# Evaluating Improvements to the Shapelet Transform

Aaron Bostrom, Anthony Bagnall and Jason Lines

University of East Anglia
Norwich, Norfolk
United Kingdom

## ABSTRACT

The Shapelet tree algorithm was proposed in 2009 as a novel way to find phase independent subsequences which could be used for time series classification. The shapelet discovery algorithm is O($n^2m^4$), where n is the number of cases, and m is the length of the series. Several methods have sought to increase the speed of finding shapelets. The ShapeletTransform reduces the finding to a single pass, and FastShapelets smooths and reduces the series lengths through PAA and SAX. However neither of these techniques can enumerate all shapelets on the largest of the datasets present in the UCR repository. We first evaluate whether the FastShapelet algorithm is better as a transform, and secondly provide a contract classifier for the shapelet transform, by calculating the number of fundamental operations we can estimate the run time of the algorithm, and sample the data to fulfil this contract. We found that whilst the FastShapeletTransform does drastically reduce the operation count of finding shapelets it is not significantly better than FastShapelets, nor can it compete with the ShapeletTransform. The factory method for sampling the data is competitive with the ShapeletTransform and in some cases we see minor improvements despite being much faster.

## 1. INTRODUCTION

Time series classification (TSC) is a subset of the general classification problem. The main difference is that the attributes are ordered temporally, and the ordering of values within an instance is fundamental. For a set of $n$ time series, T $= \{T_1, T_2, ..., T_n\}$ each time series has $m$ ordered real-valued observations $T_i = \{t_{i1}, t_{i2}, ..., t_{im}\}$ and a class value $c_i$.

A technique that was recently proposed was Shapelets [1]. Shapelets were proposed as a way of identifying the best subsequence within a dataset, and use that to best split the data. The advantage shapelets have over more traditional approaches is that they find phase independent features which measures such as dynamic time warping (DTW) or Euclidean distance (ED) would fail to find. The original shapelet algorithm embedded the classification process in a decision tree using information gain (IG) to assess the quality of each shapelet [1]. The shapelet transform (ST) was proposed in 2012 and since then has been shown through extensive experimental evidence to be the best shapelet based approach to TSC [2, 3]. More recently we have made a number of improvements to the shapelet transform, these include more effective early abandons in the distance calculations, binary representation of class values to improve classification accuracy on multi class problems and leverage entropy pruning speed ups [4].

A number of different techniques have been developed from the original shapelet algorithm including Fast Shapelets and Learn Shapelets [5, 6]. The shapelet approach has not only been applied in the classification domain, shapelets have been used across machine learning. Examples of shapelets in clustering [7, 8, 9, 10] and in early classification [11, 12, 13, 14].

Shapelets are unique in TSC because they find phase independent features, the best shapelets are interpretable and can map back onto the original series to give insight into a problem. Shapelets have been applied to a number of different domains these include: leaf outlines [1, 15], gesture recognition [16], gait recognition [17] , classifying mutant worms [7], identifying electric devices and classifying hand outlines [18, 19].

Our aim in this paper is to present two different techniques to speed up the finding of shapelets. We propose the FastShapelets transform, which is an adaptation of the Fast Shapelet tree, and the factory transform which uses a run time condition to sample the data.

## 2. RELATED WORK AND BACKGROUND

### 2.1 Shapelet Transform

The shapelet transform involves a single scan of the data enumerating all possible shapelets, from all possible start positions, and all possible lengths between 3 and $m$. Each shapelet is evaluated on all the other time series. The quality of a shapelet is determined by calculating the distance from a series to a shapelet. Using a sliding window, calculate the Euclidean distance for the shapelet against a series, attempting to find the minimum distance match. Euclidean distance

is defined as:

$$dist(A, B) = \sqrt{\sum_{i=1}^{m}(a_i - b_i)^2} \quad (1)$$

We define the sliding window function for calculating the distance between a shapelet and a series as, where $W$ is the set of all subsequences which are the same length as $S$ in $T$:

$$sDist(S, T) = min_{w \in W}(dist(S, w)) \quad (2)$$

With the distance values for each series and the shapelet we use information gain to calculate how best it splits the data. A number of alternative distance measures have been assessed, however there was not a significant increase in quality for the other measures, so for simplicity we have opted to continue using Information Gain [20].

Shapelet based classification for ST involves finding the $k$ best shapelets. For each shapelet, we calculate the distance between the shapelet and each series, creating $k$ features for each series, this is shown in algoritgm 1. The main advantage of using transformed data is that we can use any classifier, and that we do not need to search for shapelets at each node of the tree.

The main issue with shapelets is the time complexity. Enumerating through the possible space of shapelets and evaluating them is O($n^2m^4$). Several heuristic techniques have been made to speed up this search and were described in [1, 21, 5, 20, 4]. The most recent improvements to the shapelet algorithm were presented in [4]. We demonstrated the effectiveness of a binary representation of class values on multi-class problems. Our research has been focussed on improving findDistances and assessCandidate functions within the shapelet transform, shown in algorithm 1. Our new aim is to consider alternative finding functions, where we use alternate generateCandidates functions in algorithm 1.

**Algorithm 1** ShapeletTransform(**T**, *min*, *max*, *k*)

**Parameters:** A list of time series **T**, *min* and *max* length shapelet to search for and $k$,the maximum number of shapelets to find)
**Return:** A list of $k$ Shapelets

```
numClasses ← getClasses(T)
kShapeletsMap ← ∅
prop ← k/numClasses
for all T_i in T do
    shapelets ← ∅
    for l ← min to max do
        W_{i,l} ← generateCandidates(T, i, l)
        for all subseries S in W_{i,l} do
            D_S ← findDistances(S,T)
            quality ← assessCandidate(S, D_S)
            shapelets.add(S, quality)
    sortByQuality(shapelets)
    removeSelfSimilar(shapelets)
    kShapelets ← kShapeletsMap.get(T.class)
    kShapelets ← merge(prop, kShapelets, shapelets)
    kShapeletsMap.add(kShapelets, T.class)
return kShapeletsMap.asList()
```

**Algorithm 2** generateCandidates(**T**, $i$, $l$)

**Parameters:** A list of time series **T**, $i$ and $l$, the times series to search and the length to consider.

```
shapelets ← ∅
for pos ← 0 to m − l + 1 do
    shapelets ∪ T^l_{i,pos}
return shapelets
```

## 2.2 Shapelet Algorithm

## 2.3 Fast Shapelets

Fast shapelets (FS) were proposed in 2013 [5]. The algorithm is a refinement of the original shapelet selection algorithm and employs a number of techniques to speed up the finding and pruning of shapelet candidates [15]. The major change made to the Shapelet algorthm is the introduction of symbolic aggregate approximation (SAX) [22, 23] and random projection.

The first stage of the shapelet finding process is to create a List of SAX words [22, 23]. The basic concept of SAX is a two stage process, firstly using piece-wise aggregate approximation (PAA), to transform a time series into a number of smaller averaged sections, reducing the length and smoothing the series. With a given alphabet size of 4 the aggregate series is discretized into a long sequence of characters. The sequence of characters are evaluated similarly to shapelets, where certain letters within a word are masked (random projection) and these words are evaluated based on both the number of occurrences, and on whether they only occur in one class. If a word occurs in one class often and not in another, it should be a good discriminatory feature, as opposed to a feature which occurs often in both, which is likely to be a poor feature. The best words are then converted back into there original time series, and evaluated in the same way as the other shapelet approaches, with information gain. The best shapelet forms the splitting rule in the decision tree. This process recursively divides the data into subsets finding the best shapelets until the data is sufficiently subdivided.

# 3. FAST SHAPELET TRANSFORM

In previous work the Shapelet transform significantly out performs the shapelet tree [2]. So our hypothesis was: "is the classification model of Fast Shapelets limiting its accuracy?". Our modification to the fast shapelet algorithm is minimal. We wanted to maintain as much similarity to the original approach whilst only removing the decision tree. We identified the search function in algorithm 1 to ensure the Fast Shapelet search works within our framework, and can benefit from all the improvements we have implemented. We consider the fast shapelets in essence a heuristic search, rather than searching the entire space of possibilities, we use the FS algorithm to reduce the number shapelets that are fully evaluated. Instead of testing and calculating the orderline for every shapelet candidate, we use the collision table to prune a number of shapelets because as the original authors pointed out the best shapelets are near the top of this list.

The fast shapelets algorithm considers the entire time series dataset and evaluates all the lengths when scoring the

**Algorithm 3** generateCandidatesFS(**T**,$i$ ,$l$)

**Parameters:** A list of time series **T**, $i$ and $l$ and the times series to search and the length to consider.

1: $shapelets \leftarrow \emptyset$
2: $SAXList$ =FindSAXWords(**T**, $l$)
3: RandomProjection($SAXList$)
4: $ScoreList$ =ScoreAllSAX($SAXList$)
5: $shapelets$ =FindTopSAXFromSeries($ScoreList$, $SAXList$, $i$)
6: **return** $shapelets$

SAX representations. To ensure parity we also find all the shapelets of a particular length, but on each subsequent call only return the best shapelets from within the given series. This ensures we can evaluate them in the series by series framework we have built, and so that we do not alter the finding technique as to make them incomparable.

# 4. FACTORY TRANSFORM

As the size and lengths of the datasets increase using the shapelet transform in reasonable time frames becomes increasingly difficult. This was also the case for the Fast Shapelet transform, these timing improvements are discussed in section 6. We wanted to evaluate simple sampling techniques and skipping parameters before exploring complex search algorithms. In previous work we were forced to heavily sample the larger datasets in the UCR repository as the number of instances and the length of the series meant that a number of them would take a matter of weeks or months to complete [3]. To better estimate a datasets runtime, we define the opCount formula (equation 3), in this definition we have also included the length and position skipping parameters ($p$ and $q$. These parameters are very simple. However, they allow us to avoid considering all enumerations of the shapelets. For example with both $q$ and $p$ equal to 2, every other shapelet candidate is considered, effectively halving the amount of shapelets considered.

By defining the fundamental operation as the addition within the Euclidean distance function, we can calculate the run time complexity function of any dataset. This formula also demonstrates the big-O bounding of the algorithm as we defined earlier as O($n^2m^4$).

The $min$ and $max$ are assumed to be 3 and $m$.

We denote the position skipping parameter as $p$ and the length skipping parameter as $q$.

We define the size of the set of possible lengths that exist with length skipping as $s = (m-3)/q$ The possible lengths are

$$L =< l_1, ..., l_s >$$

where

$$l_i = ((i-1)q) + 3$$

We define the opCount formula as:

$$opCount = \sum_{i=1}^{|L|} \left\lceil \frac{m - l_i + 1}{p} \right\rceil (m - l_i + 1)(l_i)(n-1)(n) \quad (3)$$

We expand the summation to:

$$\frac{(m-3)(n^2-n)(m^3+7m^2-m(q^2-18q+27)+5q^2-24q+27)}{12pq} \quad (4)$$

This formula will be useful in optimising the skipping parameters. For the case of just sampling the original data without any skipping we can set $q$ and $p$ equal to 1. The formula is simplified to:

$$opCount = \frac{(m-3)(n^2-n)(m^3+7m^2-10m+8)}{12} \quad (5)$$

Rearranging this formula making $n$ the subject, means for a given operation count we can calculate the size of the dataset.

$$n \approx \sqrt{\frac{12opCount}{(m-3)(m^3+7m^2-10m+8)}} \quad (6)$$

Equation 6 is the re-arranged form, so that given a constraint on the number of operations the shapelet transform is allowed to perform, we can calculate the new size of the dataset. We add one constraint to the sampling, we set a threshold of at least 25 cases of the least represented class.

If the proportion of $n$ is less than this threshold then the data is being over sampled, and the length and position skipping parameters may be required to fulfil the operation limit. Often if the data is sampled and is not able to run in the allotted time either there are a significant number of classes, or the series are very long. To reduce the work done we introduce the skipping parameters, by fixing them to the same value we can solve the opCount equation for the new dataset size, and we calculate the new skipping value.

We propose a modification to the existing shapelet transform algorithm, where we define a contract algorithm to preprocess the training data to tune the parameters $n$, $q$ and $p$. Reducing the operation count to below the allotted run time.

**Algorithm 4** calculateNPQ(**T**, $opThreshold$)

**Parameters:** A list of time series **T**, $opThreshold$ is the threshold of operations.

1: $q = 1$, $p = 1$
2: $opCount$ = calculateOpCount($n$,$m$,$q$,$p$)
3: **if** $opCount > opThreshold$ **then**
4: $sampleProp$= calculateProportion($n$,$m$,$opThreshold$)
5: $leastRepProp$ = calculateLeastRep(**T**)
6: **if** $sampleProp < leastRepProp$ **then**
7: $sampleProp = leastRepProp$
8: $n = n * sampleProp$
9: $q, p$ = calculateQP($n$, $m$, $opThreshold$)
10: **return** $n$, $p$, $q$

With this algorithm we create the Factory Transform, where given a contract op threshold which is directly tied to run time we can specify how long we want our shapelet transform to spend calculating the optimal shapelets. Where we randomly prune instances from the dataset whilst maintaining the original distribution, and skip shapelets of different positions and lengths equally throughout the full search.

# 5. DATA AND EXPERIMENTAL DESIGN

## 5.1 Datasets

The 85 datasets are described in detail on the website [24]. The collection is varied in terms of data characteristics: the length of the series ranges from 24 (ItalyPowerDemand) to 2709 (HandOutlines); train set sizes vary from 16 to 8926; and the number of classes is between 2 and 60. The data are from a wide range of domains, with an over representation of image outline classification problems (29 problems). Other categories are sensor readings (16), motion capture (14), food spectrographs (7), ECG measurements (7), electric device profiles (6) and simulated data (6). We include a further dataset, HeartbeatBIDMC, to demonstrate the scalability of the shapelet factory. HeartbeatBIDMC is an ECG data set archived on physionet that has been used in TSC research [25, 26] to assess scalability. The problem involves identifying 15 individuals with heart conditions based on their heartbeat. The train series are length 3750 and the test data length 1125. We truncate the test data to length 3750. There are 600 train cases and 600 test cases.

## 5.2 Experimental Design

To thoroughly test and improve the shapelet algorithm, we have designed and built upon the WEKA Java framework. For shapelets we have created a number of optimisations, and implemented some of the other work in the literature. This includes, but is not limited to, the work presented in [15, 21, 4, 20, 27]. These optimisations all reduce the amount of fundamental operations through various heuristics. To assess the quality of these optimisations and reduce the number of calculations we perform, we developed the operation count measure (opCount). This measure is essentially the number of addition operations in the Euclidean distance between two subsequences. Early abandon, entropy pruning and the plethora of other optimisations all reduce this count, however none of these techniques reduce the worst case time complexity.

We have designed a series of experiments to assess both the run time performance and the accuracy. We wanted to identify how much of a stand alone reduction the FS search provides in comparison to a full search. We also wanted to show the amount of work the current state-of-the-art heuristics reduce operation count by, and then finally how much work we can reduce by combining all approaches. All of the experiments we outline operate under the experimental standard we set in [3]. We perform 100 re-samples where possible, some of the larger datasets were not possible to run in a timely fashion (greater than 7 days runtime). This illustrates our point on a greater need for more drastic dataset reduction in subsequent experiments.

We measure four approaches to the shapelet speed up these are; Full Shapelet search without optimisations, Full Shapelet search with optimisations, Fast Shapelet search without optimisations, and Fast Shapelet search with optimisations. Time taken for the full shapelet search without optimisations can be calculated, and is our reference point for how much work we have avoided.

Our first set of experiments were designed to assess the speed improvement of the fast shapelet search. We also wanted to test the search with and without any pruning to test its individual contribution. In Table 3 we present the opCounts for the four approaches, and the amount of work they performed with respect to the full transform, as a percentage.

We also assess the effect the Fast Shapelet search had on accuracy, and whether the Fast Shapelet Transform was better than the original algorithm (FS) and whether it was as good as the shapelet transform. In Table 6.1 we present the accuracies of a number of baseline classifiers; 1 nearest neighbour with dynamic time warping setting the window through cross validation, rotation forest and the accuracies for the FastTransform, FastShapelets (FS) and the Shapelet Transform (ST).

The second set of experiments we perform are for the Factory transform. We proposed the factory transform in section 4 as a way of finding shapelets in a fixed time by dynamically resizing large datasets, as well as skipping on lengths and positions when searching for shapelets. The factory transform calculates the estimated time to create the shapelet transform and resamples only if this exceeds the threshold, therefore a large portion of the 85 datasets will not need to be sampled. We set our time threshold to one day and compare the factory transforms accuracy with that of the full shapelet transform. The results are presented in table 4. We present the parameters that were selected for each dataset by the estimation function. We also opted to fix $q$ and $p$ to the same value for simplicity. We present the results for the HeartBIDMC dataset.

# 6. RESULTS

## 6.1 Fast Shapelet Transform vs. Full Shapelet Transform

Table 6.1 presents the results for the accuracies of the Fast shapelet transform compared with the shapelet transform, fast shapelets, rotation forest(RotF) and 1nn-DTW (DTW) a baseline comparison. In Figure 1 we present the critical difference diagram for these classifiers. Completing the 100 resamples on the very large datasets for the Fast shapelet transform, even with its speed increase would have still required sampling of cases, which is not the purpose of these experiments, and so we chose not evaluate it on the very large datasets. The design of our current framework allows us to very simply enable and disable certain optimisations, for further analysis using the FactoryTransforms subsampling for large datasets could be considered to further evaluate the technique on the large datasets. The results show that the Fast shapelet transforms is neither significantly better than FastShapelets, nor significantly worse than DTW, this is a marked improvement, as the Fast Shapelets is in a separate clique.

In Table 6.1 we show the operation counts for the four shapelet approaches, these operations are in the order of millions. We were unable to present results for all the datasets with the Shapelet transform as the missing datasets require some minor sampling to complete in a timely manner, and this would prove an unfair comparison. The fast shapelet search reduces on average the amount of operations by 65% and the speed up techniques for distance calculations reduce the average amount of operations by 60%. The combination of both search and speedup techniques reduces the opcount on average by 95%, this highlights both the importance of the current research into distance early abandon heuristics, as well as into alternative search techniques for shapelets.

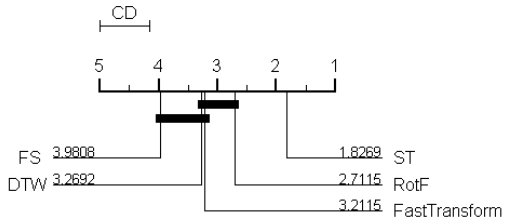


**Figure 1: Critical difference diagram for 5 classifiers, showing the effectiveness of the FastTransform against Shapelet Transform, Fast Shapelets and 2 benchmark classifiers.**

## 6.2 Factory Tranform vs. Full Shapelet Transform

In Table 4 we show the result for the factory transform compared with the shapelet transform. The factory transform wins on 32 out of the 42 datasets and comparing both the classifiers on a Wilcoxon signed rank test and a binomial test the factory transform is significantly better at the 5% level. When the shapelet transform was proposed the scalability on larger problems was tackled by a simple method for setting the min and max lengths of shapelets. This method drastically reduced the search space for the large problems, at the expense of accuracy. In subsequent research we've increased the shapelet min and max parameters, to 3 and $m$ so that we consider the entire problem space, and for the larger problems a simple sampling technique of 10% has been sufficient. The Factory Transform proposed is more accurate on the larger problems than the Shapelet Transform because the previous sampling required was much more coarse.

The HeartBIDMC dataset was estimated to have a run time, assuming no optimisations, of 68,000 days. We reduced this by heavily sampling both train set and skipping parameters, to calculate a transform in one day. We show the results for the HeartBIDMC dataset in Table 3 and compare the FactoryTransform to a plethora of elastic measures [28]. The size of the dataset and the length of the series meant that getting either LearnShapelets or FastShapelets to complete in a timely manner was not possible. In figure 2 we show an example of two of the shapelets we found for class 1 in the dataset.

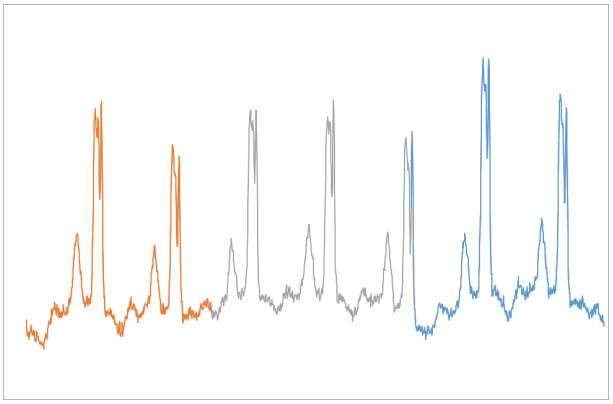

**Figure 2: An example of the best two shapelets for class 1**

**Table 3: Comparing the FactoryTransform with other common TSC approaches**

| Algorithm | Accuracy |
|---|---|
| FactoryTransform | 0.972 |
| Euclidean 1NN | 0.21 |
| DTW | 0.805 |
| WDTW-1NN | 0.818 |
| DDTW-Rn-1NN | 0.806 |
| WDDTW-1NN | 0.79 |
| LCSS-1NN | 0.8817 |
| MSM-1NN | 0.915 |
| TWE-1NN | 0.905 |
| ERP-1NN | 0.833 |
| EE | 0.835 |

# 7. CONCLUSIONS AND FUTURE WORK

Our research aimed to evaluate two separate techniques for reducing the average runtime of the shapelet search algorithm. The first technique we defined was an extension to the Fast Shapelets algorithm, originally proposed in [5]. The second technique we defined was the Factory transform, before we start considering more complex techniques for reducing the run time complexity of shapelets, we wanted to the effectiveness of more simplistic approaches; these included sampling of cases, and skipping length and position parameters when finding shapelet candidates.

Firstly we evaluated whether we could improve the Fast Shapelets algorithm, leveraging an alternative search algorithm for finding shapelets but utilising the transform to enable a broader range of classifiers. This research had two aims, to show that the Fast Shapelet transform was superior to Fast shapelets, which was not the case. Secondly we wanted to show that it was comparable to the DTW benchmark whilst being faster than the Shapelet transform, this was shown in figure 1 and the difference in operation counts from 60% work reduction down to a 95% when paired with the Fast shapelet search.

Secondly we evaluated the more simplistic speed up technique of dynamically sampling cases on large datasets. We set the time limit of 1 day for our contract on the UCR datasets

**Table 1: Comparison of Fast Transforms accuracy, average accuracy of 100 fold resampling**

| Datasets | FastTransform | ST | FS | RotF | DTW |
|---|---|---|---|---|---|
| ArrowHead | 0.808 | 0.851 | 0.675 | 0.789 | **0.871** |
| Beef | 0.682 | 0.736 | 0.502 | **0.819** | 0.61 |
| BeetleFly | **0.964** | 0.875 | 0.796 | 0.791 | 0.85 |
| BirdChicken | **0.947** | 0.927 | 0.862 | 0.747 | 0.853 |
| Car | 0.807 | **0.902** | 0.736 | 0.788 | 0.75 |
| CBF | 0.945 | **0.986** | 0.924 | 0.898 | 0.545 |
| ChlorineConcentration | 0.747 | 0.682 | 0.566 | **0.846** | 0.69 |
| Coffee | 0.991 | **0.995** | 0.917 | 0.995 | 0.982 |
| DiatomSizeReduction | **0.933** | 0.911 | 0.873 | 0.881 | 0.92 |
| DistalPhalanxOutlineAgeGroup | 0.809 | **0.819** | 0.745 | 0.807 | 0.699 |
| DistalPhalanxOutlineCorrect | 0.826 | **0.829** | 0.78 | 0.812 | 0.759 |
| DistalPhalanxTW | 0.665 | 0.69 | 0.623 | **0.692** | 0.622 |
| ECG200 | 0.846 | 0.84 | 0.806 | 0.851 | **0.855** |
| ECG5000 | 0.94 | **0.943** | 0.922 | 0.942 | 0.926 |
| ECGFiveDays | 0.741 | 0.955 | **0.986** | 0.86 | 0.685 |
| FaceAll | 0.862 | **0.968** | 0.772 | 0.905 | 0.936 |
| FaceFour | 0.719 | 0.794 | **0.869** | 0.853 | 0.732 |
| FacesUCR | 0.73 | **0.909** | 0.701 | 0.784 | 0.873 |
| FiftyWords | 0.685 | 0.713 | 0.512 | 0.675 | **0.763** |
| Fish | 0.691 | **0.974** | 0.742 | 0.859 | 0.91 |
| GunPoint | 0.885 | **0.999** | 0.93 | 0.924 | 0.973 |
| Herring | 0.584 | **0.653** | 0.558 | 0.608 | 0.505 |
| InsectWingbeatSound | 0.618 | 0.617 | 0.488 | **0.633** | 0.531 |
| ItalyPowerDemand | 0.96 | 0.953 | 0.909 | **0.967** | 0.956 |
| Lightning7 | 0.655 | **0.724** | 0.101 | 0.701 | 0.616 |
| Meat | 0.872 | 0.966 | 0.924 | **0.994** | 0.821 |
| MedicalImages | 0.732 | 0.691 | 0.609 | **0.756** | 0.668 |
| MiddlePhalanxOutlineAgeGroup | **0.72** | 0.694 | 0.613 | 0.669 | 0.591 |
| MiddlePhalanxOutlineCorrect | 0.78 | 0.815 | 0.716 | **0.82** | 0.781 |
| MiddlePhalanxTW | **0.579** | 0.579 | 0.519 | 0.568 | 0.507 |
| MoteStrain | 0.837 | **0.882** | 0.793 | 0.859 | 0.756 |
| OliveOil | 0.701 | 0.881 | 0.765 | **0.889** | 0.784 |
| Plane | 0.97 | **1.0** | 0.97 | 0.986 | 0.994 |
| ProximalPhalanxOutlineAgeGroup | 0.839 | 0.841 | 0.797 | **0.847** | 0.781 |
| ProximalPhalanxOutlineCorrect | 0.802 | **0.881** | 0.797 | 0.875 | 0.857 |
| ProximalPhalanxTW | 0.782 | 0.803 | 0.716 | **0.808** | 0.75 |
| ShapeletSim | 0.554 | 0.934 | **1.0** | 0.488 | 0.662 |
| SonyAIBORobotSurface1 | 0.881 | 0.888 | **0.918** | 0.814 | 0.745 |
| SonyAIBORobotSurface2 | 0.863 | **0.924** | 0.849 | 0.846 | 0.852 |
| Strawberry | 0.907 | 0.968 | 0.917 | **0.974** | 0.959 |
| SwedishLeaf | 0.842 | **0.939** | 0.758 | 0.884 | 0.906 |
| Symbols | 0.862 | 0.862 | 0.908 | 0.842 | **0.935** |
| SyntheticControl | 0.962 | **0.987** | 0.92 | 0.967 | 0.563 |
| ToeSegmentation1 | 0.606 | **0.954** | 0.904 | 0.578 | 0.702 |
| ToeSegmentation2 | 0.764 | **0.947** | 0.873 | 0.646 | 0.804 |
| Trace | 0.752 | **1.0** | 0.998 | 0.932 | 0.995 |
| TwoLeadECG | 0.678 | **0.984** | 0.92 | 0.928 | 0.94 |
| TwoPatterns | 0.92 | 0.952 | 0.696 | 0.928 | **0.997** |
| Wafer | 0.991 | **1.0** | 0.981 | 0.995 | 0.995 |
| Wine | 0.58 | **0.926** | 0.794 | 0.919 | 0.85 |
| WordSynonyms | 0.59 | 0.582 | 0.461 | 0.586 | **0.728** |
| Yoga | 0.821 | 0.823 | 0.721 | **0.854** | 0.835 |

| Datasets | FastTransform | FullTransform |
|---|---|---|
| Wins | 12 | 40 |

| Datasets | FastTransform | FastShapelets |
|---|---|---|
| Wins | 36 | 16 |

**Table 2: opCounts(in millions) of four different versions of the shapelet transform on 47 datasets**

| Dataset | Fast | Full | SpeedUps | Combined | Fast | SpeedUps | Combined |
|---|---|---|---|---|---|---|---|
| ArrowHead | 137405.000 | 423196.000 | 127217.000 | 46764.750 | 0.32 | 0.30 | 0.11 |
| Beef | 1132100.000 | 3567380.000 | 1016010.000 | 415402.000 | 0.32 | 0.28 | 0.12 |
| BeetleFly | 1333850.000 | 2192860.000 | 1221600.000 | 860176.000 | 0.61 | 0.56 | 0.39 |
| BirdChicken | 647070.000 | 2192860.000 | 883095.000 | 330075.000 | 0.30 | 0.40 | 0.15 |
| Car | 2453470.000 | 32921900.000 | | 880795.000 | 0.07 | | 0.03 |
| CBF | 17522.927 | 20033.384 | 14534.000 | 12455.808 | 0.87 | 0.73 | 0.62 |
| ChlorineConcentration | 3680240.000 | 14087100.000 | | 1954840.000 | 0.26 | | 0.14 |
| Coffee | 99638.013 | 427243.000 | 38227.000 | 13061.743 | 0.23 | 0.09 | 0.03 |
| DiatomSizeReduction | 20391.452 | 286551.000 | 25235.000 | 2562.230 | 0.07 | 0.09 | 0.01 |
| DistalPhalanxOutlineAgeGroup | 44971.005 | 569421.000 | 106319.000 | 11741.437 | 0.08 | 0.19 | 0.02 |
| DistalPhalanxOutlineCorrect | 138160.000 | 1282270.000 | 279203.000 | 41800.481 | 0.11 | 0.22 | 0.03 |
| DistalPhalanxTW | 45080.854 | 569421.000 | 101613.000 | 1075.419 | 0.08 | 0.18 | 0.01 |
| ECG200 | 32243.889 | 72758.961 | 32585.000 | 1671.359 | 0.44 | 0.45 | 0.02 |
| ECG5000 | 2351580.000 | 8203050.000 | 3998320.000 | 129349.000 | 0.29 | 0.49 | 0.02 |
| ECGFiveDays | 6912.204 | 14825.684 | 4838.000 | 219.910 | 0.47 | 0.33 | 0.01 |
| FaceAll | 6343870.000 | 7903390.000 | 5466050.000 | 409327.000 | 0.80 | 0.69 | 0.05 |
| FaceFour | 533955.000 | 698003.000 | 394843.000 | 26959.794 | 0.76 | 0.57 | 0.04 |
| FacesUCR | 855320.000 | 1004840.000 | 673159.000 | 55698.130 | 0.85 | 0.67 | 0.06 |
| GunPoint | 21388.225 | 105975.000 | 38627.000 | 972.567 | 0.20 | 0.36 | 0.01 |
| Herring | 2481450.000 | 23267400.000 | | 48211.737 | 0.11 | | 0.01 |
| InsectWingbeatSound | 6519320.000 | 17505600.000 | | 380356.000 | 0.37 | | 0.02 |
| ItalyPowerDemand | 65.203 | 136.489 | 54.000 | 2.677 | 0.48 | 0.40 | 0.02 |
| Lightning7 | 1984710.000 | 4219010.000 | 2935880.000 | 136956.000 | 0.47 | 0.70 | 0.03 |
| Meat | 781272.000 | 11987500.000 | | 9179.359 | 0.07 | | 0.01 |
| MedicalImages | 450407.000 | 1202180.000 | 641756.000 | 30612.930 | 0.37 | 0.53 | 0.03 |
| MiddlePhalanxOutlineAgeGroup | 30638.186 | 569421.000 | 87209.000 | 676.306 | 0.05 | 0.15 | 0.01 |
| MiddlePhalanxOutlineCorrect | 67418.459 | 1282270.000 | 206842.000 | 1512.536 | 0.05 | 0.16 | 0.01 |
| MiddlePhalanxTW | 31195.745 | 566573.000 | 89467.000 | 673.167 | 0.06 | 0.16 | 0.01 |
| MoteStrain | 1034.774 | 1644.874 | 950.000 | 60.285 | 0.63 | 0.58 | 0.04 |
| OliveOil | 173962.000 | 7706080.000 | 70298.000 | 456.460 | 0.02 | 0.01 | 0.01 |
| Plane | 157276.000 | 401574.000 | 182699.000 | 7208.427 | 0.39 | 0.45 | 0.02 |
| ProximalPhalanxOutlineAgeGroup | 21829.624 | 569421.000 | 77454.000 | 332.152 | 0.04 | 0.14 | 0.01 |
| ProximalPhalanxTW | 21373.217 | 569421.000 | 78038.000 | 421.790 | 0.04 | 0.14 | 0.01 |
| ShapeletSim | 1672210.000 | 1994760.000 | 1355260.000 | 129571.000 | 0.84 | 0.68 | 0.06 |
| SonyAIBORobotSurface1 | 664.006 | 799.063 | 317.000 | 25.187 | 0.83 | 0.40 | 0.03 |
| SonyAIBORobotSurface2 | 984.563 | 1101.050 | 618.000 | 55.713 | 0.89 | 0.56 | 0.05 |
| Strawberry | 6855060.000 | 96915700.000 | | 118914.000 | 0.07 | | 0.01 |
| SwedishLeaf | 1683130.000 | 5745210.000 | 2235150.000 | 75636.194 | 0.29 | 0.39 | 0.01 |
| Symbols | 216693.000 | 1266960.000 | 762049.000 | 14815.894 | 0.17 | 0.60 | 0.01 |
| SyntheticControl | 85263.579 | 102522.000 | 78777.000 | 6203.079 | 0.83 | 0.77 | 0.06 |
| ToeSegmentation1 | 510802.000 | 776099.000 | 470739.000 | 30078.875 | 0.66 | 0.61 | 0.04 |
| ToeSegmentation2 | 750774.000 | 1469900.000 | 813917.000 | 47968.920 | 0.51 | 0.55 | 0.03 |
| Trace | 1387840.000 | 4785000.000 | 3031780.000 | 104463.000 | 0.29 | 0.63 | 0.02 |
| TwoLeadECG | 575.925 | 1990.827 | 404.000 | 12.442 | 0.29 | 0.20 | 0.01 |
| TwoPatterns | 14450300.000 | 23003900.000 | | 1000580.000 | 0.63 | | 0.04 |
| Wine | 24924.705 | 810711.000 | 18938.000 | 80.178 | 0.03 | 0.02 | 0.01 |
| WordSynonyms | 13696600.000 | 31906000.000 | | 870405.000 | 0.43 | | 0.03 |

**Table 4: Accuracy and paramters for each dataset to run in the allotted time of 1 day, on fold 0**

| Dataset | Factory | Full |
|---|---|---|
| Adiac | **0.7697949** | 0.76838875 |
| Car | 0.8931973 | **0.90183336** |
| ChlorineConcentration | **0.7273969** | 0.6820625 |
| CinCECGtorso | **0.9839249** | 0.91826814 |
| Computers | **0.8065306** | 0.78464 |
| CricketX | **0.78375196** | 0.7771026 |
| CricketY | **0.7629775** | 0.7621795 |
| CricketZ | **0.8024594** | 0.7977812 |
| Earthquakes | 0.7363088 | **0.7372662** |
| FiftyWords | **0.7444271** | 0.71298903 |
| Fish | **0.97463554** | 0.9741651 |
| FordA | 0.94346786 | **0.96535605** |
| FordB | 0.9039557 | **0.91508645** |
| Ham | 0.7689019 | **0.80838096** |
| HandOutlines | 0.9178434 | **0.9239189** |
| Haptics | **0.52988344** | 0.51185066 |
| Herring | 0.6481186 | **0.65343213** |
| InlineSkate | **0.42346933** | 0.3930182 |
| InsectWingbeatSound | **0.6274324** | 0.61653537 |
| LargeKitchenAppliances | **0.93287075** | 0.9325067 |
| Lightning2 | 0.5913348 | **0.65885246** |
| Mallat | **0.97428316** | 0.9723497 |
| Meat | **0.9685374** | 0.96566665 |
| NonInvasiveFatalECGThorax1 | 0.9122605 | **0.94676846** |
| NonInvasiveFatalECGThorax2 | 0.9309809 | **0.9538524** |
| OSULeaf | **0.97529095** | 0.9340909 |
| Phoneme | **0.3389144** | 0.3290559 |
| RefrigerationDevices | **0.79137415** | 0.7608 |
| ScreenType | **0.7057415** | 0.6761067 |
| ShapesAll | **0.9041326** | 0.8542333 |
| SmallKitchenAppliances | **0.80742854** | 0.80248 |
| StarlightCurves | **0.97814846** | 0.9774138 |
| Strawberry | **0.96924984** | 0.9684324 |
| TwoPatterns | **0.9548572** | 0.9517175 |
| UWaveGestureLibraryAll | **0.9553351** | 0.94207704 |
| UWaveGestureLibraryX | **0.81106496** | 0.8059464 |
| UWaveGestureLibraryY | **0.74265033** | 0.736957 |
| UWaveGestureLibraryZ | **0.7518175** | 0.74676436 |
| WordSynonyms | **0.59562725** | 0.58238244 |
| Worms | **0.7434405** | 0.7194805 |
| WormsTwoClass | **0.78293157** | 0.7787013 |
| Yoga | **0.86662924** | 0.82251 |

| dataset | originalN | newN | q and p |
|---|---|---|---|
| Adiac | 390 | 390 | 2 |
| Car | 60 | 60 | 2 |
| ChlorineConcentration | 467 | 411 | 2 |
| CinCECGtorso | 40 | 40 | 11 |
| Computers | 250 | 50 | 3 |
| CricketX | 390 | 390 | 4 |
| CricketY | 390 | 343 | 4 |
| CricketZ | 390 | 390 | 4 |
| Earthquakes | 322 | 138 | 4 |
| FiftyWords | 450 | 450 | 4 |
| Fish | 175 | 175 | 4 |
| FordA | 3601 | 51 | 2 |
| FordB | 3636 | 51 | 2 |
| Ham | 109 | 54 | 1 |
| HandOutlines | 1000 | 69 | 50 |
| Haptics | 155 | 155 | 19 |
| Herring | 64 | 64 | 2 |
| InlineSkate | 100 | 100 | 35 |
| InsectWingbeatSound | 220 | 220 | 2 |
| LargeKitchenAppliances | 375 | 75 | 4 |
| Lightning2 | 60 | 60 | 3 |
| Mallat | 55 | 55 | 6 |
| Meat | 60 | 60 | 2 |
| NonInvasiveFatalECGThorax1 | 1800 | 1267 | 70 |
| NonInvasiveFatalECGThorax2 | 1800 | 1267 | 70 |
| OSULeaf | 200 | 200 | 4 |
| Phoneme | 214 | 214 | 23 |
| RefrigerationDevices | 375 | 75 | 4 |
| ScreenType | 375 | 75 | 4 |
| ShapesAll | 600 | 600 | 16 |
| SmallKitchenAppliances | 375 | 75 | 4 |
| StarLightCurves | 1000 | 164 | 17 |
| Strawberry | 613 | 183 | 1 |
| TwoPatterns | 1000 | 894 | 2 |
| UWaveGestureLibraryAll | 896 | 221 | 20 |
| UWaveGestureLibraryX | 896 | 221 | 3 |
| UWaveGestureLibraryY | 896 | 221 | 3 |
| UWaveGestureLibraryZ | 896 | 221 | 3 |
| WordSynonyms | 267 | 267 | 2 |
| Worms | 181 | 181 | 15 |
| WormsTwoClass | 181 | 59 | 5 |
| Yoga | 300 | 55 | 1 |

| | |
|---|---|
| Full Wins | 10 |
| Factory Wins | 32 |

| Dataset | originalN | newN | q and p | Accuracy |
|---|---|---|---|---|
| HeartbeatBIDMC | 600 | 375 | 513 | 0.971564571 |

and conducted experiments with 100 resamples. Of the 85 UCR datasets we identified 42 problems that could not fulfil the contract without sampling. Using the operation count formula we derived we were able to estimate the amount of sampling we needed to perform, and with an arbitrary threshold of 25 cases of the least represented class we sample to the larger of the two. The factory was shown to be significantly better than the shapelet transform, and on 32 out of the 42 datasets we had superior accuracy.

The contract classifier and the sampling technique has been very effective in reducing the dataset size on problems with only a few number of classes, and with relatively short series. As the series length increases, and we become unable to sample the number of cases because of the threshold, we have to increase the length and position skipping dramatically to fulfil the contract. This means large portions of the potential shapelet candidates are missed, some of which could have formed part of the shapelet set on a full enumeration. The fast shapelet transform demonstrated the efficacy of searching smaller portions of the dataset, and our next area of research will be implementing and assessing a number of local search techniques to speed up the transform. Our research aims to improve the shapelet transform, whilst finding shapelets that exist within the original dataset.

Finally we presented a case study of HeartBIDMC which presents significant challenges, both due to its size, the length of the series, and the type of data each class represents. We were unable to calculate values for other shapelet based approaches, demonstrating that despite considerable speed ups that as the size of datasets increases more drastic sampling is required, and can be effective. We also presented the results for a number of elastic measures, which were significantly worse than the shapelet approach.

## Acknowledgment

The experiments were carried out on the High Performance Computing Cluster supported by the Research and Specialist Computing Support service at the University of East Anglia. We would particularly like to thank Leo Earl for his help and forbearance.

# 8. REFERENCES

[1] L. Ye and E. Keogh, "Time series shapelets," *Proc. 15th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '09*, p. 947, 2009.

[2] J. Lines, L. Davis, J. Hills, and A. Bagnall, "A shapelet transform for time series classification," *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*, pp. 289–297, 2012.

[3] A. Bagnall, J. Lines, A. Bostrom, and J. Large, "The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version," *eprint arXiv:1602.01711*, p. 19, Jan. 2016.

[4] A. Bostrom and A. Bagnall, "Binary shapelet transform for multiclass time series classification," in *Int. Conf. Big Data Anal. Knowl. Discov.*, vol. 9263, pp. 257–269, 2015.

[5] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *Proc. Thirteen. SIAM Conf. ...*, pp. 668–676, 2013.

[6] J. Grabocka, N. Schilling, M. Wistuba, and L. Schmidt-Thieme, "Learning Time-series Shapelets," *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '14*, pp. 392–401, 2014.

[7] J. Hills, *Mining Time-series Data using Discriminative Subsequences*. PhD thesis, University of East Anglia, 2014.

[8] J. Zakaria, A. Mueen, and E. Keogh, "Clustering time series using unsupervised-shapelets," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 785–794, 2012.

[9] J. Zakaria, A. Mueen, E. Keogh, and N. Young, "Accelerating the discovery of unsupervised-shapelets," *Data Min. Knowl. Discov.*, vol. 30, no. 1, pp. 243–281, 2016.

[10] L. Ulanova, N. Begum, and E. Keogh, "Scalable Clustering of Time Series with U-Shapelets," *Proc. 2015 SIAM Int. Conf. Data Min.*, pp. 900–908, 2015.

[11] Z. Xing, P. S. Yu, and K. Wang, "Extracting Interpretable Features for Early Classification on Time Series," *Siam Int. Conf. Data Min.*, pp. 247–258, 2011.

[12] Z. Xing, J. Pei, and P. S. Yu, "Early classification on time series," *Knowl. Inf. Syst.*, vol. 31, pp. 105–127, Apr. 2012.

[13] M. P. Griffin and J. R. Moorman, "Toward the early diagnosis of neonatal sepsis and sepsis-like illness using novel heart rate analysis.," *Pediatrics*, vol. 107, no. FEBRUARY 2001, pp. 97–104, 2001.

[14] L. Bernaille, R. Teixeira, I. Akodjenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 2, pp. 23–26, 2006.

[15] L. Ye and E. Keogh, "Time series shapelets: A novel technique that allows accurate, interpretable and fast classification," *Data Min. Knowl. Discov.*, vol. 22, no. 1-2, pp. 149–182, 2011.

[16] B. Hartmann and N. Link, "Gesture recognition with inertial sensors and optimized DTW prototypes," *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, pp. 2102–2109, 2010.

[17] S. T. and P. B. Sivakumar, "Human Gait Recognition and Classification Using Time Series Shapelets," *2012 Int. Conf. Adv. Comput. Commun.*, pp. 31–34, Aug. 2012.

[18] J. Lines and A. Bagnall, "Time series classification with ensembles of elastic distance measures," *Data Min. Knowl. Discov.*, vol. 29, pp. 565–592, June 2014.

[19] A. Bagnall, L. Davis, J. Hills, and J. Lines, "Transformation based ensembles for time series classification," *Proc. 12th SDM*, pp. 890–901, 2012.

[20] J. Hills, J. Lines, E. Baranauskas, J. Mapp, and A. Bagnall, "Classification of time series by shapelet transformation," *Data Min. Knowl. Discov.*, vol. 28, pp. 851–881, May 2014.

[21] A. Mueen, E. Keogh, and N. Young, "Logical-shapelets," *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. data Min. - KDD '11*, p. 1154, 2011.

[22] L. Wei, E. Keogh, and A. Xi, "SAXually explicit images: Finding unusual shapes," *Proc. - IEEE Int. Conf. Data Mining, ICDM*, pp. 711–720, 2006.

[23] J. Lin, E. Keogh, L. Wei, and S. Lonardi, "Experiencing SAX: a novel symbolic representation of

time series," *Data Min. Knowl. Discov.*, vol. 15, pp. 107–144, Apr. 2007.

[24] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," July 2015. www.cs.ucr.edu/ eamonn/time_series_data/.

[25] P. Schäfer, "Scalable time series classification," *Data Min. Knowl. Discov.*, no. February, 2015.

[26] B. Hu and S. Evans, "Discovering the Intrinsic Cardinality and Dimensionality of Time Series using MDL," *Icdm*, pp. 1086–1091, 2011.

[27] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, pp. 262–270, 2012.

[28] J. Lines, *Time Series classification through transformation and ensembles*. PhD thesis, University of East Anglia, 2015.