Accelerating Dynamic Time Warping Clustering with a Novel Admissible Pruning Strategy

Nurjahan Begum Liudmila Ulanova Jun Wang[1] Eamonn Keogh
University of California, Riverside University of Texas at Dallas[1]
{nbegu001, lulan001, eamonn}@cs.ucr.edu wangjun@utdallas.edu[1]

ABSTRACT

Clustering time series is a useful operation in its own right, and an important subroutine in many higher-level data mining analyses, including data editing for classifiers, summarization, and outlier detection. While it has been noted that the general superiority of Dynamic Time Warping (DTW) over Euclidean Distance for *similarity search* diminishes as we consider ever larger datasets, as we shall show, the same is not true for *clustering*. Thus, clustering time series under DTW remains a computationally challenging task. In this work, we address this lethargy in two ways. We propose a novel pruning strategy that exploits both upper and lower bounds to prune off a large fraction of the expensive distance calculations. This pruning strategy is admissible; giving us provably identical results to the brute force algorithm, but is at least an order of magnitude faster. For datasets where even this level of speedup is inadequate, we show that we can use a simple heuristic to order the unavoidable calculations in a *most-useful-first* ordering, thus casting the clustering as an anytime algorithm. We demonstrate the utility of our ideas with both single and multidimensional case studies in the domains of astronomy, speech physiology, medicine and entomology.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database Application – *Data Mining*

Keywords

Clustering; Time Series; Anytime Algorithms

1. INTRODUCTION

Given the ubiquity of time series data in scientific, medical and financial domains, the research community has made substantial efforts to create efficient algorithms for classification, clustering, rule discovery, and anomaly detection for this data type [1] [3][11][16][27]. In particular, time series *clustering* is useful, both as an exploratory technique and also as a sub-module for solving higher-level data mining problems. As a concrete example, consider Figure 1, which illustrates a subset of a cluster we discovered in a social media dataset [32]. This clustering allows us to at least partly address two problems:

- **Synonym Discovery**: In this example, we have the hashtag *#Michael*. It is not clear to whom this refers to: Michael Phelps? Michael Caine? However, by noting that this cluster also contains *#MichaelJackson*, this ambiguity is resolved.

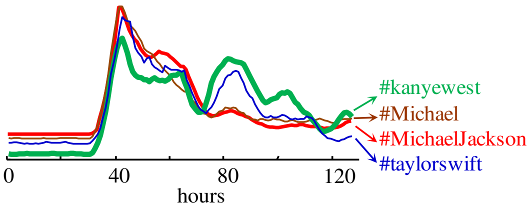

Figure 1: A cluster of four Twitter hashtag usage time series (normalized for volume) over ~6 days starting from June 12, 2009 [32]. (Best viewed in color.)

- **Association Discovery**: Here we see that *#kanyewest* and *#taylorswift* have highly similar time series representations, but are clearly not synonyms. If we test to see whether this relationship existed prior to the illustrated timeframe, we find it does not. This suggests the existence of an *event* that caused this temporary association, and with a little work we can discover the famous "*I'mma let you finish*" event at the 2009 Video Music Awards [35].

In this example, the knowledge gleaned is clearly trivial; however, similar ideas have been used to track the levels of disease activity and public concern during the recent influenza A H1N1 pandemic [26]. Note that while we discovered this example using DTW, it might have been discovered with the computationally efficient Euclidean distance. However, in some cases there may be a *causal relationship* (rather than just an *association*) between events, resulting in a local lag between peaks. DTW is an ideal way to capture/be invariant to such out-of-sync relationships.

We begin by demonstrating that the problem we plan to address, robustly clustering large time series datasets with invariance to irrelevant data, has not been solved in previous work.

For most time series data mining algorithms, the quality of the output depends almost exclusively on the distance measure used [27]. A consensus has emerged that the Dynamic Time Warping (DTW) distance measure is the best in most domains, almost always outperforming the Euclidean Distance (ED) [27]. As a concrete example, consider the two clusterings of three randomly chosen mammals shown in Figure 2. The input data is the mitochondrial DNA after it was converted to a time series representation (converting DNA to time series is a commonly used operation [18][19]). Two types of DNA mutations, *insertions* and *deletions*, have the effect of "warping" the time series. At least in this case, we can see that DTW is invariant to these mutations and correctly unites *Bos taurus* (cattle) and *Hyperoodon ampullatus* (bottlenose whale), with *Talpa europaea* (mole) as the out-group.

KDD '15, August 11 - 14, 2015, Sydney, NSW, Australia
© 2015 ACM. ISBN 978-1-4503-3664-2/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2783258.2783286

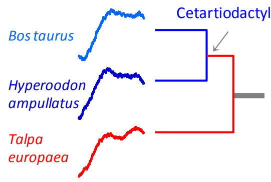

Figure 2: Single-linkage hierarchical clusterings of DNA using DTW (*left*) and Euclidean distance (*right*).

While this example[1] is on a small and somewhat specialized dataset, in Section 6 we will show that the superiority of DTW extends to large datasets in many domains.

1.1 Why This Problem Is Hard

Given that DTW is intrinsically slow because of its quadratic time complexity, there are two ideas that are commonly used to mitigate the problem of using such a slow distance measure [19]. We briefly discuss them here only to dismiss them as solutions to the task at hand.

- *The convergence of DTW and Euclidean distance results for increasing data sizes.* It has been noted that for many problems, including motif discovery [16] and classification [27], the results returned by DTW and Euclidean distance tend to become increasingly similar as the dataset sizes increase. This suggests that we can simply use the more efficient Euclidean distance to cluster large datasets.
- *The increasing effectiveness of lower-bounding pruning for increasing data sizes.* For some problems, notably similarity search, the lower-bounding pruning of unnecessary calculations is the main technique used to produce speedup. The effectiveness of this lower-bounding tends to *improve* for larger datasets [19].

Unfortunately, neither of these observations helps us for *clustering* under DTW. To demonstrate why the first observation does not help, we performed a simple experiment in which we measured the leave-one-out training error of 1NN classification using both DTW and ED, for various numbers (50 to 2000) of exemplars from the CBF dataset [13]. With 50 objects, the error rates differ by a factor of 4.6 (7% and 1.75%, respectively), but as shown in Figure 3.*top*, by the time we consider the 2000 object dataset, this difference is essentially zero.

This effect is well known for time series *classification* [22][27], and it might be imagined that this applies to *clustering*. To show that this is not the case, we performed a parallel experiment in which we *clustered* the same objects and measured the performance using the Rand Index [21]. As shown in Figure 3.*bottom*, DTW *clustering* maintains its superiority over Euclidean distance as the datasets get larger.

Similarly, the second observation above does not help significantly. It is true that lower bounds are increasingly effective for larger datasets when attempting a *similarity search*. This is because for larger datasets, we can expect to have a smaller *best-so-far* early on, allowing more effective pruning [19][22][27]. However, in *clustering*, we need to know the distance between all pairs [11], or at least all distances within a certain range, rendering the typical use of lower-bounding pruning ineffective.

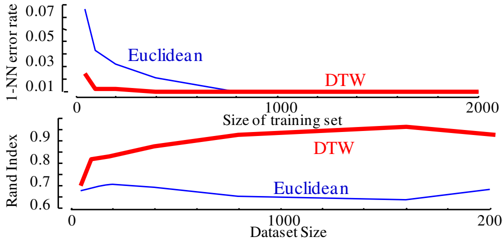

Figure 3: *top*) The classification error rates of DTW and ED tend to converge as we see more training data. *bottom*) In contrast, for clustering, DTW retains its great superiority over ED for increasingly large datasets.

1.2 Why Existing Work Is Not the Answer

More generally, many clustering algorithms achieve scalability by exploiting a spatial access method. For example, the scalable version of the ubiquitous DBSCAN uses an R*tree [6]. However, because DTW is not a metric, it is very difficult to index, especially for long (i.e., high-dimensional) time series objects.

Beyond the need to scalably support DTW, we note the need for a clustering algorithm that supports invariance to outliers. That is to say, unlike some clustering methods such as k-means, which attempt to explain *all* the data, we believe it is critical to allow the clustering algorithm the freedom to ignore some data.

Consider the example in Figure 4. We took twelve objects from a heraldic shield dataset [36], and clustered them using k-means and DP, the algorithm we propose to augment (described in detailed in Section 4.1). Because we are using the (non-metric) DTW measure, which may prevent k-means from converging, we used the variant in [10] which performs k-means clustering using the all-pair distance matrix. Note that for ease of visualization, we used multidimensional scaling to cast high-dimensional time series objects to two dimensions. After we ran the algorithms, both of them gave the perfect Rand Index. We then inserted a single outlier object (object 13) from this dataset, and reran the algorithms. As we can see from Figure 4.*bottom.left*), k-means assigned objects 8-12 to the cluster of the outlier object. In addition to this, k-means falsely identified objects 1 and 2 as a separate cluster from the cluster of objects 3-6. In contrast, from Figure 4.*bottom.right*) we can see that DP only clustered object 8 in the cluster of the outlier object, but did not change the cluster labels of the rest of the dataset.

This toy example is contrived and anecdotal, but conformed by more rigorous experiments on real data [37].

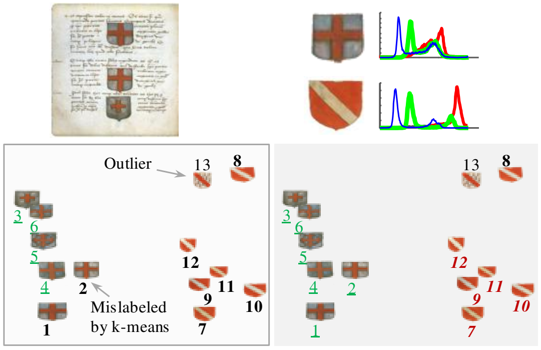

Figure 4: *top.left*) Leaf 18V of a 15th-century book, *Treatises on Heraldry*[36]. *top.right*) The colorful heraldic shields can be converted to 3D RGB "time series" of color distribution. *bottom.left*) Even the insertion of a single outlier can confuse

[1] We defer a discussion of our experimental philosophy until Section 5, but we note that all experiments in this work are made reproducible by our unrestricted sharing of code/data.

k-means. *bottom.right*) In contrast, the performance of the DP algorithm is not sensitive to outliers.

In this work, we address all the considerations above. We adapt DP (Density Peaks), a relatively new clustering framework that is able to ignore outlying data points [24]. While robust to outliers, DP is relatively slow, as it requires O(N^2) DTW calculations. We augment DP such that it can exploit *both* DTW upper and lower bounds, to compute only the absolutely necessary DTW calculations, and do so in a *best-first* manner, giving our algorithm the desirable *anytime algorithm* behavior [2][34].

2. RELATED WORK

The field of clustering is vast, and even the subfield of clustering time series has an enormous literature [1][11][31][33]. Much of the works on time series clustering are concerned with clustering based on time series *features* [31], which are at best tangentially related to our goals. Here, we are only interested in clustering based on time series *shapes*. In the latter case, there are two important and interrelated choices that define most of the literature: the choice of distance measure, and the choice of clustering algorithm.

Most of the literature on time series shape-based clustering uses metric measures like Euclidean distance [31]. The ubiquity of Euclidean distance seems to derive more from its familiarity and ease of indexing than any data-driven assessment of its effectiveness. As Figure 3.*top* illustrates on a single representative example, the general superiority of DTW over ED is well understood in the community (cf. [27]), at least for *classification*. As Figure 3.*bottom* hints at, and as we later empirically confirm on many diverse datasets, the dominance of DTW over ED for *clustering* is, if anything, greater.

The plethora of shape-based clustering algorithms [11][20][33] can be divided at the highest level into those that insist on explaining (i.e., *clustering*) all the data [33] vs. those that have the representational power to leave some data unclustered (a small minority) [20]. We believe that this distinction is underappreciated and critical to the success of most efforts. For clarity, consider the following analogy: If we were clustering *people*, then surely every person in our database would belong to some group, even if (due to the small size of our sample) the size of some groups were just *one*. In contrast, imagine we are clustering subsequences from a speech articulation database (see Section 6.1). We hope that the subsequences will cluster into well-defined words or phrases. However, it is highly likely that we will have some examples of coughing, sneezing or harrumphing. Such sequences are likely to be very dissimilar to the rest of the database. It is not just the case that we do not want/need them to be clustered; we do not want them to affect the clustering of the clusterable words or phrases (recall Figure 4). However k-means and its variants insist on explaining these instances, and because of k-means's sum of squares objective function, these highly dissimilar items have a huge effect on the overall clustering.

There exist works [15] in the literature that perform clustering on top of DBSCAN [6]. The problems with such approaches are the inheritance of the non-determinism of DBSCAN, and the use of *only* lower bounds to prune expensive distance calculations. A handful of research efforts [33] have attempted to mitigate the slow performance of DTW clustering by casting it to an anytime framework. Most such efforts reduce to the following: Until there is a user interrupt, these frameworks keep replacing the (fast to compute) approximate DTW distances with true (slow to compute) DTW distances. If there is no user interrupt, such frameworks would calculate the full distance matrix (generally in some *clever* "most-likely-to-be-useful" order), and return the exact clustering. Our proposed algorithm goes beyond this in several ways. Most importantly, we show that calculating the full distance matrix is unnecessary in the general case. By exploiting both upper and lower bounds to DTW, and, more critically, by exploiting the *relationship* between these bounds, we can compute the *exact* clustering while only calculating a tiny fraction of the full distance matrix.

3. BACKGROUND

There has been significant research on clustering datasets that are too large to fit in main memory [4]. This problem setting typically assumes inexpensive distance measures, but costly disk accesses [4]. However, the problem we wish to solve exploits DTW, which itself is a very expensive distance measure. Therefore, in situations when even the data can be stored in main memory, the time needed to do the clustering may be on the order of days/weeks. The problem we are interested in is therefore, CPU constrained, not I/O constrained.

3.1 Anytime Algorithms

For most clustering algorithms, it is well known that not all distance measurements contribute equally to the final clustering assignments. For example, a recent paper on hierarchical clustering demonstrates (under some mild assumptions) that it is possible to capture the true structure of the clustering with just *carefully chosen* O($n \log^2 n$) distance computations [14]. This fact that some distance computations are more important than others immediately suggests the use of *anytime algorithms*, assuming only that we can find an efficient and (even *somewhat*) effective test to identify these influential distance computations.

Recall that anytime algorithms are algorithms that can return a valid solution to a problem, even if interrupted before ending [33][34]. Starting with a negligibly small amount of setup time, these algorithms always have a *best-so-far* answer available and the quality of the answer improves with the increase of execution time. The desirable properties of anytime algorithms are interruptibility, monotonicity, measurable quality, diminishing returns, preemptibility, and low overhead [34]. Note that this is a very brief introduction to anytime algorithms; we refer the interested reader to [34], which contains an excellent survey. As Figure 5 shows, anytime algorithms are in essence optimizing the tradeoff between execution time and quality of the solution.

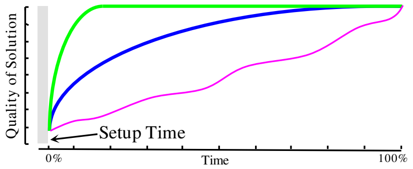

Figure 5: An abstract illustration of an anytime algorithm. The three curves show a comparison of the possible performances of three hypothetical anytime algorithms. The bottommost curve (pink) is only improving linearly over time, but the topmost curve (green) demonstrates *diminishing returns*, making most of its improvements early on.

For clarity, we reiterate that the anytime algorithm approach is just one of the two contributions of this paper. We propose to make the clustering *absolutely faster* by admissible pruning. This is *in addition* to rearranging the order the non-prunable calculations are considered to produce the best possible diminishing returns *anytime algorithm* behavior. We are now in a position to explain the DP framework.

4. ALGORITHM

4.1 Density Peaks Algorithm Overview

Our proposed solution is inspired by DP, the density-based clustering algorithm recently proposed in [24]. We chose to augment the DP framework for solving large time series clustering problems because of the following:

- Recent literature [20] and our own experience on real datasets (cf. Section 6) suggest that the successful clustering of time series requires the ability to ignore some data objects. It is not merely that anomalous objects themselves are unclusterable; it is that the presence of these objects can affect the labels of objects that *are* clusterable in unpredictable ways. The DP algorithm has been shown to be able to ignore anomalous data points.
- The DP algorithm is able to handle datasets whose clusters can form arbitrary shapes. This is in contrast to k-means and related algorithms which assume the clusters are "balls" in space. This observation is particularly important for DTW, which is not a metric. While we cannot exactly visualize DTW clusters in a metric space, it is clear that some classes of objects under DTW form complex manifolds in DTW "space."
- Many clustering algorithms require the user to set many parameters. In contrast, the DP algorithm requires only two. Moreover, they are relatively intuitive and not particularly sensitive to user choice.
- Finally, it happens to be the case that the DP algorithm is amiable to optimization and conversion to an anytime algorithm.

For concreteness, we will take the time to explain the clustering algorithm [24] we adapt and augment in our framework. The DP algorithm assumes that the cluster centers are surrounded by lower local density neighbors and are at a relatively higher distance from any point with a higher local density. Therefore, for each point i in the dataset, the DP algorithm computes two quantities:

- Local density (ρ_i)
- Distance from points with higher local density (δ_i).

We can formally define these two quantities:

Definition 1 The *Local Density* ρ_i of point i is the number of points that are closer to it than some cutoff distance $d_{c.}$

Definition 2 The *Distance from Points of Higher Density* is the minimum distance δ_i from point i to all the points of higher density. For the special case of the highest density point, this distance is the maximum of the distances of all the points from their higher density points.

We give the algorithm to compute ρ_i in Table 1 and δ_i in Table 2.

Table 1: Local Density Calculation Algorithm

```
Input   D,all-pair distance matrix
        dc, cutoff distance
Output  ρ,the local density vector for all n points in the
        dataset
1       for i = 1:n
2         ρ(i) = count(D(i,otherObjects)<dc)
3       end
```

Given the all-pair distance matrix D and a cutoff distance d_c, for each point i in the dataset, ρ_i is calculated in lines 1-3 of Table 1.

In Table 2, using the local densities ρ from Table 1, for each point i, the list of the points with higher densities is calculated (line 2). In line 4, this list is sorted in descending order. From lines 5 – 7, for each point in the sorted order, the distances from their higher density points are calculated. For the special case of the highest density point (which by definition does not have a higher density neighbor), this distance is calculated in line 8.

Given the ρ_i and δ_i for each object i¸ the DP algorithm calculates the cluster centers χ, and performs the cluster assignments based on these centers.

Table 2: Distance to Higher Density Points Algorithm

```
Input   D,all-pair distance matrix
        ρ,the local density vector
Output  δ,NN distance vector of higher density points
1       for i = 1:n
2         δ_list(i) = findHigherDensityItems(i,ρ)
3       end
4       [sorted_δ_list, sortIndex] = sort(δ_list,'descend')
5       for j = 2:n
6         δ(sortIndex(j)) = NNDist(sorted_δ_list(j))
7       end
8       δ(sortIndex(1)) = max(δ(2:n))
```

The cluster centers are selected using a simple heuristic: *points with higher values of* ($\rho_i \times \delta_i$) *are more likely to be centers*. We give the cluster center selection algorithm in Table 3.

Table 3: Cluster Center Selection Algorithm

```
Input   δ,NN distance vector of higher density points
        ρ,the local density vector
        k, number of clusters
Output  χ, cluster centers
1       χ = topK(sort(ρ*.δ, 'descend'),k)
```

Given the sorted values of ($\rho_i \times \delta_i$) in descending order, the top k items are selected as cluster centers (line 1). The value of k can be specified by the user, or found automatically using a "knee-finding" type of algorithm [24].

The final step of the DP algorithm is the cluster assignment. Each data item gets the cluster label of its nearest neighbor (NN) from the list of points with higher local densities than it has. We give the cluster assignment algorithm in Table 4.

Table 4: Cluster Assignment Algorithm

```
Input   χ, cluster centers
        δ,NN distance vector of higher density points
        sortIndex, sorted index of items based on
        descending ρ
Output  C, clusters
1       for i = 1:size(χ)
2         C(χ(i)) = i //assign cluster labels for centers
3       end
4       for j = 1:n
5         if C(sortIndex(j)) == empty //no cluster label yet
6            C(sortIndex(j)) = C(NN(sortIndex(j)))
7         end if
8       end
```

In lines 1-3 the cluster labels of the centers are assigned. After this initialization, each of the points in the dataset (other than the centers themselves) gets the cluster label of its nearest neighbor from the higher density list in the descending order of local density (lines 4-8). It is important to note that this algorithm allows the clusters to have arbitrary, possibly non-convex shapes, unlike k-means and its variants, which are restricted to a Voronoi partitioning of the input space. We are now in a position to describe our augmented version of the DP framework.

4.2 TADPole: Our Proposed Algorithm

We call our algorithm, TADPole (Time-series Anytime DP). As stated in Section 1, in order for the original DP algorithm to cluster a dataset, we need to know the distances between all pairs. The time needed to compute these all-pair distances becomes untenable for a quadratic time distance measure such as DTW. In order to mitigate this undesirable time complexity, our thoughts naturally turn to attempts to speed up other (non-clustering) algorithms that need to compute DTW frequently. Most such algorithms exploit linear time lower bounds like LB_Keogh [12], LB_Kim, LB_Yi [31], etc. Moreover, some algorithms exploit the fact that ED is an *upper* bound to DTW, and can also be computed in O(n) time.

In our TADPole algorithm, we augment the DP clustering framework and exploit the upper and lower bounds of DTW to prune unnecessary distance computations, resulting in at least an order of magnitude speedup. For datasets where even this level of speedup is inadequate, we show that we can use a simple heuristic to order the unavoidable calculations in a most-useful-first ordering. As a result, our algorithm can be cast to an *anytime* clustering framework, quickly producing a good answer, and rapidly refining it until it converges to the exact answer.

The inputs to the TADPole algorithm are the lower bound and upper bound matrices for the true DTW distances of all the objects of the dataset. Note that the time needed to compute these is inconsequential (<1%) relative to the overall clustering time.

The only parameters we need are the cutoff distance (d_c) and optionally, the number of clusters (k), if the user wishes to specify this value rather than use the knee-finding heuristic suggested in [24]. Note that our use of these two additional upper bound and lower bound matrices increases the space complexity of the algorithm by 200%. However, this is not an issue because:

- The DP algorithm (especially when using DTW or another expensive measure) is *CPU* bound, not *space* bound.
- If really necessary, we could greatly mitigate this space overhead. The lower bound matrix will have many elements that are zeros, and thus would be amiable to encoding as a sparse matrix.

For clarity of presentation, we present our contributions in two different sections, although the final algorithm incorporates both ideas. In Sections 4.2.1 to 4.2.4, we show how to accelerate the TADPole algorithm by admissibly pruning the distance computations during the calculation of local densities (ρ) and NN distances (δ) from a higher density list for each item. In Section 4.2.5, we show how to reorder these computations to give us the *diminishing returns* property of anytime algorithms [2][34].

4.2.1 Pruning during Local Density Calculation

Consider the four cases shown in Figure 6.

In this step of the TADPole algorithm, the inputs are the fully computed lower (LB_{Matrix}) and upper bound (UB_{Matrix}) matrix. For each object pair (i,j), while calculating their local densities (lines 1 to 3 in Table 1), we prune their distance (D_{ij}) computation according to the following four cases shown in Figure 6:

Case A: *Objects i and j are identical*

The DTW distance of two identical objects, i and j, is equal to their ED distance. It is a simple lookup in the upper bound distance matrix, and requires no actual DTW distance computation. This case is logically possible but very rare (Figure 6.A)).

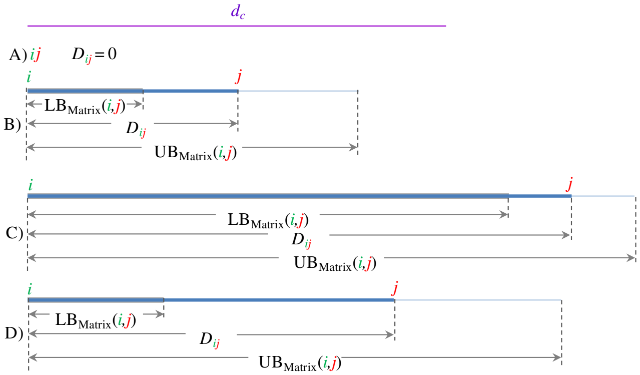

Figure 6: The four mutually exclusive and exhaustive cases of distance computation pruning during local density calculation. Note that the cutoff distance d_c, represented by the purple line at the top, applies to all four cases below it. Case A is difficult to visually represent, as *i* and *j* coincide.

Case B: $UB_{Matrix}(i,j) < d_c$

If the upper bound distance between objects i and j is less than the cutoff distance (d_c), then i and j are definitely within d_c distance to each other (Figure 6.B)). Therefore, we can prune the DTW distance computation of these two objects.

Case C: $LB_{Matrix}(i,j) > d_c$

If the lower bound distance of i and j is greater than the cutoff distance, then these two objects are definitely *not* within d_c distance to each other (Figure 6.C)). We can therefore admissibly prune their DTW distance computation.

Case D: $LB_{Matrix}(i,j) < d_c$ and $UB_{Matrix}(i,j) > d_c$

In this case, we cannot tell whether or not the actual DTW distance between i and j is within d_c. Therefore, *only in this case* do we need to compute D_{ij} (Figure 6.D)).

With this intuition in mind, we specify the formal distance pruning algorithm during the local density calculation in Table 5.

As we can see from Table 5, in lines 5 - 21, for all the object pairs in the data, the TADPole algorithm checks which of the four cases applies in order to determine whether or not these objects are within the cutoff distance.

The occurrence of case B tells us that the object pair in question are definitely within d_c (lines 10 -11) without having to calculate the expensive true DTW distance. Cases A (lines 8 -9) and C (lines 12 -13) specify that the object pair is not within $d_{c.}$ It is only the occurrence of case D that forces the algorithm to calculate the true DTW distance of the object pair in question (lines 14 -19).

At the end of this section of TADPole, for each object i we have all the local densities (ρ_i) computed. Using lines 1 - 3 of Table 2, we can now find the δ list, the list of the points with higher densities. Next we will describe our pruning strategy for this step.

Table 5: Pruning Algorithm during Local Density Calculation

Input	**LB_{Matrix}**, full computed lower bound matrix **UB_{Matrix}**, full computed upper bound matrix **Data**, the dataset **d_c**, cutoff distance
Out put	**ρ**,local density vector for all points in dataset **D_{Sparse}**, partially filled distance matrix

```
D_Sparse = empty
for i = 1:size(Data)
 objectsWithin_dc = empty
  for j = 1: size(Data)
    if i == j
        continue;
    else
      if LB_Matrix(i,j) == UB_Matrix(i,j) //case A)
        continue
      elseif UB_Matrix(i,j) < d_c       //case B)
        objectsWithin_dc = [objectsWithin_dc j]
      elseif LB_Matrix(i,j) > d_c      // case C)
        continue
      //case D)
      elseif LB_Matrix(i,j) < d_c and UB_Matrix(i,j) > d_c
        D_Sparse(i,j) = calculateDist(Data(i),Data(j))
        if D_Sparse(i,j) < dc
           objectsWithin_dc = [objectsWithin_dc j]
        end if
      end if
    end if
  end for
  ρ(i) = length(objectsWithin_dc)
end for
```

4.2.2 *Pruning during NN Distance Calculation from Higher Density List*

Our pruning strategy for this step works in two phases. First, for each item we find an upper bound of the NN distance from its higher density list. In the second phase we perform the actual pruning based on these upper bounds. The distance computation of TADPole terminates when for all objects in the dataset, we are done finding their actual NN distance from their respective higher density lists.

Phase 1: Upper bound calculation

Given D_{Sparse} and ρ_i for each item *i*, we initialize the upper bound of its NN distance from its higher density list, ub_i, to `inf`. For each item *j* in the higher density list of *i*, we either have the actual DTW distance (D_{ij}) computed already or have access to the upper bound ($UB_{Matrix}(i,j)$) to this distance. We scan the higher density list of item *i,* and if the current $ub_i > D_{ij}$ or $ub_i > UB_{Matrix}(i,j)$*,* we update the current ub_i to D_{ij} (if available already), or to $UB_{Matrix}(i,j)$ otherwise. Therefore, we can guarantee that the NN distance from the higher density list for item *i* can be no larger than ub_i. We give a visual intuition of this upper bound calculation in Figure 7.

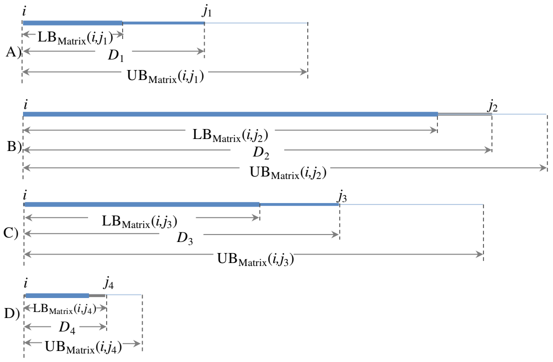

Figure 7: An illustration of the distance pruning during the NN distance calculation from a higher density list of an object. From object *i*, the elements in the higher density list are $j_1 - j_4$. After Phase 1, ub_i will be $UB_{Matrix}(i,j_4)$. In Phase 2, the distance computations of D_{ij2} and D_{ij3} are pruned.

In Figure 7, the elements on object *i*'s higher density list are $j_1 - j_4$. Assume that we only know the DTW distances from object *i* to objects j_1 and j_3, (D_1 and D_3 respectively, shown in blue). Because we do not know D_2 and D_4, we have shown these distances in gray in Figure 7. When Phase 1 starts, ub_i is initialized to `inf.` Now our TADPole algorithm scans object j_1 and updates ub_i to D_1. Because $UB_{Matrix}(i,j_2)$ and D_3 are both are greater than ub_i, we do not need to update ub_i. In the last step, given that $UB_{Matrix}(i,j_4) < ub_i$, we update ub_i to $UB_{Matrix}(i,j_4)$. This ub_i is an upper bound of the NN distance from object *i*'s higher density list.

We give the upper bound calculation algorithm for the NN distance computation from a higher density list in Table 6. We initialize the upper bound vectors of NN distances of objects from their higher density list, *ub* to `inf` (line 1). Next, considering each of the item on the higher density list of an object *i*, $\delta\_list_i(j)$,we check whether *i*'s current upper bound can be tightened (lines 5 -13). In lines 5 - 8 we see if the actual distance between *i* and $\delta\_list_i(j)$ has been computed already, then whether or not this distance can tighten ub_i. If the distance has not been computed yet, then in lines 10 - 12 we check whether we can tighten ub_i by replacing it with the upper bound distance between *i* and $\delta\_list_i(j)$.

Table 6: Upper Bound Calculation Algorithm for NN Distance Computation from Higher Density List

Input	**UB_Matrix**, full computed upper bound matrix **Data**, the dataset **D_Sparse**, partially filled distance matrix δ_list, list of the points with higher densities
Output	*ub*, upper bound vector of NN distances from higher density points

```
1   ub = inf(size(Data))
2   for i = 1:size(Data)
3     for j = 1:size(δ_list_i)
4       highDensityItem = δ_list_i(j)
5       if D_Sparse(i, highDensityItem)≠ empty
6         if ub_i > D_Sparse(i, highDensityItem)
7           ub_i = D_Sparse(i, highDensityItem)
8         end if
9       else
10        if ub_i > UB_Matrix(i,highDensityItem)
11          ub_i = UB_Matrix(i,highDensityItem)
12        end if
13      end if
14    end for
15  end for
```

At the end of this phase of TADPole, we have *ub*, the upper bound vector of NN distances from higher density points, computed. We now describe exploiting *ub* to prune the distance calculations during the computation of the higher density list.

Phase 2: Pruning

We give the pruning algorithm during the computation of NN distances from the higher density lists of all objects in Table 7.

We begin by scanning the higher density list of each of the objects again. In line 5 of Table 7, for an object *i,* we test whether $LB_{Matrix}(i, \delta\_list_i(j))$ is greater than ub_i we calculated in Table 6 . If this is the case, we prune the distance computation (line 6) for $\delta\_list_i(j)$. Otherwise, if the *true* distance between *i* and $\delta\_list_i(j)$ is already calculated, then we consider this distance as one of the potential NN distances from *i's* higher density list (line 9). If the true distance is not yet known, *only then* do we compute it (line 11-12). Finally, we compute the NN distance vector for all objects from their higher density lists (line 17).

In Figure 7, we see that both $LB_{Matrix}(i,j_2)$ and $LB_{Matrix}(i,j_3)$ are greater than ub_i. Therefore, we can prune D_2 and D_3. In this example, we assumed we know D_1; therefore, after the pruning done by Phase 2, we only need to calculate D_4.

Table 7: Pruning Algorithm during the Computation of the NN Distances from the Higher Density Lists of All Objects

Input	**LB$_{Matrix}$**, full computed lower bound matrix **Data**, the dataset **D$_{Sparse}$**, partially filled distance matrix δ_list, list of the points with higher densities *ub*, upper bound vector of NN distances from higher density points
Output	δ,NN distance vector of higher density points

```
1   for i = 1:size(Data)
2     temp_δ = empty
3     for j = 1:size(δ_list_i)
4       highDensityItem = δ_list_i(j)
5       if LB_Matrix(i,highDensityItem)> ub_i
6         continue //prune distance computation
7       else
8        if D_Sparse(i, highDensityItem)≠ empty
9         temp_δ = [temp_δ D_Sparse(i, highDensityItem]
10       else      // calculate distance
11        D_Sparse(i, highDensityItem) =
12        calculateDist(Data(i),Data(highDensityItem))
13        temp_δ = [temp_δ D_Sparse(i, highDensityItem]
14       end if
15      end if
16     end for
17     δ(i) = min(temp_δ)
18  end for
```

After this phase of TADPole, for each item *i* we have access to the NN distance from points with higher local densities (δ_i). At this point, given ρ_i and δ_i for each object *i*¸ the TADPole algorithm calculates the cluster centers χ using the algorithm in Table 3, and performs the cluster assignments based on these centers according to the algorithm in Table 4.

4.2.3 Multidimensional Time Series Clustering

While most of the research efforts on time series clustering have considered only single-dimensional cases [11][20], the increasing prevalence of medical sensors (c.f. Section 6.2) and wearable devices [23] (c.f. Section 6.1) has given urgency to the need to support multidimensional clustering [28]. Fortunately, our extension of TADPole to the multidimensional case requires changing only a *single* line of code. For clarity, we highlight these changes for multidimensional clustering for Table 5 and Table 7 in Table 8 and Table 9, respectively.

Table 8: Pruning Algorithm during Local Density Calculation for Multidimensional Data (see Table 5)

Input	**LB$_{Matrix}$**, full computed lower bound matrix along *d* dimensions **UB$_{Matrix}$**, full computed upper bound matrix along *d* dimensions **Data**, the dataset **d$_c$**, cutoff distance
16	... D$_{Sparse}$(i,j) = $\sum_{dim=1}^{d}$ calculateDist(Data$_{dim}$(i), Data$_{dim}$(j)) ...

Table 9: Pruning Algorithm during NN Distance Computation from Higher Density List, Multidimensional Case (see Table 7)

```
10      else      // calculate distance
11        D_Sparse(i, highDensityItem) =
12        $\sum_{dim=1}^{d}$ calculateDist(Data_dim(i), Data_dim(highDensityItem))
```

Recall that in Table 5, we gave the full lower bound and upper bound distance matrices as inputs to the algorithm. To perform multidimensional clustering, for each dimension we wish to consider, we calculate the corresponding lower/upper bound distance matrices *independently* along those dimensions. We take the sum of all lower bound matrices/upper bound matrices and give these cumulative matrices as inputs to our algorithm described in Table 5. In addition, when we actually calculate the distances (line 16 in Table 5 and lines 10-12 in Table 7), we take the summation of the distances along all the dimensions. All other components of TADPole will remain the same.

As we shall show empirically now, by using the pruning method described so far, TADPole can obtain *at least* an order of magnitude speedup over the original DP algorithm.

4.2.4 How Effective Is Our Pruning?

Before generalizing to allow anytime behavior in the next section, in this section we will demonstrate *just* the utility of our pruning strategy. In order to intuitively calibrate the effectiveness of our pruning, we compare to the best and worst possible cases:

- In order to perform clustering, the DP algorithm needs the all-pair distance matrix computed [24]. Therefore, in terms of distance computation, the brute force DP algorithm itself is the obvious worst-case strawman.
- The best possible variant of DP is the one that performs a distance computation *only* when it is necessary. Therefore, during density computation, this variant of DP considers *only* those distance computations that contribute to the actual density of an object. In addition to this, during the computation of the NN distance from the higher density list of an object, this variant considers *only* the actual NN distances. We call this algorithm the *oracle* variant of DP. Note that we obviously cannot compute this in real-time, but only by doing an expensive post-hoc study.

We compare the amount of distance pruning we achieve against these two variants of the DP algorithm. For this experiment, we consider the StarLightCurves dataset [13]. We vary the number of objects in the dataset we need to cluster (by randomly sampling) and record the number of *true* DTW distance computations. As we can see from Figure 8.*left*), the number of distance computations increases quadratically using the brute force algorithm. In contrast, the oracle algorithm requires very few distance computations; moreover, we can see that our TADPole algorithm performs *almost* as well as the oracle algorithm.

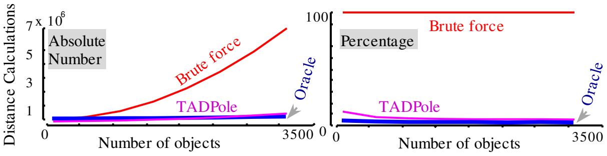

Figure 8: A comparison of the amount of pruning TADPole achieves compared to an oracle and the brute force algorithm. TADPole is very close to the oracle algorithm.

This claim is reinforced in Figure 8.*right*, in which we see that the *percentage* of distance computations TADPole requires is very close to the oracle algorithm. As we can see, as the datasets get larger, TADPole converges closer and closer to the oracle algorithm.

Also note that a similar performance is observed in *all* datasets we considered (archived in [37] for brevity). Moreover, we obtain similar results if we measure the CPU time instead. As we can see from Figure 9, to cluster the StarLightCurves data, TADPole requires only 9 minutes, whereas the DP algorithm needs 9 hours.

In spite of these very promising results, demonstrating a sixty-fold speedup, there exist datasets where even this amount of speedup is not adequate. In order to address similar scalability issues for other types of data mining analyses, including classification [29] and outlier detection [2][3], researchers have attempted to create *anytime* versions of their algorithms [2][33]. One significant advantage of the DP algorithm (and our modifications to it) is that it is particularly amiable to casting as an anytime framework. In essence, the computations discussed in this section can be computed in any order. Thus far, we have simply computed them in a top-to-bottom, left-to-right order. However, we should expect that not all such computations of the true DTW are equally significant in terms of their impact on the final clustering, and that if we could find even an approximate "most-significant-first" order, we could converge more quickly. In the next section, we describe such an ordering heuristic.

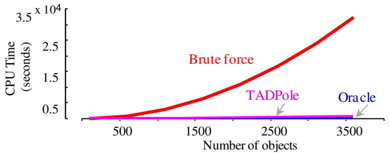

Figure 9: A comparison of the CPU time spent by TADPole against an oracle and the brute force algorithm to cluster the StarLightCurves dataset. As before, the performance of TADPole is very close to the oracle algorithm.

4.2.5 Distance Computation-Ordering Heuristic

Recall from the above that the DP algorithm may be considered a two-step algorithm; calculating the local densities (Table 1) first, and then finding the NN distances from the higher density lists (Table 2) of the objects. Only the latter step is amiable to anytime ordering; the former step may be regarded as the *setup time* [2][29][33].

Before attempting to create an anytime ordering function, it will be instructive to consider two baselines: what is the *best* we could possibly do, and what would we have to do in order to claim we are beating the most obvious strawmen?

- The *best* ordering heuristic we consider is an *oracle* ordering. We can compute this by allowing the algorithm to *cheat*. In each step of the algorithm, this order chooses the object that maximizes the current Rand Index. The algorithm is cheating, because by definition, a clustering algorithm normally does not have access to class labels.
- The most basic strawman is top-to-bottom, left-to-right ordering, but that is brittle to "luck". A *random* ordering is much less so, so we consider random ordering as the baseline we would like to improve upon.

To understand the performance of these two algorithms, we took the Insect dataset from [13] and measured the Rand Index, as the two algorithms above refined the mixture of true DTW distances and upper bound distances that we have at the end of phase 1 (i.e., Table 6), into the set of all *necessary* DTW computations needed (i.e., Table 7). The results are shown in Figure 10 (for the moment, ignore the blue line). For completeness, we also show the accuracy achieved using the Euclidean distance with the DP algorithm. If the Euclidean distance *was* competitive, it would be fruitless to waste time computing expensive DTW calculations. The results clearly show that in *this* dataset, DTW is needed.

After initially getting worse, the random ordering linearly (in a stepwise fashion) converges on the true clustering. In contrast, the oracle algorithm achieves a perfect Rand Index after calculating the true DTW distances for the NN list of just five objects.

As impressive as the oracle's performance is, we can actually come *very* close to it, as shown by the blue curve in Figure 10. The ordering heuristic TADPole exploits is the descending order of the local density (ρ) × the upper bound distance (*ub*) from the higher density list of the objects.

With a little introspection, it is easy to see why our distance computation-ordering heuristic is as close as the *oracle* ordering. Recall from Section 4.1 that points with higher values of ρ×δ are more likely to become cluster centers. Until we calculate all the NN from each object's higher density list, we do not have access to their δ. However, we can estimate δ by the tight *upper* bound *ub* to δ. Our distance computation-order heuristic exploits *ub* to prioritize distance computations for items that are more likely to be centers. Because the centers are selected earlier, we achieve a higher Rand Index with very few actual distance computations.

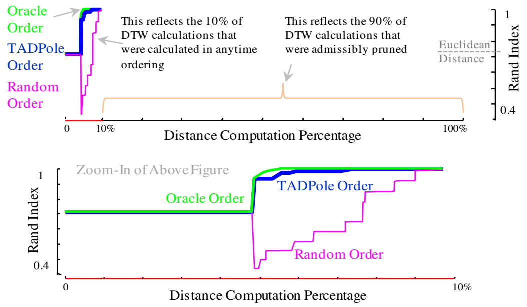

Figure 10: *top*) A comparison of different distance computation-order heuristics on the Insect dataset [13]. An oracle ordering (green) converges stunningly quickly. The random ordering (pink) converges very slowly. Our proposed ordering (blue) is very close to the oracle. *bottom*) A zoomed-in view of the figure shown at the *top*).

From Figure 10.*top*) we see that *in conjunction* with all the pruning strategies described in Sections 4.2.1 and 4.2.2, TADPole achieves a perfect Rand Index after doing *only* ~6% of all possible distance computations. Of course, it does not "realize" this, and must compute ~10% of all possible distance computations before admissibly terminating.

5. EXPERIMENTAL EVALUATION

All experiments in this paper (including the ones above) are completely reproducible. We have archived all experimental code, parameter settings and data at [37]. The goal of our experiments is to show that our algorithm is more efficient and effective than current algorithms. We also show that our algorithm is not particularly sensitive to the *only* parameter choice we have to make. In addition to this, we demonstrate the utility of our approach on three real-world case studies.

5.1 Comparison with State-of-the-Art Clustering Algorithms

The principle strawman we need to compare to is the brute force version of DP with DTW. This comparison is explicitly encoded in Figure 10 and the similar figures below. In these experiments we also replace DTW with Euclidean distance to demonstrate that DTW is really necessary. In Table 10, we show a comparison of the clustering performance of TADPole to some well-known state-of-the-art clustering algorithms (which we carefully tuned) under DTW in five randomly chosen datasets from [13]. As we can see, the cluster quality returned by TADPole is usually better than the best-performing clustering algorithm. Note that we are not claiming DP is always superior, rather we chose DP because it is *at least* competitive with the state-of-the-art, and amiable to acceleration as we have demonstrated.

The greatly superior accuracy of TADPole makes the timing results somewhat irrelevant, but TADPole is at least an order of magnitude faster than the rival methods (exact numbers at [37]).

Table 10: Clustering Quality (in Terms of Rand Index) of TADPole vs. Some State-of-the-Art Clustering Algorithms

Dataset	TADPole$_{DTW}$ (*TADPole$_{ED}$*)	k-means[10] DTW$_{version}$	Hierarchical DTW$_{version}$	DBSCAN [6] DTW$_{version}$	Spectral [17] DTW$_{version}$
CBF	1 (*0.66*)	0.78	0.73	0.77	0.76
FacesUCR	0.92 (*0.86*)	0.87	0.85	0.77	0.94
MedicalImages	0.66 (*0.67*)	0.67	0.62	0.65	0.69
Symbols	0.98 (*0.81*)	0.93	0.78	0.91	0.95
uWaveGesture_Z	0.86 (*0.84*)	0.85	0.83	0.8	0.86

The greatly superior accuracy of TADPole makes the timing results somewhat irrelevant, but TADPole is at least an order of magnitude faster than the rival methods (exact numbers at [37]).

5.2 Parameter Sensitivity Experiments

To demonstrate that TADPole is not sensitive to parameter choices, we took the *Symbols* dataset [13], and performed TADPole on it with k = 6. We then varied the cutoff distance (d_c) parameter and measured the Rand Index obtained with the alternative settings. As we can see from Figure 11, there is a very wide range of choices for the values of d_c, which gives high-quality clustering.

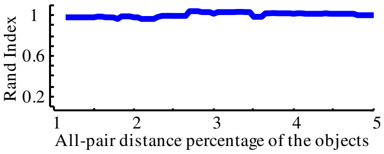

Figure 11: A parameter sensitivity test of TADPole shows stability of clustering over a very wide range of parameters.

6. CASE STUDIES

6.1 Electromagnetic Articulograph Dataset

The Electromagnetic Articulograph (EMA) is a device that is increasingly used for mouth movement studies [30]. The apparatus consists of a set of unobtrusive accelerometers that are attached to multiple positions on the tongue, lips, jaw, nose and forehead, and can record high-resolution 3D movement/position data in real-time. Recent research has suggested that articulographs may eventually allow a "silent speech" interface that translates non-audio articulatory data to speech output, an idea that has significant potential for facilitating oral communication after a laryngectomy [30]. The most common use of articulographs is in speech therapy for a plethora of speech disorders. However, EMA use has a significant burden: setting up the system can take up to 30 minutes per session (this time is spent carefully gluing the sensors to the participant's face and tongue). Given this significant setup time, practitioners are anxious to get the most from each session, yet the goals of the session are not typically fixed, but change in reaction to the participant's progress and areas of difficulty. Thus, there is a need to cluster the utterances of speakers in an *interactive* fashion, so that sessions can be adapted on the fly. We consider a dataset of lower-lip accelerometer time series movement data of 18 words collected from multiple speakers, for a total of 414 objects. The duration of utterance of each of the words is ~0.7 seconds. In Figure 12.*left*), we show the data collection process for one of our subjects. In Figure 12.*right*), we show two examples of the utterances of the word "*fate"* by two different individuals. These examples are clearly warped, suggesting this is an appropriate domain for TADPole.

We tested the power set of combinations of axes, confirming that axis Z with axis Y gives us the best clustering, with a Rand Index of 0.94 (the other results are archived at [37]).

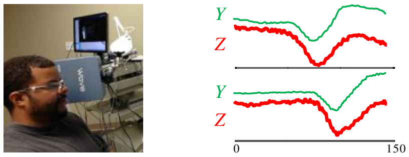

Figure 12: *left*) One of our volunteers wearing the articulograph apparatus. *right*) Two examples of the 3D time series produced by enunciating the word "*fate*" show inter-subject warping. The X axis is omitted here, as it only has useful information for patients with facial asymmetry.

We can now ask how effective our pruning strategy is when performing this clustering. We show the result in Figure 13.

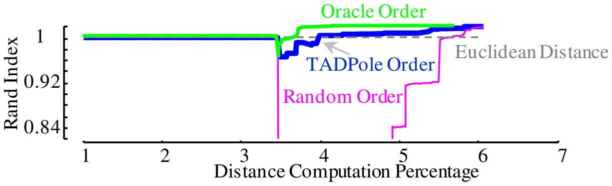

Figure 13: The amount of distance pruning achieved by TADPole is ~94% for the Articulographs [30]. Moreover, the ordering heuristic is almost as good as the oracle ordering.

As we can see from Figure 13, TADPole achieves ~94% pruning, converging almost as quickly as the oracle algorithm. In wall clock terms, TADPole takes only 2.89 seconds, allowing interactive analysis and feedback to the patient.

6.2 Pulsus Dataset

Pulsus Paradoxus is defined as a significant decline in the pulse with inspiration. It is a symptom of *Cardiac Tamponade*, a life-threatening condition where high-pressure fluid fills the sac surrounding the heart, impairing cardiac filling and causing 20,000 deaths per year in the USA alone. Of the several ways to detect Pulsus, the least invasive and simplest uses the PPG (PhotoPlethysmoGram) shown in Figure 14.*top*).

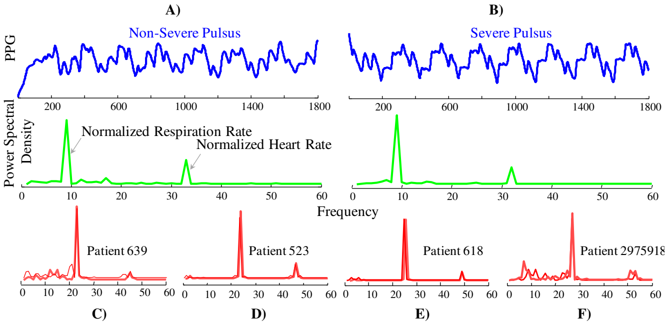

Figure 14: *top*) PPG and Power Spectral Density (PSD) signal of a patient with non-severe Pulsus (*top.left*)) and severe Pulsus (*top.right*)). *bottom*) Four PSDs of four patients forming four different clusters within the non-Pulsus objects. From these four clusters, we can see the objects are clearly warped.

For this case study we consider a dataset of 500 PPGs from two sources, the MIMIC II Waveform Database [8][25] and our collaborators. The latter dataset has the advantage that our collaborators followed up on the patients (in some case, *post-mortem*); thus, we have access to unusually rich annotations and external knowledge to evaluate our result. As shown in Figure 14.*top*), the raw PPG data is very complex, and following the suggestions of Dr. John Criley (UCLA School of Medicine), we converted the PPGs to *amplitude spectrums* (Figure 14.*middle*) and clustered in that space. Dr. Criley's intuition is that for Tamponade patients, the fluid that fills the sac surrounding the heart will cause a "shadow" signal to show up during respiration. For this dataset, TADPole produced a perfect clustering, with a pruning rate of 88%, making it an order of magnitude faster than brute-force. In Figure 15.*left*), we show the PPG measurement process. From Figure 15.*right*), we can see that the Pulsus instances are within a compact cluster, and the non-Pulsus instances seem to form a number of sub-clusters. Our medical collaborator suggests this reflects the fact that there is *one way* to have Tamponade, but *multiple* ways to have a healthy heartbeat.

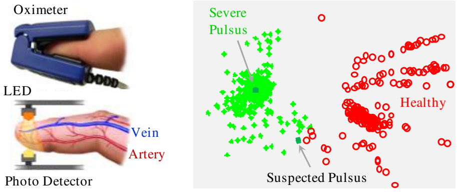

Figure 15: *left*) A PPG apparatus. *right*) The Pulsus dataset projected into two dimensions using multidimensional scaling, and color coded by the output of TADPole.

6.3 Person Re-Identification Dataset

Person re-identification is the task of recognizing individuals across spatially disjointed cameras [7], and an important problem for understanding human behavior in areas covered by surveillance cameras. As shown in Figure 16, we can extract color histograms from the video, thus treating the problem as a multidimensional time series problem. We considered the PRID dataset [9], randomly extracting 1,000 images of 12 different individuals. After we ran TADPole in this dataset to cluster the images of these 12 individuals, we achieved a Rand Index of 95.4% and distance pruning of 80% (*anytime* plot at [37]). In contrast, Euclidean distance only achieves a Rand Index of 89%.

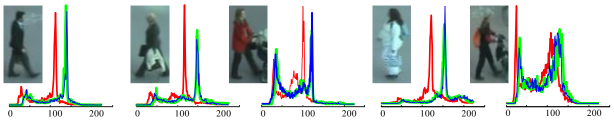

Figure 16: Representative images from the dataset [9] with their corresponding color histograms.

7. CONCLUSIONS

By introducing novel pruning strategies that exploit *both* upper and lower bounds, we have produced a robust DTW clustering algorithm that is both absolutely faster, and able to compute the clustering in an anytime fashion. We have demonstrated the utility of our algorithms on diverse domains, including two in which our algorithm is currently actively deployed (EMA/Pulsus).

8. ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support for our research provided by NSF IIS-1161997 II.

9. REFERENCES

[1] Aggarwal, C. C., & Reddy, C. K. (Eds.). *Data Clustering: Algorithms and Applications*. CRC Press, 2013.

[2] Assent, I., et al. *Anyout: Anytime Outlier Detection on Streaming Data*. In Database Systems for Advanced Applications, pp. 228-242, 2012.

[3] Begum, N. et al. *Rare Time Series Motif Discovery from Unbounded Streams*. Proceedings of the VLDB Endowment, 8(2), 2014.

[4] Cao, F., Ester, M., Qian, W., & Zhou, A. *Density-Based Clustering over an Evolving Data Stream with Noise*. SIAM SDM, 2006.

[5] Ding, R. et al. *YADING: Fast Clustering of Large-Scale Time Series Data* Proceedings of the VLDB Endowment 8.5, 2015.

[6] Ester, M., Kriegel. et al. *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*. ACM SIGKDD, pp. 226-231, 1996.

[7] Gheissari, N. et al. *Person Reidentification using Spatiotemporal Appearance*. IEEE CVPR, vol. 2, pp. 1528-1535, 2006.

[8] Goldberger, A. L. et al. *Physiobank, Physiotoolkit, and Physionet Components of A New Research Resource for Complex Physiologic Signals.* Circulation, 101(23), e215-e220, 2000.

[9] Hirzer, Martin, et al. *Person Re-identification by Descriptive and Discriminative Classification. Image Analysis*. Springer Berlin Heidelberg, pp. 91-102, 2011.

[10] Jang, J. S. R. *Machine Learning Toolbox*, available at *mirlab.org/jang/matlab/toolbox/machineLearning*, (*Dec 1, 2014*).

[11] Keogh, E., & Lin, J. *Clustering of Time Series Subsequences is Meaningless: Implications for Previous and Future Research*. KAIS, 8(2), 154-177, 2005.

[12] Keogh, E. & Ratanamahatana, C.A. *Exact Indexing of Dynamic Time Warping*. KAIS 7, no. 3, 358-386, 2005.

[13] Keogh, E., et al. *The UCR Time Series Classification Page*

[14] Krishnamurthy, A., Balakrishnan, S., Xu, M., & Singh, A. *Efficient Active Algorithms for Hierarchical Clustering*. arXiv preprint arXiv:1206.4672, 2012.

[15] Mai, Son T., et al. *Efficient Anytime Density-Based Clustering*. SDM, 2013.

[16] Mueen, A., Keogh, E. J., Zhu, Q., Cash, S., & Westover, M. B. *Exact Discovery of Time Series Motifs.* SDM, pp. 473-484, 2009.

[17] Ng, A. Y., Jordan, M. I., & Weiss, Y. *On Spectral Clustering: Analysis and An Algorithm.* Advances in Neural Information Processing Systems, 2, pp. 849-856, 2002.

[18] Rakthanmanon, T. et al. The UCR Suite: Fast Subsequence Search (DNA) *www.youtube.com/watch?v=c7xz9pVr05Q*, 2012.

[19] Rakthanmanon, T., et al. *Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping*. ACM TKDD, 7(3), 10, 2013.

[20] Rakthanmanon, T., et al. *Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data*. ICDM 2011.

[21] Rand, W. M. *Objective Criteria for the Evaluation of Clustering Methods*. J. Am. Statist. Assoc. 66.336, pp. - 846-850, 1971.

[22] Ratanamahatana, C. A., & Keogh, E. *Everything You Know About Dynamic Time Warping is Wrong*. In 3rd Workshop on Mining Temporal and Sequential Data, pp. 22-25, 2004.

[23] Rawassizadeh, R et al. *Wearables: Has the Age of Smartwatches Finally Arrived?* Communications of the ACM 58.1, pp. - 45-47, 2014.

[24] Rodriguez, A., & Laio, A. *Clustering by Fast Search and Find of Density Peaks*. Science, 344(6191), 1492-1496, 2014.

[25] Saeed, M., et al. *Multiparameter Intelligent Monitoring in Intensive Care II (MIMIC-II): A Public-access Intensive Care Unit Database*. Critical care medicine, 39(5), 952, 2011.

[26] Signorini, A., Segre, A. M., & Polgreen, P. M. *The Use of Twitter to Track Levels of Disease Activity and Public Concern in the US During the Influenza A H1N1 Pandemic*. PloS one, 6(5), 2011.

[27] Shieh, J., & Keogh, E. *iSAX: Indexing and Mining Terabyte Sized Time Series*. ACM SIGKDD, pp. 623 – 631, 2008.

[28] Shokoohi-Yekta, M. et al. *Generalizing Dynamic Time Warping to the Multi-Dimensional Case Requires an Adaptive Approach*. SDM 2015.

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

[30] Wang, J. et al. *Preliminary Test of A Real-time, Interactive Silent Speech Interface Based on Electromagnetic Articulograph*, SLPAT, pp. - 38 - 45, 2014.

[31] Wang, X., et al. *Experimental Comparison of Representation Methods and Distance Measures for Time Series Data*. DMKD, 26(2), 275-309, 2013.

[32] Yang, J., & Leskovec, J. *Patterns of Temporal Variation in Online Media*. ACM WSDM, pp. 177-186, 2011.

[33] Zhu, Q. et al. *A Novel Approximation to Dynamic Time Warping allows Anytime Clustering of Massive Time Series Datasets*. SDM, 2012.

[34] Zilberstein, S. *Using Anytime Algorithms in Intelligent Systems*. AI magazine, 17(3), 73, 1996.

[35] 2009 MTV Video Music Awards *en.wikipedia.org/wiki/2009_MTV_Video_Music_Awards*

[36] Unknown Author. (15th cent., second half). *Treatises on Heraldry*. Bodleian Library collection, MS. Lat. misc. e.

[37] Supporting Webpage: *www.cs.ucr.edu/~nbegu001/SpeededClusteringDTW*