Scalable Clustering of Time Series with U-Shapelets

Liudmila Ulanova Nurjahan Begum Eamonn Keogh
University of California, Riverside
{lulan001, nbegu001, eamonn}@cs.ucr.edu

Abstract

A recently introduced primitive for time series data mining, *unsupervised shapelets* (*u-shapelets*), has demonstrated significant potential for time series clustering. In contrast to approaches that consider the *entire* time series to compute pairwise similarities, the u-shapelets technique allows considering only *relevant* subsequences of time series. Moreover, *u-shapelets* allow us to bypass the apparent chicken-and-egg paradox of defining *relevant* with reference to the clustering itself. U-shapelets have several advantages over rival methods. First, they are defined even when the time series are of different lengths; for example, they allow clustering datasets containing a mixture of single heartbeats and multi-beat ECG recordings. Second, u-shapelets mitigate sensitivity to irrelevant data such as noise, spikes, dropouts, etc. Finally, u-shapelets demonstrated ability to provide additional insights into the data. Unfortunately, the state-of-the-art algorithms for u-shapelets search are intractable and so their advantages have only been demonstrated on tiny datasets. We propose a simple approach to speed up a *u-shapelet* discovery by two orders of magnitude, without any significant loss in clustering quality.

1 Introduction

Time series clustering is an area of research that has attracted a significant amount of effort in the last two decades [1][9]. Virtually all research has focused on introducing novel similarity measures and/or novel clustering techniques. In contrast, a recent technique, u-shapelets [26], uses the Euclidean distance as the similarity measure, and a k-means-like technique as the clustering algorithm. The novelty of u-shapelets is in selectively ignoring most of the data, and only using a small number of subsequences for clustering.

Before introducing our contributions to u-shapelet discovery, we will first (re)argue the case for u-shapelets in the crowded literature of time series clustering techniques.

1.1 Why U-shapelets are the Technique of Choice

The most compelling feature of u-shapelets is that they can ignore irrelevant data when clustering. Consider the tiny dataset shown in Figure 1. While the correct clustering would be obvious to the human eye, even without our color-coded hint, most clustering algorithms would perform poorly for two reasons. The first reason is that the data are not aligned. This could be partially mitigated here by using Dynamic Time Warping (DTW). However, if we considered eight-minute instead of eight-second snippets, DTW would require hours even for just six objects. More important, however, is the fact that perhaps half the data in each time series is *simply irrelevant* to the class, consisting of random environmental background sounds. Any clustering algorithm that is forced to consider and "explain" such irrelevant data is doomed to failure.

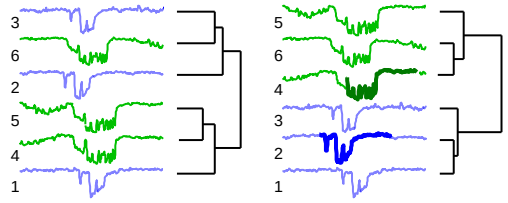

Figure 1: Six 8-second snippets of bird songs in MFCC space, three from the Olive-sided Flycatcher (*Contopus cooperi*) and three from the White-crowned Sparrow (*Zonotrichia leucophrys*). *left*) The clustering using Euclidean distance is essentially random. *right*) The clustering using u-shapelets (denoted dark/bold) correctly separates the two species and gives insight as to the most telling differences between them

An additional advantage of u-shapelet clustering is that it is defined for datasets in which the individual objects are of different lengths. This is not the case for most techniques in the literature. For example, a recent paper that considers the clustering of motion capture time series data tells us, "*While the original motion sequences have different lengths, we trim them* (to have) *equal duration*" [9]. Furthermore, the location of this trimming is subjective, relying on the (human) ability to find the region *"...most significant in telling human motion apart*" [9]. Note that these authors are to be commended for stating their assumptions so concretely. In the vast majority of cases, no such statements are made; however, the "equal length" assumption is implied, and the trimming to equal length is done by exploiting expensive human skill.

The final advantage of u-shapelets is that they are much more expressive in terms of representational power. In particular, they allow separating data belonging to one class and assigning the remaining data a "non-class" label. Figure 2 illustrates this idea in two-dimensional space. If we task k-means with clustering the data shown in Figure 2.*top.left*, it will produce the intuitive cluster labels shown in Figure 2.*top.right*. Here the three classes are simple Gaussian "balls." However, let us now consider the case shown in Figure 2.*bottom.left*. Here we have two of the clusters used in the previous case, but the remaining third of the data comes from a uniform "background" distribution. In this case, k-means produced the clustering shown in Figure 2.*bottom.right*.

The important observation is not that k-means cannot correctly label the background cluster; it is that the presence of the background cluster can cause some of the data that is highly clusterable to be mislabeled. As shown in Figure 3,

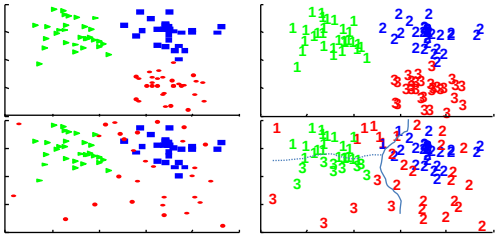

Figure 2: *top.row*) k-means can recover clusters when *all* objects belong to *some* cluster. However, *bottom.row* shows that k-means has difficulty recovering the *same* clusters in the presence of items that do not belong to *any* cluster. Note that the colors shown are for human introspection only; they are not available to the algorithm.

we desire an algorithm that can cluster data that is clusterable, but ignore the non-clusterable data, i.e., non-clusterable data should not affect the outcome of the algorithm. As we shall show, u-shapelets solve this for the time series case.

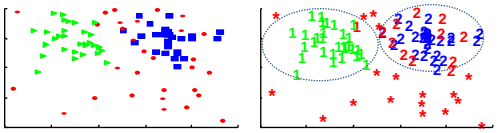

Figure 3: An ideal clustering algorithm would recover the two obvious clusters in this data (enclosed by the dashed circles), without being affected by the non-clusterable data

Many research efforts in time series clustering have focused on modifying the distance measure to be invariant to some property of the data [2][7], but still use k-means or some similar partitional clustering algorithm as the underlying clustering mechanism. These efforts assume that every time series belongs to *some* cluster. However, it is not clear why this should be the case, and in Section 4 we will show it is *not* the case for many real-world datasets. With a little introspection it is easy to see why this assumption is more often unwarranted for time series than for other types of data. Recall the bird calls shown in Figure 1. While both of these birds are relatively vocal, if we attempt to estimate population density by clustering a full day of data on a minute-by-minute basis [3], we may expect to find that the majority of snippets will not contain *any* bird sounds. Thus, we argue that the representational power of the clusterings algorithm is of key importance with time series, and that we *must* have the ability to leave some (perhaps *most*) of the data unclustered.

1.2 Scalability Issues

Although u-shapelets have shown considerable promise for time series clustering, the u-shapelet extraction algorithms proposed to date are intractable for large datasets [26]. To mitigate this, the algorithm in [26] resorts to computing gap scores for the subsequences of just the *first* time series in the dataset, making it *order dependent* and brittle to an unusual instance being the first item encountered. To eliminate this undesirable property, we must compute the gap score for *every* subsequence of each time series in the dataset. Each score requires a nearest neighbor search of the subsequence in question to *each* time series in the dataset. While this algorithm can be improved by pruning and early abandoning techniques from [14], this only produces a relatively modest speedup.

As we noted, the bottleneck of the state-of-the-art approach is that it requires computing all of the distances between time series subsequences in the dataset and then choosing the best subsequence (in a sense explained in Section 2) as a u-shapelet. Our work leverages the observation that most of these computations are not necessary if we can identify a small fraction of all possible u-shapelet candidates to compute the actual distances, and by the further observation that a hashing algorithm can identify a very small set of u-shapelet candidates, which will contain the best u-shapelet with very high probability.

1.3 Summary of Contributions

We conclude this section with our contributions:

- We introduce SUSh (Scalable U-Shapelet) – a hash-based algorithm that allows u-shapelet discovery two orders of magnitude faster than current techniques.
- We produce the first taxonomy of u-shapelets. In particular, we show that while there is only one way to be a high-scoring u-shapelet, there are two distinct ways to be a low-scoring u-shapelet. This observation is important because it informs our speedup strategies, and may be further exploited by others in the community.
- We make all of our code and data available [19] to allow confirmation and extension of our work.
- Finally, while (supervised) shapelets now have a significant user base and have seen applications in domains as diverse as gesture recognition [10], severe weather prediction [13], and biometrics [17], there is currently sparse evidence for the utility of u-shapelets. Here we forcefully provide such evidence, showing that domain-agnostic u-shapelets can outperform rival techniques, even after those algorithms are carefully tuned to the problem at hand by human experts.

The rest of this paper is organized as follows. In Section 2 we present the formal definitions and background. Section 3 describes our approach. Section 4 contains experimental results and comparisons to rival methods. Section 5 offers conclusions and avenues for future work.

2 Definitions and Background

We begin by defining the key terms used in this work. Some of these definitions are restated or adapted from [26], but are included here for completeness.

As visually hinted at in Figure 1, we are not interested in global properties of time series, but in the properties of local time series subsequences:

DEFINITION 1. An *unsupervised-shapelet* (*u-shapelet*) *candidate Ś* is *any* subsequence that has a number of data

points less than or equal to the number of data points of the shortest time series in the dataset.

Figure 4 shows examples of u-shapelet candidates. Note that this definition does not require the u-shapelet to be a subsequence of a time series existing in the dataset, as was the case with Figure 1. However, constraining the u-shapelet to be a subsequence of an existing time series makes the search space finite.

DEFINITION 2. The subsequence distance *sdist*(*S*, *T*) between a time series *T* and a subsequence *S* is the minimum of the distances between the subsequence *S* and all possible subsequences of *T* of length equal to the length of *S*.

This definition opens the question of which distance measure to use for *sdist*. We use the ubiquitous Euclidean distance (ED), and exploit the recent speedup techniques for its calculation proposed by Mueen et al. [14]. ED is known to be very competitive for time series problems [24]. Dynamic Time Warping *can* be more accurate on some problems, but this is because it is able to be invariant to the cumulative distortions in the time axis which are inevitable in a *long* sequence. Our relatively *short* u-shapelets neither require nor benefit from this invariance. Following standard practice in the community, we z-normalize all subsequences before any ED calculations [14][24][26].

By computing the *sdist* between a u-shapelet candidate and all time series in a dataset, we create an orderline:

DEFINITION 3: An *orderline* is a vector of subsequence distances *sdist*($\acute{S}$, T_i) between a u-shapelet candidate $\acute{S}$ and all time series T_i in the dataset.

The time required to calculate an orderline for a single u-shapelet candidate is $O(NM \log M)$, where N is the number of time series in the dataset and M is the average length of the time series. This computation in itself is not too daunting; however, brute-force search *requires* K such calculations, where K is the number of subsequences. The size of K depends on the length of the u-shapelets, but is O(NM). Thus, our ultimate goal is to only compute a tiny fraction of such orderlines.

A u-shapelet candidate can be "good" or "bad," that is to say, high-scoring or low-scoring.

DEFINITION 4: A *good u-shapelet candidate* $\acute{S}$ is a subsequence having the following property: *sdist* between $\acute{S}$ and any time series in one group D_A is significantly smaller than *sdist* between $\acute{S}$ and any time series in another group D_B: *sdist*($\acute{S}$, D_A) << *sdist*($\acute{S}$, D_B).

Thus, a u-shapelet candidate $\acute{S}$ has a separation power: we can split the time series in the dataset into two groups by considering the subsequence distances to $\acute{S}$.

A *bad u-shapelet candidate* is any subsequence that does not have such separation power. Note that u-shapelet candidates can be bad in two ways: they may be either *stop-word* u-shapelets or *outlier* u-shapelets.

DEFINITION 5: A *stop-word u-shapelet* is a subsequence that has similar subsequences in the majority of the time series in the dataset.

A stop-word u-shapelet does not have separation power because its *sdist* to all or the majority of time series in the dataset is small. By analogy to text retrieval, a stop-word u-shapelet is like the words "*the*" or "*and*." Since these words appear in virtually every document, they are useless as features.

An alternative way for a u-shapelet candidate to be bad is to be a "unique" *outlier* subsequence.

DEFINITION 6: An *outlier u-shapelet* is a subsequence that is close to too small a fraction of the dataset to be considered worthy of representing a cluster.

Again by analogy to text retrieval, we can think of an outlier u-shapelet as being a *hapax legomena*. They are simply too rare to be useful features for discrimination.

Figure 4 demonstrates the three types of u-shapelet candidates in the raw data and their orderlines.

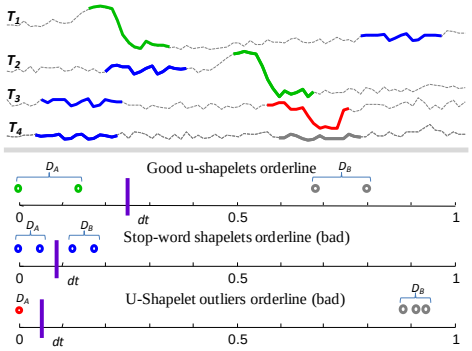

Figure 4: *top*) Good u-shapelet candidates presented as green subsequences in time series T_1 and T_2; bad u-shapelets shown in red in time series T_3 (*outlier*) and blue (*stop-word*). *bottom*) Orderlines for the three different types of u-shapelets (best viewed in color)

As the reader may have intuited from the orderlines shown in Figure 4, a good u-shapelet will produce a large "gap" between D_A and D_B. To measure its "quality," we take the definition from [26]:

$$gap = \mu_B - \sigma_B - (\mu_A + \sigma_A), \quad (1)$$

where μ_A and μ_B denote *mean*(*sdist*($\acute{S}$, D_A)) and *mean*(*sdist*($\acute{S}$, D_B)), and σ_A and σ_B represent *std*(*sdist*($\acute{S}$, D_A)) and *std*(*sdist*($\acute{S}$, D_B)), respectively. If D_A or D_B consists of only one element (or of an insignificant number of elements that cannot represent a separate cluster) the gap score is assigned to zero in order to ensure the high-scored u-shapelet candidates have true separation power.

While there are several u-shapelet clustering algorithms we can define (cf. Section 3.4), all of them require optimizing this gap score. However, as hinted at above, even if we confine our attention to u-shapelet candidates that are subsequences from our dataset, this means that we must invoke the Euclidean distance $O(NM^2 \log M)$ times. We

propose to vastly reduce this number with *a hashing-based* algorithm, which requires the review of a common time series discretization technique in Section 2.2.

One last item we need to introduce does not refer to the u-shapelets directly, but refers to the clusterings. In most cases we want to prevent pathological results with clusters having a very small number of items or the vast majority of the dataset. To enforce this "balance" of cluster sizes, [26] suggested constraints on relative sizes of D_A and D_B using the ratio:

$$(1/r) < |D_A|/|D_B| < (1 - 1/r), \qquad r > 1 \qquad (2)$$

Note that r in our work does not correspond to the *number* of clusters obtained by clustering with u-shapelets; it simply filters out the u-shapelet candidates that are not within the desirable range of separation ratio. If r is set to one, we will have *perfectly* balanced cluster sizes as $|D_A|/|D_B| = 1$. If we increase r we are allowing increasingly unbalanced cluster sizes.

While this constraint is undoubtedly helpful in domains where the user has enough knowledge to steer the clustering towards (or away from) certain solutions, for simplicity in this work we hardcode r=5.

2.1 A Motivating Observation

Consider the much-studied *Trace* dataset [8] which contains 200 time series of length 275 in four equal-sized classes. If we run a brute-force u-shapelet discovery on this dataset we find the best u-shapelet has a gap score of 0.75. If we use this u-shapelet to separate data we obtain a Rand index [15] of 1 (perfect clustering). However, as Figure 5 shows, *any* u-shapelet that scores above 0.65 produces the same Rand index.

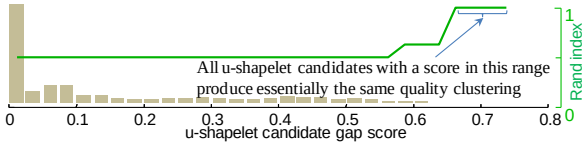

Figure 5: *gray*) The distribution of all u-shapelet scores computed during a brute force search. *green*) The minimum Rand index of these u-shapelets. Once the u-shapelet score is greater than about 0.65, it can achieve the same Rand index as the best u-shapelet

While the shape of this distribution differs on other datasets, the general rule seems universally true. Of the huge number of possible u-shapelets (say, millions) there will be a large number (say, thousands) that will differ in u-shapelet score only slightly, and in external quality metrics (e.g., Rand index) not at all. Thus, in order to obtain a high-quality clustering it is sufficient to find one of these "*good enough*" u-shapelets, a significantly easier task.

DEFINITION 7: Let the best u-shapelet in the dataset have a gap score of n_{best} and the left part of its orderline contain a set of time series D_{A_best}. We call a u-shapelet having a gap score n_{good} and containing the same set of time series on the left part of its orderline $D_{A\_good} = D_{A\_best}$ as a *good enough* u-shapelet if it has the following property: there is no u-shapelet candidate with a gap score of $n_{any} > n_{good}$ and left part of its orderline D_{A_any} such that $(D_{A\_any} \neq D_{A\_best})$.

2.2 SAX Overview

Symbolic Aggregate approXimation (SAX) [11] allows the transformation of a *real-valued* time series of any length n to a discrete *string* of length l, where $l \ll n$. This change of representation reduces not only the dimensionality of the time series, but, more critically for our application, the cardinality. This finite (and small) cardinality allows us to avail of algorithms defined only for discrete data, such as hashing.

We denote a time series T, reduced to cardinality c and dimensionality d, as SAX$(T)_{c,d}$. The first step towards obtaining a SAX word from a time series is to reduce the dimensionality of a time series via the Piecewise Aggregate Approximation (PAA), as shown in Figure 6. After the PAA representation is obtained, it is then discretized into the symbolic representation.

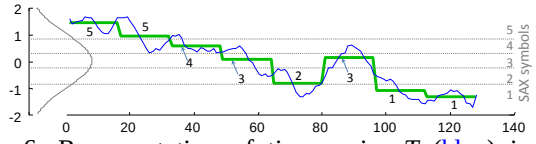

Figure 6: Representation of time series T (blue) in PAA (green/bold) converted into a SAX word, SAX$(T)_{5,8}$ = {5,5,4,3,2,3,1,1}, with c = 5 and d = 8

It is important to note that we do not convert the entire time series into one SAX word. We are not interested in the *global* properties of the times series, but in *local* subsequences. Therefore, we perform a subsequence extraction from the time series using a sliding window to convert each subsequence into an individual SAX word. The SAX representation of the time series will consist of a *set* of SAX words, as shown in Figure 7.

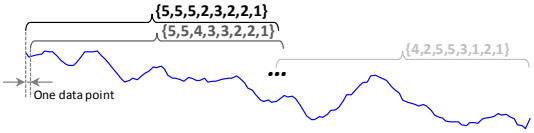

Figure 7: Time series T converted into a set of SAX words, {5,5,5,2,3,2,2,1}, {5,5,4,3,3,2,2,1}, ..., {4,2,5,5,3,1,2,1}, using a sliding window of length 64

Our hashing algorithm presented in Section 3 uses a sliding window of the u-shapelet candidate's length; thus, each time series in the dataset will be represented as a set of SAX words, whereas each u-shapelet candidate is represented as a *single* SAX word.

We can use SAX representations as proxies for the real-valued time series. However, it is important to preempt the discussion of an *apparent* solution to our problem. One might imagine that given this discrete, SAX version of our dataset, one could simply avail of one of the many algorithms developed by the bioinformatics community to approximately solve a variant of the NP-complete

Distinguishing Substring Selection problem [5], a very close analogue of our current problem[1]. However, while it is possible to define a distance measure on the SAX words that lower bounds the Euclidean distance [11], it is possible (although *rare*) that two *distinct* strings could have a vanishingly small Euclidean distance, and that two identical strings could refer to time series with a large Euclidean distance. Nevertheless, as we shall show in Section 3.2, a SAX-guided search allows us to quickly find high-quality u-shapelets.

3 Our Approach

Speedup techniques based on SAX have been proposed for both motif discovery [11] and supervised shapelet discovery [16]. We exploit similar techniques of using hashing with randomized "don't care" masks for unsupervised-shapelet discovery, but must consider some non-trivial modifications to the algorithms.

3.1 Overview of SUSh Algorithm

We are now in a position to explain the intuition that allows us to use hashing to efficiently find u-shapelets.

First, we converted the time series into a SAX representation using a sliding window of the u-shapelet candidate's length. We expected similar time series to map to similar, but not necessarily identical strings. To increase the probability of similar time series mapping to the same symbolic representation, we randomly assigned some positions in the SAX word as "don't cares," an idea we adapted from the bioinformatics community [20][22], and generally called *random masking*.

As shown in [16], the approach of using SAX with random masking is relatively insensitive to parameter choices. Thus, we adopted and fixed the parameters from [16], assigning the cardinality of SAX words to 4, the dimensionality to 16 and the number of rounds of random masking to 10 in all experiments. In extensive experiments (relegated to [19] for brevity) we confirmed that our algorithm is not sensitive to parameter choice. Figure 8 illustrates the result of applying random masking to identify similar subsequences from their SAX representation on the toy dataset we presented in Figure 4. The green subsequences $\acute{S}_1$ and $\acute{S}_3$, while near identical in the original space, have SAX representations that differ by a single symbol: $SAX(\acute{S}_1)_{6,6}$={6,6,***3***,2,2,2}, while $SAX(\acute{S}_3)_{6,6}$ = {6,6,***4***,2,2,2}. It is this situation that prevents the direct application of solutions to the problem based on tests of string *equality* [5]. This is unavoidable for all cardinality reduction schemes though it *is* sometimes possible that a different choice of cardinality or dimensionality would have resulted in two identical strings. However by applying random masking we find that subsequences $\acute{S}_1$ and $\acute{S}_3$ often share their *masked* SAX representation (in the figure under the first mask, *both* $\acute{S}_1$ and $\acute{S}_3$ have the representation {*,6,*,2,2,2} and under the second, {6,6,*,2,2,*}). As we apply more random projections we expect similar subsequences to collide more often.

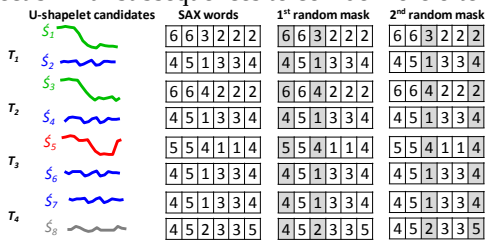

Figure 8: U-shapelet candidates, their original SAX representation (c = 6 and d = 6) and SAX words after two rounds of random masking of two symbols

Obviously, u-shapelet candidates that share the exact same SAX word will always collide (u-shapelet candidates $\acute{S}_2$, $\acute{S}_4$ and $\acute{S}_6$ in Figure 8). However, for our purposes it is more important to find some subset of u-shapelet candidates that only appear in some time series in the dataset, but not in the majority of them. Consider u-shapelet candidate $\acute{S}_8$: under the 2nd random mask it will collide with four other candidates which makes the number of collisions highly variable.

For *supervised* ("classic") shapelets it would be sufficient to compute how often (under different random masks) each candidate appeared in each *labeled* class of time series and to choose those that appeared often in one class but not in another. However, as we do not have class labels, we cannot exploit this property, but we have discovered some other relationships discussed below.

First, the taxonomy presented in Section 2 tells us we can expect the u-shapelet candidates having too many or too few collisions to be stop-words or outliers, respectively. After several rounds of random masking were applied we counted how many time series had a subsequence sharing the masked SAX signature with each u-shapelet candidate. We filtered out all of the candidates that shared their SAX signature with most of the time series (i.e., *stop-word u-shapelets*) or with only a tiny fraction of them (i.e., *outlier u-shapelets*).

Second, we have observed that the variability in the number of collisions for a SAX subsequence is a good predictor of the eventual quality of the corresponding u-shapelet. Figure 9 demonstrates the distribution of the *maximum* gap scores of u-shapelet candidates for each standard deviation range after we filtered out the outliers and stop-word u-shapelet candidates. This result is very suggestive, telling us we should compute the expensive gap scores in the order of the lowest variance first, and that this ordering can allow us to "early" abandon the search with little chance of missing the best u-shapelet.

[1] The Distinguishing Substring Selection problem is: given a set of "*good*" strings and a set of "*bad*" strings, create a string which is, under Hamming distance, "*far*" from the good strings but "*close*" to the bad strings [4]. Note that we are tasked with *creating* a string that may not exist in the dataset, whereas u-shapelets task us with finding a subsequence from *within* the data.

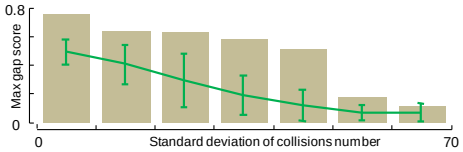

Figure 9: *gray*) Distribution of maximum values of gap score per interval. *green*) Mean and standard deviation of gap score values per interval

Figure 10 illustrates how we count the number of time series sharing a masked SAX signature with each u-shapelet candidate we are testing.

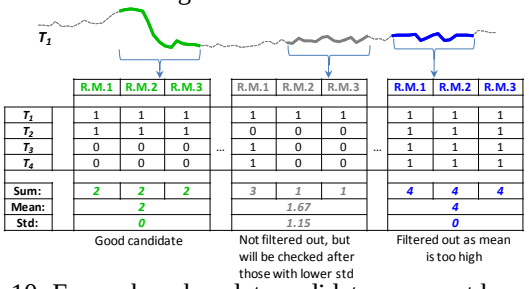

Figure 10: For each u-shapelet candidate we count how many time series share the masked SAX signature with it (number of collisions). U-shapelet candidates having a low variability of the number of collisions are very likely to be better candidates

Having sorted u-shapelet candidates as described above, we need to compute the gap score of some small fraction of first candidates. We will show empirically that it suffices to compute the gap score of less than 1% of u-shapelet candidates to find a "good enough" u-shapelet. Hashing and sorting can be performed very quickly; therefore, we can obtain a two orders of magnitude speedup.

3.2 Algorithm for U-shapelet Discovery

Given the intuition behind our approach, we are finally in a position to give a formal definition of Algorithm 1.

Algorithm 1. *GetU-shapelet*(*Data, sLen, projectionsNum*)

Input: *Data*: dataset; *sLen*: u-shapelet length
Output: *Ś*: u-shapelet

```
1:   allUsh ← GetSubsequences(Data, sLen)
2:   ushList ← ConvertToSAX(allUsh)
3:   for i ← 1 to projectionsNum
4:      randProjections ← GetRandomProjections(ushList)
5:      cCount (:, i) ← CountCollisions (randProjections)
6:   end for
7:   sMean ← mean(cCount, 1)
8:   [allUsh, cCount] ← Filter(cCount, sMean< lb | sMean> ub)
9:   order ← Sort (std(cCount, 2)), allUsh ← allUsh(order)
10   m ← size(allUsh)/fraction
11:  bsfGap← 0
12:  for i ← 1 : m
13:     gap ← ComputeGap (allUsh(i))
14:     if gap > bsfGap then Ś ← allUsh(i)
15:  end for
16:  return Ś
```

Line 1 extracts all subsequences of a given length from the time series in the dataset. In line 2 the subsequences are converted to SAX. Having obtained a SAX representation of all u-shapelet candidates, we hash their random projections (line 4) and then count the collisions on line 5. In line 8 we filter candidates that appear in too few or too many time series in the dataset under the majority of masks, as we have argued that such candidates are almost certainly *outlier* or *stop-word* u-shapelets, respectively. In line 9 the u-shapelet candidates that survived this pruning step are sorted by the standard deviation of the number of collisions (lower standard deviation first). After the u-shapelet candidates havebeen sorted we can begin to compute the gap score according to the algorithm proposed in [26]. We show the algorithm for the gap score computation in Algorithm 2. Here the algorithm is simply a direct implementation of equation (1).

Algorithm 2. *ComputeGap(s, Data, lb, ub)*

Input: *Data*: dataset; *s*: u-shapelet candidate; *lb, ub*: lower/upper bound of reasonable # of time series in cluster
Output: *gap*: gap score

```
1:   dis ← ComputeOrderline (s, Data),  gap ← 0
2:   for i ← lb to ub
3:      D_A ← dis ≤ dis(i), D_B ← dis > dis(i)
4:      if (lb ≤ | D_A | ≤ ub) then
5:         m_A ← mean(D_A), m_B ← mean(D_B)
6:         s_A ← std(D_A), s_B ← std(D_B),
7:         currGap ← m_B - s_B - (m_A + s_A)
8:      if currGap > gap then gap ← currGap
9:   end for
10   return gap
```

Up to this point we have neglected to discuss the stopping criteria for the Algorithm 1. If we allow it to exhaustively search all candidates, it will *still* be faster than a naive brute force search because the filtering step in line 8 will generally have reduced the candidate set size. The fraction of data pruned depends on the dataset, but is typically at least 50%. If we exhaustively search the remaining items, our algorithm has the flavor of an *anytime algorithm* [23], as the "best-first" heuristic sorting in line 9 gives us the desirable diminishing returns property [23]. However, for simplicity we propose to simply stop searching after we have examined 1% of the original candidate size. This gives up a two order of magnitude speedup, and as we shall empirically show, produces results that are nearly indistinguishable from an exhaustive 100% brute force search.

3.3 Justification of Our Approach for Gaining Speedup

As we explained in Section 2.1, our goal is to find a "*good enough*" u-shapelet, not necessarily the best one. We do not know in advance how many "good enough" u-shapelets can be found in the dataset, but if we assume that 0.1% of all u-shapelet candidates can be considered "good enough," then the probability of finding at least one such candidate within the first 1% of candidates checked as a function of the number of u-shapelet candidates is presented in Figure 11 as the blue/**bold** line.

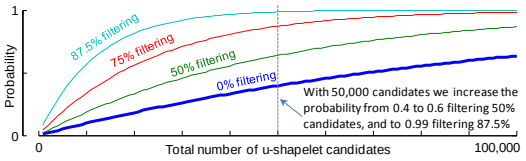

Figure 11: Probability of finding a "good enough" u-shapelet after searching 1% of candidates

Our speedup mechanism introduced in Algorithm 1 can be seen as having two related elements: discarding unpromising candidates (line 8) and sorting the remaining candidates by the most promising first (line 9). Here we consider *just* the utility of the former, as it may not be intuitive that simply discarding some unpromising candidates can produce significant speedup. In Figure 11, we show the effects of discarding 50%, 75% and 87.5% of the candidates, based on the probability of finding a good enough candidate after examining just 1% of the u-shapelet candidates. This plot shows that simply discarding a large fraction of the unpromising data makes a huge difference in the probability of early success.

3.4 Clustering with U-shapelets

We have discussed algorithms for u-shapelet search, but did not explain how to cluster the time series given that we found a good enough u-shapelet(s).

3.4.1 State-of-the-art Algorithm Overview

In the initial work on u-shapelets, Zakaria et al. [26] suggested the use of a modified k-means algorithm. At each step (i.e., with each u-shapelet discovered) they added the orderline of the u-shapelet to the collection of previously discovered u-shapelets' orderlines, and gave this collection to the k-means algorithm. Thus, the input to the k-means was not the *subsequences of the time series*, but the *distances* of each time series to each of the u-shapelets extracted. At each step they analyzed whether a newly-added u-shapelet's orderline affected the clusters assigned by the k-means. The u-shapelet extraction stopped when the clusters of time series did not change with the addition of new u-shapelets' orderlines. This approach has a drawback: it requires the explicit specification of the number of clusters *k*. We would like to provide an approach that allows the algorithm to decide how many clusters the data have (though it is possible to specify the number of clusters if this information is available).

3.4.2 Algorithm for Clustering with U-shapelets

In contrast to [26], we do not use k-means to assign time series to clusters. We iteratively split the data with each discovered u-shapelet: we consider the left part of the u-shapelet's orderline to be a separate cluster; thus, we remove the time series of the part D_A of the u-shapelet's orderline and continue a new u-shapelet search with the rest of the data. This approach is a direct implementation of the u-shapelet definition: u-shapelets separate time series in the dataset by the *sdist* to this u-shapelet. Thus, time series having a small *sdist* will form a cluster, as depicted in Figure 3. As a stopping criterion for the number of u-shapelets extracted we examine the decline of the u-shapelet gap score: we stop when the gap score of the newly-found u-shapelet becomes less than half of the gap score of the first discovered u-shapelet. In essence, this may be regarded as a form of heuristic "knee-finding" [18].

4 Experimental Evaluation

To ensure reproducibility we created a webpage with all datasets and code used in this work [19]. In addition, the site contains supplementary results and multimedia to demonstrate the utility of our ideas.

Our empirical evaluation has the following goals:

- Thus far, only a single paper has shown the utility of u-shapelet clustering [26]. Here we provide significant additional evidence of their utility.
- We demonstrate that our algorithm is two orders of magnitude faster than the current state-of-the-art, while producing results that do not differ in external evaluation metrics (i.e., the Rand index).

We take great care to be fair to rival methods. This is often difficult, since many methods in the literature are not even defined for datasets unless all time series are of the same length. Indeed, bypassing this significant limitation is one of the main advantages of u-shapelets. While [6] argues forcefully that this "equal-length" assumption is unwarranted for most real-world problems and has led to "*unwarranted optimism*" about the performance of algorithms in the related task on classification, we make the effort to consider such datasets here. In order to assess the quality of clustering with a certain u-shapelet we need to have some metric to compare clusterings to the ground truth. We use the Rand index [15], which essentially indicates what fraction of data was clustered correctly.

4.1 Speedup Evaluation

An overview of the datasets is presented in Table 1. For addition details and the data itself we refer the interested reader to the project's supporting page [19].

Table 1: Dataset Information

Dataset	Source	TS #	TS length	Classes #
Trace	[8]	200	275	4
PAMAP	[17]	345	500	7
Birds(all)	[25]	177	500	2
AMPds	[12]	400	400	2
ECG_PVC	[4]	166	144-698	2
ECG_APB	[4]	164	140-699	2
ECG_RT	[4]	126	146-700	2

We begin by comparing the original brute force algorithm with the proposed SUSh algorithm on the 7 datasets. The results are presented in Table 2. It is clear that our approach to ordering allows discovering good u-shapelet candidates much more quickly than computing all candidates' gap scores. The mean speedup is 73.4; it varies because of the properties of the datasets. If the data are noisy (as is the case with PAMAP, Birds and Trace) datasets, the random masking produces many collisions that have to be

counted. With less noisy data similar subsequences have a higher chance of sharing their SAX representation; thus, random masking produces fewer new collisions that must be counted.

Table 2: Running Time and Speedup

Dataset	Brute force time (*hours*)	SUSh time (*min*)	Speedup
Trace	1.89	1.76	64.4
PAMAP	13.65	16.23	50.5
Birds(all)	1.36	1.26	64.5
AMPds	9.37	6.89	82.9
ECG_PVC	1.68	1.37	73.6
ECG_APB	1.21	0.77	94.1
ECG_RT	0.82	0.59	83.8

In Table 3 we present the Rand index for clustering using the u-shapelet found by brute force and SUSh. It is clear that our claim that our approach produces essentially the same quality of clustering as the exhaustive brute force approach is borne out.

Table 3: Rand Index

Dataset	k-means (true clusters #)	Brute force	SUSh
Trace	0.75	1	1
PAMAP	0.75	0.92	0.92
Birds(all)	0.77	0.97	0.97
AMPds	0.66	0.78	0.78
ECG_PVC	*Not defined*	0.98	0.98
ECG_APB	*Not defined*	0.97	0.97
ECG_RT	*Not defined*	0.95	0.95

4.2 Case Study: Physical Activity Dataset

The PAMAP project (Physical Activity Monitoring for Aging People) aims to monitor physical activity of elderly people [17]. As shown in Figure 12, the data is comprised of signals from accelerometers located on the subjects as they performed different activities.

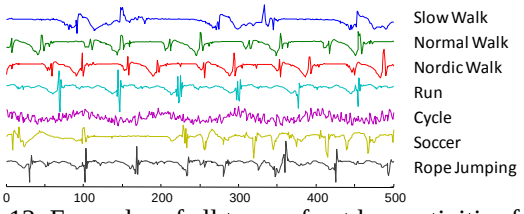

Figure 12: Examples of all types of outdoor activities from x-accelerometer located at the foot position

The subjects recorded their activities as a single performance, resulting in data that is one long multidimensional time series where each activity was *weakly* labeled. In other words, for each time frame, what the subject was supposed to do was annotated without looking at the data itself. Thus, the data may contain various *transition stages* (e.g., a part of a time series labeled "cycling" may contain regions of unlocking a bike and walking it to the street, etc.). For simplicity we considered data only from a single sensor (the x-axis accelerometer on the foot). We extracted sections of 500 data points or 5 seconds long "blindly": we did not examine the data for the most typical or noise-free regions. We considered 345 such time series of length 500 from the same subject.

U-shapelet clustering gives a Rand index of 0.92. k-means clustering, with k = 7 (the true number of clusters) gives us a Rand index of 0.75. We further considered clustering the data using several variants of the Fourier methods proposed in [7]. The *best* of these methods obtained a Rand index of 0.88, significantly less than u-shapelets. The order-dependent state-of-the-art approach *fails* to cluster the data because the first time series in the dataset mostly represents a transition stage and it is not possible to extract a u-shapelet from it [26]. If we "fix" this algorithm by making it examine *every* time series in the dataset (not just the first), then its Rand index is essentially the same as our approach, but it then takes more than 30 hours, as opposed to just 16 minutes using our method. Note that our experiments are focused on showing we can reduce the *relative* time by two orders of magnitude. Our *absolute* times may seem less impressive, but by porting to a lower-level language and availing of (orthogonal to our contributions) additional speedup techniques [14], we believe a further two orders of magnitude are obtainable. In any case, even in this case we clustered the data more quickly than it was collected (i.e., we clustered 28.7min of data in 16min).

4.3 Case Study: Bird Calls

We investigated the utility of our u-shapelet discovery algorithm for separating distinct bird songs. We considered several recordings from [25], some containing songs of the white-crowned sparrow (examples of data presented in Figure 1) and the remainder containing other miscellaneous species or background noise. As in Figure 1, we converted the sounds into MFCC space and used the 2nd coefficient. We fixed the number of time series containing white-crowned sparrow songs and added increasing amounts of the background noise dataset. To allow comparison with k-means, we made all of the time series of equal length (500 data points) and aligned sparrow song snippets by hand. The results of clustering with k-means and u-shapelets are presented in Figure 13. It can be clearly seen that the addition of spurious data hurts k-means clustering, whereas u-shapelets separate bird songs essentially with the same quality regardless of the presence of spurious data. In essence, this was the claim we made in discussing Figure 3.

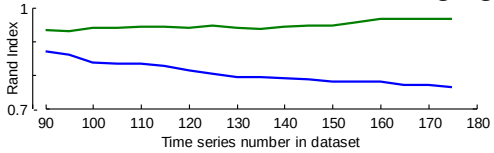

Figure 13: Rand index for k-means clustering (blue) and clustering with u-shapelets (green). Spurious data addition does not hurt the quality of clustering with u-shapelets (averaged by 10 random runs)

4.4 Comparisons to Rival Methods

Recall that this paper's contribution is making u-shapelet discovery tractable; we are not presenting a novel clustering technique per se. However, since there is currently only one work showing the utility of u-shapelets, we will take the time to compare to some existing methods. In order to be

scrupulously fair to rival methods, we compare directly to the results in the original papers, without reimplementing (and possibly "crippling") the algorithms, and we test only on data the original authors chose to showcase in their work.

In [9] the authors introduce a clustering method called CLDS (complex-valued linear dynamical systems), and note that the *"approach produces significant improvement in clustering quality, 1.5 to 5 times better than well-known competitors on real motion capture sequences."* The method is hard to summarize, involving graphical models, Kalman filters and a complex-fit EM algorithm, so we refer the interested reader to the original paper [9]. The authors demonstrate the utility of their work on the publicly available MOCAPANG-Subject-35, right-foot-marker dataset. Unlike u-shapelets, their method is not defined for time series with different lengths, so they manually trim the data to the same length. The evaluation method is based on the conditional entropy, and they manage to score 0.1015. This metric is not as intuitive as, say, the Rand index; however, suffice it to say that smaller is better, and their score is better than random guessing. In contrast, we ran u-shapelets on the same dataset, and achieved a Rand index of 1 (a perfect score), and a conditional entropy of zero. Thus, on this dataset we are more accurate than the rival method, and more general (we did not need to trim sequences).

The authors of [1] also deal with only a part of the data in the time series. They propose a clustering approach for the time series based on the similarity of only some fragments in the time series, ignoring the rest of the data. Our u-shapelet discovery algorithm was able to identify the synthetically implanted trends in their synthetic dataset and performed the correct clustering.

5 Conclusion and Future Work

Two decades ago, Timmer et al., working on one of the first practical applications of time series clustering, noted that: "*The crucial problem is not the classifier, but the selection of well-discriminating features."* [21]. In essence, this is what u-shapelets are. However, rather than use statistical features such as entropy or the zero-crossing rate, u-shapelets use *subsequences* from the data itself as the discriminating features. We have demonstrated the utility of u-shapelets in diverse datasets and proposed the techniques that make u-shapelet discovery tractable, thus allowing its application to much larger datasets. Our somewhat arbitrary "*examine only a special 1% of the data*" rule heuristic does empirically give us the same accuracy as brute-force algorithms, and in future work we hope to prove a (probabilistic) bound on performance.

Acknowledgements. We gratefully acknowledge funding from NSF IIS-1161997.

6 References

[1] G. Atluri,. *Discovering Groups of Time Series with Similar Behavior in Multiple Small Intervals of Time*, Proceedings of the 14th SIAM International Conference on Data Mining, 2014.

[2] G. Batista, X. Wang, E. Keogh, *A Complexity-Invariant Distance Measure for Time Series*, Proceedings of the 11th SIAM International Conference on Data Mining 2011, pp. 699-710

[3] D.K. Dawson, M.G. Efford, *Bird population density estimated from acoustic signals*, J. Appl. Ecol., 46 (2009), pp. 1201–09

[4] A.L. Goldberger et al. *PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals*, Circulation 101(23):e215-e220

[5] J. Gramm, J. Guo, R. Niedermeier, *On exact and approximation algorithms for Distinguishing Sub-string Selection*, Proceedings of the 14th FCT. Springer-Verlag, Berlin, August 2003, pp. 25-42

[6] B. Hu, Y. Chen, E. Keogh, *Time Series Classification under More Realistic Assumptions*, Proceedings of the 13th SIAM International Conference on Data Mining, 2013, pp. 578-586

[7] K. Kalpakis, D. Gada, V. Puttagunta, *Distance measures for effective clustering of ARIMA time-series*, Proc' of IEEE International Conference on Data Mining, 2001, pp. 273-280

[8] E. Keogh et al., The UCR Time Series Classification/Clustering Homepage: www.cs.ucr.edu/~eamonn/time_series_data/

[9] L. Li, B. Prakash, *Time Series Clustering: Complex is Simpler!* Proceedings of the 28th ICML, 2011, pp. 185-192

[10] J. Liu et. al., *uWave: Accelerometer-based personalized gesture recognition and its applications*, Pervasive and Mobile Computing 5(6), 2009, pp. 657-675

[11] J. Lin, E. Keogh, S. Lonardi, B. Chiu *A symbolic representation of time series, with implications for streaming algorithms*, Proceedings of the 8th ACM SIGMOD DMKD workshop, 2003, pp. 2-11

[12] S. Makonin et al., *AMPds: A Public Dataset for Load Disaggregation and Eco-Feedback Research*, Electrical Power and Energy Conference (EPEC), 2013 IEEE, 2013, pp. 1-6

[13] A. McGovern, D. Rosendahl, R. Brown, K. Droegemeier, *Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction*, Data Mining and Knowledge Discovery. 22 (1), 2011, pp. 232-258

[14] A. Mueen, E. Keogh, N. Young, *Logical-Shapelets: an expressive primitive for Time Series classification*, Proceedings of the 17th ACM SIGKDD, 2011, pp. 1154-1162

[15] W. Rand, *Objective criteria for the evaluation of clustering methods*, Journal of the American Statistical Association, Vol. 66, No. 336, Dec., 1971, pp. 846-850

[16] T. Rakthanmanon, E. Keogh, *Fast shapelets: A scalable algorithm for discovering time series shapelets*, Proc' of the 13th SIAM conference on Data Mining, 2013, pp. 668-676

[17] A. Reiss, M. Weber, D. Stricker, *Exploring and extending the boundaries of physical activity recognition*, IEEE SMC'11, pp. 46-50

[18] S. Salvador, P. Chan, *Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms*, Proc of 16th IEEE ICTAI 2004, pp. 576-584

[19] Supporting page: https://sites.google.com/site/ushapelet/

[20] K. Tharakaraman et al., *Alignments anchored on genomic landmarks can aid in the identification of regulatory elements*, ISMB. 2005, pp. 440-448

[21] J. Timmer, C. Gantert, G. Deuschl, J. Honerkamp, *Characteristics of hand tremor time series*, Biological cybernetics 70.1, 1993, 75-80

[22] M. Tompa, J. Buhler, *Finding motifs using random projections*, Proceedings of the 5th Int'l Conference on Computational Molecular Biology, 2001, pp. 67-74

[23] K. Ueno, X. Xi, E. Keogh, D.-J. Lee, *Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining*, Proc' of the IEEE ICDM 2006, pp. 623-632

[24] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, E. Keogh, *Experimental comparison of representation methods and distance measures for time series data*, Data Min. Knowl. Discov. 26(2): pp. 275-309 (2013)

[25] Xeno-canto bird sounds collection http://www.xeno-canto.org/

[26] J. Zakaria, A. Mueen, E. Keogh, *Clustering Time Series using Unsupervised-Shapelets*, Proceedings of the IEEE International Conference on Data Mining, 2012, pp. 785-794