A Novel Approximation to Dynamic Time Warping allows Anytime Clustering of Massive Time Series Datasets

Qiang Zhu Gustavo Batista Thanawin Rakthanmanon Eamonn Keogh
University of California, Riverside
{qzhu, gbatista, rakthant, eamonn}@cs.ucr.edu

ABSTRACT

Given the ubiquity of time series data, the data mining community has spent significant time investigating the best time series similarity measure to use for various tasks and domains. After more than a decade of extensive efforts, there is increasing evidence that Dynamic Time Warping (DTW) is very difficult to beat. Given that, recent efforts have focused on making the intrinsically slow DTW algorithm faster. For the similarity-search task, an important subroutine in many data mining algorithms, significant progress has been made by replacing the vast majority of expensive DTW calculations with cheap-to-compute lower bound calculations. However, these lower bound based optimizations do not directly apply to clustering, and thus for some realistic problems, clustering with DTW can take days or weeks.

In this work, we show that we can mitigate this untenable lethargy by casting DTW clustering as an *anytime algorithm*. At the heart of our algorithm is a novel data-adaptive approximation to DTW which can be quickly computed, and which produces approximations to DTW that are much better than the best currently known linear-time approximations. We demonstrate our ideas on real world problems showing that we can get virtually all the accuracy of a batch DTW clustering algorithm in a fraction of the time.

Keywords

DTW, Clustering, Anytime Algorithm

1 INTRODUCTION

The extraordinary ubiquity of time series data has resulted in the data mining community spending significant resources in investigating algorithms to mine time series archives. Much of this effort has focused on finding the best distance measure to use for the given domain. After more than a decade of extensive research, there is increasing evidence that Dynamic Time Warping (DTW), which includes Euclidean Distance (ED) as a special case, is *very* difficult to beat [5][41]. Given this, significant attention has focused on making the intrinsically quadratic-time DTW algorithm faster [5][16][17][43]. For the problem of *query-by-content*, significant progress has been made by replacing the vast majority of expensive DTW calculations with cheap-to-compute lower bound calculations. However, these lower bound based optimizations do not directly apply to clustering, and thus for some realistic problems, clustering with DTW can take days or weeks. As a concrete example, consider the two following real world problems:

- A star light curve is the measurement of the brightness of a celestial object as a function of time [13]. The study of light curves in astronomy has led to the discoveries of pulsars, extra solar planets, supernovae, the rate of expansion of the universe, etc. In addition to automatically recorded digital star light curves, there are over 100 million analogue observations from archives dating back 130 years [37]. One stage of the digitization process involves *clustering* the data to look for outliers [13], and recent work has forcefully shown that DTW is much better than ED for this task [28]. However, as we shall show below, clustering a mere 9,236 curves under DTW takes about 127 days using a batch algorithm, a severe bottleneck in digitization efforts.
- A fundamental idea in autonomous robotics is that the robot should learn to adapt to its environment by clustering its experiences and using those clusters as the basis of classification and/or outlier detection algorithms [8][10][25][44]. For at least a decade, DTW has been the distance measure of choice in this domain [8][25]. However, in general we cannot anticipate how long the robot will have to do clustering before external events force it to invoke a classification decision, thus an anytime framework is ideal in this context.

In this work, we show that we can mitigate the untenable lethargy of DTW clustering by casting it as an *anytime algorithm* [7][45]. Anytime algorithms are algorithms that trade execution time for quality of results. In particular, an anytime algorithm always has a *best-so-far* answer available, and the quality of the answer improves with execution time. The user may interact with the clustering, examining an answer at any time and choose to terminate, temporarily suspend, or continue the algorithm's execution until completion [33]. For example, in the star light curve scenario above, rather than waiting 127 days for the batch algorithm to finish, the user may temporarily suspend the algorithm after a few hours, glance at the approximate solution and if she sees obvious outliers, she can then examine these outliers offline while the resumed algorithm runs in the background.

The idea of interactively exploring clustering results has been shown to be useful for at least a decade [31], but thus far has been limited to Euclidean distance or other inexpensive distance measures.

At the heart of our technique for porting DTW clustering to an anytime framework is a novel data-adaptive approximation to DTW which can be quickly computed, but produces approximations to DTW which are much closer than any of the known linear-time approximations [29]. Our ideas are general enough to be used for hierarchical, partitional or spectral clustering; and the *overhead* (the extra time over the batch algorithm that our anytime algorithms take if allowed to run to completion) is inconsequential. The fundamental contributions of our work are:

- A novel approximation to Dynamic Time Warping, which is both fast to compute and accurate, by exploiting the domain dependent relationship between DTW and its upper and lower bounds.
- A heuristic ordering function that tells the anytime algorithm the best order in which calculate the *exact* DTW distances. In essence, this ordering function predicts which of the currently *approximated* distances are most likely to benefit from being replaced by *exact* DTW calculations.

The rest of the paper is organized as follows. In Section 2 we review related work and give three assumptions that inform our ideas. In Section 3 we introduce the framework of our anytime clustering algorithm, expanding the discussion of the two main subroutines in Section 4 and Section 5. We perform an extensive empirical evaluation on real datasets in Section 6, and offer conclusions in Section 7.

2 ASSUMPTIONS AND RELATED WORK

We begin with a statement of the assumptions that inform our work. We denote the number of time series to cluster as M and their length as N[1].

2.1 Assumptions

Assumption 1**:** The time taken to cluster the data is negligible compared to the time to calculate *all* required DTW distances.

As a concrete example, once we are given a full distance matrix for the "start light curves" dataset with M = 9,236 and N = 1,024, hierarchical clustering with average linkage only takes 4 seconds, whereas it takes 127 days to actually *fill* the matrix with DTW distances.

[1] For datasets with different length of time series, we can normalize them to the same length with little or no impact on accuracy [26].

This assumption not only motivates our work (if DTW could be quickly computed, the batch algorithm would suffice), but allows the desirable *interruptibility* property of anytime algorithms (cf. Section 2.2).

Note that this assumption includes the implicit assumption that the problems we are interested in are CPU constrained, not I/O constrained. There is significant research on clustering with (relatively) inexpensive distance measures on datasets that are too large to fit into main memory [3]. In contrast, our distance measure (DTW) is so expensive that even when the amount of data we have is trivially retained in main memory, the time needed to cluster may be on the order of weeks.

Assumption 2**:** Both a lower bound *and* upper bound to the true DTW distance can be calculated in a time that is negligible compared to the time taken to calculate DTW.

This assumption is not difficult to satisfy: while the time complexity for DTW is O(N^2), several lower bounds (LB_Keogh [16], LB_Kim [17], LB_Yi [43], etc.) are available which can be calculated in just O(N). Moreover, the Euclidean distance is an upper bound to the DTW, and it can also be calculated in O(N). As we will show in Section 4, a very accurate approximation to DTW can be obtained based on combining lower and upper bounds.

Assumption 3**:** DTW *can* produce superior clustering results for time series than the Euclidean distance.

Clearly if this assumption is not true, then we are wasting our time trying to cast DTW clustering into an anytime framework, since we should just do efficient clustering with the Euclidean distance. It has been shown recently that for one-nearest neighbor *classification*, on 38 diverse datasets, that DTW significantly outperforms Euclidean distance on a majority of datasets [5]. DTW achieves its robustness by allowing non-linear alignments between two time series, as shown in Figure 1.*left*.

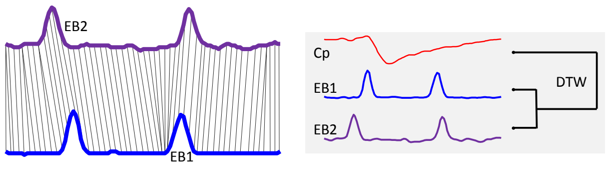

Figure 1: *left*) The DTW alignment between two Eclipsing Binary (EB) star light curves shows an intuitive peak-to-peak matching. *right*) Three star light curves clustered under the DTW distance with complete linkage. The two eclipsing binaries are correctly linked together, in contrast to Euclidean clustering (cf. Figure 2)

For such *similar* but locally out of phase time series, however, the Euclidean distance would report an unexpectedly large distance. We illustrate this in Figure 2.*left* for star light curves also shown in Figure 1.*left*. If we contrast the clusterings obtained in these two figures, we can gain some intuition as to the utility of DTW for clustering.

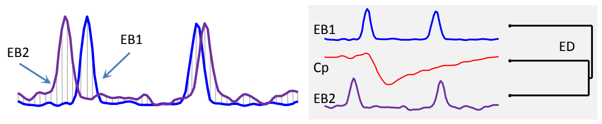

Figure 2: ***left*****) Time series are aligned one-to-one by the Euclidean distance, the distance reported is proportional to the length of the gray hatch lines.** ***right*****) Three star light curves clustered under the Euclidean distance with complete linkage are subjectively and objectively incorrect. We would have expected that the two Eclipsing Binaries (EB) would link before either linked with the Cepheid (Cp).**

However, it has also recently been shown that as datasets get larger, the difference in performance between DTW and Euclidean distance converges for one-nearest neighbor *classification* [32]. Presumably this is because an item to be classified is more likely to find a close match without the need for excessive warping as the dataset gets larger. Thus, while we can show that Euclidean distance can produce poor clustering results for *small* datasets, as in Figure 2.*right*, we must be careful assuming this can happen for larger datasets. This assumption is the only one not directly answered by the current literature. Therefore, we tested all datasets containing at least 2,000 time series from the UCR repository [38], which contains the majority of all publicly available, labeled time series datasets in the world. We applied three clustering algorithms to eleven large datasets under Euclidean and DTW distance respectively, and compared the results against the ground truth by *Adjusted Rand Index* [12]. The results are visually summarized in Figure 3.

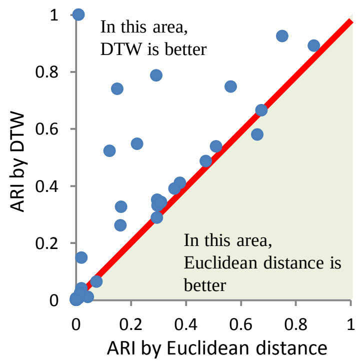

Figure 3: DTW vs. Euclidean distance tested using three types of clustering algorithms. The clustering results are measured by Adjusted Rand Index (ARI). Each point corresponds to one dataset with one clustering algorithm.

As we can see, DTW won the majority of times, and had a very close score even when it lost (cf. [36] for all raw numbers). Nevertheless, note that our assumption only postulates DTW's superiority for clustering of *some* real dataset(s); it is not necessary that DTW always outperforms Euclidean distance.

It is worth noting that once the above assumptions are satisfied, our anytime algorithm framework does *not* make any additional assumptions about the concrete clustering algorithms or objective functions that need to be optimized, and is thus very generally applicable.

2.2 Related Work

2.2.1 DTW Approximation

The time taken to compute the best alignment of two time series as shown in Figure 1.*left*, and thus the DTW *distance*, is O(N^2), we refer the reader to [16] and the references therein for more details on the DTW.

Because of the long known utility of DTW, and the fact that *exact* algorithm is intrinsically slow, there has been at least two decades of effort to improve its performance by *approximation* [4][29]. Note that these efforts are orthogonal to the efforts to create tight lower bounds of DTW [5]. While lower bounds can be seen as special cases of approximation, they are designed for a single purpose, allowing lower-bound based indexing [5][16][17][43]. These lower bound functions return a "distance" of zero for most sequences that are somewhat similar. This is not a problem within the context of a lower-bound based nearest neighbor search, but clearly lacks fine discrimination power for clustering.

Most of the work on approximating DTW for direct use in data mining algorithms leverages of the idea of doing DTW on a reduced dimensionality approximation [29], possibly at multiple levels of reduced dimensionality, using the results at a coarse level of approximation to seed the search at the next level [4]. This idea is attractive because if we down sample the data by a factor of *C*, the speedup obtained is approximately C^2.

However, all such methods are either still O(N^2) but with a lower constant factor, or O(N) but with such high constant factors that they may be slower than fast O(N^2) methods. In either case, none of methods we are aware of produce a good approximation to the true DTW in anything close to less than one-tenth of the time for full DTW algorithm. As we shall see in Section 4.1, our method can produce *very* accurate approximations to the true DTW for diverse datasets in a tiny fraction of the time needed for the full DTW algorithm.

2.2.2 Anytime Algorithms

As illustrated in Figure 4.*left*, anytime algorithms are algorithms that trade execution time for quality of results [45]. In particular, after some small amount of *setup-time* an anytime algorithm always has a *best-so-far* answer available, and the (expected) quality of the answer improves with execution time, until the anytime algorithm eventually terminates with the same answer that the batch algorithm would have achieved. Because data miners are interested in increasingly large datasets and, at least in some cases, increasingly complex analysis of these datasets, there has been a recent explosion of interest in using anytime (and anyspace [44]) algorithms for data mining [19][30][33][39][42].

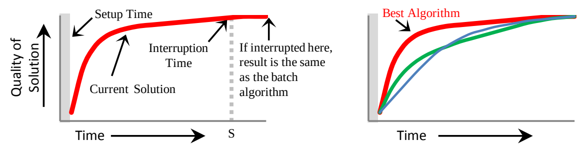

Figure 4: *left*) An abstract illustration of an anytime algorithm. Note that the quality of the solution keeps improving up to time S, when the algorithm is interrupted by the user. *right*) A comparison of possible performances of anytime algorithms. Here the top line shows an algorithm that dominates the other two at all times.

The most desirable properties of anytime algorithms have been outlined by Zilberstein and Russell [45]:

- **Interruptibility**: After some small amount of setup time, the algorithm can be stopped at any time and provide a tentative or partial answer.
- **Monotonicity**: The quality of the result is a non-decreasing function of computation time. Note that we desire this to be true *on average*; however, it is not necessary that on any particular run the quality must be strictly monotonic.
- **Measurable quality**: The quality of an approximate result can be determined. Note that it is only necessary that we can measure the quality in the same sense as we could measure the quality of the batch algorithm. Fully measuring the quality/validity of a clustering solution is still an area of active research [9].
- **Diminishing returns**: The improvement in solution quality is largest at the early stages of computation.
- **Preemptability**: The algorithm can be suspended and inspected, and then resumed with minimal overhead.
- **Low Overhead**: The time taken by the anytime algorithm to run to completion (assuming it is not interrupted) must be only slightly longer than the time taken by the batch algorithm.

Figure 4 illustrates many of these desirable properties. An obvious question is how can we compare the performance of several rival candidate anytime algorithms? In the best case, a single algorithm may dominate its rivals at every time point, as in Figure 4.*right* with the bold/red line. However, if we ignore this line, there are two hypothetical algorithms dominating each other at different times. In such a case, we would generally prefer the one which gains the most improvement at the beginning of the run.

2.2.3 *Anytime Clustering*

In contrast to anytime *classification* [19][22][30] [33][39][42], anytime *clustering* has gained much less attention from the data mining community. In [11] the authors proposed a technique called "active data clustering", which actively selects new distances to calculate, and uses tentative knowledge to estimate the relevance of missing data. However, they defined their own objective function and a special clustering method to minimize it, which limits its application. One of the most exciting ideas in this area recently appeared in [6], where the authors considered the problem of how to reconstruct the hierarchical clustering based on just a small subset of all pair-wise distances. The authors formally prove that if the "tight clustering" (TC) condition (the distance between two items in the same sub-tree is always smaller than the distance from each of them to any item out of the sub-tree) holds, the exact clustering of M items can be determined with $3M\log M$ distance calculations. Strictly speaking, their method is not presented as an anytime algorithm; however, it could trivially be seen as such, although the $3M\log M$ calculations that must be done would result in a long setup time. Moreover, it is not clear how the technique will work for non-metric distance functions (such as DTW), and we desire a more general technique that also allows *partitional* or *spectral* clustering.

Finally there are several research efforts that are titled *anytime* clustering, but are perhaps better understood as *incremental* clustering, suggesting techniques to *maintain* clusters in the face of newly arriving objects [18].

In summary, we are not aware of any generic techniques that allow for anytime hierarchical/ partitional/spectral clustering under expensive non-metric distance measures such as DTW.

2.2.4 *Quality Measurement of Clusterings*

We need to measure the quality of clustering in two contexts. First, we simply need to demonstrate the *monotonicity* and *diminishing returns* of our algorithm by creating plots like Figure 4. Second, to allow interactive clustering [14][31], it is useful (but not necessary) to have a quantitative measure of the current clustering.

The former requirement is easy to deal with. We can plot the quality of the *approximate* solutions at every stage of the anytime algorithm by comparing it to the final solution based on the *exact* DTW. After extensive empirical and theoretical evaluation, the authors of [20] recommended *Adjusted Rand Index* (ARI) as the most robust index to measure the similarity of two clusterings, and it has become one of the most widely adopted validation measurements [35]. The expected value of ARI is zero if the current solution is a random clustering, and it reaches its maximal value of one when the current clustering is identical to the final result. We refer the reader to [12] for more details of ARI.

While we will focus on ARI for the quality measurement in Section 6, to minimize the danger of producing optimistic results by using only a single measure, we also tested several other widely used measurements, such as the *cophenetic correlation coefficient* [34] which measures the linear correlation of pair-wise distances between two dendrograms, *normalized mutual information* [40] which compares two clusterings from an information theoretical perspective, etc. We find these measurements are highly related, and we thus mostly omit redundant plots for brevity. The interested reader can find *all* results in our supporting website [36].

As for the problem of evaluating the quality of clustering when a user preempts a run of the anytime algorithm, there are many possibilities available, from visual inspection (surprisingly scalable up to at least 10,000 objects for hierarchical clustering [31]) to various statistical tests [9].

3 ANYTIME CLUSTERING FRAMEWORK

The basic outline of our anytime clustering algorithm is described in Table 1. The first three lines in Table 1 correspond to the setup stage of the anytime algorithm (shown as the gray region in Figure 4). The algorithm can only be interrupted after the completion of this stage. As the reader may recall from the *interruptability property* (cf. Section 2.2), this stage should only last for a very short period of time. We first build an *a*pproximation of the DTW distance matrix[2] (*a*DTW) in line 1. As DTW is a symmetric measure, the matrix is saved as an upper triangular matrix in the row-major order. Once the distance matrix is available, clustering (either hierarchical, partitional, or spectral) is performed to obtain the first approximate solution we can report to the user (line 2).

Table 1. Basic framework for anytime clustering

```
Algorithm [Clusters] = AnytimeClustering(Dataset)
1   aDTW = BuildApproDistMatrix(Dataset);
2   Clusters = Clustering(aDTW,Dataset);
3   Disp('Setup is done, interruption is possible');
4
5   O = OrderToUpdateDTW(Dataset);
6   for i = 1:Length(O)
7       aDTW(O(i)) = DTW(Dataset,O(i));
8       if UserInterruptIsTrue()
9          Clusters = Clustering(aDTW,Dataset);
10         if UserTerminateIsTrue(Clusters)
11            return;
12         endif
13      endif
14  endfor
15  Clusters = Clustering(aDTW);
```

Lines 6 to 14 calculate the true DTW distances to *incrementally* replace the values in *a*DTW, in the order specified in *O* obtained in line 5. During this stage, clustering is performed (line 9) *only* if the user requests an answer (line 8). If the user interrupts the algorithm in this manner, she has a choice to terminate the algorithm after checking the current clustering result (line 10). This may happen because the approximate solution already satisfies the user's evaluation criteria; or in contrast, the user might believe that there is no hope to obtain a meaningful clustering even if the full DTW matrix had been calculated. For example, she sees a very poor clustering, and realizes that she forgot to normalize the data, or that the time series smoothing parameter she used was too aggressive, etc.

When all $(M \times (M-1))/2$ distances have been updated, the *a*DTW matrix becomes the true DTW matrix (*t*DTW), and we have the *same* answer (line 15) as the batch algorithm.

Given this framework, there are just two things we must define: how do we create the approximate DTW matrix *a*DTW required in line 1, and how we define the update ordering list *O* in line 5? Note that both these decisions are independent of the choice of *which* clustering algorithm the user will use (lines 2 and 9).

Before giving our solutions to these two sub-problems in the next two sections, we would like to emphasize that the proposed framework in Table 1 is generic enough to support virtually all distance-based clustering algorithms with little or no modifications.

4 APPROXIMATION OF THE DTW DISTANCE MATRIX

In order to build an approximation of the DTW distance matrix (*a*DTW) as required in line 1 of Table 1, we have many potential choices. We could initialize *a*DTW with the Euclidean distance, or with one of the many lower bounds to DTW proposed in the literature such as LB_Keogh [16], LB_Kim and LB_Yi [43], or with the DTW distance calculated on downsampled versions the time series [29], etc.

The most critical constraint on the plethora of choices we have is that the time to build *a*DTW must be a tiny fraction of the time to calculate all DTW distances; otherwise, this would impose a long setup time. Fortunately, as noted in Section 2.1, there are both lower and upper bounds available to DTW that can be computed in $O(N)$. Among the lower bounds, *LB_Keogh* (denoted as LB for short below) has been shown to be the tightest lower bound in [5] and elsewhere, and Euclidean distance (ED) is a tight upper bound to DTW and can also be computed in $O(N)$.

As illustrated in Figure 5, the DTW distance between any two sequences can be bound between LB and ED.

[2] Note that although we approximate the full distance matrix, our algorithm can also be applied to those clustering methods not requiring all pair-wise distances (once *Assumption 1* holds), where the algorithm can be terminated if every queried distance from *a*DTW is the exact distance.

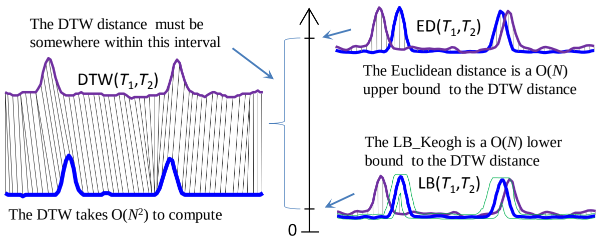
The DTW distance must be somewhere within this interval
DTW(T_1,T_2)
The DTW takes O(N^2) to compute
ED(T_1,T_2)
The Euclidean distance is a O(N) upper bound to the DTW distance
The LB_Keogh is a O(N) lower bound to the DTW distance
LB(T_1,T_2)
0

Figure 5: The DTW distance can be bounded by *fast-to-compute* upper and lower bounds.

While either bound could be *separately* used as approximations to DTW, a fundamental contribution of this work is to show that we can derive a much more accurate approximation by using these two bounds *together*. We propose to *learn* the best "mixing weight" of the upper/lower bounds by sampling a tiny fraction of the true DTW distances. We begin by defining the *DTW_Ratio* for a pair of time series (T_1, T_2) as:

$$DTW\_Ratio(T_1,T_2) = \frac{DTW\ (T_1,T_2) - LB\ (T_1,T_2)}{ED\ (T_1,T_2) - LB\ (T_1,T_2)} \quad (1)^3$$

DTW_Ratio(T_1,T_2) gives the relative position of DTW(T_1,T_2) between its lower and upper bounds. We can readily notice from (1) that it equals zero when DTW(T_1,T_2) = LB(T_1,T_2); it equals one when DTW(T_1,T_2) = ED(T_1,T_2); and the range of DTW_Ratio(T_1,T_2) is constrained to [0,1] (cf. Figure 5). Assume for the moment that the DTW_Ratio for two time series *is* known, and that we have calculated the corresponding LB and ED, then we can rearrange (1) to solve for the DTW distance:

$$\begin{aligned} DTW\ (T_1,T_2) &= LB(T_1,T_2) \\ &+ DTW\_Ratio(T_1,T_2) \times (ED\ (T_1,T_2) - LB(T_1,T_2)) \end{aligned} \quad (2)$$

Based on this we have transformed the problem of estimating DTW distance for each pair of time series to the problem of estimating the appropriate DTW_Ratio. Let us preview the results in the next section by considering three possibilities for the DTW_Ratio.

- If it is the case that the DTW_Ratio between any two pairs of time series is about the same value for any dataset, then we could find this value *once* and have a universal O(N) (since both LB and ED can be calculated in O(N)) estimator of DTW for any dataset. Regrettably, as we shall see, this is not true.
- If it is the case that the DTW_Ratio between any two pairs of time series is about the same value for a *particular* dataset, then we could learn that value for just that particular dataset, and have a domain specific O(N) estimator of DTW. Unfortunately, as we shall see, this is only *sometimes* true.
- If the DTW_Ratio between any two pairs of time series has some distribution that depends on LB and ED, and if we can cheaply estimate this distribution, we have a O(N) estimator of DTW for that domain.

[3] We define DTW_Ratio(T_1,T_2) = 0 if ED(T_1,T_2) - LB(T_1,T_2) = 0.

In the following section we flesh out these ideas.

4.1 Observations about the DTW_Ratio

Assume for the moment that we are only allowed to use a single value, r, to approximate *all* DTW_Ratios in a given dataset. We can use the ubiquitous *Root Mean Square Error* (*RMSE*) to measure the quality of the estimation:

$$RMSE = \sqrt{\frac{1}{\binom{M}{2}} \sum_{1 \le i < j \le M} (DTW\_Ratio(T_i,T_j) - r)^2} \quad (3)$$

RMSE has a unique minimum when setting r to the mean of all DTW_Ratios (also defined as the standard deviation of DTW_Ratios); however, how well this *one-size-fits-all* mean estimator works (i.e., how small *RMSE* is) depends on the distribution of DTW_Ratio values. The ideal case would be if they were all in a tightly restricted range, since the *mean* value would be a good approximation to *all* values. However, if the distribution of the DTW_Ratios is large, then the wide spread of their values will result in a lack of precision in the estimation of the DTW values.

To understand these distributions, we examined all large datasets from the UCR archive [38], plotting the histograms of all (M×(M-1))/2 DTW_Ratios for two of them in Figure 6.

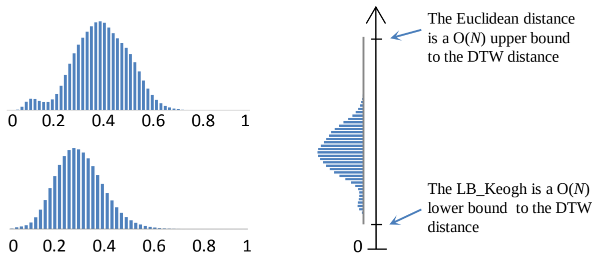

Figure 6: *left*) Histograms of DTW_Ratios of all pairs of time series for two large datasets from [38]. *left.top*) "Two Patterns" dataset containing 5,000 samples. *left.bottom*) "Face(all)" dataset containing 2,250 samples. *right*) The histograms can be seen in the context of the upper and lower bounds, see also Figure 5.

As we can see in Figure 6, in these two cases both distributions form a unimodal distribution with most of the values close to the mean (i.e, with low variance). We find that the *RMSE* of the *one-size-fits-all* mean estimator is 0.11 and 0.10, respectively. This result suggests that, in spite of its simplicity, a reasonable approximation technique is simply to use the mean. To summarize our findings thus far:

Observation 1**:** The *one-size-fits-all* mean estimator *can* be a good approximation to all DTW_Ratios from a single dataset.

Do the DTW_Ratio values always have a unimodal distribution with a small variation for all datasets?

Unfortunately we find this is not the case. We show two typical counterexamples in Figure 7:

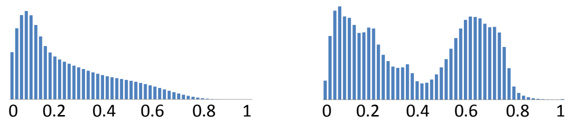

Figure 7: Histograms of DTW_Ratios of all pairs of time series for two large datasets, which are against *Observation 1*. *left*) "star light curves" dataset [27] (9,236 samples) *right*) "wafer" dataset [38] (7,174 samples)

As we can see, the histogram in Figure 7.*left* is heavily *right*-skewed and the one in Figure 7.*right* clearly forms a bimodal distribution. For such cases, the above *one-size-fits-all* mean estimator is much less accurate: their *RMSE* are 0.19 and 0.25 respectively.

To obtain some intuition as to how we might improve the current estimation model, let us consider a toy example. Suppose we have nine pairs of time series, whose DTW_Ratios are {0.3, 0.2, 0.4, 0.5, 0.4, 0.2, 0.3, 0.1, 0.3}. Using the *one-size-fits-all* mean estimator, we set *r* to their mean 0.3 and obtain a *RMSE* of 0.12, as illustrated in Figure 8.*a*.

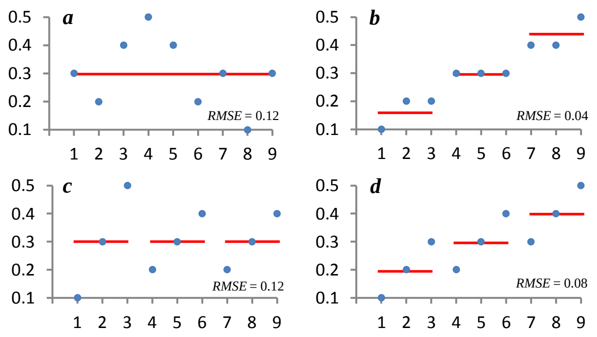

Figure 8: Illustration of different mean estimators. *a*) *one-size-fit-all* mean estimator. *b*), *c*) and *d*) are *multiple-for-all* (*k*=3) mean estimators based on *optimal*, *random* and *approximate* sorting, respectively.

Suppose instead that we are allowed to use *multiple*, say *k* (1 < *k* ≤ *n*), values to approximate *all* DTW_Ratios? In this case, all pairs of time series are divided into *k* equal-size[4] groups and each of the *k* groups is assigned its own estimated value *r*. If *k* = 3, it is simple to find the optimal grouping is {{0.1, 0.2, 0.2}, {0.3, 0.3, 0.3}, {0.4, 0.4, 0.5}}, which reduces the *RMSE* to 0.04 by setting each *r* to the mean value of the corresponding group, as shown in Figure 8.*b*.

Are we guaranteed to obtain a smaller *RMSE* if we use this *multiple-for-all* mean estimator? The answer is no. Figure 8.*c* shows that the grouping {{0.1, 0.3, 0.5}, {0.2, 0.3, 0.4}, {0.2, 0.3, 0.4}} generates the same *RMSE* as the *one-size-fits-all* mean estimator. The reader may have noticed that the difference between the above two groupings is that the optimal one (Figure 8.*b*) is based on the *sorted* DTW_Ratios. It is easy to see why the sorting-based grouping works better, since the sorting puts closer values into the same group, and thus reduces their differences to the mean of the group. However, the actual values of DTW_Ratios are unknown (recall that our task is to *estimate* them). It seems that the sorting heuristic cannot be applied because of this chicken-and-egg paradox.

[4] A size-adaptive division is also possible; we use equal-size for simplicity because it achieves very good approximation (see 6.1) and tentative tests suggest the room for improvement is marginal.

Perhaps all is not lost; we can ask: could an *approximate* sorting of DTW_Ratios still help? For clarity, consider an approximate sorting based grouping for our toy example: {{0.1, 0.2, 0.3}, {0.2, 0.3, 0.4}, {0.3, 0.4, 0.5}}, as shown in Figure 8.*d*. It is clear that, although approximate, the sorting still helps to make values in the same group closer, and therefore the *RMSE* is reduced to 0.08.

Fortunately, we can obtain a good approximate sorting of DTW_Ratios based on only ED and LB. We first divide the numerator and denominator of the right part of (1) by $LB(T_1,T_2)$ to obtain:

$$DTW\_Ratio(T_1,T_2) = \frac{DTW\ (T_1,T_2)\,/\,LB\ (T_1,T_2) - 1}{ED\ (T_1,T_2)\,/\,LB(T_1,T_2) - 1} \quad (4)$$

From (4), it is clear that the DTW_Ratio depends on two distinct quantities: the ratios DTW(T_1,T_2)/LB(T_1,T_2) and ED(T_1,T_2)/LB(T_1,T_2). To explore their properties, we randomly sampled 1,000 pairs from the "star light curves" dataset and plotted their LB distances in increasing order together with respective DTW and ED values in Figure 9.

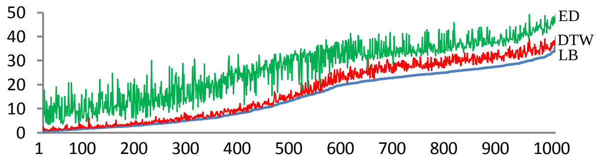

Figure 9: ED, DTW and LB of 1,000 randomly sampled pairs of time series from "star light curves" dataset, in the increasing order of LB.

We can observe that:

- LB is a tighter bound to DTW than ED, and thus DTW/LB is usually much smaller than ED/LB.
- The curve of DTW is smoother than the curve of ED, which demonstrates that the variance of DTW/LB is lower than ED/LB (both curves are ordered by LB).

This pattern is observed in almost all large datasets we have checked (see also Figure 11). Therefore, the DTW_Ratio in (4) tends to be largely determined by the denominator term ED(T_1,T_2)/LB(T_1,T_2), and we can use this intuition to approximately sort the DTW_Ratios.

We tested the *multiple-for-all* mean estimator with 100 groups based on the approximate sorting, and compared the result to the *one-size-fits-all* mean estimator. *RMSE* is reduced from 0.19 to 0.12 for the "start light curves" dataset and from 0.25 to 0.13 for the "wafer" dataset[5]. Before moving on, we summarize our new findings about DTW_Ratio as follows:

***Observation 2*:** We can further reduce *RMSE* by the *multiple-for-all* mean estimator based on the approximate sorting of DTW_Ratios.

4.2 Estimating the DTW_Ratio

For the *multiple-for-all* mean estimator proposed above, we had set the number of groups to 100. This is the *only* important parameter of our approximation method. To explore its sensitivity to *RMSE*, we tested on all four datasets discussed in Section 4.1 using different numbers of groups. Figure 10 shows the results:

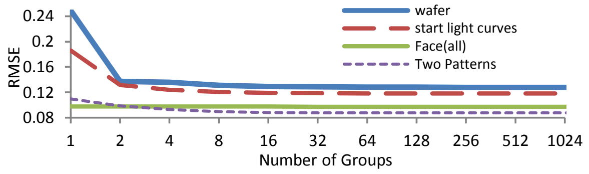

Figure 10: RMSE vs. Number of Groups. Tested on four large datasets using different number of groups (one group corresponds to the *one-size-fits-all* mean estimator). RMSE varies little when number of groups is more than 16 for all datasets. Note log scale in x axis.

The plot clearly suggests that the number of groups is not critical to the estimation error of our *multiple-for-all* mean estimator, once it is larger than, say 16. Therefore, for the rest of this paper, we have fixed the number of groups to 20.

There is one more issue to resolve the mean estimator: how do we obtain the mean of DTW_Ratios for each group? We achieve this by *sampling*. Due to the central limit theorem, the mean of a large collection of DTW_Ratios is approximately normally distributed, thus the mean of *randomly sampled* DTW_Ratios follows a Student's t distribution. A t-test can be applied to tell if the true mean falls in any given confidence interval. For all the remaining experiments in this paper, we sample DTW_Ratios *incrementally* until the true mean falls in [sample mean-0.01, sample mean+0.01] with a 90% confidence.

As we shall show in Section 6.1, the number of sampled DTW_Ratios (i.e., full DTW calculations) is only a tiny portion for all datasets we tested. Furthermore, the property of t-test tells us that the number of sampled DTW_Ratios for a given confidence interval is *only* related to the standard deviation of the sampled DTW_Ratios, *not* the size of the data set.

5 GENERAL ORDERING HEURISTICS

Having initialized with a good approximation of the DTW distance matrix, the next step is to update each value in *a*DTW, one-by-one, till it becomes *t*DTW. As we noted in Section 3, we would like to find a good updating order *O* which is *general* enough to be applied to any clustering algorithm. Therefore, we will focus on how to reduce the approximation error of *a*DTW quickly. Generally, we expect the approximate clustering result to be closer to the final clustering result if *a*DTW is more similar to *t*DTW.

We first define the Normalized DTW Approximation Error (*NDAE*) for a given pair of time series (T_1, T_2) as:

$$NDAE(T_1,T_2)=\frac{|\,DTW\ (T_1,T_2)-DTW'\ (T_1,T_2)\,|}{DTW\ (T_1,T_2)} \quad (5)$$

$NDAE(T_1,T_2)$ equals zero if the approximation distance DTW'(T_1,T_2) is identical to the DTW(T_1,T_2), in which case we do not need to update the distance for that pair, while we should give priority to update DTW for those pairs with a larger *NDAE* . It is difficult to order *NDAE* based on (5), thus we replace DTW in (5) by (2), and obtain another equivalent expression for $NDAE(T_1,T_2)$:

$$\frac{|\,DTW\_Ratio\ (T_1,T_2)-DTW\_Ratio'\ (T_1,T_2)\,|}{1/(ED(T_1,T_2)/LB(T_1,T_2)-1)+DTW\_Ratio\ (T_1,T_2)} \quad (6)$$

Let us consider the numerator / denominator:

- We cannot compare the *numerator* for pairs of time series (since the DTW_Ratio(T_1,T_2) is unknown); however, based on tests in Section 4.1, its expected value is very small.
- We can approximately sort the *denominator*. Since ED and LB are known, and as we have previously shown (in Section 4.1), a pair of time series with a larger ED/LB is more likely to have a smaller DTW_Ratio.

Therefore, we can make a similar claim that a pair of time series with a larger ED/LB is more likely to have a larger *NDAE*, and therefore should be given priority to update its true DTW distance.

We have compared our proposed ordering technique to a dozen alternatives: random ordering, smallest/largest value in *a*DTW first, items from lowest/highest variance column in *a*DTW first, etc. Surprisingly, considering all datasets and clustering algorithms, random ordering is a very competitive technique, and thus we use it as our straw man technique in the next section.

6 EMPIRICAL EVALUATION

To obtain a thorough evaluation of our ideas, we:

- Tested on all eleven datasets containing at least 2,000 time series from the UCR Time Series repository [38], the world's largest collection of time series datasets.
- Evaluated the similarity of *133,601,661* (non-duplicated) pairs of time series from diverse

[5] We did extensive experiments on diverse large datasets, and *all* of them gained from the approximate sorting. Due to the limited space, all results are archived at [36].

domains, such as astronomy, sensor networks, robotics, etc.

- Applied three different *types* of clustering algorithms (hierarchical, partitional and spectral), and measured their performance by 5 indices.
- Created a website [36] which contains all datasets and code used in this work, together with spreadsheets which contain the raw numbers displayed in all the figures. In addition, the website contains additional experiments which we could not fit into this work (including a YouTube video illustrating the utility of our ideas: http://youtu.be/43nKmEuum2c).

6.1 DTW Approximation

We begin our experiments with a comparison of the mean *NDAE* (the smaller the better, see (5)) of all pair-wise approximations by three methods: LB, ED and our proposed DTW_Ratio-based approximation (*DTW_Ratio* for short below). Recall that the latter needs to randomly sample a tiny fraction of true DTW distances. To be fair to the other methods, we also updated those sampled DTW for LB and ED. For each dataset, we calculated mean *NDAE* of all (M×(M-1))/2 pairs approximations, repeating each test ten times, and reported the mean in Figure 11:

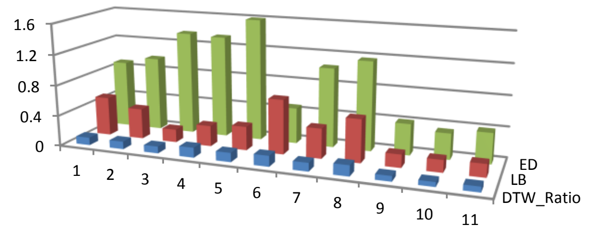

Figure 11: Mean of *NDAE* of all pair-wise approximations by three DTW approximation methods, tested on eleven large datasets. Smaller values are better. The first four datasets are discussed in Section 4 and [36] contains a description/key for all datasets.

The dominance of DTW_Ratio is quite obvious in the above figure. Notably, the mean *NDAE* (averaged on all datasets) by DTW_Ratio is just *0.1*, which means the expected error of our approximation is only *one-tenth* of the true DTW. The results bode well for our proposed anytime algorithm that exploits this approximation.

Having shown that we can estimate the initial *a*DTW very accurately, we are ready to evaluate two desirable properties (cf. Section 2.2) of the anytime algorithm.

The first one is the *interruptibility*, here in terms of how long it takes to obtain the first *a*DTW. While LB and ED can be calculated quickly for all pairs, the number of sampled DTW distances must be a small number. For the above tests, the sampling rate for each dataset varies from 0.02% to 0.25%, with an average of just 0.1%. Furthermore, as analyzed in Section 4.2, the number of DTW calculations required by sampling will not increase as the dataset gets larger. In fact, the largest sampling rate above comes from the smallest dataset we tested. To give the reader a concrete sense of how much speed up we can gain from the approximation, we consider the "star light curve" as an example. The time to calculate the full distance matrix of ED and LB is 23 minutes and 116 minutes, respectively. Just 7,760 (averaged over 10 runs) DTW distances are sampled from all 42,647,230 possible pairs by our initial approximation, which takes about 33 minutes. So, the overall time the user has to wait for *a*DTW to be built is less than 3 hours, in contrast to the 127 days for the batch algorithm. We archived full experimental results on all eleven datasets at [36], showing a speed-up from 113 times to more than 1000 times.

The second property is the *low overhead*. Compared to the batch algorithm, we have to calculate two more distance matrices of LB and ED. However, as shown above, this overhead time is inconsequential (just 0.08% of the time of the batch algorithm for the "star light curve" dataset).

6.2 Anytime Clustering

In this section we focus on evaluating the *monotonicity* and *diminishing returns* properties (cf. Section 2.2) of the anytime algorithm. We will also demonstrate the generality of our anytime clustering framework (Table 1) by using different types of clustering algorithms.

6.2.1 Hierarchical Clustering

The output of hierarchical clustering is a tree structure called a dendrogram, which allows the user to view the clusters at different granularities. Hierarchical clustering has been widely used in a variety of research and application domains. For example, the authors of [31] have shown that hierarchical clustering is an effective tool for microarray data analysis to identify similar genes in large datasets.

Here we considered the agglomerative hierarchical clustering with average linkage (UPGMA), which begins with M clusters and merges two most similar clusters in each step till all items are merged.

For tests on each dataset, we first initialized *a*DTW with ED, and compared four combinations of strategies: updating *a*DTW by DTW_Ratio or not (line 1 of Table 1), and arranging DTW updating order by ED/LB or randomly (line 5 of Table 1). The quality of the current clustering is measured at the beginning (no DTW calculations), just after the *setup* is complete (cf. Section 2.2.2); when a tiny fraction[6] of the DTW distances have been sampled; and at 10% steps as replacing distances in *a*DTW with the true distances. We repeated all four

[6] We pick 0.5% for consistency and simplicity, because the sampling rate of DTW_Ratio varies for each dataset and each run (from 0.02% to 0.25%).

anytime strategies on each dataset ten times, and report the mean values. Figure 12 shows the results for the "Two Patterns" dataset.

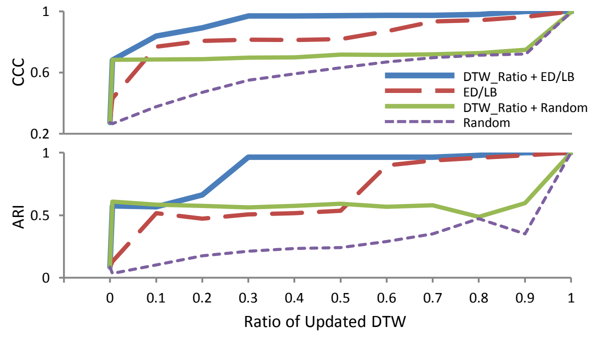

Figure 12: Quality of anytime hierarchical clustering on the "Two Patterns" dataset. Four anytime strategies are compared by Cophenetic Correlation Coefficient (*top*) and Adjusted Rand Index (*bottom*).

We can see from Figure 12 that our championed strategy "DTW_Ratio (approximation) + ED/LB (updating order)" (the top line) beats all other three at almost all time steps. The second observation is that the two techniques using DTW_Ratio achieve a huge improvement very early, when just 0.5% of all DTW calculations have been performed. In contrast, the clustering quality of "Random" (the bottom line) is still very poor even after it has calculated 90% of all DTW distances. Another interesting finding is that the "ED/LB" updating order still helps without the boosting by DTW_Ratio (the thicker dashed line); its clustering quality surpasses or becomes very close to "DTW_Ratio + Random" after 10% of DTW calculations.

Because the performance of "ED/LB" and "DTW/Ratio + Random" almost always resides between the other two, to enhance the visual clarity, we will not show their results in the rest of this paper. Moreover, as different measurements (*CCC* and *ARI* as shown in Figure 12, and see other three[7] at [36]) are often highly related, we will only show one measure in the plots below to reduce redundancy. The reader can find the full (*all* strategies compared by *all* measurements on *all* eleven datasets) results at [36].

For four datasets discussed in Section 4, we show their evaluation results measured by *CCC* in Figure 13. We choose *CCC* because it considers all levels of dendrogram. We see very similar results. "DTW_Ratio + ED/LB" achieves almost all possible improvement at the very beginning and dominates "Random" all the time. If we compare the first check point of four datasets, we can find that the scores for "Two Patterns" and "Face(all)" are much lower than those of the other two (notably, also holds for other two clustering algorithms, see Figure 14 and Figure 15).

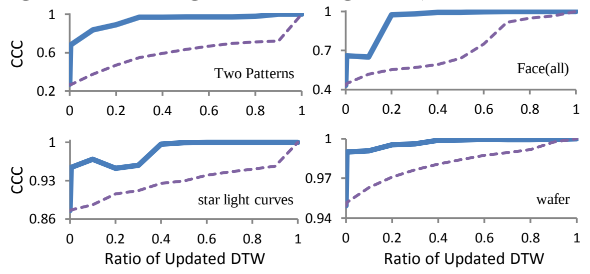

Figure 13: Quality of anytime hierarchical clustering on four large datasets (note we include "Two Patterns" shown in Figure 12 for completeness). "DTW_Ratio + ED/LB" (the solid/top line) beats "Random" (the dashed/bottom line) all the time for all datasets.

This is probably due to the "highly-warping" property of these two datasets, making the clustering solutions by ED and DTW quite different.

6.2.2 K-medoids Clustering

K-medoids clustering belongs to another category of clustering called *partitional* clustering, which tries to find the best *K* partitions of the dataset. Instead of averaging data points (which is still an open problem under warping [24]) as K-means, K-medoids chooses a data point from the cluster as its centroid (called medoid). There exist several variations of K-medoids based algorithms, and we have chosen the classic PAM algorithm [15], since it does not require any parameter settings beyond *K*. Although it is slower than hierarchical and spectral clustering (as discussed below), taking minutes rather than seconds, it is still dwarfed by the DTW calculations (thus, *Assumption 2* holds).

The quality of anytime K-medoids clustering measured by *ARI* (*CCC* is only defined for hierarchical clustering) on the same four datasets is compared in Figure 14 (similar results on all other datasets can be found at [36]):

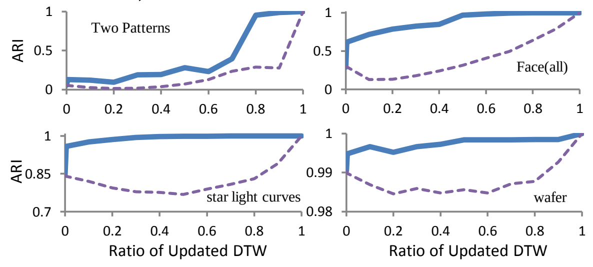

Figure 14: Quality of anytime K-medoids clustering on four large datasets. "DTW_Ratio + ED/LB" (the solid/top line) always beats "Random" (the dashed/bottom line).

Again, we see the dominance of "DTW_Ratio + ED/LB" in all four datasets. However, we do find it does not work well enough for the "Two Patterns" dataset if we consider the *diminishing returns*. We take some comfort in noting this is the only synthetic dataset among the four. Moreover, we have done additional

[7] Including several measurements which do *not* require the final exact DTW distance matrix, hence we have a heuristic to suggest when to stop the anytime algorithm.

studies to further understand this outlying example, but pushed our tentative explanations to [36] for brevity. Another surprising observation is that for “Random”: the quality of clustering can decrease even after calculating more than 50% of DTW distances.

6.2.3 *Spectral Clustering*

The final clustering technique we considered is spectral clustering, which transfers the original data into a new space that is more amiable to clustering. Among various implementations, we tested the most popular, *normalized spectral clustering* by Ng et al. [23].

Figure 15 shows results of anytime spectral clustering:

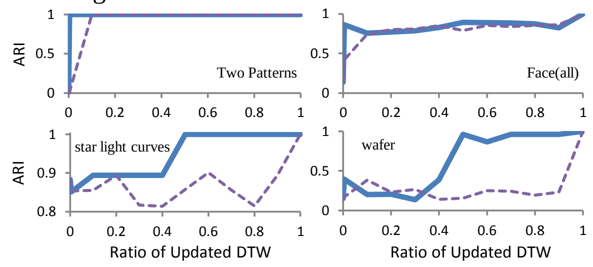

Figure 15: Quality of anytime spectral clustering on four large datasets. “DTW_Ratio + ED/LB” (the solid line) still shows a better performance than “Random” (the dashed line) for all datasets.

In this set of experiments, two strategies appear closer performance than in the above two clustering algorithms we have tested. This might be because the similarity graphs in the spectral clustering only model local neighborhood relationships [1], and therefore less DTW updating is required. For example, in the top two plots of Figure 15, even for “Random” its *ARI* increases to an almost perfect score after 10% of DTW calculations, while this happens even faster for “DTW_Ratio + ED/LB”. However, the latter is still favorable especially when the time to calculate the full DTW distance matrix is long. For example, it took 15 hours for the batch algorithm to calculate the DTW distance matrix for the “Two Patterns” dataset, so 10% of the calculation would still take as long as one and a half hours, while our DTW_Ratio-based approximation only took 7 minutes. Note that clusterings based on these three methods are almost identical (see the top left of Figure 15). We did the same experiment on a larger “Two Patterns” dataset with 25,000 time series: In this case the DTW_Ratio approximation reduced the time from 15.7 days (by the batch algorithm) to 2.8 hours, while still generating a *very* similar clustering result (see a visual demonstration as a video or a sequence of high resolution images at [36]).

7 DISCUSSION AND CONCLUSIONS

While we have focused on anytime *clustering* of *time series* using *DTW* in this work, we believe that our ideas may have applications to other expensive distance measures for time series, such as LCSS, EDR, ERP, SpADE and Swale [5], so long as both an upper and lower bound can be defined for them. However, we do not consider these in great detail for two reasons. First, it allows a simpler and more concrete exposition of our ideas, and second, a recent extensive empirical study has suggested that DTW is at least competitive with all other measures for the highly related problem of nearest neighbor *classification* [5].

Likewise, our ideas may have utility for clustering other data types, where the most effective distance measure is expensive, but inexpensive upper and lower bounds are available. Such examples could include strings, where quadratic-time edit distance is often the best measure, and simple bounds exist (The distance is always *at least* the difference of the lengths of the two strings. It is *at most* the length of the longer string), or Earth Movers Distance which is quadratic but has tight bounds available [2].

Beyond clustering, our distance matrix that can be built in an anytime fashion can also be utilized in other problems. For example, in [21] all pair-wise correlations/distances for tens of thousands of time series from data warehouses are needed for discovery of patterns and anomalies in an application in data center management. This problem may be amiable to a very similar anytime framework. Again we leave such considerations to future work in order to do the special case justice in both presentation and evaluation.

We have shown the first example of an anytime clustering algorithm for large datasets under the DTW measure. Our algorithm has an inconsequential cost in terms of additional time, but allows us to achieve most of the benefit of the batch algorithm in a just a tiny fraction of the time. We have tested on datasets with up to 25,000 objects, which is at least an order of magnitude larger than any other time series clustering efforts that we are aware of. Finally we have made all our code and data freely available to allow replication, extension and adoption of our ideas.

8 REFERENCES

[1] von Luxburg, U. 2007. A Tutorial on Spectral Clustering. Statistics and Computing 17(4).

[2] Assent, I., Wichterich, M., Meisen, T. and Seidl, T. 2008. Efficient similarity search using the Earth Mover's Distance for large multimedia databases. ICDE 2008: 307-16.

[3] Bradley, P., Fayyad, U. and Reina, C. 1998. Scaling clustering algorithms to large databases. In Proceedings of the 4th KDD, pp 9-15.

[4] Chu, S., Keogh, E., Hart, D. and Pazzani, M. 2002. Iterative Deepening Dynamic Time Warping. In 2nd SIAM International Conference on Data Mining.

[5] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X. and Keogh, E. 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. PVLDB 1(2) :1542-1552.

[6] Eriksson, B., Dasarathy, G., Singh, A. and Nowak, R. 2011. Active Clustering: Robust and Efficient Hierarchical Clustering using Adaptively Selected Similarities. 14th ICAIS.

[7] Grass, J. and Zilberstein,S. 1996. Anytime algorithm development tools. SIGART Artificial Intelligence. Vol 7, No. 2, ACM Press.

[8] Großmann, A., Wendt, M. and Wyatt, J. 2003. A Semi-supervised Method for Learning the Structure of Robot Environment Interactions. IDA 2003: 36-47

[9] Halkidi, M., Gunopulos, D., Vazirgiannis, M., Kumar, N., Domeniconi, C. 2008. A clustering framework based on subjective and objective validity criteria. TKDD 1(4).

[10] Hamagami, T. and Hirata, H. 2005. State space partitioning and clustering with sensor alignment for autonomous robots. IEEE Intl Conference on Systems, Man and Cybernetics: 2655–60.

[11] Hofmann, T. and Buhmann, M.J. 1998. Active data clustering. In Advances in Neural Information Processing Systems, 10, 528-534.

[12] Hubert, L. and Arabie, P. 1985. Comparing partitions. Journal of Classification, pp 193–218.

[13] Innis, J. L.; Heil, P.; Thompson, K.; Coates, D. W. 2004. An Historical Light Curve of CF Octantis from Digitised Images of the Bamberg Plate Archive. PASA 21:3, pp. 284-89.

[14] Jiang, D., Pei, J. and Zhang, A. 2003. Towards interactive exploration of gene expression patterns. ACM SIGKDD Explorations, 79–90.

[15] Kaufman, L. and Rousseeuw, P.J. 1990. Finding Groups in Data: an Introduction to Cluster Analysis. Wiley.

[16] Keogh, E. 2002. Exact indexing of dynamic time warping. In 28th International Conference on Very Large Data Bases. pp 406-417

[17] Kim, S., Park, S. and Chu, W. 2001. An index-based approach for similarity search supporting time warping inlarge sequence databases. In: Proceedings of the 17th ICDE, pp 607–614.

[18] Kranen, P., Assent, I., Baldauf, C. and Seidl, T. 2009. Self-Adaptive Anytime Stream Clustering. ICDM 2009.

[19] Kranen, P., Günnemann, S., Fries, S. and Seidl, T. 2010. MC-Tree: Improving Bayesian Anytime Classification. SSDBM 2010: 252-269.

[20] Milligan, G. and Cooper, M. 1986. A study of the comparability of external criteria for hierarchical cluster analysis. Multivariate Behavioral Research, 21:441-458.

[21] Mueen, A., Nath, S. and Liu, J. 2010. Fast approximate correlation for massive time-series data. In the Proceedings of ACM SIGMOD 2010. pp. 171-182.

[22] Myers, K., Kearns, M. J., Singh, S. P. and Walker M. A. 2000. A Boosting Approach to Topic Spotting on Subdialogues. ICML 2000.

[23] Ng, A., Jordan, M., and Weiss, Y. 2001. On spectral clustering: analysis and an algorithm. NIPS 2001.

[24] Niennattrakul, V. and Ratanamahatana, C.A. 2007. On Clustering Multimedia Time Series Data Using K-Means and Dynamic Time Warping. MUE 2007.

[25] Oates, T., Schmill, M., and Cohen, P. 2000. A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgments. AAAI/IAAI: 846-851.

[26] Ratanamahatana, C. and Keogh, E. Three Myths about Dynamic Time Warping Data Mining. SDM 2005.

[27] Rebbapragada, U., Protopapas, P., Brodley., C. E. and Alcock, C. 2009. Finding Anomalous Periodic Time Series: An Application to Catalogs of Periodic Variable Stars, Machine Learning, Vol. 74, Iss 3.

[28] Sart, D., Mueen, A., Najjar, W., Niennattrakul, V. and Keogh, E. 2010. Accelerating Dynamic Time Warping Subsequnce Search with GPUs and FPGAs. ICDM 2010.

[29] Salvador, S. and Chan, P. 2004. FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. KDD Workshop on Mining Temporal and Sequential Data, pp. 70-80.

[30] Seidl, T., Assent, I., Kranen, P., Krieger, R. and Herrmann, J. 2009. Indexing density models for incremental learning and anytime classification on data streams. EDBT 2009: 311-322.

[31] Seo, J. and Shneiderman, B 2002. Interactively Exploring Hierarchical Clustering Results. IEEE Computer 35(7): 80-86.

[32] Shieh, J. and Keogh, E. 2008. *i*SAX: Indexing and MiningTerabyte Sized Time Series, SIGKDD. p.623-31.

[33] Shieh, J. and Keogh, E. 2010. Polishing the Right Apple: Anytime Classification Also Benefits Data Streams with Constant Arrival Times. ICDM 2010.

[34] Sokal, R. and Rohlf, F.J. 1962. The Comparison of Dendrograms by Objective Methods. Taxon 11: 33-40.

[35] Steinley, D. Properties of the Hubert-Arabie adjusted Rand index. Psychol Methods, 2004, 9(3): 386-96.

[36] Supporting website. *www.cs.ucr.edu/~qzhu/anytimeclustering/*

[37] Templeton, M. 2001. Archival Data Digitization -- Work In Progress. accessed 6/4/11. www.aavso.org/archival-data-digitization-work-progress/

[38] Keogh, E., Zhu, Q., Hu, B., Hao. Y., Xi, X., Wei, L. and Ratanamahatana, C. A. 2011. The UCR Time Series Classification/Clustering Homepage. *www.cs.ucr.edu/~eamonn/time_series_data/*

[39] Ueno, K., Xi, X., Keogh, E. and Lee, D.J. 2006. Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining. ICDM 2006.

[40] Witten, I.H. and Frank, E. 2005. Data Mining: Practical Machine Learning Tools and Techniques.

[41] Xi, X., Keogh, E., Shelton, C., Wei, L. and Ratanamahatana, C.A. 2006. Fast time series classification using numerosity reduction. In Proc of the 23rd ICML. p.1033-1040.

[42] Yang, Y., G.I. Webb, K. Korb, and K-M. Ting 2007. Classifying under Computational Resource Constraints: Anytime Classification Using Probabilistic Estimators. Machine Learning 69(1). pp. 35-53.

[43] Yi, B., Jagadish, H. and Faloutsos, C. 1998. Efficient retrieval of similar time sequences under time warping. In:ICDE 98, pp 23–27

[44] Ye, L., Wang, X., Keogh, E. and Mafra-Neto, A. 2009. Autocannibalistic and Anyspace Indexing Algorithms with Application to Sensor Data Mining. SDM 2009.

[45] Zilberstein, S. and Russell, S. 1995. Approximate reasoning using anytime algorithms. In Imprecise and Approximate Computation, Kluwer Academic Publishers..