm only searches the first element in the dataset for u-shapelets. However, in Sect. 6.8 we will revisit this limitation.

6.2 Comparison to rival methods

We begin by comparing the performance of our method with the clustering method proposed in Zhang et al. (2005). Zhang et al. proposed an unsupervised feature extraction algorithm for time series clustering (Zhang et al. 2005) where they have used an orthogonal wavelet transform (*Haar wavelet*) to automatically choose the dimensionality of features. They showed a performance improvement over other feature extraction techniques, such as singular value decomposition (SVD) and discrete Fourier transform (DFT).

We compare the clustering quality of extracted features—Zhang's method—and the clustering quality of u-shapelets in Table 10. We tested our method on three benchmark datasets: *Trace*, *Synthetic Control*, *Gun Point*, from the UCR time series archives (Keogh et al. 2011) and four datasets from Zhang et al. (2005) and Kalpakis et al. (2001). Below, we present a brief description of all the datasets.

- *Synthetic Control*: contains 100 instances for each of the six different classes of control chart (Ding et al. 2008).
- *Gun Point*: contains 100 instances for each of the two classes and the dimensionality of the data are 150 (Ding et al. 2008).
- *ECG*: contains seventy time series from three different groups of people. Each time series is 2 s long. The groups are {*malignant ventricular arrhythmia*; *normal sinus rhythm*, *supraventricular arrhythmia*}.
- *Population*: contains time series representing the population estimates from 1900 to 1999 in 20 U.S. states. The data is partitioned based on demographics (Zhang et al. 2005).
- *Temperature*: contains thirty time series of the daily temperatures. The given ground truth is based on geographical distance and climatology (Zhang et al. 2005).
- *Income*: contains 25 time series representing the per capita income from 1929 to 1999 in 25 states of the USA.

Note that in order to be rigorously fair to Zhang et al., we use the Rand index numbers *they* reported (i.e. Table 8 of Zhang et al. 2005) on these datasets, not our own reimplementation.

From the results in Table 10, it is clear that the use of u-shapelets generally gives us a better clustering result than the feature extraction method (Zhang et al. 2005). In Table 10, we also present the result when entire time series is used for clustering, as this has shown to be a surprisingly good straw man (Ding et al. 2008; Keogh and Kasetty 2002).

In the following sections, we present case studies from diverse domains, showing that our *general* technique is competitive with *specialized* techniques, even when we

Table 10 Comparison to rival methods

Dataset (# of class)	Rand index			Number of u-shapelets used
	Extracted features (Zhang et al. 2005)	u-shapelets	Time series ED	
Trace (4)	0.74	**1**	0.75	2
Synthetic-Control (6)	0.85	**0.94**	0.87	5
Gun Point (2)	0.49	**0.74**	0.49	1
ECG (3)	0.4	**0.7**	Not-defined	1
Population (2)	0.8	**0.9**	0.5	1
Temperature (2)	0.8	0.9	**1**	1
Income (2)	0.5	0.5	0.5	1

Highest rand index are given in bold

test on the datasets proposed by the original authors as ideal showcases for their own techniques.

6.3 Rock categorization

Hyperspectral Remote Sensing collects and processes optically recorded information from the electromagnetic spectrum. It is used *worldwide* in applications as diverse as agriculture, mineralogy, and environmental monitoring and has seen lunar and Martian deployments in the last decade.

We consider a dataset used in Cerra et al. (2011) to test a proposed compression-based similarity measure for time series, the normalized compression distance (NCD). The authors in Cerra et al. (2011) compared NCD with Euclidean distance, Spectral Angle, Spectral Correlation, and Spectral Information Divergence on a set of 41 spectra from three different classes of rock (*Felsic, Mafix, Shale*) and showed that their proposed distance measure gives the best clustering results in this domain.

As shown in Fig. 16 (right), in this dataset, our method extracts just two u-shapelets.

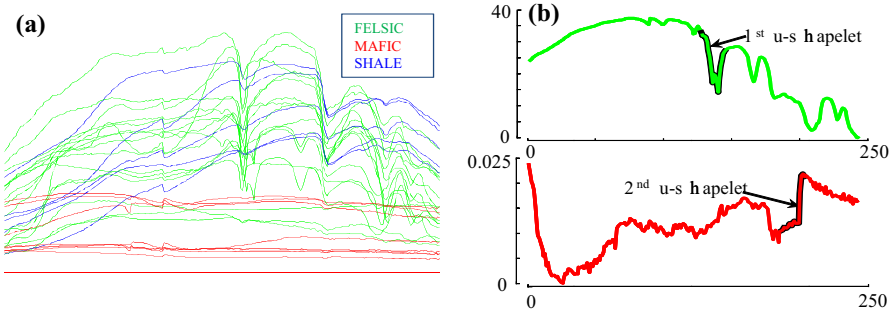

Fig. 16 (*left*) 41 spectra analyzed. (*right*) u-shapelets used for clustering (Color figure online)

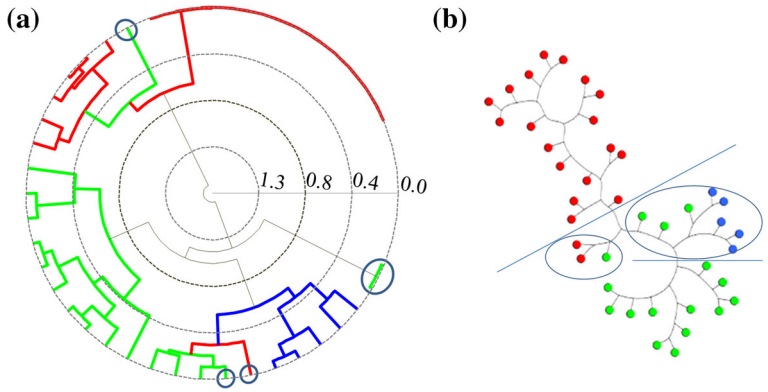

Fig. 17 Hierarchical clustering, (*left*) using u-shapelets and (*right*) using NCD (screenshot of Fig. 4 from Cerra et al. 2011). All *blue lines* are annotations added by us (Color figure online)

In Fig. 17 (left), we present the result of hierarchical clustering using the u-shapelets. To help the reader compare our clustering result with Cerra et al. (2011), we have included a screen capture of their *best* clustering result.

Because they produce an unrooted tree, it is difficult to calculate formal metrics of success. Cerra et al. (2011) noted, "The evaluation is done by visually inspecting if spectra belonging to the same class are correctly clustered in some branch of the tree, i.e. by checking how much each class can be isolated by 'cutting' the tree at convenient points." If we cut the tree of Fig. 17 (right) at the points where we placed the blue lines, we find that the *eleven* nodes encircled are not clustered correctly. Whereas, if we use the two u-shapelets, only *three Felsic* and *two Mafic* rocks are clustered incorrectly. This simple experiment suggests that our clustering method can be used to cluster spectral signatures and it outperforms a variety of methods tuned for this application domain.

6.4 Synthetic dataset (Shahriar)

While *real* datasets such as the one used in the last section are the most convincing demonstrations of our algorithm, in this section (and the next) we consider *synthetic* datasets created by other researchers. We do this to demonstrate two things. First, the u-shapelets returned by our algorithm can offer insight into the data, and second, our very general *unsupervised* approach is competitive even with approaches where researchers design an algorithm, design synthetic datasets to showcase it, and provide *supervision* (i.e. class labels to their algorithm).

We first consider a synthetic dataset consisting of ten examples from two classes of univariate sequences (Shariat and Pavlovic 2011) (the small size we consider is to allow direct comparison to the original paper (Shariat and Pavlovic 2011), as it happens we get *perfect* results over a wide ranges of data sizes). The first class is a sinusoidal signal, while the second class is a rectangular signal, both of which are randomly embedded within segments of Gaussian noise. In Fig. 18, we show two examples

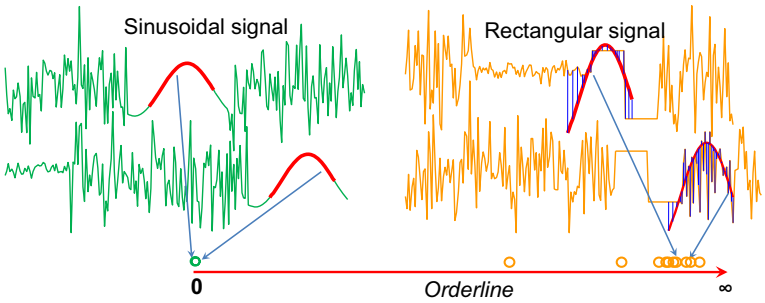

Fig. 18 Two examples of sinusoidal signals and rectangular signals. Sinusoid u-shapelet and its nearest neighbors in the signals are marked with *red*. The orderline shows the subsequence distances of the u-shapelet (Color figure online)

of sequences from each class. Because the time series are of different lengths, the *Euclidean distance* is not defined here. *DTW* can be used, but as the authors (Shariat and Pavlovic 2011) show, the presence of noise and the misalignment of sinusoidal and rectangular signals greatly degrade DTW's performance.

In order to deal with the noise and the misalignment of signals, Shariat and Pavlovic (2011) proposed an approach for sequence alignment based on canonical correlation analysis (CCA). Their method, Isotonic CCA (IsoCCA), generalizes DTW to cases where the alignment is accomplished on arbitrary subsequence segments, instead of on individual samples. They used a 1-NN classifier and the leave-one-out cross validation to classify the twenty signals and reported that the *classification* accuracy is 90 % when using IsoCCA and 60 % when using DTW.

It is important to note that these results are for *classification* accuracy; that is to say, this method requires class labels during training time. In contrast, our clustering algorithm does *not* see class labels when it runs and evaluates the clusters using classification accuracy on holdout data after the clustering is complete.

Using our method, the *sinusoid* signal is extracted as the sole u-shapelet, and by using that u-shapelet, we can get 100 % classification accuracy. In Fig. 18, we mark the sinusoid u-shapelet discovered with its nearest neighbors in three other randomly chosen time series in red/bold. The orderline illustrates the subsequence distances to the sinusoid u-shapelet. We can see from the orderline in Fig. 18 that the Rand index will be one when we cluster the dataset using the subsequence distances of the sinusoid u-shapelet.

To summarize, here we can beat a rival approach, on the data created by the proposed authors, even though they exploit class labels during training time, a luxury that we deny our approach.

6.5 Synthetic dataset (Hartmann)

The attentive reader may wonder whether the superior performance of u-shapelets in the previous section was due to the fact that there is no intra class variability within the sinusoidal or rectangular signals. Fortunately, the datasets created in Hartmann et al.

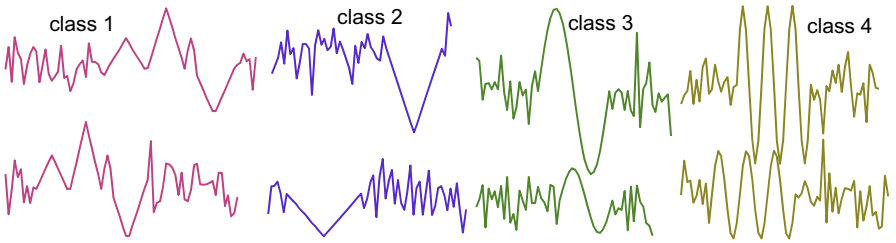

Fig. 19 Sample time series of synthetic dataset 2. Classes 1 and 2, and classes 3 and 4, are confused by most clustering methods (Color figure online)

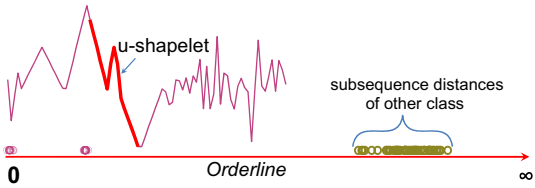

Fig. 20 U-shapelet from class 1 and the corresponding orderline (Color figure online)

(2010) allows us to test this. The dataset consists of four classes. All the time series are composed of embedded class signals and additional noise; however, the class signals have significant randomly-generated variations in length and height. Moreover, the defining shapes of classes two and three are composed of the "sub-signal" shape of classes one and four. In Fig. 19, we present two time series from each class of the dataset.

Hartmann et al. (2010) shows that when they use their method on just class one and class four, they can get good results. However, their method performs poorly if required to consider all four classes. We applied our algorithm to the same set of permutations of classes as the original paper (Hartmann et al. 2010). In all cases, the Rand index is much better than the original author's results (modulo slight difference in how the results are reported, Hartmann et al. 2010). For example, only one u-shapelet is used for the class-1 versus class-4 variant, and the Rand index is perfect (equal to one); Fig. 20 illustrates the u-shapelet and the corresponding orderline. The Rand index is also perfect when we consider the more difficult class 2 versus class 3 variant, but our algorithm uses four u-shapelets in this case.

Note that, once again, we are comparing with an algorithm that was co-developed *with* a synthetic dataset designed to showcase the algorithms strength, and once again, we easily beat the proposed algorithm.

Furthermore, the original authors solved the problem of *supervised* learning by using prototypes or subsequences of the data, which they learned from the training data. In contrast, we are clustering time series using u-shapelets or without any prior knowledge about the dataset.

This experiment provides us with the perfect opportunity to clarify an important point. There is no strict relationship between the number of clusters k and the number of u-shapelets required to create k clusters. In particular, our algorithm requires *four* u-shapelets to cluster class-2 and class-3 into the correct *two* clusters. To illustrate,

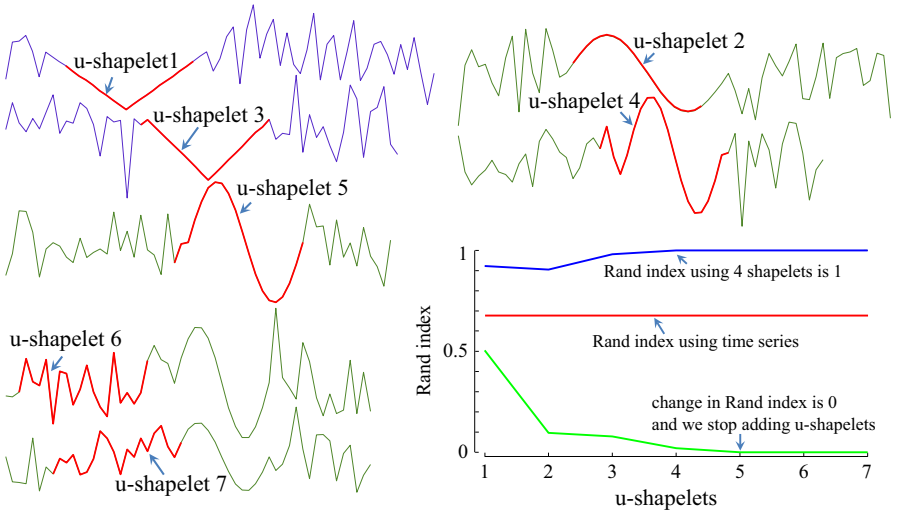

Fig. 21 All the u-shapelets from class-2 versus class-3 version of Hartmann et al. (2010). (*bottom right*) The *blue curve* shows Rand index relative to the ground truth, as we add u-shapelets. The *red line* shows the Rand index if the *whole* time series is used for clustering. The *green curve* shows the change in Rand index with the addition of u-shapelets. This curve correctly predicts that four u-shapelets is best for this problem (Color figure online)

we show all the u-shapelets extracted by our method in Fig. 21 and show the change in the Rand index as we add seven u-shapelets one by one. The last two u-shapelets are just random noise because by that stage of the extraction, all meaningful patterns have been discovered by our algorithm. Note that even though essentially random u-shapelets are extracted in the late stages of our algorithms run, they do not affect the clustering result, as shown by the blue curve in Fig. 21.

6.6 Clustering household devices

In order to reduce carbon emissions, the UK government has planned to equip 27 million households with an intelligent metering system (cf. Fig. 22) at a cost of approximately £10 billion. As a useful by-product, these devices allow individuals to observe their electricity consumption and decrease their carbon footprint. This project is supported by a Cambridge-based company, Green Energy Options (GEO), which has installed the monitoring devices in 187 homes across East Anglia and recorded the usage of individual devices (cf. Fig. 22 (left)) at 15 min intervals for approximately a year.

The clustering problem here is difficult because of the high intra-class variability. Different houses use different devices during different times of the day. Moreover, there is also variability among the power consumption of the devices depending on the brand, etc.

Lines et al. (2011), classifies household appliances using these electricity usage profiles. They derived a set of features (min, max, mean, skewness, etc.) to classify

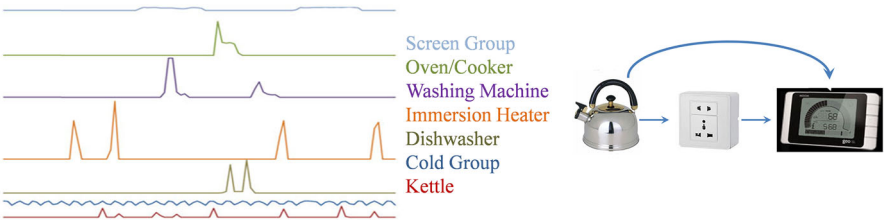

Fig. 22 (*left*) Examples of electricity usage profiles of a single day for the seven devices considered (screen shot from Lines et al. (2011)), (*right*) smart energy management display system

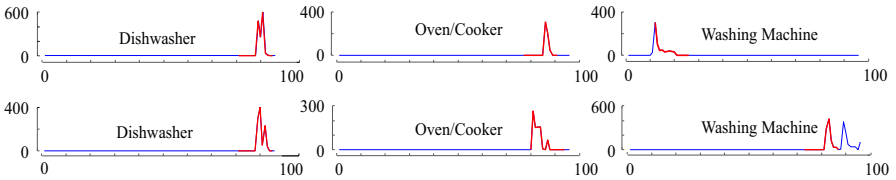

Fig. 23 Sample u-shapelets used for clustering (Color figure online)

the data. We used a subset of the dataset that contains 2,073 time series to test our clustering method. Each time series corresponds to the electric usage profile of a household item over 24 h. If we use whole time series to cluster the dataset, the Rand index is 0.56, whereas by using the u-shapelets, we can achieve a Rand index of 0.86.

In Fig. 23, we have shown some of the u-shapelets that also provide insight about the data. For example, domain experts tell us that the two spikes shown in the dishwasher class correspond to the *rinse and clean* cycles.

6.7 Clustering non-invasive fetal ECG

Automatic analysis of ECG signals is needed to improve diagnostic systems. For this experiment, we have collected non-invasive fetal electrocardiograms (ECGs) signals from PhysioNet (Goldberger et al. 1997), an online medical archive containing digital recordings of physiological signals. As we have mentioned before, most classification and clustering algorithms consider only single heartbeats. We have extracted signals which are 3,000 points long. Each signal contains multiple heartbeats. The signals are taken from a single subject during the 22nd and 38th weeks of pregnancy. The dataset contains 948 such signals. As the reader can see from Fig. 24, the signals are not aligned perfectly. As a result, when we use entire time series for clustering, the Rand index is 0.63, which is close to random. However, using the u-shapelets, we achieve a Rand index of 0.99.

Figure 25 presents the first u-shapelet and the corresponding orderline. Note that for visual clarity, we plotted subsequence distances for ECG signals of week 22 and week 38 in two different lines with two different colors. When we asked cardiologist Dr. Helga Van Herle about the u-shapelets, she suggested that it corresponds to the *p-wave* of the ECG signal, known to change as a function of the developing fetus' changing heart morphology.

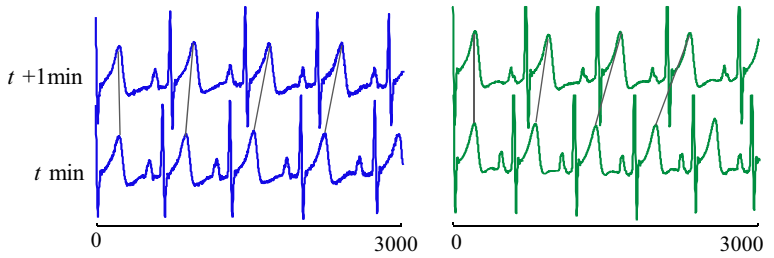

Fig. 24 The misalignment between ECG signals (indcated by the *gray hatch lines*) recorded during (*left*) 22nd week and (*right*) 38th week of pregnancy (Color figure online)

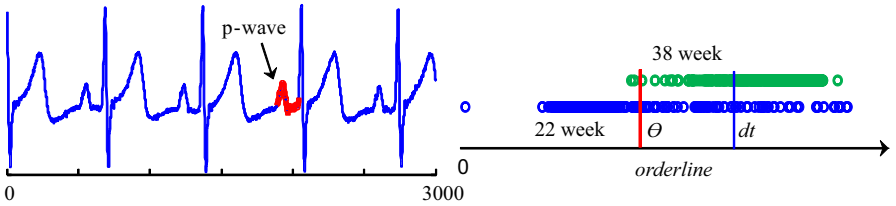

Fig. 25 (*left*) The 1st u-shapelet (*red*/*bold*) and (*right*) its corresponding orderline (Color figure online)

6.8 Comparison of exhaustive search with subset search

The experimental results that we have presented in Sects. 6.2–6.8 are based on searching just a *subset* of the candidate u-shapelets. In particular, we only searched the first time series in the dataset. As we have shown, the u-shapelets that we get from the *subset search* can cluster the datasets very well, equaling or beating the obvious rival methods. However, the u-shapelet that we get from a subset search is not guaranteed to give us the *best* clustering result. There may exist datasets where the u-shapelet, which we obtain from *exhaustive search,* gives us a better clustering result. In Fig. 26, we have presented one such dataset where exhaustive search is better than the subset search. We have used the Starlight-curve dataset from Keogh et al. (2011). There are 120 time series in the dataset, and each time series has 1024 points in it. The exhaustive search gives us the red/bold marked u-shapelet that also has highest shapelet-score, as measured by the Rand index. In contrast, we also ran subset search 120 times, each time putting a different time series as the first one in the dataset. As we can see by the blue/light marks in Fig. 26, searching a subset of the data generally produces inferior results. Here, exhaustive search took about 34 min, whereas *each* subset search took 5.3 min. However, the extra time for the exhaustive search clearly pays dividends.

6.9 Scalability experiment

In this section, we present experimental results that test the scalability of our u-shapelet discovery algorithm in Table 6. The dataset we consider contains 600 time series of length 1,024 from the Mallat dataset [18]. We consider only a u-shapelet length of 100, and thus, there are 555,000 candidate u-shapelets for which we need to compute

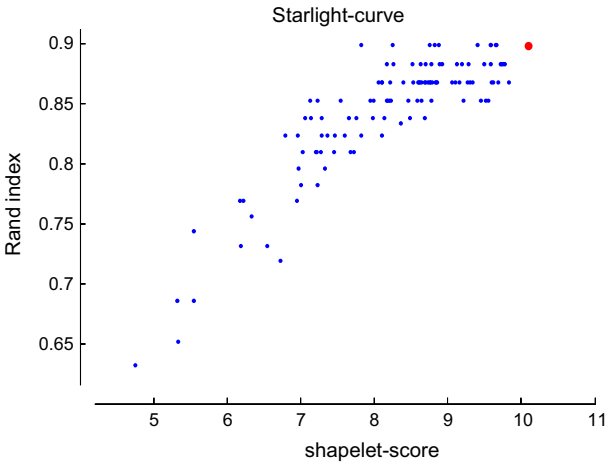

Fig. 26 Comparing *exhaustive* search with 120 invocations of *subset* search on randomly shuffled data. Using exhaustive search, we found the u-shapelet denoted by the *red*/*bold* mark has the highest shapelet-score and (tied) highest Rand index (Color figure online)

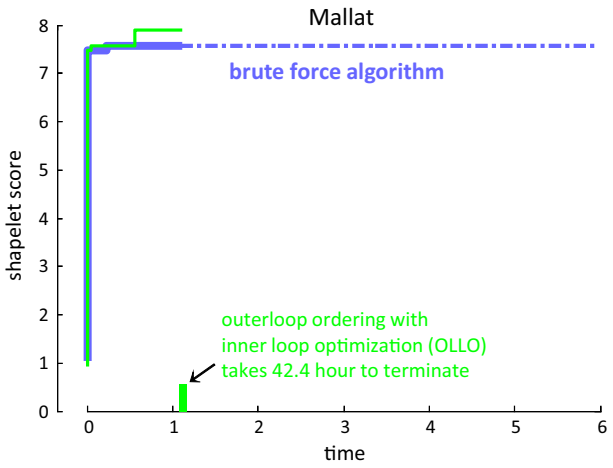

Fig. 27 A test case where outer loop ordering with inner loop optimization takes 42.4 h to complete searching all the candidate u-shapelets, while brute-force search completes searching only 18.2 % of the candidate u-shapelets (Color figure online)

the shapelet-score. In order to compare the performance of our algorithm (OLLO) to the brute-force algorithm, we tested OLLO and the algorithm in Table 6 on the same dataset. As shown in Fig. 27, OLLO (green or thin curve) completes searching all the candidate u-shapelets around five times faster than the brute-force search (blue or thick curve).

6.10 A discussion of absolute time requirements

In the experiments in the previous sections we have confined our timing evaluations to the *relative* improvements over the original version of our algorithms, and measured progress mostly in terms of number of candidates the algorithm must consider. Here we address the questions of the scalability of rival methods, and the *absolute* times.

The most obvious rival methods are clustering on *extracted* features with an off-the-shelf clustering algorithm such as K-means. In Sect. 6.2 we choose clustering method proposed in Zhang et al. (2005) as a representative example, however the literature is replete with examples. Another obvious strawman is simply to feed the raw data (not *extracted* features) into an off-the-shelf clustering algorithm. As we showed in Table 10, these methods are always inferior to u-shapelets in terms of accuracy. This is not surprising, these methods are explicitly required to consider *all* the data when clustering, whereas u-shapelets are able, and almost always do, ignore *most* of the data. The poor quality of their clusterings is not accompanied with bad news about scalability, these methods are extremely fast. The *slowest* experiment using one of these rival methods was the few seconds extracted features (Zhang et al. 2005) took to cluster the ECG data. In contrast our method took around 2.5 h on the same dataset.

Such lethargy is naturally undesirable, but consider the following; the *Population* dataset took ten years to record, the *Temperature* dataset took one year to record, the *ECG* dataset would cost about $22,000 USD to record (according to USC cardiologist, Dr Helga Van Herle), the Household Device data took one year to record, and some of the rock data considered in Sect. 6.3 required a trip to the Moon. Thus, given the enormous cost in money and/or time require obtaining certain datasets, we feel that few people will baulk at a day or two of CPU time.

7 Conclusion and future work

We have illustrated the importance of ignoring *some* data in order to cluster time series in real world applications under realistic settings. We have further introduced unsupervised shapelets and showed their utility. Additionally, we have proposed novel methods that can speed up the u-shapelet discovery process significantly. Our method can select representative u-shapelets from a time series without any human intervention. We have shown the utility of our algorithm in very diverse real world settings, including rock categorization, clustering household devices based on their electricity usage profiles, and clustering ECG signals. Our very general method is highly competitive, even when tested on datasets generated by other authors to explain their own proposed algorithms. We have also tested our method with statistical based clustering, the most obvious strawman.

Future work includes attempts to further exploit the complexity-based lower bound introduced in this work, as it may have unity for speeding up the discovery of shapelets, time series motifs, and time series discords.

Acknowledgments Thanks to all the donors of the datasets. This work was funded by NSF IIS—1161997 and a gift from Siemens.

Appendix: Proof of Theorem 1

Theorem 1

$$\sqrt{\sum_{i=1}^{n-1}(A_i - A_{i+1})^2} - \sqrt{\sum_{i=1}^{n-1}(B_i - B_{i+1})^2} \leq 2\sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} \quad (10)$$

We will prove (10).

Proof Without loss of generality, assume

$$(\mathrm{A}_1 - \mathrm{B}_1)^2 \leq (\mathrm{A}_\mathrm{n} - \mathrm{B}_\mathrm{n})^2 \quad (11)$$

(If not, reverse both A and B. That is, replace A with $(A_n, A_{n-1}, \ldots, A_1)$, and likewise for B. Note that (10) holds for A and B iff it holds for A and B reversed.)

Note that the inequality (10) is equivalent to

$$\mathrm{d}\left(\mathrm{A}, \mathrm{A}'\right) - \mathrm{d}(\mathrm{B}, \mathrm{B}') \leq 2\mathrm{d}(\mathrm{A}, \mathrm{B}) \quad (12)$$

where $d(X, Y)$ is Euclidean distance, i.e., $d\,(X, Y) = \sqrt{\sum_i^n (X_i - Y_i)^2}$, and

$$A' = (A_1, A_1, A_2, A_3, \ldots A_{n-1})\,,$$
$$B' = (B_1, B_1, B_2, B_3, \ldots B_{n-1})\,.$$

So, to prove (10), it suffices to prove (12). Here is a proof of (10).

The triangle inequality holds for Euclidean distance. Applying it twice shows

$$\mathrm{d}\left(\mathrm{A}, \mathrm{A}'\right) \leq \mathrm{d}\,(\mathrm{A}, \mathrm{B}) + \mathrm{d}\left(\mathrm{B}, \mathrm{A}'\right) \leq \mathrm{d}\,(\mathrm{A}, \mathrm{B}) + \mathrm{d}\left(\mathrm{B}, \mathrm{B}'\right) + \mathrm{d}(\mathrm{B}', \mathrm{A}').$$

Rearranging gives

$$\mathrm{d}\left(\mathrm{A}, \mathrm{A}'\right) - \mathrm{d}\left(\mathrm{B}, \mathrm{B}'\right) \leq \mathrm{d}\,(\mathrm{A}, \mathrm{B}) + \mathrm{d}(\mathrm{A}', \mathrm{B}') \quad (13)$$

By inspection, $d(A', B')^2 = d(A, B)^2 - (A_n - B_n)^2 + (A_1 - B_1)^2$, so

$$\mathrm{d}\left(\mathrm{A}', \mathrm{B}'\right) = \sqrt{\mathrm{d}(\mathrm{A}, \mathrm{B})^2 - (\mathrm{A}_\mathrm{n} - \mathrm{B}_\mathrm{n})^2 + (\mathrm{A}_1 - \mathrm{B}_1)^2} \quad (14)$$

Together with (11), this gives

$$\mathrm{d}\left(\mathrm{A}', \mathrm{B}'\right) \leq \sqrt{\mathrm{d}\,(\mathrm{A}, \mathrm{B})^2} = \mathrm{d}(\mathrm{A}, \mathrm{B}) \quad (15)$$

Substituting this into (13) gives,

$$\mathrm{d}\left(\mathrm{A}, \mathrm{A}'\right) - \mathrm{d}\left(\mathrm{B}, \mathrm{B}'\right) \leq \mathrm{d}\,(\mathrm{A}, \mathrm{B}) + \mathrm{d}\,(\mathrm{A}, \mathrm{B}) = 2\mathrm{d}(\mathrm{A}, \mathrm{B}) \quad (16)$$

proving (12) which proves (10). □

References

Andino SLG et al (2000) Measuring the complexity of time series: an application to neurophysiological signals. Human Brain Mapp 11:46–57

Aziz W, Arif M (2006) Complexity analysis of stride interval time series by threshold dependent symbolic entropy. Euro J Appl Phys 98:30–40

Batista G, Wang X, Keogh E (2011) A complexity-invariant distance measure for time seies. In: SDM

Cerra D, Bieniarz J, Avbelj J, Reinartz P, Mueller R (2011) Compression-based unsupervised clustering of spectral signatures. In: Hyperspectral image and signal processing: evolution in remote sensing (WHISPERS 2011)

Ding H, Trajcevski G, Scheuermann P, Wang X, Keogh E (2008) Querying and mining of time series data: experimental comparison of representations and distance measures. Proc VLDB Endow 1:1542–52

Garilov M, Anguelov D, Indyk P, Motwani R (2000) Mining the stock market: which measure is best? In: Proceedings of the ACM KDD

Goldberger AL et al (1997) PhysioBank, PhysioToolkit, and PhysioNet: circulation. Discovery 101:1

Halkidi M, Batistakis Y, Vazirgiannis M (2001) On clustering validation techniques. J Intell Inform Syst 17:107–145

Hartmann B, Schwab I, Link N (2010) Prototype optimization for temporarily and spatially distorted time series. In: AAAI spring symposium series (SSS 2010), pp 15–20

Hirano S, Tsumoto S (2006) Cluster analysis of time-series medical data based on the trajectory representation and multiscale comparison techniques. In: Proceedings of the ICDM

Hu B, Chen Y, Keogh E (2013) Time series classification under more realistic assumptions. In: SDM, pp 578–586

Hu B, Rakthanmanon T, Hao Y, Evans S, Lonardi S, Keogh E (2011) Discovering the intrinsic cardinality and dimensionality of time series using MDL. In: Proceedings of the ICDM

Jiang D, Tang C, Zhang A (2004) Cluster analysis for gene expression data: a survey. IEEE Trans. Knowl Data Eng 16(11):1370–1386

Jesin's Webpage (2013) https://sites.google.com/a/ucr.edu/clusteringtsusingushapelet/. Accessed 19 April 2015

Kalpakis K, Gada D, Puttagunta V (2001) Distance measures for effective clustering of ARIMA time-series. In: ICDM

Keogh E, Kasetty S (2002) On the need for time series data mining benchmarks: a survey and empirical demonstration. In: Proceedings of the ACM KDD, pp 102–111

Keogh E, Lin J, Truppel W (2003) Clustering of time series subsequences is meaningless: implications for past and future research. In: Proceedings of the IEEE ICDM, pp 115–122

Keogh E, Zhu Q, Hu B, Hao Y, Xi X, Wei L, Ratanamahatana CA (2011) The UCR time series classification/clustering homepage. http://www.cs.ucr.edu/~eamonn/time_series_data/. Accessed 19 April 2015

Kosala R, Blockeel H (2000) Web mining research: a survey. In: ACM SIGKDD

Li M, Vitanyi P (1997) An introduction to Kolmogorov complexity and its applications, 2nd edn. Springer, Berlin

Liao TW (2005) Clustering of time series data—a survey. Pattern Recognit 38:1857–1874

Lines J, Bagnall A, Smith PC, Anderson S (2011) Classification of household devices by electricity usage profiles. In: IDEAL, LNCS, vol 6936

Lin J, Khade R, Li Y (2012) Rotation-invariant similarity in time series using bag-of-patterns representation. J Intell Inform Syst 39:287–315

Möerchen F (2003) Time series feature extraction for data mining using DWT and DFT. Technical report no. 33. Philipps-University Marburg, Marburg

Muata K (2007) Post-pruning in decision tree induction using multiple performance measures. Comput Oper Res 34:3331–3345

Mueen A, Keogh E, Young N (2011) Logical-Shapelets: an expressive primitive for time series classification. In: Proceedings of the ACM SIGKDD, pp 1154–1162

Rakthanmanon T, Keogh E, Lonardi S, Evans S (2011) Time series epenthesis: clustering time series streams requires ignoring some data. In: Proceedings of the IEEE ICDM

Rand WM (1971) Objective criteria for the evaluation of clustering methods. J Am Stat Assoc 66:846–850

Ratanamahatna CA, Keogh E (2004) Making time-series classification more accurate using learned constraints. In: SDM

Ruiz EJ, Hristidis V, Castillo C, Gionis A (2012) Correlating financial time series with micro-blogging activity. In: WSDM

Salton G, Wong A, Yang CS (1975) A vector space model for automatic indexing. Commun ACM 19:613–620

Shariat S, Pavlovic V (2011) Isotonic CCA for sequence alignment and activity recognition. In: Proceedings of the IEEE international conference on computer vision (ICCV 2011), pp 2572–2578

Silva C, Ribeiro B (2003) The importance of stop word removal on recall values in text categorization. In: Proceedings of the international joint conference on neural networks 2003

Wang X, Smith K, Hyndman R (2006) Characteristic-based clustering for time series data. Data Min Knowl Discov 13:335–364

Xing Z, Pei J, Yu P, Wang K (2011) Extracting interpretable features for early classification on time series. In: Proceedings of the SDM

Ye L, Keogh E (2009) Time series shapelets: a new primitive for data mining. In: Proceedings of the ACM SIGKDD, pp 947–956

Zakaria J, Mueen A, Keogh E (2012) Clustering time series using unsupervised-shapelets. In: Proceedings of the IEEE ICDM, pp 785–794

Zakaria J, Rotschafer S, Mueen A, Razak KA, Keogh E (2012) Mining massive archives of mice sounds with symbolized representations. In: Proceedings of the SDM

Zhang H, Ho TB, Zhang Y, Lin MS (2005) Unsupervised feature extraction for time series clustering using orthogonal wavelet transform. J Inform 30:305–319

Zhang M, Sawchuk AA (2012) Motion primitive-based human activity recognition using a bag-of-features approach. In: ACM SIGHIT international health informatics symposium (IHI 2012), pp 1–10

Zilberstein S (1996) Using anytime algorithms in intelligent systems. AI Mag 17:73–83