Fast Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets

Thanawin Rakthanmanon
Department of Computer Engineering
Kasetsart University, Thailand
thanawin.r@ku.ac.th

Eamonn Keogh
Department of Computer Science & Engineering
University of California, Riverside, USA
eamonn@cs.ucr.edu

ABSTRACT

Time series shapelets are a recent promising concept in time series data mining. Shapelets are time series snippets that can be used to classify unlabeled time series. Shapelets not only provide interpretable results, which are useful for domain experts and developers alike, but shapelet-based classifiers have been shown by several independent research groups to have superior accuracy on many datasets. Moreover, shapelets can be seen as generalizing the *lazy* nearest neighbor classifier to an *eager* classifier. Thus, as a deployed classification tool, shapelets can be many orders of magnitude faster than any rival with comparable accuracy.

Although shapelets are a useful concept, the current literature bemoans the fact that shapelet discovery is a time-consuming task. In spite of several efforts to speed up shapelet discovery algorithms, including the use of specialist hardware, the current state-of-the-art algorithms are still intractable on large datasets. In this work, we propose a fast shapelet discovery algorithm that outperforms the current state-of-the-art by two or three orders of magnitude, while producing models with accuracy that is not perceptibly different.

1 INTRODUCTION

Shapelets are a recently introduced concept in time series data mining. In essence, shapelets are prototypical time series "snippets" that can be used to classify unlabeled time series that contain an occurrence of the shapelet within some previously learned distance threshold. The utility of shapelets has been confirmed and extended by many independent groups of researchers [1][6][9][10][13][15][16][29][30]. The exploding interest in shapelets can be attributed to the following factors. First, they generalize the *lazy* nearest neighbor algorithm, widely understood to be the state-of-the-art technique for time series [27], to an *eager* decision-tree-like classifier, allowing orders of magnitude improvement in classification time. Second, they are *interpretable*, and can give insights as to what defines the differences between two classes [30]. Finally, on some problems, shapelets can be simply more accurate than any known rival method [19][30].

Given these advantages, we have recently seen shapelets (and very similar ideas) applied to creating classifiers in various domains such as gesture recognition [2][17], sensor networks [12], motion capture [26], cardiology, climatology [18], robotics [26], electrical power demand [1][9] and health care [22]. In addition, we have begun to see several generalizations of shapelets, such as *logical shapelets* [19], which classify objects based on conjunctions/ disjunctions of shapelets, and *local shapelets* [29], which impose constraints on where valid shapelets may appear within an object.

Before continuing further, we will take time to develop the reader's intuition for shapelets. Figure 1 shows six examples of reptile skulls [3] and their time series representations. Three of them are horned lizards (*Phrynosoma coronatum*, *P. braconnieri* and *P. mcallii*), and the other three are turtles (*Elseya dentate*, *Glyptemys muhlenbergii*, and *Annemys sp*).

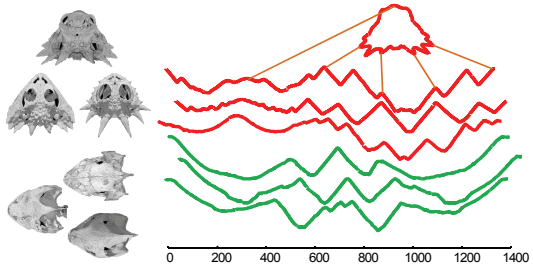

Figure 1: *left*) Skulls of horned lizards and turtles. *right*) The time series representing the images. The 2D shapes are converted to time series using the technique in [14]

For more details on how shapes are converted to time series, we refer the reader to [14][25]; however, Figure 1.*top.right* visually hints at how a shape can be "unwound" into a time series.

As we can see from Figure 1, the two classes here have a lot of intraclass variability, something that does not bode well for traditional classifiers that consider the *entire* time series [27]. Suppose instead we run the shapelet discovery algorithm on this small dataset [30]. Doing so, we find the shapelet that can best distinguish between the two types of reptiles. This shapelet is presented in Figure 2, both in the time series space and "brushed" back onto the original shape space.

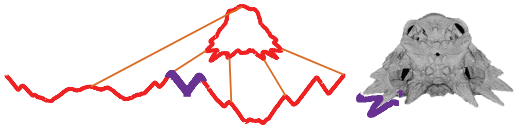

Figure 2: *left*) The shapelet that best distinguishes between skulls of horned lizards and turtles, shown as the purple/bold subsequence. *right*) The shapelet projected back onto the original 2D shape space

The discovered shapelet corresponds to two horns of the horned lizard, an intuitive and interpretable result.

This toy example demonstrates the great strength of shapelets. With zero parameters to tweak, we obtained a shapelet that is visually intuitive and allows perfect classification accuracy in this (admittedly contrived) domain. However, this example also exhibits the current weakness of shapelets it took several seconds to find the shapelet in this *tiny* dataset.

The best-known running time for the shapelet discovery algorithm is $O(n^2m^3)$, where n is the number of objects or time series in the dataset, and m is the length of the longest time series. In this work, we propose an $O(nm^2)$ algorithm for finding shapelets. Our algorithm is *heuristic*; it is not guaranteed to find the same shapelet as [19][30]. We exploit a random projection technique [24][21] on the SAX representation [14][28] to find potential shapelet candidates. However, our experimental results in Section 5 demonstrate that the classification accuracy of the proposed algorithm is not significantly different from the accuracy obtained by exact brute-force algorithms [19][30].

The rest of this paper is organized as follows. In Section 2, we introduce definitions and notations. The basic shapelet algorithm is discussed in [30] and the current state-of-the-art algorithm [19] is reviewed in Section 3. In Section 4, we explain our algorithm in detail. Section 5 demonstrates the performance and accuracy of our algorithm. Case studies showing the advantage of our proposed algorithm are described in Section 6 and we offer conclusions in Section 7.

2 DEFINITIONS AND NOTATION

We begin by introducing all necessary notation and definitions. First, we define a *time series*:

Definition 1: A *time series* T is an ordered list of numbers; $T = t_1, t_2, \ldots, t_m$. Each value t_i can be any finite number and m is the length of time series T.

A local subsection of a time series is called a *time series subsequence*:

Definition 2: A *time series subsequence* S is a contiguous sequence of a time series. Subsequence S of length l of time series T starting at position i can be written as $S = T_i^l = t_i, t_{i+1}, \ldots, t_{i+l-1}$.

For the classification task, many time series are grouped together with their corresponding class labels in a container called a *dataset*:

Definition 3: A *dataset* D is a set of pairs of time series, T_i, and its class label, c_i. Formally, $D = \langle T_1,c_1\rangle, \langle T_2,c_2\rangle, \langle T_3,c_3\rangle, \ldots, \langle T_n,c_n\rangle$. For the rest of this paper, we use n as the number of time series inside the dataset D. Note that the lengths for each time series are not necessarily equal.

To measure the similarity between subsequences, we define the *distance between two subsequences:*

Definition 4: The *distance* between subsequence S and $\hat{S}$ of the same length is the length-normalized Euclidean distance between subsequences S and $\hat{S}$. If both subsequences are Z-normalized with *mean*=0 and *std*=1, the distance is defined as:

$$dist(S, \hat{S}) = \sqrt{\frac{1}{l}\sum_{i=1}^{l}(s_i - \hat{s}_i)^2}$$

Shapelets can be of any length up to m. In order to allow meaningful comparisons between candidate shapelets of different lengths, *length normalization* must be used.

We therefore define the *distance* between a time series and a given *subsequence*:

Definition 5: The *distance* between subsequence S of length l and time series T is defined as the minimum distance between subsequence S and any subsequence of T of the same length as subsequence S. Formally, $dist(S, T) = min_i\ dist(S, T_i^l)$.

Suppose that dataset D contains n time series from c different classes. The number of time series in class i is n_i and we define class probability as $p_i = n_i / n$. Hence, we define the *entropy* of the dataset as:

Definition 6: The *entropy* of the dataset D is defined as $E(D) = \sum_{i=1}^{c} p_i log(p_i)$.

To divide the dataset into two smaller datasets, we define a *split*:

Definition 7: A *split* is a tuple $\langle s,d\rangle$ of a subsequence s and distance threshold d which can separate the dataset into two smaller datasets, D_L and D_R. The number of time series in D_L and D_R are n_L and n_R, respectively.

We next define the *information gain* of the given split:

Definition 8: The information gain of a split sp is:

$$I(sp) = E(D) - \frac{n_L}{n}E(D_L) - \frac{n_R}{n}E(D_R)$$

The distance between two different sides of the given split is a *separation gap*:

Definition 9: A *separation gap of* a split sp is:

$$gap(sp) = \frac{1}{n_L}\sum_{t_L \in D_L} dist(s, t_L) - \frac{1}{n_R}\sum_{t_R \in D_R} dist(s, t_R)$$

The definition of *shapelet* will be explained briefly here; however, for a more complete definition of *shapelet*, please refer to [30].

Definition 10: A *shapelet* is a split that separates the dataset into two smaller datasets with the maximum information gain; ties are broken by maximizing the separation gap.

We visually summarize the concept of shapelets with our toy example shown in Figure 3. A candidate (e.g., the highlighted subsequence from Figure 2.*left*) is shown in the box for reference. The distances between the candidate subsequence (i.e., tentative shapelet) and all time series are calculated; all corresponding objects are placed on the *orderline* according to the calculated distance. In Figure 3, three skulls from horned lizards, whose distances to the shapelet are small, are shown as red rectangles on the left-hand side of the orderline. In contrast, the distances between three time series of turtle skulls and the candidate subsequence are shown as green triangles on the right-hand side of the orderline, because their distances to the candidate shapelet are larger. Note that, in this example, the candidate shapelet corresponds to the horns of a lizard (cf. Figure 2).

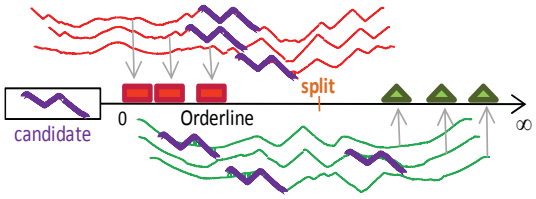

Figure 3: The *orderline* shows the distances between the candidate subsequence and all time series as positions on the x-axis. The three objects on the left-hand side of the line correspond to horned lizards and the three objects on the right correspond to turtles

After the orderline is created, we can calculate the split point, the separation gap, and the information gain of the candidate shapelet. When all candidates have been processed in this manner, the best one will be reported as the final shapelet [19][30].

3 RELATED AND BACKGROUND WORK

Our work extends the original work by Ye which introduced the concept of shapelets and showed an algorithm to allow their discovery [30]. However the general intuition behind shapelets, the idea of using small sub-patterns to identify the class of a larger object, is known in other domains (especially. bioinformatics), including *class prototypes* [6], *discriminative patterns* [2][5][8], and *predictive motifs* [18], etc.

In the next section, we briefly consider the current state-of-the-art shapelet discovery algorithm.

3.1 Brute Force Shapelet Discovery

The brute force shapelet discovery algorithm shown in Table 1 is a simple algorithm that generates and tests all possible candidates and returns the best one.

Table 1. Brute Force Algorithm

Algorithm: *BruteForceShapelet*

Input: *D* : Dataset containing time series and class labels
Output: *shapelet*: the final shapelet

```
1   [TS,Label] = ReadData(D)
2   bsf_gain = 0
3   for len = 1 to m
4      Candidates=GenerateAllCandidates(TS,len)
5      for each cand in Candidates
6         create a split sp from cand
7         gain = ComputeInfoGain(D,sp)
8         if (gain > bsf_gain)
9            bsf_gain = gain
10           shapelet = s
11        end if
12     end for
13  end for
```

The dataset *D* is a list of pairs of time series with their labels; thus, in line 1, we extract the time series and label from *D*. Because the final shapelet can be of any length, all subsequences of every length in the dataset will be generated as candidates in line 4.

Thereafter in line 6, each candidate is used to compute a split, *sp*, as shown in Figure 3 in the previous section. Finally, the information gain is computed in line 7. The returned shapelet is the candidate with maximum information gain.

If n is the number of time series in the dataset and m is the length of the longest time series in the dataset, the number of candidates in a time series is $O(m^2)$ and the total number of all candidates in the dataset is $O(nm^2)$. A distance computation from one candidate to all time series takes time $O(nm^2)$ to compute. Hence, the total running time of the brute force algorithm is $O(n^2m^4)$.

3.2 Current State-of-the-Art Algorithm

The first improvement of the brute force algorithm was introduced in [30]. They proposed a technique to calculate a cheap-to-compute upper bound of the information gain and use it to admissibly prune some candidates.

The current state-of-the-art algorithm is given at [19]. This algorithm can also find the exact shapelet [30], but does so more quickly. The speedup comes from using a classic pruning technique — triangular inequality — to prune some candidates, and from caching some tricks. The latter idea trades speedup for memory, and may run into space problems for large datasets.

Using the same notation as above, the worst-case running time of the current state-of-the-art is $O(n^2m^3)$ and the algorithm requires a memory footprint as large as $O(nm^2)$.

To concretely ground this analysis, on the *Wafer dataset* [11], with n=1000 and m=152, the state-of-the-art takes more than 12 hours to find the shapelet. However, the algorithm we will introduce in the next section can find essentially the same shapelet in just 23 seconds.

4 FAST SHAPELET DISCOVERY

We are finally in a position to explain our algorithm in detail. First, we describe the key ideas using our reptile toy example; then, we give formal details in Section 4.2.

4.1 Overview of the Algorithm

We propose to solve the shapelet discovery problem with a *change of representation*. In particular, we will transform the raw *real-valued* and *high-dimensional* data into a *discrete* and *low-dimensional* representation. Searching over a smaller representation is obviously more efficient; more importantly, however, having a discrete representation will allow us to *hash* our data, and use the collision history to inform our search.

4.1.1 Generating SAX Words

For each object in the dataset, we transform the time series into a symbolic representation using Symbolic Aggregate approXimation (SAX) [14][28]. For brevity, we assume the reader is familiar with SAX, and refer the interested reader to [14] for additional details.

From our toy example in Section 1, a time series from *P.coronatum* is shown once again in Figure 4. The top part of the figure shows an example of a SAX word **adbacc**, created by the first subsequence. Multiple SAX words will be generated for a given time series using the sliding window technique [14]. SAX has two parameters, which are desired (reduced) dimensionality, d, and cardinality, c. Although some techniques (e.g., [7]) could be applied for setting these parameters, for simplicity, in our implementation we set the cardinality

to 4 and word length to 16 so we can represent a SAX word with a simple 4-byte integer.

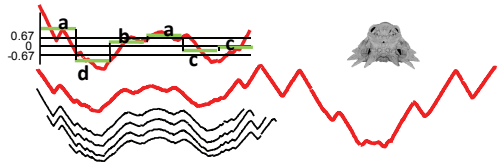

Figure 4: *top.left*) The SAX word adbacc created from a subsequence of the time series corresponding to *P. coronatum*. *bottom*) The sliding window technique

4.1.2 Random Masking

It is very important to recall that a *single* time series creates *multiple* SAX words, which correspond to the multiple subsequences we can obtain as we slide the shorter subsequence length across the longer time series. This is hinted at in Figure 4, and shown explicitly in Figure 5.*left*, where each time series (derived from a skull) creates two or three SAX words of length five.

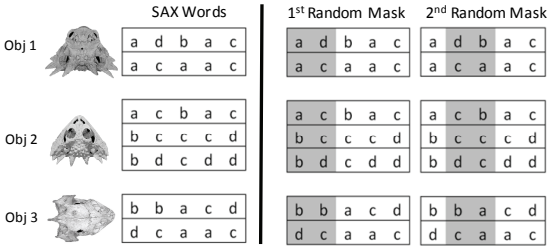

Figure 5: *left*) SAX words for each object. *right*) SAX words after masking two symbols. Note that the masking positions are randomly picked

Having created the SAX representation of our data, we have an *apparent* solution to the shapelet discovery problem. We could conduct a brute force search for the shapelets in the SAX space. This would require only slight modifications of the algorithm shown in Table 1, and because of the reduced dimensionality of SAX it would be faster than working with the raw data. However, there are two problems with this idea. The search would be faster, but still quadratic in the number of SAX words. More importantly, we have the problem of *false dismissals*.

The problem of false dismissals is caused by the fact that two time series that differ only by a tiny epsilon could produce two different SAX words[1]. Thus, it is possible that the best shapelet in the raw data spaces maps to slightly different SAX words, such as the SAX words **adbac** and **acbac**, created by the lizard skulls in Figure 5.*left*.

The solution to this problem is to exploit *random projection*, a mature idea from bioinformatics [24]. The idea is to project all SAX words of high dimensionality to a smaller dimensionality. This is illustrated in Figure 5.*right*, where all SAX words of dimensionality five have been randomly masked at two positions, reducing the dimensionally to three. Note that the first such random projection *does* take our two different SAX words, **adbac** and **acbac**, and makes them identical, **ad**bac and **ac**bac.

The reader will appreciate two potential problems with the idea of random projection. In our toy example we *contrived* our "random" choice of a mask, but we cannot generally be sure that a single projection helps us. The second problem is that if we mask too many locations, our decrease in the likelihood of false *dismissals* comes at the cost of an explosion of false *positives*, all of which must be checked. Again, we can turn to the bioinformatics literature for the answer [24]. As hinted at by the multiple masks in Figure 5.*right*, if we mask conservatively, but do multiple random masks, we can make the probability of false dismissals arbitrarily low while not incurring a measurable increase in false positives [24]. The remainder of Section 4 makes these ideas more concrete.

4.1.3 Counting Similar Objects

To avoid all-to-all distance computations, we apply hashing (i.e., random masking) on all of our data objects. The intuition is that two objects that are similar in the original space have a very high probability of collisions, even if they happen to have been mapped to slightly different SAX words.

Figure 6.A shows that, after hashing, the SAX words **adbac** and **acbac** share the same signature, ****bac**. All SAX words that have the signature **bac** have their counters incremented in the relevant table shown in Figure 6.A.*right*. Similarly, in the second iteration the words **adbac** and **acbac** once again randomly hash to the same word, this time **a**ac**.

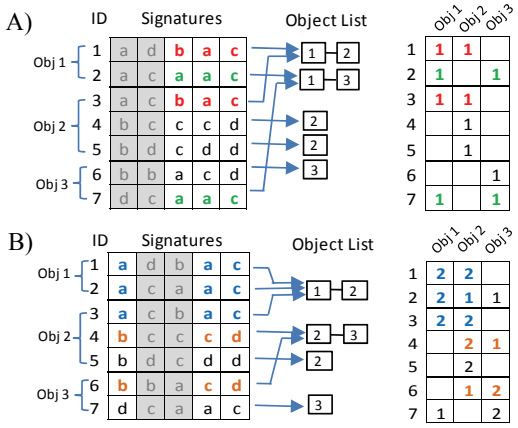

Figure 6: The first (A) and second (B) iterations of the counting process. *left*) Hashing process to match all same signatures. Signatures created by removing marked symbols from SAX words. *right*) Collision tables showing the number of matched objects by each words

Thus, after *r* iterations of random projection, we expect the collision table shown in Figure 6.A.*right* to remain mostly sparse, but to contain some locations that have values that are a significant fraction of *r*. As we shall

[1] This is of course true for *any* discretization method.

show in the next section, this information can guide our shapelet search.

4.1.4 *Finding the Best Candidates*

Continuing with our toy example, let us assume that we have done random projections for five iterations and the collision table is shown as in Figure 7.A.

As shown in Figure 7.B, we can condense the collision table by summing all of the object-based counts to the class-based counts, and create the complementary data structure in Figure 7.C. From these two tables we can calculate the *distinguishing power* of each SAX word using the simple equation shown in Figure 7.D. Note that the *distinguishing power* is high if the reference words appear frequently in one class but rarely in another class.

In this example, the highest score is from *word1* because it is close to objects in *class1* 10 times (*obj1* 5 times and *obj2* 5 times) and far from objects in *class2* 10 times; hence, its distinguishing power is 10+10 = 20. In contrast, SAX *word5* receives a power score of zero because this reference word is similar to objects from *class1* 5 times but also far from objects in *class1* 5 times, and has the same distribution in *class2*; hence, the score is (5-5)+(5-5) = 0. This suggests a pattern that is equally frequent in both classes, rather like a "stop-word" in text classification.

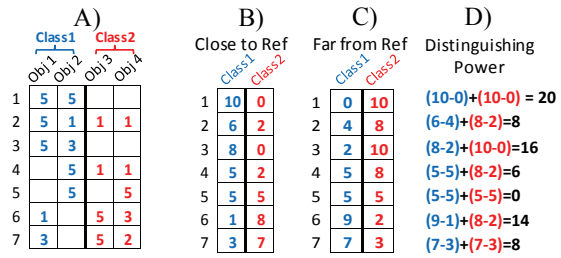

Figure 7: A) The collision table for all words after five iterations. The scores in the table represent the number of times an objects shares the same signature with the reference words. B) Grouping scores from objects in the same class. C) Complement of (B) to show that the number of times objects in each class do not share the same signature with the reference word. D) The distinguishing power of each SAX word

This list of SAX words with high distinguishing power is *almost* a solution to our problem, as it very highly correlates with the quality of the corresponding shapelets (i.e., their *information gain*) in the original raw data space. Empirically we can be certain that the best shapelet is near the top of this list. However, we do need to spend some time searching the top candidates in the original space to confirm we have a high-quality shapelet.

This is the complete intuition behind our algorithm. In the next section, we formalize these ideas.

4.2 Fast Shapelet Algorithm

Our shapelet discovery algorithm is shown in Table 2. In line 1, we extract all time series with their class labels from the current dataset D. Note that the dataset D will be iteratively made smaller as we descend deeper into the decision tree.

Table 2. Fast Shapelet Algorithm

Algorithm: *FastShapelet*

Input: D : Dataset containing time series and class labels
r : number of random iterations
k : number of SAX candidates

Output: *shapelet*: the final shapelet

```
1   [TS,Label] = ReadData(D)
2   for len = 1 to m
3      SAXList = CreateSAXList(TS,len)
4      Score = {}
5      for i = 1 to r
6         Count = RandProjection(SAXList,TS)
7         Score = UpdateScore(Score,Count)
8      end for
9      SAXCand = FindTopKSAX(SList,Score,k,r)
10     TSCand = Remap(SAXCand,TS)
11
12     max_gain=inf, min_gap=0
13     for i = 1 to |TSCand|
14        cand = TSCand[i]
15        DList = NearestNeighbor(TS,cand)
16        [gain,gap] = CalInfoGain(DList)
17        if (gain>max_gain) ||
18           ((gain==max_gain)&&(gap>min_gap))
19              max_gain = gain
20              min_gap = gap
21              shapelet = cand
22        end if
23     end for
24  end for
```

The process is split into two phases. In the first phase (lines 3-10), we select potential subsequences after a search in the SAX space (lines 12-23); this is the process we informally discussed in detail in the previous section. In the second phase, we measure the quality of those potential candidates in the raw data space and return the best candidate as the final shapelet.

To select the candidates, all subsequences of length `len` from all time series are created using the sliding window technique, and we create their corresponding SAX word and keep them in `SAXList` (line 3/Figure 5.*left*). After the list of SAX words has been created, we use these discrete representations to do hashing with `RandProjection()` by creating a hash signature of each SAX word, and give one count for each SAX word based on its signature. Then, we update the total score from multiple iterations (line 7/Figure 6).

Next, each SAX word is given a score to show how many times each word occurs in each object. We then calculate a distinguishing power for each SAX word, and pick the top k subsequences that have the highest score (line 9/Figure 7). We remapped these SAX words back to their original raw data subsequences (line 10).

Note that we have two parameters, r and k, here. However, according to our experiments in Section 5, our algorithm is not sensitive to them; thus, we simply fix $r = 10$ and $k = 10$.

We are now ready for the second phase (lines 12-23); we calculate the information gain for each candidate in the top k list and pick the best one as the shapelet. More specifically, each candidate is considered one at a time (line 13). The body of the loop calculates the distance between the candidate subsequences and each time series using the equation in Definition 5. After

these calculations (lines 17-21), we pick the subsequence which has the highest information gain as the final shapelet, breaking ties (if any) by the maximum gap (cf. Definition 9).

This explanation is necessarily terse. We refer the interested reader to [31] for a more detailed line-by-line explanation and the original (annotated) source code.

5 EXPERIMENTAL RESULTS

We begin by noting that *all* the code and data used in this work are available on our supporting webpage [31], in addition to numerous additional experiments.

5.1 UCR Time Series Dataset

We compared our algorithm with the current state-of-the-art [19]. For fairness, we used the code provided by the original authors and set the parameters as they had recommended on their supporting webpage.

We begin by considering the accuracy on 32 datasets from the UCR Time Series archives [11] in Figure 8. Note that we attempted tests on all *45* datasets from the UCR achieves; however, we abandoned the 13 experiments in which the state-of-the-art algorithm [19] had not finished after 24 hours.

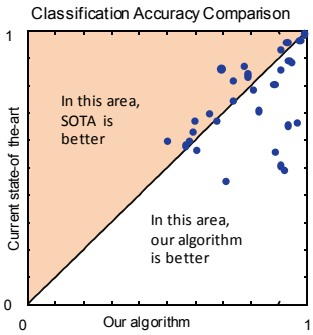

Figure 8: Classification accuracy of our algorithm and the state-of-the-art on 32 datasets from the UCR archives

Visually it is difficult to say which algorithm is better, and counting wins, ties, and losses produces similarly ambivalent results. However, the results are *strongly consistent* with our claim that our method is no worse than the state-of-the-art. The only real difference between two algorithms is scalability, as the time comparison shown in Figure 9 illustrates.

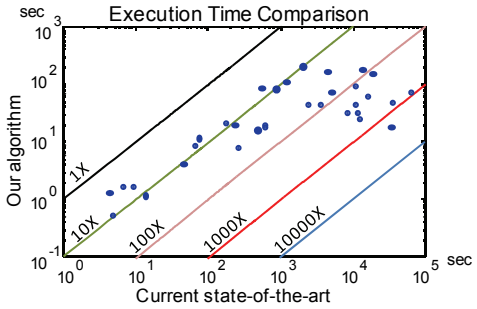

Figure 9: Running time comparison between our algorithm and the state-of-the-art on 32 datasets from the UCR time series archives

The greatest speedup we achieve on these 32 datasets is 2,030X, and we gain a speedup exceeding 100X for at least 12 other datasets. On the *Wafer* dataset, although we gain the most speedup with 2,030X, our method still achieves very high accuracy at 99.64%, which appears to be the best known result on this dataset [11][27]. In the next section we explore the issues affecting scalability in more detail.

5.2 Scalability

To compare the scalability of our algorithm and the current state-of-the-art in more detail, we tested on the largest time series dataset in the UCR time series archives, the *StarlightCurves* dataset. For *all* shapelet discovery algorithms, there are two factors that strongly determine the difficulty of the search, the number of time series in the dataset and the length of the time series; below we varied each one independently.

Figure 10.*left* shows the result when the number of time series, n, is varied from 50 to 800 and the length of all time series, m, is fixed at 100. Figure 10.*right* shows the result when n is fixed at 100 and m is varied from 50 to 800. Note that the maximum size of the experiments we consider here are constrained by the implementation of the state-of-the-art algorithm that we received from the original authors. For any experiments larger than the one below we get an *out-of-memory* error.

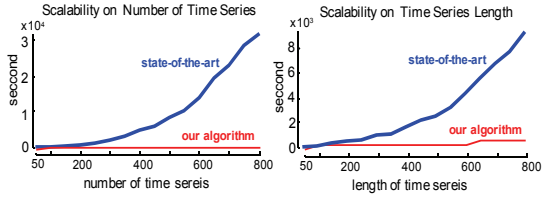

Figure 10: Scalability of our algorithm and the current state-of-the-art on *StarlightCurves* dataset. *left*) The number of time series in the dataset is varied. *right*) The length of time series is varied

The running time of the current state-of-the-art in Figure 10.*left* increases from 16 seconds to 8.7 hours from n=50 to n=800 (m is fixed at 100). However, our algorithm is significantly faster; the running time is 0.76 seconds at the beginning and less than 16 seconds when n=800. Thus, the speedup factor when n=800 is 1,970X. Figure 10.*right* shows that our algorithm achieves similar speedups when m is increased.

These empirical results are not surprising given the time complexity analyses of the algorithms. Recall that the worst-case running time of the current-state-of-the-art is $O(n^2m^3)$ and, in the best case if the triangle inequality can prune all candidates, the running time can be as low as $O(n^2m^2)$. However, our algorithm is just $O(nm^2)$.

We omit a detailed *space* complexity analysis for brevity, except to note that here we are better by an even greater margin.

5.3 When to Use Shapelets or 1NN

One of the classic questions faced by all data miners is which algorithm to use for a given task. It is well documented in the literature that the classification performance of shapelets is highly variable in the sense

that it can be significantly *better*, or significantly *worse* than the only other high-performing time series classification method, the nearest neighbor algorithm [9][10][15][30][30].

Many research papers "solve" this problem, or at least bypass it, by only reporting the holdout error. However, this ignores the question of whether we would have known ahead of time which algorithm to use? We answer *this* question below.

To answer this question, we tested on all 45 datasets from the UCR archives using a method recommended by Salzberg [23]. We split the train data from all 45 datasets in [11] to two equal parts, called **A** and **B**; the test data from the archives is preserved as it is and used as unseen data; we call this test data **C**.

For both algorithms, first **A** is used as the training data for creating a classification model, and then **B** is used as the test data for measuring the accuracy of the model, which is created from **A**. Then, we swap the roles of **A** and **B** (i.e., 2-cross validation). The *expected accuracy* is the average between these two models.

To compare two classifiers: shapelet and the Euclidean distance one nearest neighbor (1NN), we define an *expected ratio* by:

$$Expected\ Ratio = \frac{Expected\ Accuracy\ of\ Shapelets}{Expected\ Accuracy\ of\ 1\text{NN}}$$

The *actual accuracy* is measured by the accuracy of the model created by the combination of **A** and **B** (the original training data) on the test data **C** (the original test data). Similar to the expected ratio, the *actual ratio* is defined by:

$$Actual\ Ratio = \frac{Actual\ Accuracy\ of\ Shapelets}{Actual\ Accuracy\ of\ 1\text{NN}}$$

If a ratio is larger than one, the accuracy of shapelets is higher than the accuracy of 1NN. The plot in Figure 11 shows the comparison between the expected ratios and actual ratios of shapelets and the 1NN algorithm.

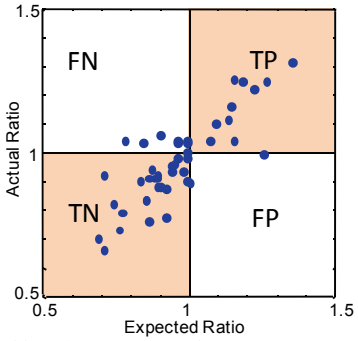

Figure 11: Accuracy ratio between *FastShapelet* algorithm and Euclidean distance-based one nearest neighbor on 45 datasets from UCR archives

The TP (True Positive) area contains the datasets in which we expected that the shapelet algorithm would be better, and it *was* better.

Likewise, the TN (True Negative) area contains the datasets in which we expected that the shapelet algorithm would be worse, and it *was* worse. Gratifyingly, most datasets fall into either TP or TN, and even remain close to the diagonal. In other words, we can generally correctly predict when shapelets are going to be useful for a particular problem.

For the handful of other datasets, the eight points in the area labeled FN (False Negative) represent a lost opportunity. We would have been slightly better off using shapelets over 1NN, but our cross validation did not realize that. The single point (just barely) inside FP (False Positive) represents the sole occasion where we expected shapelets to do well, but found we would have been (a tiny bit) better off with 1NN.

5.4 Parameter Effects

Our algorithm has two parameters that must be set by the user. They are r, the number of iterations of random projections, and k, the size of the set of potential candidates that are "promoted" from the SAX space back to the raw data space to be tested (lines 10 of Table 2). Intuitively, when the number of iterations of random projections is increased, the process should make the set of potential candidates more tolerant to noise and reduce over-fitting. Likewise, when the set of potential candidates is larger, the quality of the final shapelet should be better because we pick from a larger set. These two parameters also (approximately) linearly affect the running time of our algorithm.

As we can show empirically, these two parameters are not sensitive in terms of the accuracy produced.

On the *StarlightCurves* dataset, when r is varied from 1 to 50 and k is fixed at 10, the running time increases from 1,600 to 2,100 seconds and the accuracy changes only in a narrow range of between 93.29% and 94.37%. When r is fixed at 10 and k is varied from 1 to 50, the running time is increased from 380 to 4,900 seconds; however, once again the accuracy of the shapelet model only ranges between 93.35% and 94.30%.

The effects of parameters r and k on all datasets from the UCR archives are shown in Figure 12. In this experiment, we vary one of the two parameters r and k from 1 to 50 and fix another one to 10, and we run the experiments 30 times on all 45 datasets. Although the running time is increased linearly by both parameters, the accuracy is not sensitive to the value of parameters, as shown in Figure 12.*top*.

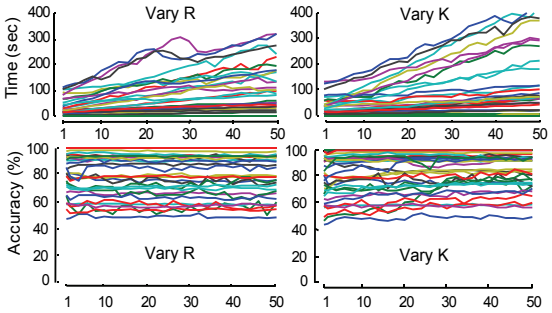

Figure 12: *bottom*) The accuracy of the algorithm is not sensitive to both parameters r and k. *top*) The running time of the algorithm is approximately linear by either parameter. Note that when we vary r (k), we fix k (r) to ten; thus, we are changing only one parameter at a time

6 CASE STUDIES

This section will demonstrate that shapelets are useful in several real-world applications.

6.1 Starlight Dataset

The *StarlightCurve* dataset is the largest dataset in the UCR time series archives. It contains 9,236 starlight curve time series of length 1,024. Three types of star objects are *Eclipsed Binaries* (*EB*), with 2,580 objects, *Cepheids* (*Cep*), with 1,329 objects, and *RR Lyrae Variables* (*RR*), with 5,236 objects. The dataset is divided into training data and test data of size 1,000 and 8,236 objects, respectively. Examples of starlight curves in each class are shown in Figure 13.

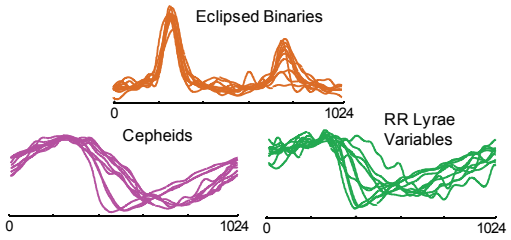

Figure 13: Examples of starlight curves in three classes: *Eclipsed Binaries*, *Cepheids*, and *RR Lyrae Variables*

Because the objects in *Cep* and *RR* are globally similar, these objects are difficult to separate. The accuracy of the one nearest neighbor algorithm using the Euclidean distance and DTW is 84.9% and 90.5%, respectively. However, using a shapelet decision tree for classification, our *FastShapelet* achieves an average accuracy of 93.68% from 30 runs. Figure 14 shows a decision tree with three leaf nodes.

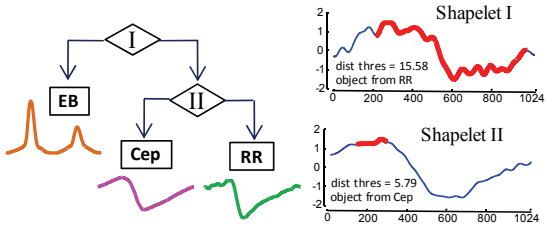

Figure 14: *left*) Decision tree of *StarlightCurve* dataset created by our algorithm. *right*) Two shapelets shown as the red/bold part in the time series

To the best of our knowledge, this is the highest accuracy ever reported on this dataset [4][11]. Moreover, of the hundred-plus papers that cited the UCR archive in the last three years (when this particular dataset was added to the archives) a significant fraction of them do not report *any* results for this dataset, because their algorithm ran out of memory or time.

By interpolation, the current state-of-the-art is expected to take 4.5 months; however, our fast shapelet algorithm can create the decision tree in Figure 14 in 3,150 seconds (just under an hour). Thus, the speedup is more than 3,000X on this dataset.

6.2 Physical Activity Dataset

This section demonstrates that shapelets also can be used as a high accuracy classification tool for activity recognition, an area drawing increasing attention due to the increasing availably of inexpensive sensors. The dataset considered is from the Physical Activity Monitoring for Aging People (PAMAP) [20]. The entire dataset contains 43 time series which are collected from multiple sensors (e.g., accelerometers, gyroscopes, magnetometers, etc.) on various parts of the body (i.e., the hand, chest, and foot). The data consists of eight subjects (seven males, one female) taking part in various sporting activities.

For simplicity we choose to use only one sensor among all 43 sensors of the original dataset to perform activity classification. Figure 15 shows examples of time series (from by z-accelerometer at hand position) of all seven outdoor activities.

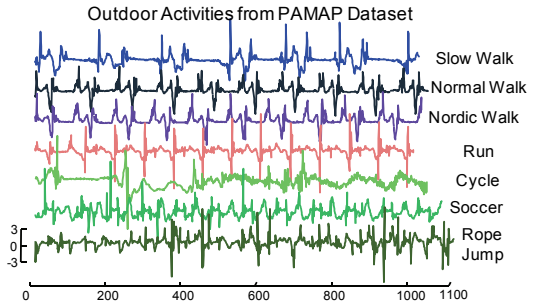

Figure 15: Examples of all outdoor activities from PAMAP dataset. Note that the time series of each activity are generally different lengths

In the original PAMAP dataset, all subjects perform all activities in one long performance so the data is a long time series that contains all (annotated) activities in one sequence. We preprocessed the data using the sliding window technique, as recommended by the original authors [22].

The accuracy of one nearest neighbor classifiers on this dataset using the Euclidean distance, DTW with 5% band size, and DTW with 10% band size are 61.16%, 81.73% and 82.03%, respectively. However, in the same task, using our shapelet algorithm we achieve an accuracy of 88.70%, which outperforms the one nearest neighbor using either the Euclidean distance or DTW.

The original authors of the dataset also created classifiers by using multiple time series [22] from *all* 43 sensors. Beyond the fact that they used all 43 sensors and we only use one, we cannot directly reproduce their experiments because of their lack of explicitness about how the data was processed. Nevertheless, they report that their specialist algorithms can obtain the highest accuracy of around 80% to 90%. Thus, domain-agnostic shapelets using a single time series can be at least competitive with highly tuned, domain-informed specialist methods using all 43 time series.

6.3 ECG Dataset

The ECG Five Days dataset from PhysioNet.Org [20] is a long ECG time series recorded on two different days with the same patient; a copy of this dataset can also be found at [11]. The dataset contains 890 objects, with 23 objects as training data and 861 as test data. Examples of the time series are shown in Figure 16. The main challenges in classifying this dataset are that the data exhibits linear drift (in medical domains, this is called *wandering baseline*), as shown in Figure 16.*top*, and the time series from two different classes are very similar, at least *globally*.

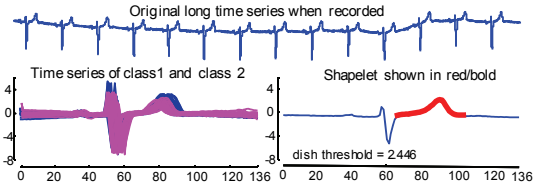

Figure 16: *top*) ECG time series when first recorded. *left*) Time series from the two classes are very similar, and it is even hard to distinguish between them with the naked eyes. *right*) The shaplet discovered by our algorithm shown in red/bold

Using one nearest neighbor classification with either Euclidean distance or DTW on this dataset, the accuracy is only 79.7%. However, the shapelet discovered by our proposed algorithm shown in Figure 16.*right* is able to obtain 99.4% accuracy from this dataset. This is, by a large margin, the best result ever reported for this dataset [4][11].

Moreover, the results are quite intuitive, according to USC cardiologist Helga Van Herle, focusing our attention on the delayed t-wave is the only medically significant difference between the two classes, and our algorithm can discover this part as the shapelet.

7 CONCLUSIONS

We proposed an algorithm for shapelet discovery that is up to three orders of magnitudes faster than the current state-of-the-art and yet has accuracy that does not significantly differ. We have made all our code freely available at [31], and as such hope to expand the scope of problems to which shapelets can be applied.

8 ACKNOWLEDGEMENT

This research was funded by NSF awards 0803410, 0808770, and 1161997, and the Royal Thai Scholarship.

9 REFERENCES

[1] K.W. Chang, B. Deka, W. M. H. Hwu, and D. Roth. Efficient Pattern-Based Time Series Classification on GPU *ICDM*. 2012.

[2] H. Cheng, X. Yan, J. Han and P. S. Yu. Direct Discriminative Pattern Mining for Effective Classification. Data Engineering, *ICDE*, 2008, 169-178.

[3] Digital Morphology 2012: http://www.digimorph.org/

[4] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1, 2, 1542-52.

[5] G. D. Fatta, S. Leue, and E. Stegantova. Discriminative pattern mining in software fault detection, *ACM SOQUA*, 2006, 62-6.

[6] B. Hartmann, Ingo Schwab, Norbert Link: Prototype Optimization for Temporarily and Spatially Distorted Time Series. *AAAI Spring Symposium: It's All in the Timing*, 2010.

[7] B. Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi, and E. Keogh: Discovering the Intrinsic Cardinality and Dimensionality of Time Series using MDL. *ICDM*, 2011.

[8] P. Geurts. Pattern Extraction for Time Series Classification. *In Proc of the 5th PKDD*, 2001, 115–127.

[9] D. Gordon, D. Hendler, and L. Rokach. Fast Randomized Model Generation for Shapelet-Based Time Series Classification. arXiv:1209.5038. 23 Sep 2012

[10] Q. He, Z. Dong, F. Zhuang, and Z. Shi. Fast Time Series Classification Based on Infrequent Shapelets. *ICMLA*. 2012.

[11] E. Keogh, Q. Zhu, B. Hu, Y. Hao, X. Xi, L. Wei, C. Ratanamahatana, The UCR Time Series Classification/ Clustering Homepage 2012: *www.cs.ucr.edu/~eamonn/ time_series_data*

[12] M. H. Ko, G. West, S. Venkatesh, and M. Kumar. Online context recognition in multisensor systems using dynamic time warping. *In Intelligent Sensors, Sensor Networks and Information Processing Conference*, 2005, 283 – 288.

[13] Z. Li, C. X. Lin, B. Ding, J. Han. Mining Significant Time Intervals for Relationship Detection. *SSTD*, 2011, 386-403.

[14] J. Lin, Eamonn J. Keogh, Li Wei, Stefano Lonardi: Experiencing SAX: a novel symbolic representation of time series. *DMKD*, *15*, 2, 2007, 107-144.

[15] J. Lines and A. Bagnall. Alternative Quality Measures for Time Series Shapelets, *IDEAL*, 2012, 475-483.

[16] J. Lines, L. M. Davis, J. Hills, and A. Bagnall. A shapelet transform for time series classification, *KDD*, 2012, 289-297.

[17] J. Liu, L. Zhong, J. Wickramasuriya, and V. Vasudevan. uWave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 2009.

[18] A. McGovern, D. Rosendahl, R. Brown, and K. Droegemeier. Identifying predictive multi-dimensional time series motifs: an application to severe weather prediction. *DMKD*, *22*, 2011.

[19] A. Mueen, Eamonn J. Keogh, Neal Young: Logical-shapelets: an expressive primitive for time series classification. *KDD*, 2011, 1154-1162.

[20] Physical Activity Monitoring for Aging People: http://www.pamap.org

[21] T. Rakthanmanon, Q. Zhu, and E. Keogh: Mining Historical Archives for Near-Duplicate Figures. *ICDM*, 2011, 557-566.

[22] A. Reiss, M. Weber and D. Stricker. Exploring and Extending the Boundaries of Physical Activity Recognition. *IEEE SMC Workshop on Robust Machine Learning Techniques for Human Activity Recognition*, 2011.

[23] S. L. Salzberg, On comparing classifiers: Pitfalls to avoid and a recommended approach. *DMKD*, *1*, 1997, 317–328.

[24] M. Tompa. & J Buhler (2001). Finding motifs using random projections. In proceedings of the 5th Int'l Conference on Computational Molecular Biology. pp 67-74.

[25] P. J. Van Otterloo. *A contour-oriented approach to shape analysis*. Prentice-Hall NJ, 1991, 90-108.

[26] D. Vail and M. Veloso. Learning from accelerometer data on a legged robot. *In Proc of the 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, 2004.

[27] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P.Scheuermann, and E. Keogh, Experimental comparison of representation methods and distance measures for time series data. to appear in *DMDK*. 2012. (Online preview)

[28] L. Wei, Eamonn J. Keogh, Xiaopeng Xi: SAXually Explicit Images: Finding Unusual Shapes. *ICDM*, 2006, 711-720.

[29] Z. Xing, J. Pei, P. S. Yu, K. Wang. Extracting Interpretable Features for Early Classification on Time Series. *SDM*. 2011, 247-258.

[30] L. Ye, Eamonn J. Keogh: Time series shapelets: a novel technique that allows accurate, interpretable and fast classification. *DMKD*, *22*, 1-2, 2011, 149-182.

[31] Supporting Webpage: *www.cs.ucr.edu/~rakthant/FastShapelet*