UC Riverside
UC Riverside Electronic Theses and Dissertations

Title
Exploiting Time Series Primitives to Solve Realistic Data Mining Problems

Permalink
https://escholarship.org/uc/item/6r725821

Author
Begum, Nurjahan

Publication Date
2016

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
RIVERSIDE

Exploiting Time Series Primitives to Solve Realistic Data Mining Problems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Nurjahan Begum

June 2016

Dissertation Committee:
      Dr. Eamonn Keogh, Chairperson
      Dr. Vassilis Tsotras
      Dr. Stefano Lonardi
      Dr. Tamar Shinar

The Dissertation of Nurjahan Begum is approved:

_____

_____

_____

_____

                                                      Committee Chairperson

University of California, Riverside

ACKNOWLEDGEMENTS

ABSTRACT OF THE DISSERTATION

Exploiting Time Series Primitives to Solve Realistic Data Mining Problems

by

Nurjahan Begum

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, June 2016
Dr. Eamonn Keogh, Chairperson

Given the ubiquity of time series data in scientific, medical and financial domains, data miners have made substantial efforts to design efficient algorithms for classification, clustering, rule discovery and anomaly detection for this data type. In particular, all these well-known problems are important as exploratory techniques, and as sub-modules for answering higher level data mining questions. In essence, this means designing simple but efficient algorithms for these primitive modules is a critical part to solve highly complex data mining problems. In real-life scenarios, where the time series data is noisy and high-scale, it is important that the algorithms are invariant to irrelevant data and are amiable to optimizations. In addition, the algorithms should be parameter-lite and should involve *least* possible human intervention. Developing efficient algorithms to solve realistic data mining problems with these design criteria has been a challenging task to researchers over the last few decades. In this dissertation, we propose techniques to design such robust algorithms in classification, clustering and motif discovery context. In particular, the core contribution of this dissertation is as follows:

First, we propose techniques for the previously unsolved problem of finding the stopping criterion of semi-supervised time series classification. The core assumption of most of the time series classification algorithms *per se* is the availability of copious amount of labeled instances. However, this assumption is often unrealistic, and requires careful attention of data experts, which is a critical bottleneck in data analysis. Semi-supervised learning is an obvious way to mitigate the need for human labor, however, most such algorithms are designed for intrinsically discrete objects, and do not work well in time series domain, which requires the ability to deal with real-valued objects arriving in streaming fashion. In this work, we demonstrate that in many cases, just a handful of human annotated examples are sufficient to perform accurate classification. Also, we devise a novel parameter-free stopping criterion for semi-supervised time series classification.

Second, we propose an algorithm for robustly clustering large time series datasets with invariance to irrelevant data. In this work, we show that for similarity search, even though there is the general superiority of Dynamic Time Warping (DTW) distance measure over Euclidean distance (ED), the same is not true for clustering. Thus, clustering time series under DTW remains a computationally challenging task. We propose a novel pruning strategy that exploits both the upper and lower bounds to prune off a large fraction of the expensive distance calculations. 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.

Finally in the last part of the dissertation, we investigate algorithms to find time series motifs which have been shown to have great utility as a subroutine in many higher-level data mining algorithms. As we show, this detection becomes much harder in cases where the motifs of interest are *vanishingly rare* or when faced with a never-ending stream of data. We demonstrate that under reasonable assumptions, we must abandon any hope of an exact solution to the motif problem as it is normally defined; however we introduce algorithms that allow us to solve the underlying problem with high probability.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The increasing prevalence of cheap sensors has created the urgency for mining time series data over the last few decades. As the data gets bigger, big data issues come into play. For mining real-life noisy time series data, researchers have made substantial efforts to design simple and efficient algorithms. However, designing simple and parameter-lite algorithms for time series domain is typically plagued by limited data. For example, it is unrealistic to assume a user will provide thousands of examples for a single object to ensure the data is 'perfect' in favor of the algorithm we will design. In addition, given that our data is large, using the entire dataset to derive useful insights might hurt in a lot of cases. The algorithms for mining time series data therefore, should be agnostic to outliers and should keep human intervention at the *very least*. In essence, this means designing simple but efficient algorithms for the primitive modules to mine time series data is a critical task.

In this dissertation, we address the mentioned problems from the perspective of three exploratory techniques of time series data mining: classification, clustering, and motif discovery. To be concrete, our core contributions in this dissertation are as follows:

1.1 Semi-supervised time series classification

Dozens of time series classification algorithms have been proposed in the past decade [10][29][32]. Most of these algorithms assume the availability of copious amount of labeled instances. However, this assumption is often unrealistic, particularly in the

medical community, as it is expensive and time consuming to manually annotate large medical datasets [9]. Nevertheless, huge amount of *unlabeled* data is readily available. For example, the PhysioBank archive contains over 700 GB of data [11].

Because of the intra-patient variability of cardiological signals such as electrocardiograms (ECGs), it is difficult to consider classification problems independent of patients. Given such challenges, we would like to be able to build accurate classifiers in the face of *very* scarce knowledge, as little as a *single* labeled instance.

In such circumstances, Semi-supervised Learning (SSL) is a suitable paradigm, because it can leverage the knowledge of both labeled and unlabeled instances. Although many SSL models have been studied in the literature, the critical subroutine of finding a robust stopping point for these models has not been widely explored.

Most of the literature [24][25][33][41] addressing this problem possess some weaknesses that have limited their widespread adoption. For example, the approach proposed in [41] suffers from the problem of *early stopping*. That is, it is overly conservative and may refuse to label examples that are (at least *subjectively*) obvious enough to be labeled. The approach introduced in [33] fails to identify class boundaries accurately, because it tends to produce too many false negatives.

The key observation that motivates our work is the critical importance of the stopping criteria. If an otherwise useful algorithm is too conservative (or too liberal) in its stopping criteria, it is doomed to produce many false negatives (or false positives).

In this work we make two contributions. First, we demonstrate that in many cases just a handful of human annotated examples are sufficient to perform accurate

classification. Second, we devise a novel parameter-free stopping criterion for semi-supervised learning.

In particular, given the limitations of the state-of-the-art stopping criteria for SSL, we propose a semi-supervised learning framework with a novel stopping criterion using Minimum Description Length (MDL). MDL has been widely used in bioinformatics, text mining, etc., yet it is still under explored in time series domain. This is mainly because MDL is defined in discrete space. Some recent publications [17][30] provide an in-depth analysis of the potential of applying MDL technique to real-valued time series data. In this work, we propose a parameter-free algorithm for finding a stopping criterion for SSL using the MDL technique. We evaluate our work with a comprehensive set of experiments on diverse medical data sources including electrocardiograms. Our experimental results show that our approach can construct accurate classifiers even if given only a single annotated instance.

1.2 Generalized density based clustering

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.

Given the ubiquity of time series data in various human endeavors, the research community has made substantial efforts to create efficient algorithms for classification, clustering, rule discovery, and anomaly detection for this data type. In particular, time series *clustering* is useful for solving higher-level data mining problems. For example, clustering the keyword occurrence frequency might give us useful insights on synonyms, association discovery, etc. The dissimilarity among objects can be computed by cheap Euclidean distance as long as there is no significant time lag in these objects. However, in cases when there exist *causal relationships* (rather than just *associations*) between events, Dynamic Time Warping (DTW) is an ideal way to capture/be invariant to such out-of-sync relationships.

In this work we propose a fast density based clustering algorithm which we call TADPole (Time Series Anytime DP). For this, we adapt DP (Density Peaks), a relatively new clustering framework that is able to ignore outlying data points [89]. While robust to outliers, DP is relatively slow, as it requires quadratic number of DTW calculations. We

augment DP such that it can exploit *both* the 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 [48][105]. Besides exploiting both the upper and lower bounds to accelerate the clustering process, we demonstrate that the TADPole framework can be generalized to cluster objects the distances of which have concrete upper and lower bound defined.

1.3 Rare Motif Discovery

Time series motifs are approximately repeated subsequences in a time series, which appear to be very similar to each other. This conserved structure in these patterns is interesting because it is representative of useful insights of the data. Since the motifs were defined in 2002, researchers have used them in many domains like medicine, biology, medicine, industry, etc.

The brute force algorithm to compute motifs is quadratic, and therefore researchers have proposed tractable algorithms for time series motif discovery [79]. In spite of the recent progress in motif discovery, there are situations when there is no easy solution. If the motifs are extremely rare, then with a comparatively small buffer, it is extremely hard to detect the motifs occurring simultaneously in the cache. The problem becomes even harder in streaming setting, where we assume that the data will be generating in a never-ending fashion. Had we assumed the storage of all data points in the main memory, the algorithm would still be bounded by the computational complexity of the fastest

algorithm known so far. For some of the datasets we wish to find the rare motifs from, this would require years of CPU time.

In this work we investigate algorithms to find such rare motifs. We demonstrate that under reasonable assumptions rather than computing the exact motifs, it is better to have approximate algorithms that allow us to solve the underlying problem with high probability. We show that for our purposes it suffices to find *any two* of the *n* repeated patterns; we do not really care about the smallest distance pair, or any specified pair. We show that this slightly relaxed assumption allows us to solve problems that are otherwise intractable, and to discover repeated patterns in never-ending streams. The patterns discovered by our algorithm can serve as an input to higher-order algorithms that do semi-supervised learning [119], or look for changes in the frequencies of the discovered patterns that may signal *anomalous* behavior [126], or can be pipelined to higher level data mining algorithms.

Chapter 2: A Minimum Description Length Based Stopping Criterion for Semi-supervised Time Series Classification

In recent years the plunging costs of sensors/storage have made it possible to obtain vast amounts of medical telemetry, both in clinical settings and more recently, even in patient's own homes. However for this data to be useful, it must be annotated. This annotation, requiring the attention of medical experts is very expensive and time consuming, and remains the critical bottleneck in medical analysis. The technique of Semi-supervised learning is the obvious way to reduce the need for human labor, however, most such algorithms are designed for intrinsically *discrete* objects such as graphs or strings, and do not work well in this domain, which requires the ability to deal with *real-valued* objects arriving in a streaming fashion.

In this chapter we make two contributions. First, we demonstrate that in many cases a surprisingly small set of human annotated examples are sufficient to perform accurate classification. Second, we devise a novel parameter-free stopping criterion for semi-supervised learning. We evaluate our work with a comprehensive set of experiments on diverse medical data sources including electrocardiograms. Our experimental results suggest that our approach can typically construct accurate classifiers even if given only a single annotated instance.

Keywords: MDL; Semi-Supervised Learning; Stopping Criterion; Time Series

2.1 Introduction

Over the past decade, the data mining community has turned a significant fraction of its attention to time series data. Such data is available in almost all aspects of human endeavor, including motion capture [3], medical science [38], finance [27], aerospace, meteorology [18], entertainment, etc. Given such ubiquity of time series data, it is not surprising that a plethora of algorithms for time series classification have been proposed in recent years [10][29][32]. Virtually all these algorithms assume the availability of plentiful labeled instances [31][37]. However, this assumption is often unrealistic, particularly in the medical community, as it is expensive and time consuming to manually annotate large medical datasets [5][9]. Nevertheless, huge amounts of *unlabeled* data are readily available. For example, the PhysioBank archive contains over seven hundred GB of data [11], only a tiny fraction of which are labeled. Similarly, Google's Total Library [15] contains millions of digitized books available, which could be mined after conversion to time series space, given that plentiful labeled data is available.

Given the enormous intra-patient variability of cardiological signals such as electrocardiograms (ECGs), it is difficult to consider classification problems independent of patients. For example [35] notes that "*...age, sex, relative body weight, chest configuration, and various other factors make ECG variant among persons with same cardiological conditions*". Thus we cannot easily generalize labeled ECG data from one patient to another. Therefore, constructing a classifier in order to detect some medical phenomena, (e.g., ECG arrhythmia) is a non-trivial problem.

In spite of all these challenges, we would like to be able to build accurate classifiers in the face of *very* scarce knowledge, as little as a *single* labeled instance.

Given such circumstances, Semi-supervised Learning (SSL) seems like an ideal paradigm, because it can leverage the knowledge of both labeled and unlabeled instances. Although many SSL models have been studied in the literature, the critical subroutine of finding a robust stopping point for these models has not been widely explored.

The vast majority of the literature [24][25][33][41] addressing this problem possess some flaws that have limited their widespread adoption. For example, the approach proposed in [41] suffers from the problem of *early stopping*. That is, it is overly conservative and may refuse to label examples that are (at least *subjectively*) obvious enough to be labeled. The approach introduced in [33] fails to identify class boundaries accurately, because it tends to produce too many false negatives.

Our work is motivated by the observation of the critical and underappreciated importance of the stopping criteria. If an otherwise useful algorithm is too conservative (or too liberal) in its stopping criteria, it is doomed to produce many false negatives (or false positives), and thus it is almost certainly never going to be deployed in the real world. We illustrate this observation in Fig. 1.
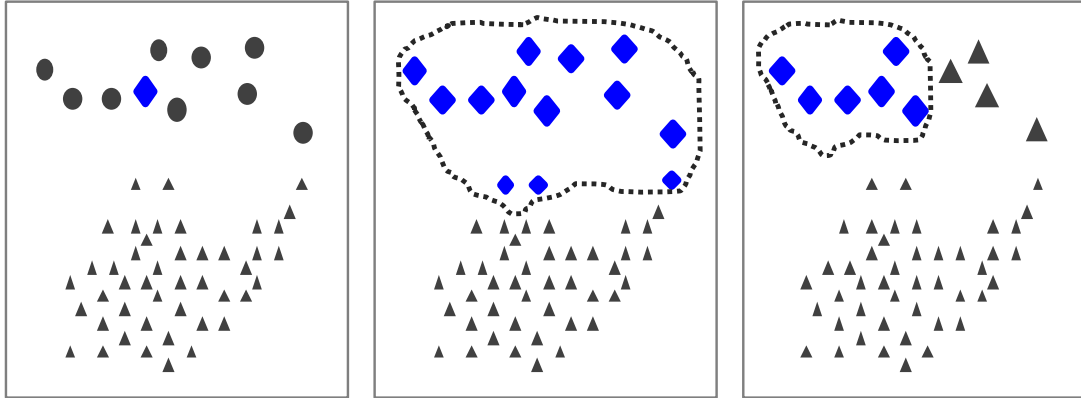
Fig. 1. *left*) **The initial positive instance (blue/square) along with unlabeled positive (grey/circle) and negative (grey/small triangle) instances.** *center*) **A too liberal stopping criterion produces lots of false positives (blue/small square).** *right*) **A too conservative stopping criterion produces lots of false negatives (grey/large triangles).**

From Fig. 1.*left*, we can see that in a dataset with a training set size as small as a single instance, we cannot avoid an unwanted production of lots of false positives with a too liberal stopping criterion (Fig. 1.*center*). Similarly, if we have a too conservative stopping criterion, we must invariably encounter lots of false negatives (Fig. 1.*right*).

In response to the noted limitations of the state-of-the-art stopping criteria for SSL, we propose a semi-supervised learning framework with a novel stopping criterion using Minimum Description Length (MDL). MDL has been widely used in bioinformatics, text mining, etc., yet it is still under explored in time series domain. This is mainly because MDL is defined in discrete space. Some recent publications [17][30] provide an in-depth analysis of the potential of applying MDL technique to real-valued time series data. In this work, we propose a parameter-free algorithm for finding a stopping criterion for SSL using the MDL technique. Its simplicity will allow adoption by experts in the medical community.

The rest of the chapter is organized as follows. In Section 2.2, we discuss related work. We give definitions and the notations in Section 2.3. We demonstrate how our semi-supervised learning algorithm works and then propose our stopping criterion using MDL in Section 2.4. In Section 2.5, we provide a detailed empirical evaluation of our ideas. Finally, we discuss conclusions and directions for future work in Section 2.6.

2.2 Related Work

Semi-supervised Learning (SSL) can be seen as intermediate between the techniques of Supervised Learning and Unsupervised Learning. SSL is generally provided with *some* supervision information in addition to unlabeled instances [4]. Labeled data is often hard to obtain, expensive, and may require human intervention. For example, there is often no replacement of a dedicated domain expert observing and labeling medical phenomena [26], human activities [28], insect behaviors [39], etc. for constructing annotated datasets. In contrast, unlabeled data is abundant and easy to collect. SSL exploits the use of unlabeled data in addition to labeled data for building more accurate classifiers [44]. Because of this, SSL is of great interest in both theory and practice. There have been many SSL techniques studied in the literature. These can be classified into five major classes: SSL with generative models, SSL with low density separation, graph based techniques, co-training techniques and self-training techniques [2][4][41][44].

Among all these classes of SSL, *Self-training* is the most common method used [1][22][34]. The idea is to first train the classifier with a small amount of data, and then use the classifier to classify unlabeled data. The most confidently classified unlabeled points, along with their predicted labels, are added to the training set. The classifier is then re-trained and the procedure repeats, i.e. the classifier uses its own predictions to teach itself [44]. Self-training approaches have been used for natural language processing tasks [21][43] and object detection systems [34]. Because of their generality and simplicity, we use self-training as the foundation of our research efforts.

In our work, we focus on the question of finding the *correct* stopping point for our self-training classifier. The question of finding a *good* stopping criterion has tentative solutions based on MDL, Bayesian Information Criterion, bootstrapping [36][44], etc. For time series domain, this question has been surprisingly under explored; to the best of our knowledge, only [24][25][33][41] have considered it. In [41], a stopping criterion based on the *minimal nearest neighbor distance* was introduced. However, this conservative approach suffers from the shortcoming that it can result in an expansion of only a small number of positives.

In order to improve the algorithm of [41], the work in [33] proposed a stopping criterion by using the previously observed distances between candidate examples from the unlabeled set to the initial positive examples. Although this refinement can add more positive examples to the labeled set, it is unable to identify accurate positive and negative instances from the unlabeled set, because it produces too many false negatives, and therefore not accurate enough in most real life applications [6].

A recently proposed method [24] is called LCLC (Learning from Common Local Cluster), which is a cluster-based approach rather than an instance-based approach. Although this method more accurately identifies the decision boundaries, it has two drawbacks. First, it assumed that *"All the instances belonging to a local cluster have the same class label"*. The same examples within the same cluster received the same class label, and therefore, LCLC can misclassify some examples if the clustering subroutine does not produce pure clusters. Second, the LCLC algorithm uses K-means algorithm, and therefore inherits K-means lack of determinism and its assumption that the data can be represented by Gaussian "balls".

To overcome the drawbacks of the LCLC approach, the same research group proposed an ensemble-based approach called En-LCLC (Ensemble based Learning from Common Local Clusters) [25]. This algorithm performs the clustering process multiple times with different settings to obtain diverse classifiers. Each instance is assigned a "soft" probabilistic confidence score and based on these scores, potential noisy instances, which can confuse the classifier, are eliminated. An Adaptive Fuzzy Nearest Neighbor (AFNN) classifier is then constructed based on the "softly labeled" set of positive and negative instances. However, this approach is *extremely* complicated and requires many unintuitive parameters to be set. It is not clear how sensitive the settings of these parameters are.

A recent work on time series SSL [6] has somewhat an ad-hoc stopping criterion, in which the classification continues until the unlabeled dataset is exhausted of true positives.

13

In this chapter, we propose a general Minimum Description Length (MDL) based stopping criterion, which works at the instance level, is parameter free and leverages the intrinsic nature of the data to find the stopping point. In time series domain, MDL has been the cornerstone of several recent efforts to solve classification and clustering problems [17][30]. However, to the best of our knowledge, this is the first work to address time series SSL using MDL.

2.3 Background and Notation

We begin by defining the data type of interest, *Time Series*:

Definition 1 *Time Series*: A time series $T = t_1,...,t_m$ is an ordered set of m real-valued variables.

We are only interested in the local properties of a time series, thus we confine our interest to *subsequences*:

Definition 2 *Subsequence*: Given a time series T of length m, a subsequence Cp of T is a sampling of length $w \leq m$ of contiguous positions from T with starting position at p, $Cp = tp,...,tp+w\text{-}1$ for $1 \leq p \leq m - w + 1$.

The extraction of subsequences from a time series is achieved by the use of a *Sliding Window*.

Definition 3 *Sliding Window*: Given a time series T of length m, and a user-defined subsequence length of w, all possible subsequences can be extracted by sliding a window of size w across T and extracting each subsequence C_p.

The most common distance measure for time series is the *Euclidean distance*.

Definition 4 *Euclidean Distance*: Given two time series (or time series subsequences) Q and C both of length m, the Euclidean distance between them is the square root of the sum of the squared differences between each pair of the corresponding data points:

$$D(Q, C) \equiv \sqrt{\sum_{i=1}^{m}(q_i - c_i)^2}$$

We illustrate these definitions in Fig. 2.
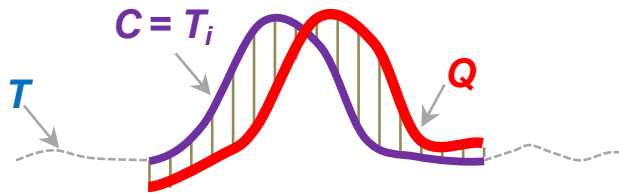
Fig. 2. A long time series *T* (grey/dashed) with a subsequence *Ti* (purple/thin) extracted and compared to a query *Q* (red/bold) under the Euclidean distance

Note that our use of Euclidean distance in this work does not preclude other distance measures; however, it has recently been forcefully shown that Euclidean distance is a very competitive distance measure [40].

Minimum Description Length (MDL) can be regarded as a formalization of Occam's Razor, which states that the best hypothesis for a given set of data is the one that leads to the best compression of the data [14]. Note that MDL is defined in the *discrete* space, but most time series are *real*-valued. To use MDL in time series domain, we need to cast the original real-valued data to discrete values. The reader may imagine that a drastic reduction in the precision of the data could sacrifice accuracy. However, recent

empirical work suggests that this is not the case. For example, [17][30] have performed extensive experiments comparing the effect of using the original real-valued data vs. its quantized version. The results showed that even drastically reduced cardinality does not result in reduced accuracy. To see this in our domain we performed a simple time series clustering experiment, as shown in Fig. 3.
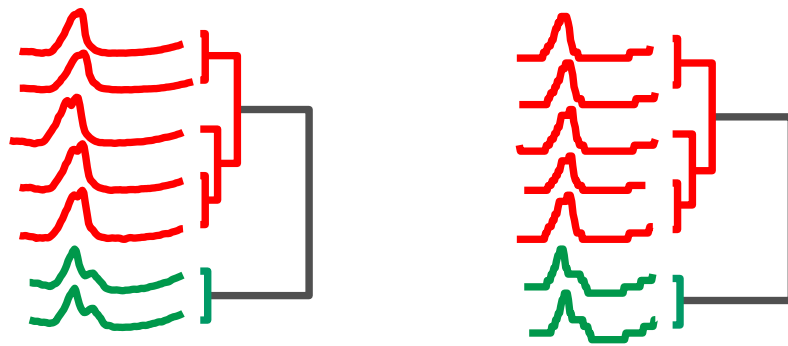
Fig. 3. A complete linkage clustering of time series incartdb I70 [11]. *left*) Original 32-bit data. *right*) the same data with cardinality reduced to eight values (3-bits).

Fig. 3 suggests that even significantly reducing the cardinality of time series has minimal effect.

To further test the claim that reducing the cardinality of the data has no *statistically significant* effect on classification accuracy, we show the error rates and ranks of the 1-NN classifier on six ECG datasets both in raw and reduced cardinality format from the UCR archive [19] in Table 1. For each dataset, we assign the best algorithm the lowest rank.

From Table 1 we can see that for the six datasets shown, classification using the raw data yields slightly better average rank. However, we cannot claim that the difference

in average ranks of the 1-NN classifier for the two different cardinality representations is

statistically significant, given of the following analysis.

Table 1. Classification error rates and ranks of 1-NN classifier for raw data (32 bit) and cardinality reduced data (4 bits). The best result for each dataset has been shown in bold/green

Datasets	Error Rate (Raw Data)	Error Rate (Cardinality 16)
ECG200	**0.12 (1)**	0.13 (2)
ECGFiveDays	0.2 (2)	**0.15 (1)**
TwoLeadECG	**0.25 (1)**	0.27 (2)
CinCECG_Torso	0.1 (2)	**0.07 (1)**
NonInvasiveFatalECG_Thorax1	**0.17 (1)**	0.24 (2)
NonInvasiveFatalECG_Thorax2	**0.12 (1)**	0.21 (2)
Average Rank	1.33	1.67

The null hypothesis we test is, "*The performances of a classifier for different cardinality representations of the data are the same and the observed differences are merely random*". In order to compare the performances of the classifier as shown in Table 1, we denote the number of datasets and the number of different cardinality representations by N and k respectively. In our case, $N = 6$ and $k = 2$. We employ the two-tailed Nemenyi test [23] to test the null hypothesis. According to this test, the performances of a classifier for different cardinality representations are significantly different if the corresponding average ranks differ by at least the critical difference (CD):

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where critical values q_α are based on the Studentized range statistic divided by $\sqrt{2}$ [6].

For a significance level α of 0.05, $q_\alpha = 1.96$, yielding a CD of 0.8. From Table 1, the difference in average ranks between the two different cardinality representations is 0.34, which is less than CD. Therefore, according to the Nemenyi test, we fail to reject the null hypothesis.

In Fig. 4, we show a critical difference diagram [6][16] of the ranked error rates from Table 1.
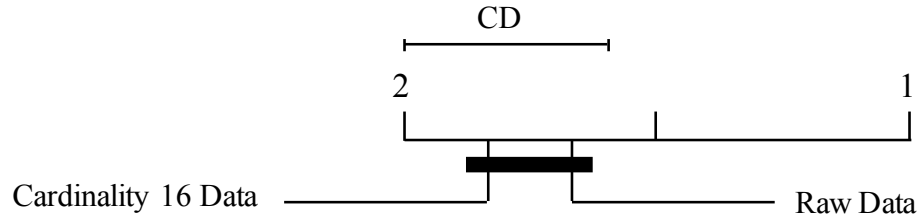
Fig. 4. Critical difference diagram representing the ranked error rates of the 1-NN classifier for the two cardinality representations of the data (Table 1).

From Fig. 4 we see that both variants of the 1-NN classifier for different cardinality representations are connected by a single clique, which means they are not significantly different from each other.

Based on the above and more extensive empirical work in [17][30], we reduce time series data to its 16-value (4-bit) cardinality version, using a discrete normalization function:

Definition 5 *Discrete Normalization Function*: A *discrete* function *Dis_Norm* is the function to normalize a real-valued subsequence *T* into *b*-bit discrete value of range $[1, 2^b]$. It is defined as below:

$$Dis\_Norm(T) = round\left(\frac{T - min}{\max - min}\right) * (2^b - 1) + 1$$

18

where min and max are the minimum and maximum value in T respectively.

As we previously noted, MDL works in discrete space, therefore we are interested in determining how many bits are required to store a particular time series *T*. We call it the *Description Length* of *T*.

Definition 6 *Description Length*: A *description length DL* of a time series *T* is the total number of bits required to represent it.

$$DL(T) = w * \log_2 c$$

where *w* is the length of *T*, *c* is the cardinality.

One of the key steps of designing our stopping criterion using the MDL technique is to find a representation or model of our data, and use this model to compress the data in a lossless fashion. We call this representation, a *hypothesis:*

Definition 7 *Hypothesis*: A hypothesis *H* is a subsequence used to encode one or more subsequences of the same length.

Naively, the number of bits required to store a time series depends only on the data format and its length. However, we are interested in knowing the *minimum* number of bits to exactly represent the data. This depends on the *intrinsic structure* of the data. In general case, this number is not calculable, as it is equivalent to the Kolmogorov complexity of the time series [20], however, there are many ways to approximate this, e.g. Huffman encoding, Shanon-Fano coding, etc.

Ultimately, we interest in how many bits are required to encode *T given H*. We call this the *Reduced Description Length* of *T*.

Definition 8 *Reduced Description Length*: A *reduced description length* of a time series T given hypothesis H is the sum of the number of bits required in order to encode T exploiting the information in H, *i.e. DL(T | H)*, and the number of bits required for H itself, *i.e. DL(H)*. Thus, the reduced description length is defined as:

$$DL(T, H) = DL(H) + DL(T \mid H)$$

We can encode T using H in many ways. One simple approach of this encoding is to store a *difference vector* between T and H, and we can easily generate the whole time series T from the hypothesis. Therefore, we use, $DL(T|H) = DL(T - H)$.

Given the definitions above, let us consider a toy example of our MDL based encoding. In Fig. 5, A is a sample time series of length 20:

A = **1 2 4 4 5 6 8 8 9 10 11 13 13 14 17 16 17 18 19 19**

H is another time series, which is a straight line of length 20:

H = **1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20**

Without encoding, the naive bit requirement to store A and H is, $2 * 20 * \log_2 20 = 173$ bits. In contrast, using H as the model of the data, we can encode A as A', which is simply the difference vector obtained by subtracting H from A. We can see that, $A' = |A - H| =$ **0 0 1 0 0 0 1 0 0 0 0 1 0 0 2 0 0 0 0 1.** From Fig. 5, the reader can see that A is a slightly corrupted version of H and that in the difference vector, there are 5 mismatches. The bit requirement is now just $20 * \log_2 20 + 5 * (\log_2 20 + \lceil \log_2 20 \rceil) = 134$ bits, which is significantly less than the naive bit requirement.

20
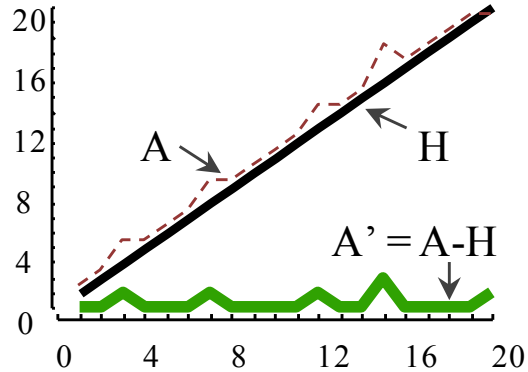
Fig. 5. : Time series *A* (maroon/dashed) is represented by the sum of hypothesis *H* (solid/black) and the difference vector *A'*(green/bold).

We are now in a position to demonstrate how our semi-supervised learning approach works with the novel MDL based stopping criterion in the next section.

2.4 Algorithms

2.4.1 Semi-supervised Time Series Classification

Semi-supervised learning technique can help build better classifiers in situations where we have a small set of labeled data, in addition to abundant unlabeled data [8]. We refer the interested reader to [4] and [44] for a more detailed treatment.

We use the self-training technique to build our classifier for the reasons noted in Section 2.2. In this work, we require only a *single* positive instance as the initial training set. During the training process, the training set will be augmented iteratively by labeling more unlabeled instances as positive.

Below are the two steps by which the classifier trains itself iteratively:

Step 1: We find the nearest neighbor of any instance of our training set from the unlabeled instances.

Step 2: This nearest neighbor instance, along with its newly acquired positive label, will be added into the training set.

The intuition of this idea is straightforward. The labeled positive example serves as a model which characterizes what a positive example should "look like". If an unlabeled instance is very similar to a positive example, its probability to have the positive class label is also assumed to be high. By adding such an example to the training set, the description of the positive class is generalized.

We give an illustrative example of the training procedure of our algorithm in Fig. 6. Each ellipse corresponds to labeling one unlabeled instance as positive. As we can see, there is a "chaining effect" of the semi-supervised learning: a set of positive examples helps labeling more positive examples.
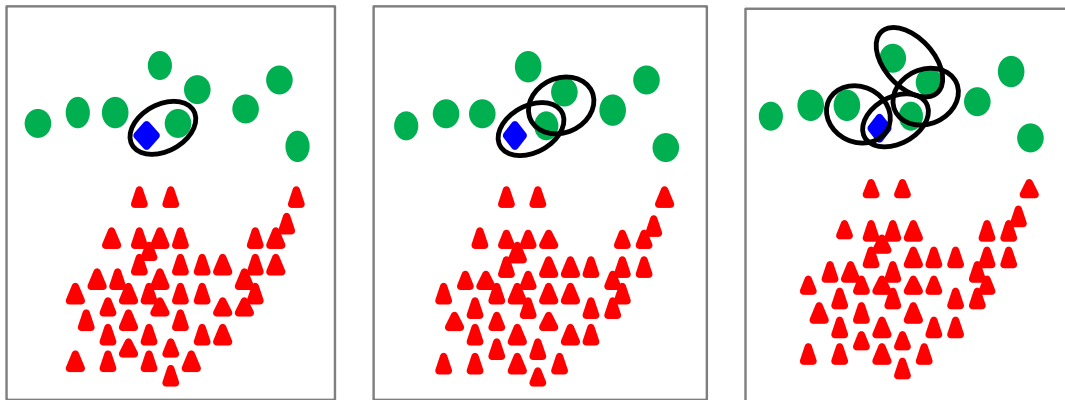
Fig. 6. left) The initial positive instance (blue/square) selects its nearest positive instance (green/circle). *middle and right)* **As we continue iterating, more unlabeled instances are labeled as positive and added to the training set.**

For concreteness, we give the algorithm for self-training time series classification using just a single labeled instance in Table 2.

In order to demonstrate the benefit of our approach, we compare our method with the naive one-nearest-neighbor straw man. The naive approach also starts with one initial seed. Instead of finding the nearest neighbor of any instance of the updated training set, the algorithm only considers the nearest neighbors of this sole seed. Thus the algorithm can be imagined as expanding concentric circles around the initial seed.

Table 2. Semi-supervised time series classification algorithm with only one labeled instance

Input	pos_Obj = **Self_Training_Classifier** (P,N)
	P, set of labeled/positive instances.
	//initially $|P| = 1$
	N, set of unlabeled/negative instances
Output	pos_Obj, set of labeled objects
1	**while** (the stopping criterion)
2	new_pos_obj = find the nearest neighbor for P
3	$P = P \cup \{new\_pos\_obj\}$
4	$N = N - \{new\_pos\_obj\}$
5	**end while**
6	$pos\_Obj = P$

However, this approach possesses a representational flaw as shown in Fig. 7. As the iteration continues, the probability of mistakenly labeling a negative instance (red/triangle) as positive also increases.
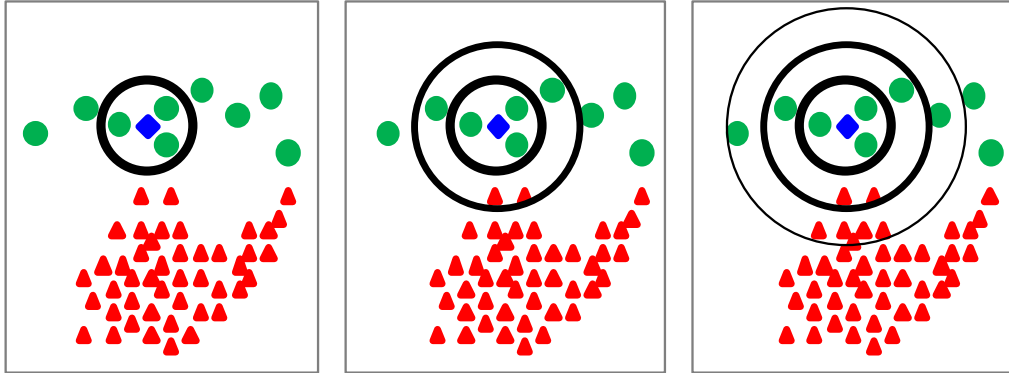
Fig. 7. Inner circles (black/thicker) correspond to the earlier iterations. *left*) **The initial positive instance (blue/square) selects its nearest neighbor as a positive instance (green/circle).** *middle and right*) **False Positives (red/triangle) are added.**

In contrast, Fig. 6 shows that the selection of an instance with the smallest distance from the current labeled positive set reduces this probability of misclassifying the instances too early. This example suggests that the expressiveness of our approach is greater than the naive nearest neighbor straw man. Note however that this greater representation power is all for naught unless it is coupled with the ability to stop adding instances at the correct time. We consider this issue next.

2.4.2 Stopping Criterion

In this section, we demonstrate how our algorithm finds the correct stopping point using an MDL based technique. As shown in Fig. 8, we consider a contrived, but real ECG dataset from [11]. Inspired by the compression-based approach of [30], our stopping criterion algorithm attempts to losslessly compress the data by finding repeated patterns, and it signals termination when further compression is not possible.

24
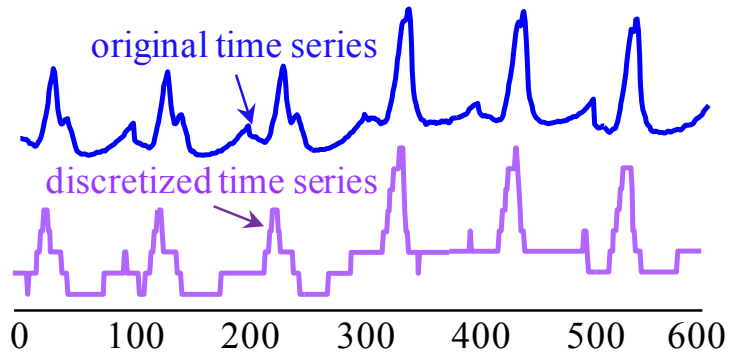
Fig. 8. An original time series is shown in blue and its discretized version is shown in purple.

In the toy example in Section 2.3, we used a *straight line* hypothesis to explain the data. Here we use the seed heartbeat as the hypothesis instead. Our algorithm will attempt to compress the purple time series by expressing all its instances in terms of this hypothesis H and the (hopefully small) residual error A'. After compression, our original time series is supposed to have shorter length, which we call its *Reduced Description Length*.

For clarity, we show the instances of the purple time series separately with their class labels ('V' and 'N') in Fig. 9.*bottom.* Note that these instances are *not* labeled by the algorithm; these labels come from the ground truth and are given for the reader's introspection only.

As the first step, we randomly give an instance as the seed to our SSL module, which is the *only* member of the training set. According to the algorithm in Section 2.4.1, this module will augment the training set. Let us imagine we do not have a stopping criterion for this module. In that case, the algorithm will eventually ingest all the instances of the dataset in the training set.

Given this, let us see how we find a stopping criterion for our SSL module using MDL. As hypothesis, we take the same initial seed we gave to our SSL module (Fig. 9.*top*). We use this hypothesis to encode the remaining five instances of our purple time series.
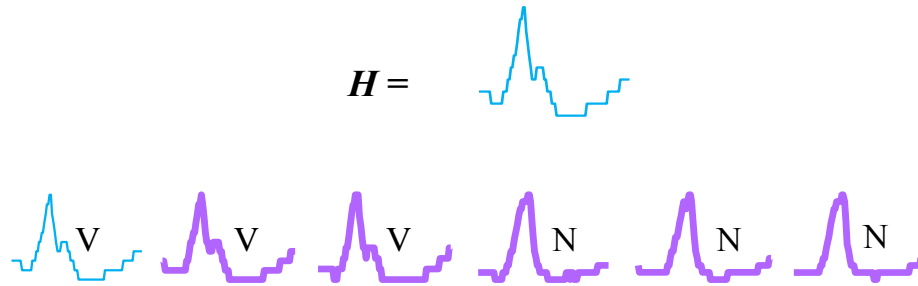
Fig. 9. *top*) **The initial seed chosen as a hypothesis.** *bottom*) **The six instances (of length 100 each) of the dataset with their class labels (from ground truth)**

Without any encoding, the description length for this time series is $DL(T) = 6 * 100 * \log_2 16 = 6 * 400 = 2400$ bits.

Our SSL module will select the nearest neighbor of any of the instance(s) in the training set from the dataset. We will encode this instance in terms of our heartbeat hypothesis and keep the rest of the dataset uncompressed. We find a total of 22 mismatches of the encoded instance with the hypothesis (Fig. 10.*top*). The reduced description length of the dataset becomes:

DL(*T,H*) = DL(*H*) + DL(*T|H*) + DL(*uncompressed*)

$$= 5*400 + 22 \times (4 + \lceil (\log_2 100) \rceil) = 2242 \text{ bits.}$$

This means we achieved data compression, which our algorithm interprets as an indication that the heartbeat in question really is a true positive.

26

We simply continue to test to see if unlabeled instances can be added to the positive pool by this "reduces the number of bits required" criteria. After the module adds the third instance in the training set, the cumulative mismatch count in terms of the hypothesis increases to 37, resulting in a reduced description length of 2007 bits, which means we are still achieving some compression (Fig. 10.*bottom*).
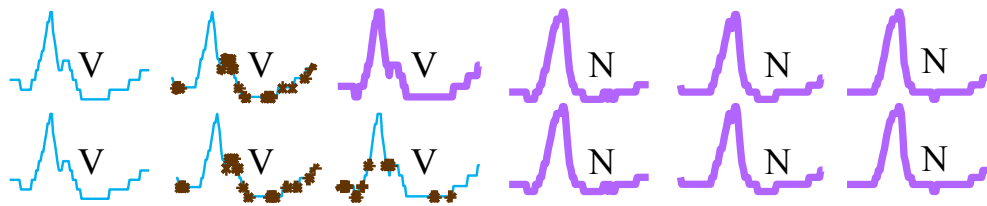
Fig. 10. The instances (cyan/thin) in the training set are encoded. The mismatches are colored as brown *.

The compression indicates that the two instances encoded are very similar to the hypothesis. By visual inspection, the reader will notice that the last three instances in Fig. 9 are *dissimilar* to the initial seed. Without a stopping criterion, the SSL module will eventually include these instances in the labeled set. This means our labeled set now starts including true *negatives*. However, our MDL criteria can save us, at least in this toy example. If we encode the last three instances using our hypothesis, the overhead to store the mismatches will not allow further compression. This means adding the true positives, and *only* the true positives, is the *best* we can do in terms of achieving compression for this example.

The effect of encoding the last three 'N' instances is shown in Fig. 11. The cumulative mismatch count jumps to 115, 192 and 273, respectively for these instances. The resulting *increase* of the reduced description lengths to 2465, 2912 and 3403 bits

respectively reinforces the observation that attempting to compress data which is unlike the hypotheses is not fruitful, and this signals an appropriate stopping criterion.

Fig. 11. After correctly encoding the three 'V' instances, attempts to encode the 'N' instances require more bits than the raw data representation.

We illustrate this observation in Fig. 12. As long as the SSL module selects instances similar to our hypothesis, we achieve data compression (shown in green). Once this module starts including instances dissimilar to the hypothesis, we can no longer achieve compression (shown in red). Therefore, this first occurrence of a subsequence that cannot be compressed with the hypotheses is the point where the SSL module should **stop.**
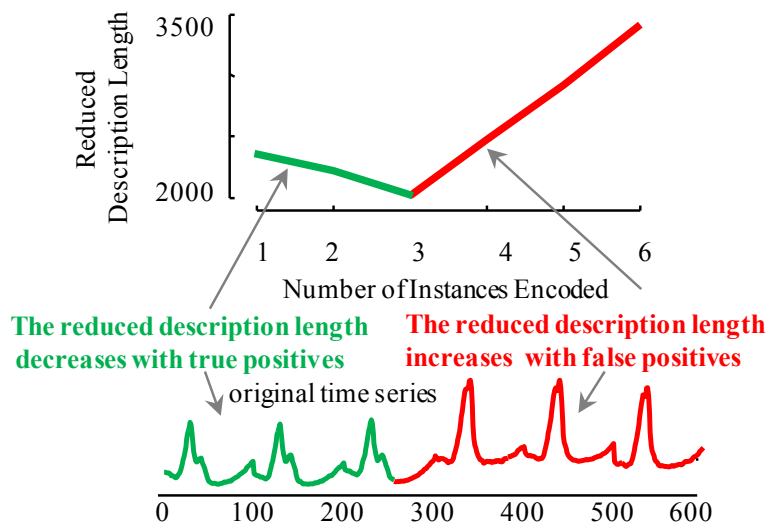
Fig. 12. *top***) Reduced Description Length curve after encoding the instances shown in** *bottom)*

In Table 3, we modify the algorithm shown in Table 2 to allow this MDL based stopping criterion.

Table 3. Stopping Criterion for Self-Training Time Series Classification Algorithm

Output Input	
	stop_flag = **find_stop_point** (*P, N*)
	P, set of labeled/positive instances.
	// initially |*P*| = 1
	N, set of unlabeled/negative instances
	stop_flag, a Boolean flag indicating the stopping point initialized to false
1	hypothesis = *P; // the initial seed is hypothesis*
2	DL_old = inf ;
3	**while** (*N* is **not** empty)
4	*pos_Obj* = Self_Training_Classifier (*P, N*);
5	DL_cur = description length of *pos_Obj*;
6	**if** DL_cur < DL_old
7	DL_old = DL_current;
8	**else**
9	*stop_flag* = true ;
10	**break**;
11	**end if**
12	**end while**

2.5 Experimental Results

We begin by stating our experimental philosophy. In order to ensure that our experiments are reproducible, we have built a website which contains all data and code [44]. In addition, this website contains additional experiments that are omitted here for brevity.

2.5.1 MIT-BIH Supraventricular Arrhythmia Database

We consider an ECG dataset from the MIT-BIH Supraventricular Arrhythmia Database (SVDB) [12]. Each recording in this database includes one or more ECG

signals and a set of beat annotations by cardiologists. Out of the 78 half-hour ECG recordings in this database beat annotations by cardiologists, we randomly chose record 801 and signal ECG1 for our experiment. Here we formulate the problem of classifying heartbeats as a 2-class problem and because identifying abnormal heartbeats are of more interest to cardiologists, our target class (i.e. the positive class) consists of only abnormal heartbeats. This dataset contains 268 abnormal heartbeats (Premature Ventricular Contraction or PVC, in this case) and our self-training classifier classifies 266 of them correctly.

Starting with a labeled set S with only one initial positive instance, our self-training classifier augments S by adding one instance at a time. As our algorithm iterates, we calculate the reduced description length DL_S of our labeled set. As expected, with the augmenting of the labeled set with more and more true positive (TP) instances, DL_S continues decreasing in our experiment. However, from the point we start adding true negatives (TN) in S, DL_S continues increasing. As a sanity check, we augmented the size of our labeled set up to 700, and found no change in the curve pattern of our DL_S vs. number of time series encoded. The results are shown in Fig. 13. From this figure, we can see that we still allow a tiny number of TP instances beyond the stopping point suggested by our algorithm. Nevertheless, this does not significantly hurt the performance of our classifier in terms of accuracy.
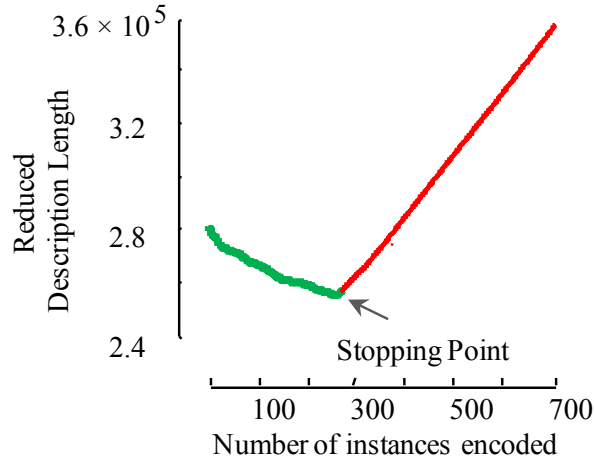
Fig. 13. Reduced Description Length vs. no. of encoded instances plot for the SVDB dataset. Green/thick and red/thin points denote TP and TN instances respectively.

2.5.2   St. Petersburg Arrhythmia Database

Next we consider classifying the heartbeats from the St. Petersburg Institute of Cardiological Technics 12-lead Arrhythmia Database (INCARTDB) [11] using our self-training classifier. This database has 75 annotated readings extracted from 32 Holter records. Each record, sampled at 257 Hz, contains 12 standard leads. Over 175,000 beat annotations are stored in the reference annotation files. An automatic algorithm produced the annotations and then corrections were made manually. We randomly chose record I70 and signal II for our experiment. This dataset contains 126 Atrial Premature Beat, and our self-training classifier classifies 120 of them correctly.

Fig. 14 shows that we obtained similar results to the last experiment.

31
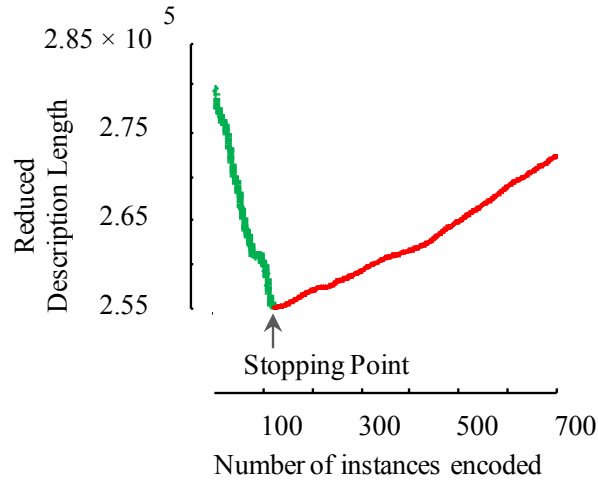
Fig. 14. Reduced Description Length of the labeled set vs. no. of encoded time series plot for the INCARTDB dataset.

As we augment our labeled set with TP instances, we obtain a decreasing trend of DL_S. The moment we exhaust the TP data and start ingesting TN instances, DL_S starts increasing. The result is obvious; the inclusion of similar patterns to the hypothesis pattern in the labeled set helps achieving further compression of the data, and we find decreasing DL_S. However, when we start including the TN instances in our labeled set, DL_S keep on increasing, because of the overhead associated with the encoding of the mismatched portions. From Fig. 14, we stop *exactly* at the instant where the TN instances start being included in the labeled set.

2.5.3   Sudden Cardiac Death Holter Database

This experiment classifies the heartbeats of the Sudden Cardiac Death Holter Database (SDDB) [13], which is archived by Physionet [11] to support research on a wide variety of substrates responsible for sudden cardiac death syndromes. Such sudden

deaths happen to 400,000 Americans and millions more worldwide each year and therefore, is of great importance.

The Sudden Cardiac Death Holter Database [13] includes 18 patients with underlying sinus rhythm. All these patients had a sustained ventricular tachyarrhythmia, and most had an actual cardiac arrest. Out of the twenty three records, we randomly chose record number 52 and signal ECG (1st) for our experiment. This dataset contains 216 R-on-T Premature Ventricular Contraction, and our self-training classifier classifies 193 of them correctly.

We show the experimental results for this dataset in Fig. 15. We obtain similar results to the two datasets described before. However for this dataset, the stopping point is not exactly at the point where the TN instances start. Beyond the stopping point, we obtain some TP instances, nevertheless, these are few in number, and the overall accuracy is not affected significantly.
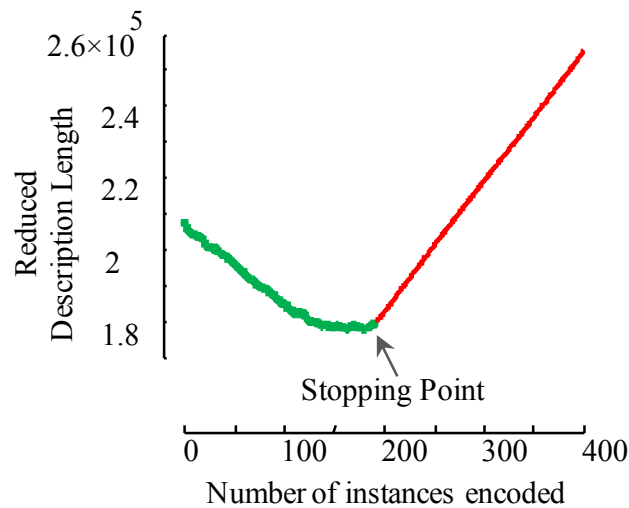
Fig. 15. Reduced Description Length of the labeled set vs. no. of encoded time series plot for the SDDB dataset.

2.5.4 Additional Examples

We conclude our experimental results for three additional datasets in Fig. 16. This datasets are not from a medical domain, and are designed to hint at the generality of our ideas.

The result in Fig. 16.*left* corresponds to the SwedishLeaf_TEST dataset [19]. For this dataset, beyond the stopping point, we obtain a tiny number of TP instances. For the Fish_TEST dataset [19], we stop exactly at the moment when the TN instances begin to be included in the labeled set (Fig. 16.*center*). However, beyond this point, we obtain *very* few TP instances, which do not hurt our classifier performance significantly.
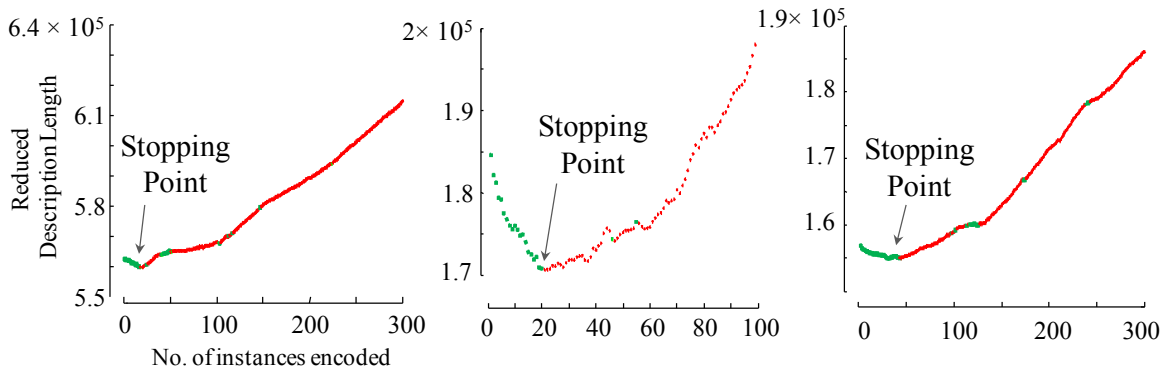
Fig. 16. Reduced description length of the labeled set vs. no. of encoded time series plots. *left*) SwedishLeaf_TEST Dataset. *center*) Fish_TEST Dataset. *right*) FaceAll_TEST Dataset.

For the FaceAll_TEST dataset [19], we obtain some TP instances beyond our stopping point (Fig. 16.*right*). This is because we started with a *single* instance to seed in our training set, and this dataset is known to be polymorphic. Given just a single example, it is hard for the classifier to distinguish between the faces of the same person, let us say, *with* and *without glasses*. Nevertheless, our algorithm could still identify the

point where the vast majority TP instances were classified already, and the TN instances started to be included in the training set.

2.5.5 Comparison with Rival Approaches

We perform a comparison between our algorithm and the state-of-the-art algorithm [41] on the Fish_TEST dataset [19]. From Fig. 17, the readers can easily observe that our algorithm suggests a better stopping point (Fig. 17.*bottom*) compared to the too conservative stopping point (Fig. 17.*top*) suggested by the state-of-the-art algorithm.
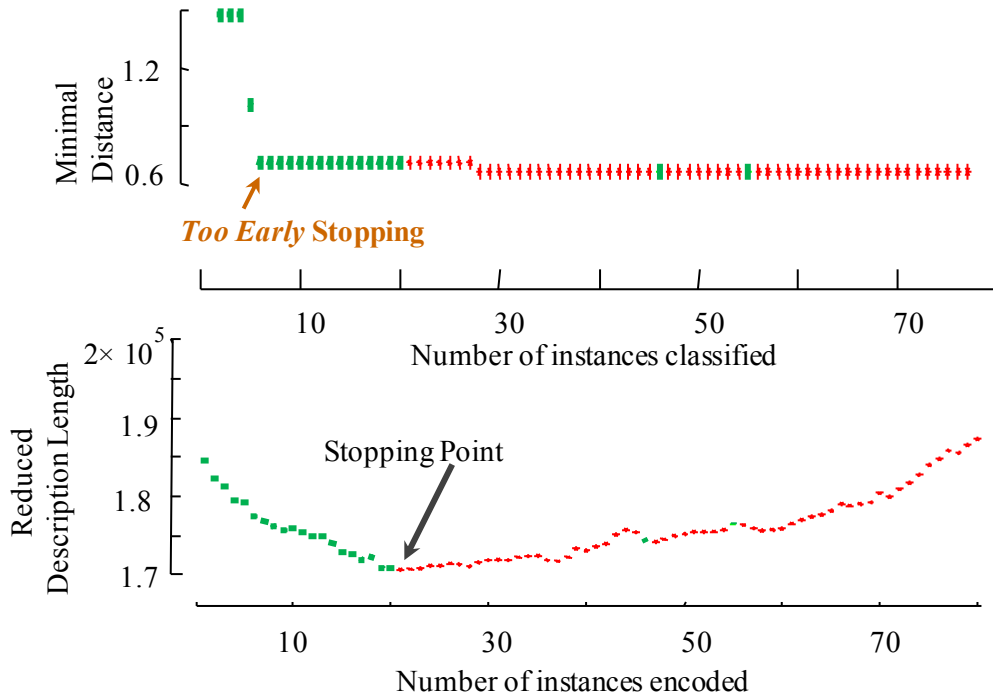
Fig. 17. Comparison of our algorithm (*bottom*) with the state-of-the-art algorithm (*top*) [41] for the Fish_TEST dataset [19]. Results show that the state-of-the-art algorithm stops too early, whereas our algorithm suggests better stopping point.

Experiments on other datasets yielded similar results. In Fig. 18, we show the results for SVDB, INCARTDB and SDDB datasets respectively.
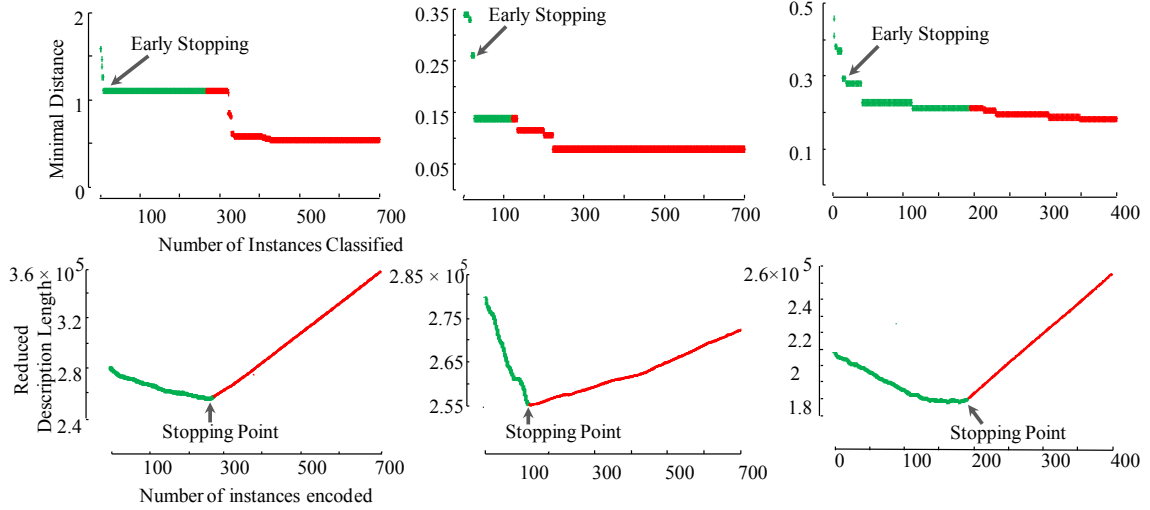
Fig. 18. Comparison of our algorithm (*bottom*) with the state-of-the-art algorithm (*top*) [41] for SVDB (*left*), INCARTDB (*center*) and SDDB (*right*) datasets.

From Fig. 18.*top*, we can see that for all the three datasets, the state-of-the-art algorithm [41] is too conservative and therefore, missed too many true positives. In contrast, our algorithm suggested a better stopping point (Fig. 18.*bottom*) with *almost* no prohibitive expansion of false positives or false negatives.

2.5.6 ROC Curve

We measure the performance of our self-training classifier by varying the discrimination threshold, which is the stopping point. We consider all instances before the stopping point as positive instances and all instances after the point as negative instances. Out of some potential stopping points, we look for the point where our algorithm actually suggests stopping the classification process. Experimental results show that our classifier stops at a point with majority of the TP instances identified with almost no FP instances selected (Fig. 19).
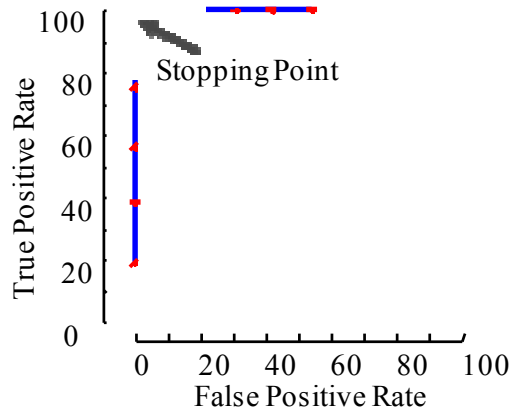
Fig. 19. ROC curve for the SVDB dataset. The stopping point denotes a point with majority of the TP instances identified, with almost no FP instances selected.

2.5.7 When does the Algorithm Fail?

Our MDL based algorithm for finding the stopping criterion of a self-training classifier does not perform well when the patterns of the two classes under consideration look very similar. As an example, we show the results for the TwoLeadECG_TEST dataset [19] in Fig. 20.

From Fig. 20, we can see that because the positive and negative instances are visually very similar, therefore, the expected increase in the Reduced Description Length separating the two class boundaries is not obvious in this case.

The reader may believe that there must be *some* difference that allowed the original annotator of the data to make class distinctions in the first place. However, it must be noted that the original annotator has access to additional information, including other telemetry recorded in parallel, which is not available to our algorithm. It is not clear if

even an expert human could do better than our algorithm, given the same view of the data.
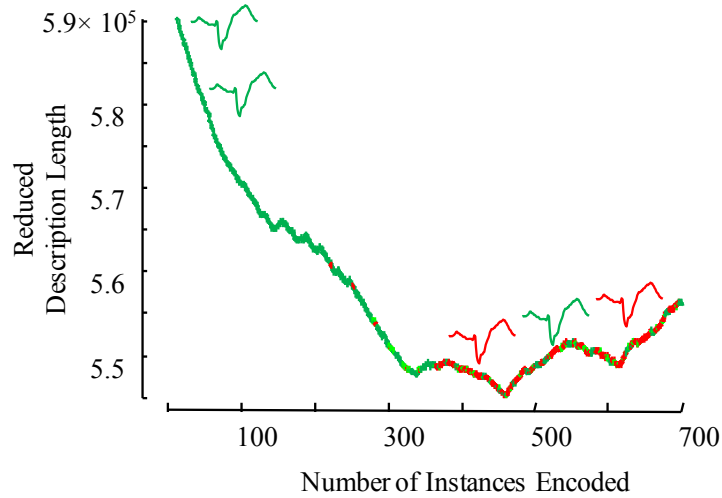
Fig. 20. Reduced description length of the labeled set vs. number of encoded time series plot for the TwoLeadECG_TEST dataset. The positive/green and negative/red instances are very similar looking, and therefore the MDL based approach does not work well.

2.6 Conclusions and Future Work

In this chapter, we proposed a novel method of semi-supervised classification of time series with as few as a single labeled instance. Previous approaches for stopping the classification process required extensive parameter tuning, and hence remain something of a black art. We devised a stopping criterion for the semi-supervised classification based on MDL, which is parameter free, and leverages the intrinsic structure of the data. To our knowledge, this is the *first* work addressing time series SSL using MDL. Extensive experiments demonstrate that our method allows us to classify real-world medical datasets with near perfect accuracy.

Our current approach works only on offline time series data; we plan extending it to perform online semi-supervised time series classification. We also plan to generalize our algorithm to the setting of multiple classes.

2.7 Acknowledgement

Chapter 3: A General Framework for Density Based Time Series Clustering Exploiting a Novel Admissible Pruning Strategy

Time Series Clustering is an important subroutine in many higher-level data mining analyses, including data editing for classifiers, summarization, and outlier detection. It is well known that for *similarity search* the superiority of Dynamic Time Warping (DTW) over Euclidean distance gradually diminishes as we consider ever larger datasets. However, as we shall show, the same is not true for *clustering*. Clustering time series under DTW remains a computationally expensive operation. In this work, we address this issue in two ways. We propose a novel pruning strategy that exploits *both* the upper and lower bounds to prune off a very large fraction of the expensive distance calculations. This pruning strategy is admissible and gives 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 into an *anytime* framework. We demonstrate the utility of our ideas with both single and multidimensional case studies in the domains of astronomy, speech physiology, medicine and entomology. In addition, we show the generality of our clustering framework to other domains by efficiently obtaining semantically significant clusters in protein sequences using the *Edit Distance*, the discrete data analogue of DTW.

Keywords: Clustering; Time Series; Anytime Algorithms

3.1 Introduction

Because of the prevalence of time series data in human endeavors, the research community has made substantial efforts to create efficient algorithms for classification, clustering, rule discovery, and anomaly detection for this data type [46][49][67][79][93]. In particular, time series *clustering* is very useful, both as an exploratory technique and as a sub-module for solving higher-level data mining problems. As a motivating example, consider Fig. 21, which illustrates a subset of a cluster we discovered in a social media dataset [103]. This clustering allows us to at least partly address two problems:

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

- **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 that the famous "*I'mma let you finish*" event at the 2009 Video Music Awards produced this relationship [106].
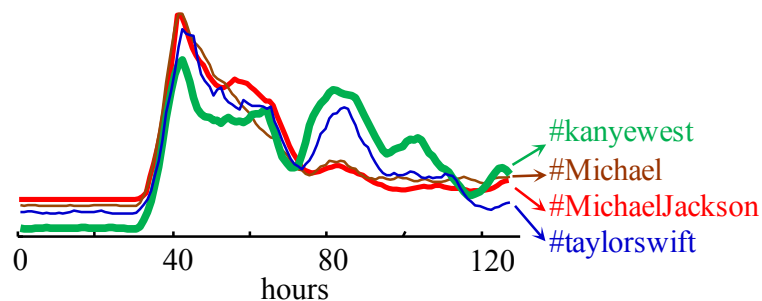
Fig. 21. A cluster of four Twitter hashtag usage time series (normalized for volume) over ~6 days starting from June 12, 2009 [103]. (Best viewed in color)

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 [92]. Note that while we discovered *this* example using Dynamic Time Warping (DTW), it might have been discovered more efficiently with the Euclidean distance (ED). However, in cases where there is a *causal relationship* (rather than just an *association*) between events, a local lag can result between peaks. It has been extensively shown in the literature that the 40-year old distance DTW is an ideal similarity measure to capture/be invariant to such out-of-sync relationships [91].

We argue that the problem we wish to solve, *robustly clustering large time series datasets with invariance to irrelevant data,* has not been solved before.

For most time series data mining algorithms, the quality of the output depends almost exclusively on the distance measure used [93]. In the last decade, a consensus has emerged that the DTW distance measure is the best measure in most domains, almost always outperforming the Euclidean Distance (ED) and other purported rivals [93]. As a concrete example, consider the two clusterings of three randomly chosen mammals shown in Fig. 22. The input data is the mitochondrial DNA after it was converted to a

42

time series representation (converting DNA to time series is a commonly used operation [84][85]). 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.
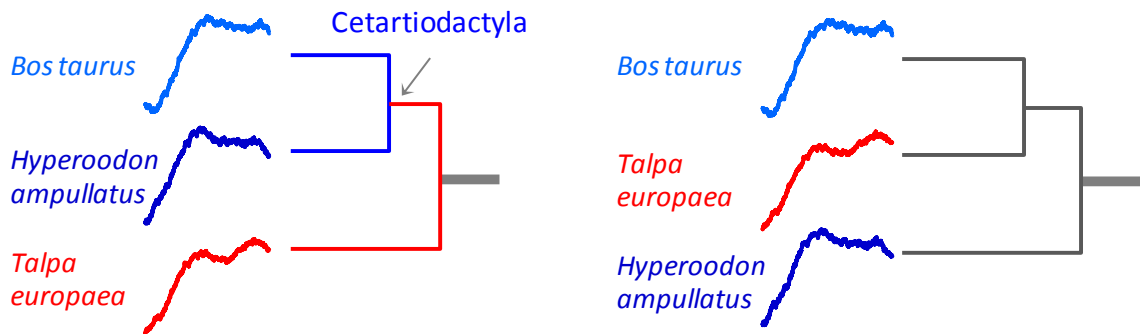
Fig. 22. Single-linkage hierarchical clusterings of DNA using DTW (*left*) and Euclidean distance (*right*).

While this example[1] is on a small and somewhat specialized dataset in the domain of bioinformatics, in Section 3.6 we will show that the superiority of DTW extends to large datasets in many domains including discrete protein data.

3.1.1   Why This Problem is Hard

Because DTW is intrinsically slow due to its quadratic time complexity, there are two ideas that are commonly used to mitigate the problem of using such a sluggish

---

[1] We defer a discussion of our experimental philosophy until Section 3.5, but we note that all experiments in this work are made reproducible by our unrestricted sharing of code/data.

distance measure [85]. We briefly discuss them here only to dismiss them as possible solutions.

- *The convergence of DTW and Euclidean distance results for increasing data sizes.* It has been noted that for many problems, including motif discovery [79] and classification [93], the results returned by DTW and Euclidean distance tend to become increasingly similar as the dataset sizes increase. This suggests that it is more efficient to use 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 dramatically as the datasets get larger [85].

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 1-NN classification using both DTW and ED, for various numbers (50 to 2000) of exemplars from the CBF dataset [69]. With 50 objects, the error rates differ by a factor of 4.6 (7% and 1.75%, respectively), but as shown in Fig. 23.*top*, by the time we consider the 2,000 object dataset, this difference is essentially zero.

This effect is well known for time series *classification* [88][93], and it might be imagined that this also 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 [87]. As shown in Fig. 23.*bottom*, DTW *clustering* maintains its superiority over Euclidean distance as the datasets get larger.
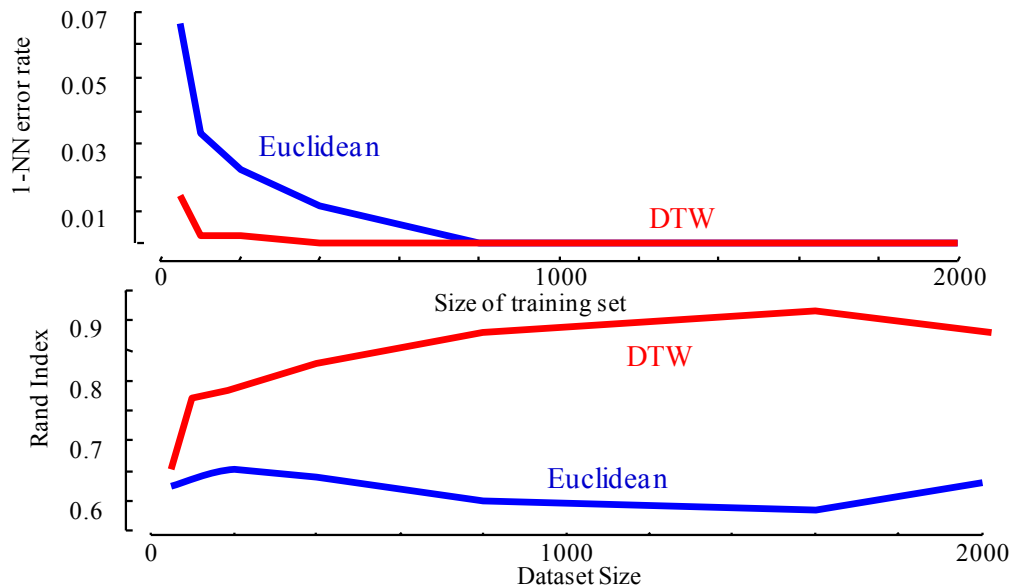
Fig. 23. *top*) **The classification error rates of DTW and ED tend to converge as we see more training data.** *bottom*) **In contrast, for clustering, DTW retains its great superiority over ED for increasingly large datasets.**

Similarly, the second observation above does not help significantly to prune DTW distance computation for clustering. 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, which allows more effective pruning [85][88][93]. However, in *clustering*, we need to know the distance between all pairs [67], or at least all distances within a certain range, which renders the typical use of lower-bounding pruning ineffective.

3.1.2   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 [56]. 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 especially important to allow a time series clustering algorithm the freedom to ignore some data.

Consider the example in Fig. 24. We took twelve objects from a heraldic shield dataset [107] and clustered these using k-means and DP, the algorithm we propose to augment (described in detailed in Section 3.4.1). Because we are using the (non-metric) DTW measure, which may prevent k-means from converging, we used the variant in [65] which performs k-means clustering using the all-pair distance matrix. Note that for the 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 a perfect Rand Index score of 1.0. We then inserted a single outlier object (object 12) from this dataset and reran the algorithms. As we can see from Fig. 24.*bottom.left*), k-means assigned objects 6-11 to the cluster of the outlier object. In addition, k-means falsely identified objects 1 and 2 as a separate cluster from the cluster of objects 3-5. In contrast,

from Fig. 24.*bottom.right*) we can see that DP only clustered object 7 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 and wide-reaching experiments on real data [108].
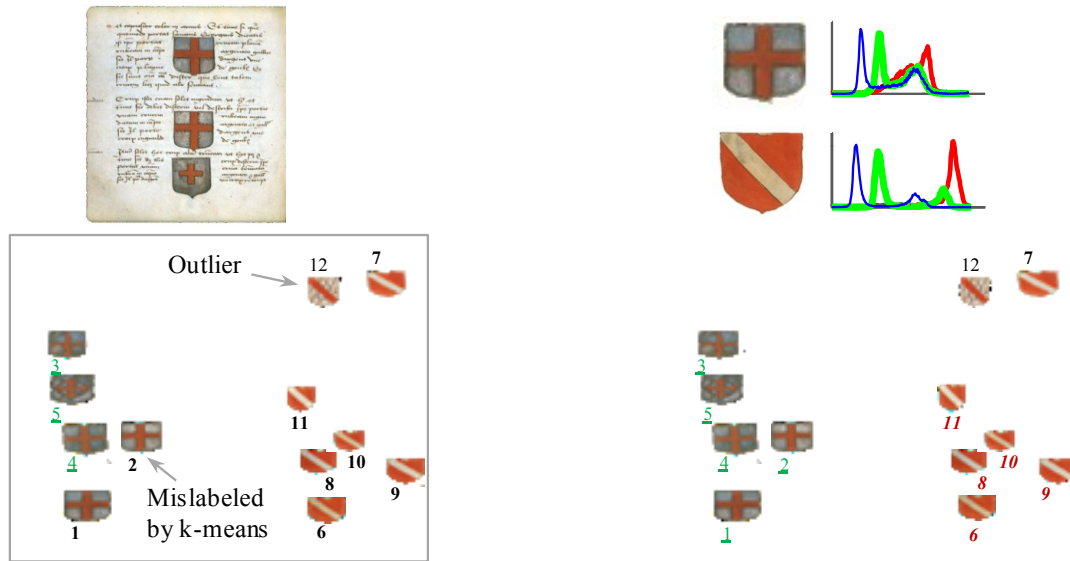
Fig. 24.: *top.left*) **Leaf 18V of a 15ᵗʰ-century book,** *Treatises on Heraldry* **[107].** *top.right*) **The colorful heraldic shields can be converted to 3D RGB "time series" of color distribution.** *bottom.left*) **Even the insertion of a single outlier can confuse k-means.** *bottom.right*) **In contrast, the performance of the DP algorithm is not sensitive to outliers.**

In this work, we address all the considerations above. We adapt DP (<u>D</u>ensity <u>P</u>eaks), a relatively new clustering framework that is able to ignore outlying data points [89]. While DP is insensitive to outliers, it 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 [48][105].

3.2 Related Work

The field of clustering is vast, and even the subfield of clustering time series has an enormous literature [46][67][101][104]. Much of the works on time series clustering are concerned with clustering based on time series *features* [101], 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 such as Euclidean distance [54][101]. 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 Fig. 23.*top* illustrates with a single representative example, the general superiority of DTW over ED is well understood in the community (cf. [93]), at least for *classification*. As Fig. 23.*bottom* suggests, and as we later empirically confirm on many diverse datasets, the dominance of DTW over ED for *clustering* is, if anything, much greater.

The plethora of shape-based clustering algorithms [67][86][104] can be divided at the highest level into those that insist on explaining (i.e., *clustering*) all the data [104] vs. those that have the representational power to leave some data unclustered (a small minority) [86]. 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*, 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, if we were clustering subsequences from a speech articulation database (see Section 3.6), we would hope that the subsequences would cluster into well-defined words or phrases. However, it is highly likely that we would have some examples of coughing, sneezing or harrumphing. Such sequences are likely to be very dissimilar to the rest of the database. Not only do we not want/need them to be clustered, but also we do not want them to affect the clustering of the clusterable words or phrases (recall Fig. 24). 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 quality of the overall clustering.

One of the basic questions in clustering problems is the notion of a 'cluster' itself. There exist partitional clustering algorithms such as k-means which typically assume data has balanced Gaussian "ball-shaped" clusters. Whereas some other algorithms (e.g. DBSCAN [56]) take the *density* of objects into consideration regardless of the shape the clusters may have. The intuition behind DBSCAN algorithm is that, each object in a cluster must have at least some number of other objects (*MinPts*) in its vicinity (ε). If two objects are *density-reachable* from each other, then they are assigned in the same cluster. However, DBSCAN is not deterministic in its assignment of the cluster border points, because the result specifically depends on the order of the objects considered. Besides, DBSCAN has two parameters, *MinPts* and ε, and there is no concrete strategy for setting these parameters.

There exist works [77] in the literature that perform clustering on top of DBSCAN [56]. 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 variant of DBSCAN called IncrementalDBSCAN [57] has been demonstrated to perform much better than the brute-force re-clustering of newly added/removed objects which is mostly appropriate for datasets changing incrementally. DBCLASD [102] is a non-parametric grid-based clustering algorithm which considers the density of objects like DBSCAN to define clusters. It exploits a grid-based approach to find polygons of clusters. However, this algorithm cannot be applied to high-dimensional time series domains. DENCLUE [62] is an algorithm which combines both density-based and grid-based approaches to define cluster centers that has the local maximum of some density function. The cluster assignment of other objects in the datasets is done by a hill-climbing approach. There exist other types of clustering algorithms like hierarchical and model-based [61]. However, such algorithms are known to be quite slow and not appropriate for our context.

A handful of research efforts [104] 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 will calculate the full distance matrix (generally in some "*most-likely-to-be-useful*" order) and return the exact

clustering. We refer readers interested in time series clustering to the detailed surveys [70][74].

Our proposed algorithm goes beyond this literature 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.

3.3 Background

There has been significant research on clustering datasets that are too large to fit in main memory [50][51]. This problem setting typically assumes inexpensive distance measures, but costly disk accesses [51]. 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.

3.3.1 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 [72]. The fact that some distance computations are more important than others, immediately suggests that we should use *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 [104][105]. 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 [105]. Note that this is a very brief introduction to anytime algorithms; we refer the interested reader to [105], which contains an excellent survey. As Fig. 25 shows, anytime algorithms, in essence, optimize the tradeoff between execution time and quality of the solution.
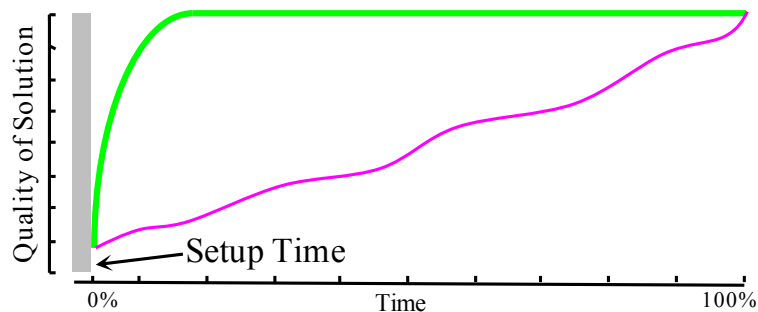
Fig. 25.: An abstract illustration of an anytime algorithm. The three curves show a comparison of the possible performances of three hypothetical anytime algorithms. The bottommost curve (pink) is only improving linearly over time, but the topmost curve (green) demonstrates *diminishing returns*, making most of its improvements early on.

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

3.4 Algorithm

We begin by reviewing the basic algorithm that we will augment and specialize to handle DTW.

3.4.1 Density Peaks Algorithm Overview

Our proposed solution is inspired by DP, the density-based clustering algorithm recently proposed in [89]. We chose to augment the DP framework for solving large time series clustering problems because of the following:

- Recent literature [86] and our own experience on real datasets (cf. Section 3.6) suggest that the successful clustering of time series requires the ability to ignore some data objects. The issue is not merely that anomalous objects themselves are unclusterable; but 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. These parameters 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.

To make our argument more concrete, we will take the time to explain the clustering algorithm [89] we adapt and augment. 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 9 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 10 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 4 and δ_i in Table 5.

Table 4: Local Density Calculation Algorithm

Input:	**D,**all-pair distance matrix
	d$_c$, 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)<d$_c$)
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 4. In Table 5, using the local densities ρ from Table 4, 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 has no 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 5: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 6.

Table 6: 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 [89].

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 7.

Table 7: 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.

3.4.2   TADPole: Our Proposed Algorithm

We call our algorithm, TADPole (Time-series Anytime DP). As stated in Section 3.1, 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 [68], LB_Kim, LB_Yi [101], 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, which results 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, which quickly produces a good answer and rapidly refines it until it converges to the exact answer (for proof, c.f. Section 3.9).

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 [89]. Because we use DTW as the underlying distance measure, the warping window size is another parameter for TADPole. We discuss the heuristic to set these parameters in Section 3.5.3. Note that our use of 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 be encoded as a sparse matrix.

- If needed, both the upper and lower bound matrix's could be computed in a *just-in-time* fashion [85]. This would greatly reduce the memory footprint, at the expense of a more complicated implementation.

For clarity of presentation, we present our contributions in two different sections, although the final algorithm incorporates both ideas. In Sections 3.4.2.1 to 3.4.2.4, 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 3.4.2.5, we show how to reorder these computations to give us the *diminishing returns* property of anytime algorithms [48][105].

3.4.2.1 Pruning During Local Density Calculation

Consider the four cases shown in Fig. 26.

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 4), we prune their distance (D_{ij}) computation according to the following four cases shown in Fig. 26.

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 (Fig. 26.A)).

59

d_c

A) ij $D_{ij}=0$

i j

$\leftarrow \text{LB}_{\text{Matrix}}(i,j) \rightarrow$

B) $\leftarrow$ D_{ij} $\rightarrow$

$\leftarrow$ $\text{UB}_{\text{Matrix}}(i,j)$ $\rightarrow$

i j

$\leftarrow$ $\text{LB}_{\text{Matrix}}(i,j)$ $\rightarrow$

C) $\leftarrow$ D_{ij} $\rightarrow$

$\leftarrow$ $\text{UB}_{\text{Matrix}}(i,j)$ $\rightarrow$

i j

$\leftarrow \text{LB}_{\text{Matrix}}(i,j) \rightarrow$

D) $\leftarrow$ D_{ij} $\rightarrow$

$\leftarrow$ $\text{UB}_{\text{Matrix}}(i,j)$ $\rightarrow$
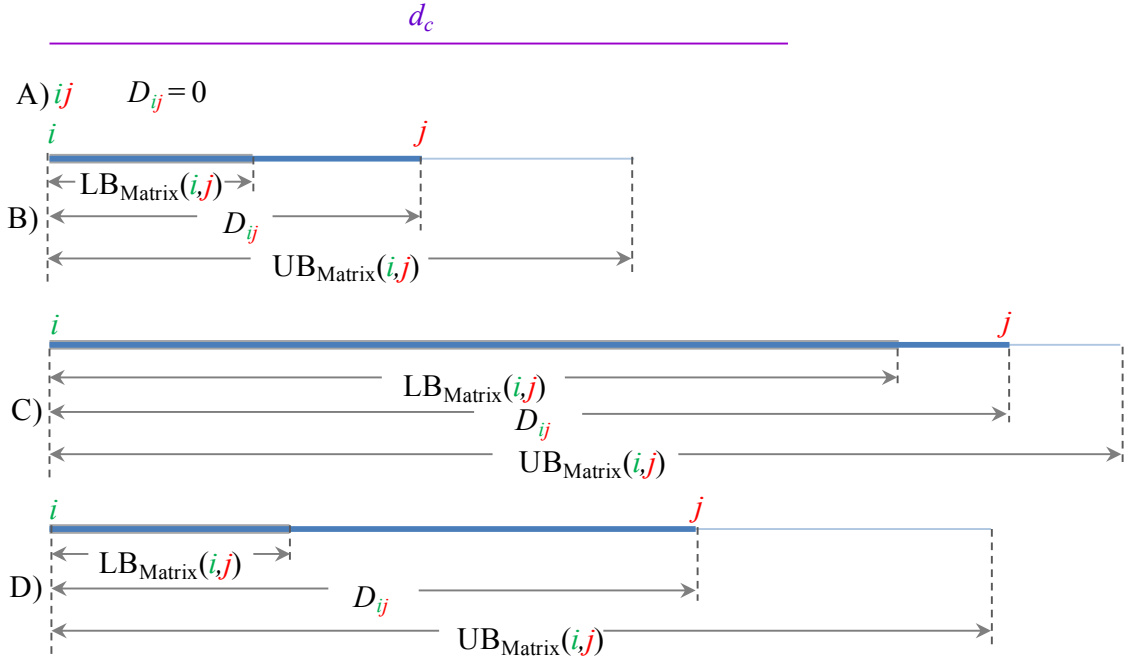
Fig. 26.: The four mutually exclusive and exhaustive cases of distance computation pruning during local density calculation. Note that the cutoff distance d_c, represented by the purple line at the top, applies to all four cases below it. Case A is difficult to visually represent, as i and j coincide.

Case B: $\text{UB}_{\text{Matrix}}(i,j) < d_c$

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 (Fig. 26.B)). Therefore, we can prune the DTW distance computation of these two objects.

Case C: $\text{LB}_{\text{Matrix}}(i,j) > d_c$

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 (Fig. 26.C)). We can therefore admissibly prune their DTW distance computation.

Case D: $\text{LB}_{\text{Matrix}}(i,j) < d_c$ and $\text{UB}_{\text{Matrix}}(i,j) > d_c$

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} (Fig. 26.D)).

With this intuition in mind, we specify the formal distance pruning algorithm during the local density calculation in Table 8.

Table 8: Pruning Algorithm during Local Density Calculation

Input:	$\mathbf{LB_{Matrix}}$, full computed lower bound matrix
	$\mathbf{UB_{Matrix}}$, full computed upper bound matrix
	Data, the dataset
	$\mathbf{d_c}$, cutoff distance
Output:	$\boldsymbol{\rho}$, local density vector for all points in dataset
	$\mathbf{D_{Sparse}}$, partially filled distance matrix

1	D_{Sparse} = empty
2	**for** i = 1:**size**(Data)
3	objectsWithin_dc = empty
4	**for** j = 1: **size**(Data)
5	**if** i == j
6	**continue**;
7	**else**
8	**if** LB_{Matrix}(i,j) == UB_{Matrix}(i,j) //case A
9	**continue**
10	**elseif** UB_{Matrix}(i,j) < d_c //case B
11	objectsWithin_dc = [objectsWithin_dc j]
12	**elseif** LB_{Matrix}(i,j) > d_c // case C
13	**continue**
14	//case D)
15	**elseif** LB_{Matrix}(i,j) < d_c **and** UB_{Matrix}(i,j) > d_c
16	D_{Sparse}(i,j) = calculateDist(Data(i),Data(j))
17	**if** D_{Sparse}(i,j) < dc
18	objectsWithin_dc = [objectsWithin_dc j]
19	**end if**
20	**end if**
21	**end if**
22	**end for**
23	ρ(i) = **length**(objectsWithin_dc)
24	**end for**

As we can see from Table 8, 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 is 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 5, 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.

3.4.2.2 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 $\text{UB}_{\text{Matrix}}(i,j)$ 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 illustration of this upper bound calculation in Fig. 27.
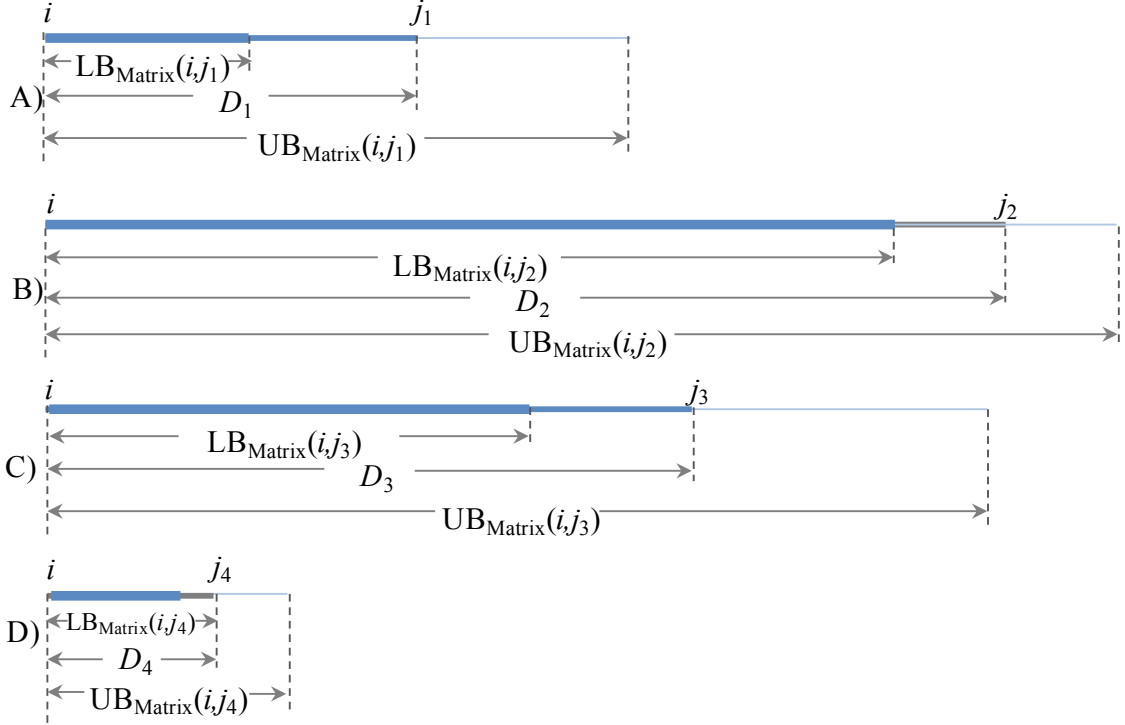
Fig. 27.: An illustration of the distance pruning during the NN distance calculation from a higher density list of an object. From object i, the elements in the higher density list are $j_1 - j_4$. After Phase 1, ub_i will be $\text{UB}_{\text{Matrix}}(i,j_4)$. In Phase 2, the distance computations of D_{ij2} and D_{ij3} are pruned.

In Fig. 27, 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 Fig. 27. 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 $\text{UB}_{\text{Matrix}}(i,j_2)$ and D_3 are both are greater than ub_i, we do not need to update ub_i. In the last step, given that $\text{UB}_{\text{Matrix}}(i,j_4) < ub_i$, 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 9. 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 9: 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 10.

We begin by scanning the higher density list of each of the objects again. In line 5 of Table 10, for an object *i,* we test whether $LB_{Matrix}(i, δ\_list_i(j))$ is greater than ub_i we calculated in Table 9 . If this is the case, we prune the distance computation (line 6) for $δ\_list_i(j)$. Otherwise, if the *true* distance between *i* and $δ\_list_i(j)$ 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 Fig. 27, 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.

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 6, and performs the cluster assignments based on these centers according to the algorithm in Table 7.

Table 10: 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)$\neq$ 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**

3.4.2.3 Multidimensional Time Series Clustering

While most of the research efforts on time series clustering have considered only the single-dimensional case [67][86], the increasing prevalence of medical sensors (c.f. Section 3.6.2) and wearable devices (c.f. Section 3.6) has given urgency to the need to support multidimensional clustering [94]. 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 8 and Table 10 in Table 11 and Table 12, respectively.

Table 11: Pruning Algorithm during Local Density Calculation for Multidimensional Data (see Table 8)

Input:	$\mathbf{LB_{Matrix}}$, full computed lower bound matrix along d dimensions
	$\mathbf{UB_{Matrix}}$, full computed upper bound matrix along d dimensions
	Data, the dataset
	$\mathbf{d_c}$, cutoff distance
Output:	$\boldsymbol{\delta}$, NN distance vector of higher density points
16	... $D_{Sparse}(i,j) = \sum_{dim=1}^{d} \text{calculateDist}(Data_{dim}(i), Data_{dim}(j))$ ...

Table 12: Pruning Algorithm during NN Distance Computation from Higher Density List, Multidimensional Case (see Table 10)

10	**else** // calculate distance
11	$D_{Sparse}(i, \text{highDensityItem}) =$
12	$\sum_{dim=1}^{d} \text{calculateDist}(Data_{dim}(i), Data_{dim}(\text{highDensityItem}))$

Recall that in Table 8, 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 8. In addition, when we actually calculate the distances (line 16 in Table 8 and lines 10-12 in Table 10), we take the summation of the distances along all the dimensions. All other components of TADPole will remain the unchanged. We explicitly evaluate the utility of TADPole clustering for multidimensional clustering in Section 3.6.4. More generally, as we shall show empirically in Section 3.5, by using the methods described above, TADPole can obtain *at least* an order of magnitude speedup over the original DP algorithm while producing identical results.

3.4.2.4 How Effective is Our Pruning?

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

- In order to perform clustering, the DP algorithm needs the all-pair distance matrix computed [89]. Therefore, in terms of distance computation, the brute force DP algorithm itself is the obvious worst-case straw man.

- 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 [69]. We vary the number of objects in the dataset we need to cluster (by random sampling) and record the number of *true* DTW distance computations. As we can see from Fig. 28.*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.
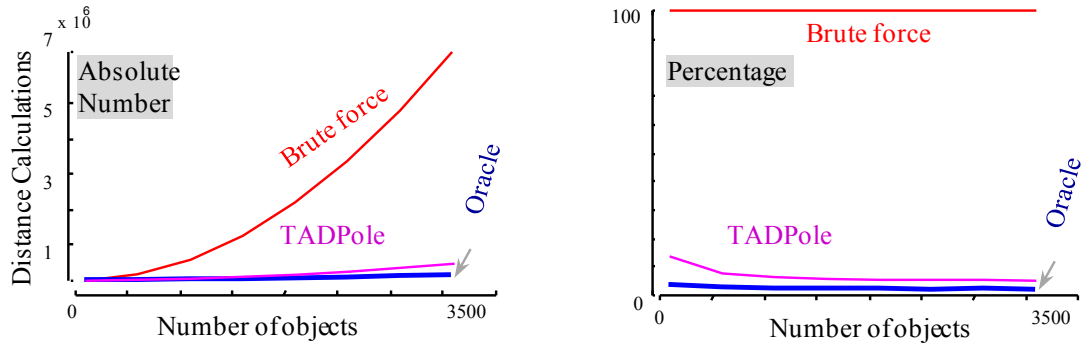
Fig. 28.: A comparison of the CPU time spent by TADPole against an oracle and the brute force algorithm to cluster the StarLightCurves dataset. As before, the performance of TADPole is very close to the oracle algorithm.

This claim is reinforced in Fig. 28.*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 [108] for brevity). Moreover, we obtain similar results if we measure the CPU time instead. As we can see from Fig. 29, to cluster the StarLightCurves data, TADPole requires only 9 minutes, whereas the DP algorithm needs 9 hours.

To put these results into context we consider the results in a very recent research effort. In [77], the authors discuss the scalability of their clustering algorithm saying that "*For a large time series dataset with 9,236 objects with the length of each object n = 8192, it costs only 2.1 seconds to transform the whole dataset and an hour for clustering with DBSCAN*". Using this as a benchmark, we did the following experiment: we took

9,236 random walks each of length 8,192, and clustered this dataset using DP algorithm. DP took only 16 seconds to cluster this dataset. Of course the hardware settings of these two experiments may not be exactly commensurate, but it is clear we have lost nothing in terms of scalability by considering adopting the DP algorithm rather than the near ubiquitous DBSCAN [77]. As we will later show, we have also lost nothing in terms of accuracy.
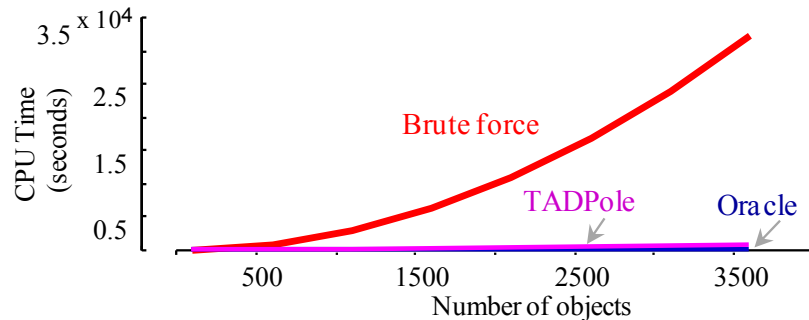
Fig. 29.: A comparison of the CPU time spent by TADPole against an oracle and the brute force algorithm to cluster the StarLightCurves dataset. As before, the performance of TADPole is very close to the oracle algorithm.

In spite of these very promising results, which demonstrate 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 [96] and outlier detection [48][49], researchers have attempted to create *anytime* versions of their algorithms [48][104]. 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.

3.4.2.5 Distance Computation-Ordering Heuristic

Recall from the above that the DP algorithm may be considered a two-step algorithm; calculating the local densities (Table 4) first, and then finding the NN distances from the higher density lists (Table 5) of the objects. Only the latter step is amiable to anytime ordering; the former step may be regarded as the *setup time* [48][96][104].

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 straw men?

- 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 straw man 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 [69] and measured the Rand Index as the two algorithms above kept refining the mixture of true DTW distances and upper bound distances that we have at the end of phase 1 (i.e., Table 9), into the set of all *necessary* DTW computations needed (i.e., Table 10). The results are shown in Fig. 30 (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.

Being initially 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 Fig. 30. 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 3.4 that points with higher values of $\rho \times \delta$ 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.

72

Because the centers are selected earlier, we achieve a higher Rand Index with very few

actual distance computations.
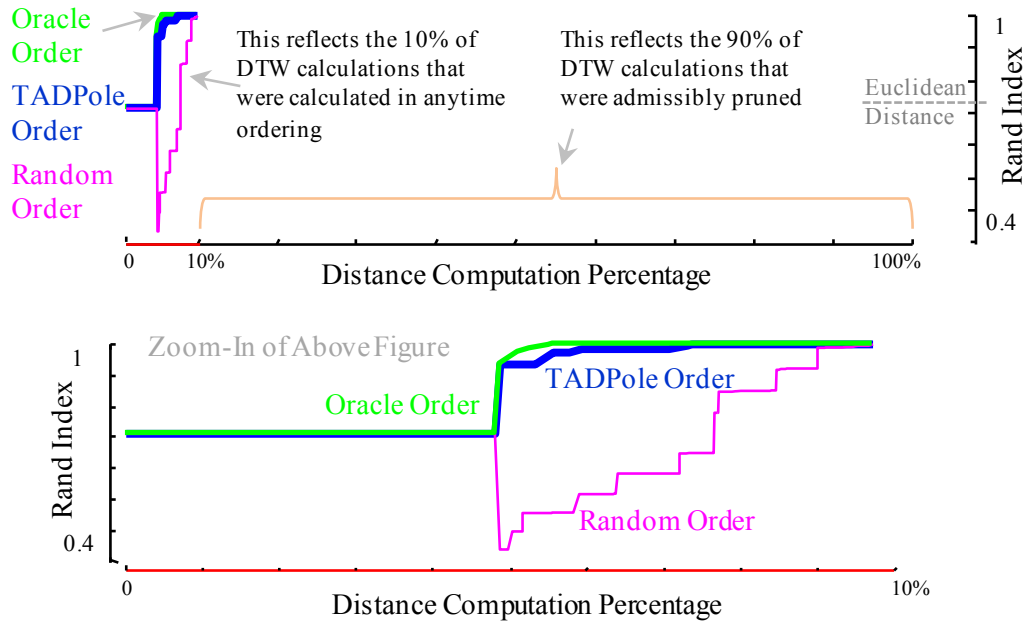
Fig. 30.: *top*) **A comparison of different distance computation-order heuristics on the Insect dataset [69]. An oracle ordering (green) converges stunningly quickly. The random ordering (pink) converges very slowly. Our proposed ordering (blue) is very close to the oracle.** *bottom*) **A zoomed-in view of the figure shown at the** *top*).

From Fig. 30.*top*) we see that *in conjunction* with all the pruning strategies

described in Sections 3.4.2.1 and 3.4.2.2, 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.

3.5 Experimental Evaluation

All experiments in this chapter (including the ones "inline" in the above text) are completely reproducible. We have archived all experimental code, parameter settings and data at [108]. 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 *few* parameter choices we have to make. In addition to this, we demonstrate the utility of our approach on multiple real-world case studies.

We conducted our experiments on a Windows 8 machine with 3.5 GHz AMD A8-6500 APU with Radeon(tm) HD Graphics processor and 16 GB RAM. All our implementations are single threaded and were written in Matlab 7.9.0.529 (R2009b).

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

The principle straw man we need to compare TADPole to is the brute force version of DP with DTW. This comparison is explicitly encoded in Fig. 30 and the similar figures below. In these experiments we also replace DTW with Euclidean distance to demonstrate that DTW is really necessary. In Table 13, 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 [69]. 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 [108]).

Table 13: Clustering Quality (in Terms of Rand Index) of TADPole vs. Some State-of-the-Art Clustering Algorithms

Dataset	TADPole$_{DTW}$ (*TADPole$_{ED}$*)	k-means[65] DTW$_{version}$	Hierarchical DTW$_{version}$	DBSCAN [56] DTW$_{version}$	Spectral [80] 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

In addition to the comparisons mentioned above, we compare against a recent partitional clustering algorithm called k-Shape [81]. The k-Shape algorithm considers the shapes of time series by using a normalized version of the cross-correlation measure. Depending on the properties of the distance measure, this algorithm computes the cluster centroids, which are used to update the assignment of objects to cluster centers in an iterative fashion. In order to demonstrate the robustness of k-Shape, the authors compare the clustering time and quality against a large number of state-of-the-art clustering algorithms. According to these experiments, k-Shape outperforms its rivals in terms of both clustering quality and time.

We compare the cluster quality and time of TADPole against k-Shape. We exhaustively run experiments on all the datasets chosen by the k-Shape authors. To be fair, we use the same data folds for both these algorithms. For TADPole, we use a fixed 5% warping window for consistency. We show the cluster quality result in Fig. 31.

As we can see from Fig. 31, for the majority of the datasets, TADPole does much better than k-Shape. It is only in the ECGFiveDays and SonyAIBORobotSurface datasets, that k-Shape performs significantly better than TADPole.

Since these two datasets are outliers, it is worth considering why they seem to favor k-Shape.

In the case of the SonyAIBORobotSurface dataset, the start and end time of the gaits are not aligned well in terms of time, which means that the first and last points of the two time series being compared may have very different values. In Fig. 32, we show two example time series from the two classes of this dataset.
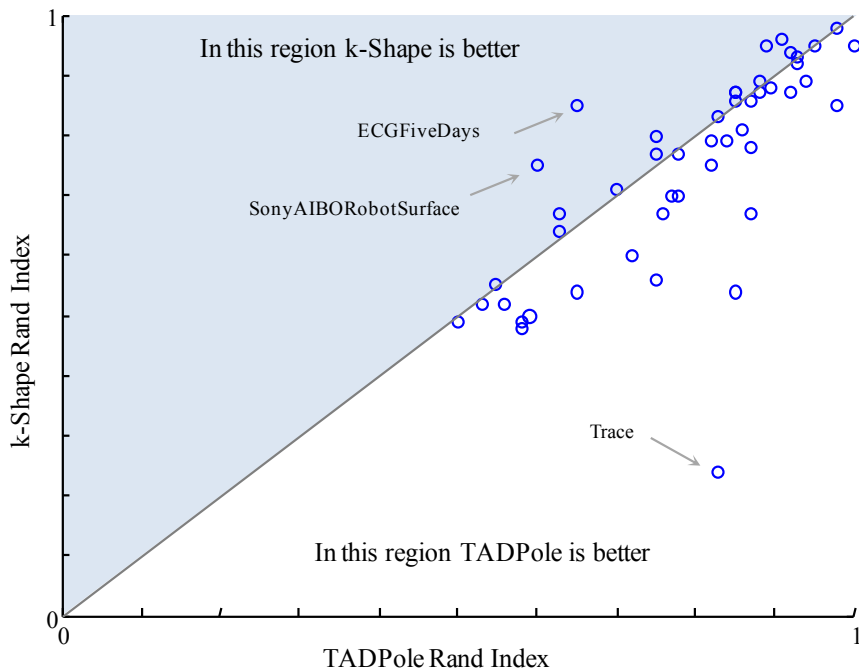
Fig. 31.: Cluster quality comparison experiment of TADPole against k-Shape in terms of Rand Index. For most of the datasets, TADPole gives significantly better clusters than k-Shape.

76

As we can see in Fig. 32, the start points of these two time series are not aligned at all, even though the end points almost agree. Because of DTW's endpoint constraint [68][88], these are forced to match, resulting in large, near random contribution to the overall DTW distance calculation. This issue (and several solutions) has been known for decades (see Section 3 of "Endpoint Variants of the DTW Algorithm" of [83]). Rather than using one of the fixes in [83], we simply smoothed the data, which resulted in a Rand Index of 0.91 by TADPole, moving it from the "*in this region k-Shape is better*" firmly into the "*in this region TADPole is better*" camp.
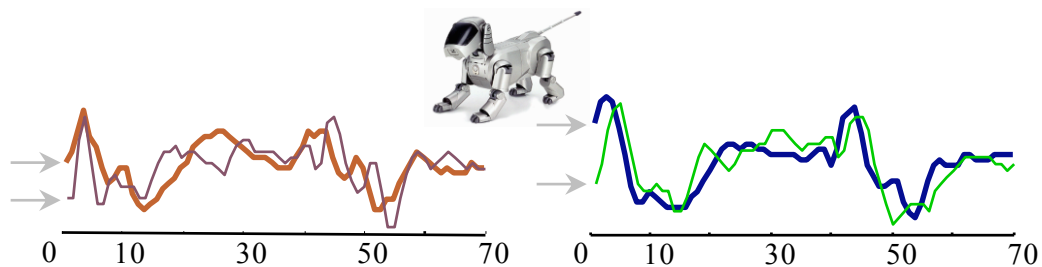
Fig. 32.: *left* **and** *right***) Two sample individual gaits of the two classes in the SonyAIBORobotSurface dataset. The intra-class variability is poorly captured by DTW because of the great difference of the first and last endpoints of two time series under question (marked with arrows).**

A similar observation explains results with the ECGFiveDays dataset, because the objects were extracted by an imperfect beat extractor, and therefore the data endpoints are not perfectly aligned (cf. Fig. 33.*left*). In addition to this, the disagreement of the start and last endpoints of the data is present (cf. Fig. 33.*right*) in this dataset. Once again, off-the-shelf smoothing (Matlab's smooth function with the default parameter) is enough to

mitigate most of the problem. We ran TADPole on the smoothed data and the improved

Rand Index for this dataset was 0.84, which is very close the k-Shape clustering (0.85).
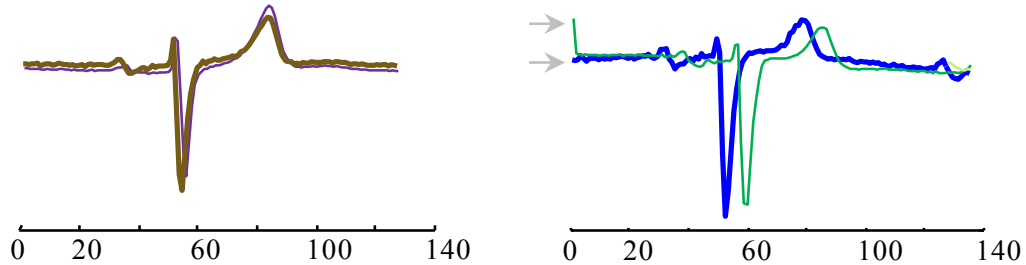
Fig. 33.: *left* and *right*) The individual heartbeats of the two classes in the ECGFiveDays dataset. *left*)The heartbeats are not perfectly aligned because of an imperfect beat extractor. *right*)Like the SonyAIBORobotSurface dataset, this data also has the disagreement of the first and last endpoints (shown with arrows).

Finally, another recently published time series clustering technique called YADING is shown to "*provide theoretical proof which... ...guarantees YADING's high performance*" [54]. However, these guarantees are *only* with respect to Euclidean distance. The only publicly available real dataset the authors of [54] test on is StarLightCurves, where they obtain a Normalized Mutual Information (NMI) score of 0.60. However, TADPole can achieve an NMI of 0.61, which is very slightly better. Likewise, in an expanded tech report that augments the paper [55], the method achieves an NMI of 0.61 on the CinC_ECG_torso dataset and 0.74 on MALLAT dataset, whereas TADPole achieves NMIs of 0.76 and 0.84, which are significantly better.

3.5.2   Parameter Sensitivity Experiments

To demonstrate that TADPole is not sensitive to parameter choices, we took the *Symbols* and *Insect* dataset [69] and performed TADPole on it with $k = 6$ and $k = 11$

respectively. We then varied the cutoff distance (d_c) parameter and measured the Rand Index obtained with the alternative settings. As we can see from Fig. 34. (a) and (b), there is a very wide range of choices for the values of d_c, which leads to high-quality clustering. We did the same experiment for several other datasets, (details available in [108]) and the results confirm our claim that TADPole is not sensitive to this cutoff distance parameter.
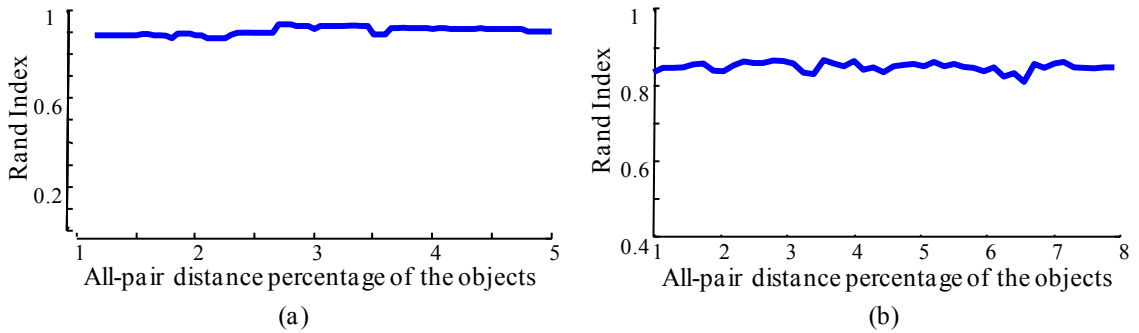
Fig. 34. (a) and (b): A parameter sensitivity test of TADPole shows stability of clustering over a very wide range of parameters.

In spite of this finding, it is clearly desirable to have some guidance in parameter setting. In the next section we introduce such a heuristic.

3.5.3 Heuristics for Setting Parameters

Up to this point we have glossed over the issue of setting the values of the threshold and the warping window width, we will now repair that omission.

First, we note that this problem is not unique to our setting; most unsupervised algorithms have this challenge. Our first proposed solution is therefore the default idea in the community. For any given problem, we can simply find the most similar dataset for

which we do have labels, and hope that the best settings there (discovered by cross validation) will generalize to the current setting.

However, beyond this we do have an idea which allows us to find good (not necessarily optimal) parameter settings in most cases. The idea is very simple, we use our unlabeled dataset at hand to build a new dataset for which we do have some labels, and use this labeled dataset to do cross validation, to set the parameters.

This idea leaves open the question of where we can find some class labeled data, our solution is to *make it*. Our basic idea is simple. Before we perform any clustering, we randomly sample objects from the dataset, which we call set R. For each object O in R, we create a copy of it that we denote as Ō. We add some warping to Ō, and place it into the dataset with the same pseudo-class label as O. The intuition is that because we know that object Ō is just a minor variant of O, we can safely assume that had Ō occurred naturally, it would have been in the same cluster as O, and our must-be-in-same cluster constraint was warranted. We denote the set containing all such warped version of objects in R, R̄.

At this point, for all object Ō in R̄, we know a pseudo label. Given this, in addition to the must-be-in-same-cluster constraint, we can say that if an object in R has a different label than a pseudo label of a new object Ō, then for this pair, the must-not-be-in-same-cluster constraint can be warranted.

This idea seems to have a tautological paradox to it. It seems that if we add *w* amount of warping to the dataset, we will discover *w* warping in that dataset. However,

this is not the case. A good setting for *w* depends not only on the intrinsic variability of the time axis and on the size of the dataset, but on the time series shapes themselves.

Having these two constraints in hand, we design a scoring function that measures the number of object pairs in $<R, \bar{R}>$ satisfying these constraints for different warping windows over all possible pairs. Therefore, our score function is:

$$score = \frac{number\ of\ pairs\ in <R,\bar{R}> satisfying\ link\ constraints\ for\ different\ warping\ window}{all\ possible\ pairs\ of <R,\bar{R}>} \quad (1)$$

We note the following:

- This idea, of generating new data, by randomly perturbing real data, is not novel [97] in general, but appears to be novel in the domain of time series.

- We are not claiming that this idea is the final or best solution to this issue. We merely introduce it as an existence proof that it is possible to set the parameters to reasonable settings, even in the absence of labeled data.

For some randomly chosen datasets, we compare the Rand Index obtained by TADPole for different warping window sizes against the score from equation 1. Our intuition is, if the shape of the score curve agrees closely with the Rand Index curve, then this score function has good correlation with the Rand Index. Having set a reasonable value of d_c, for warping window sizes where this score is relatively 'higher', we assume those warping window sizes as 'reasonable' parameter values. We illustrate our results in Fig. 35.

From Fig. 35 we can see that the score obtained corresponding to different warping window widths matches closely with the associated Rand Index for a fixed d_c. As long as the warping window width stays in relatively high scoring region, we can consider the parameter choice good.
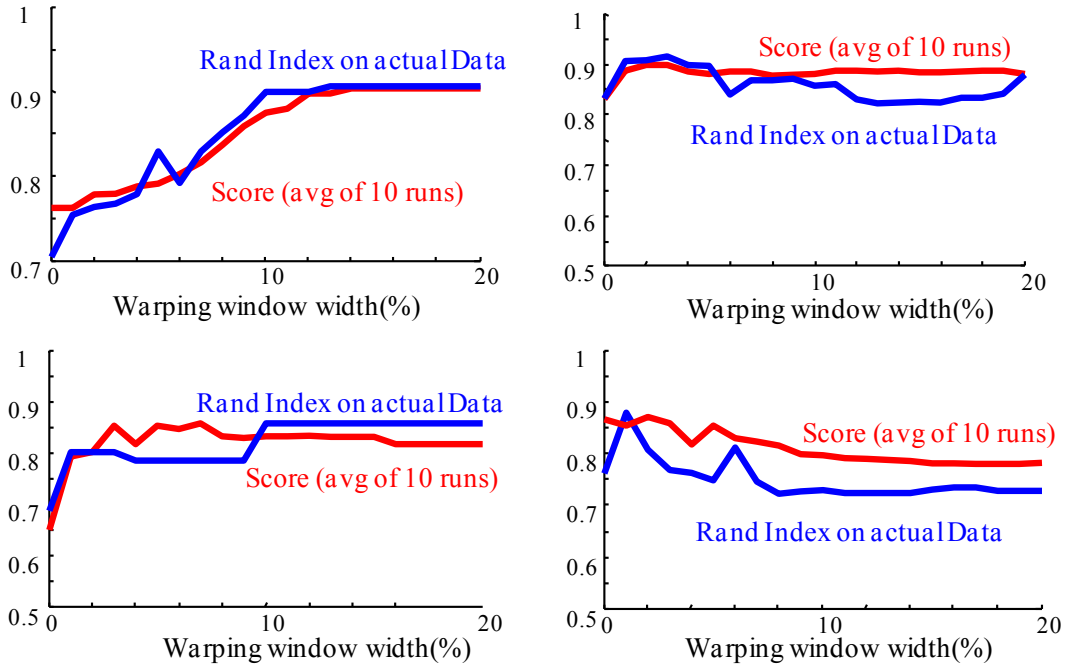
Fig. 35.: Warping window width vs. Rand Index (blue) and vs. score (red) plots for Trace (a), FaceAll (b), FaceFour (c) and SwedishLeaf (d) datasets [69]. The shapes of the red and blue curves agree closely, indicating good correlation. For a fixed d_c, the regions of the red curve with relatively high values are indicative of good parameter (here warping window width) values.

At this point we are now able to address the following question: *given a reasonably well set warping window width, how to estimate d_c?* We take the same approach to set d_c as we did for setting the warping window size. First we set a reasonably well value of the warping window size for a fixed d_c according to the scoring function described above. Now we fix the warping window width to any value in the 'good parameter zone' and change the value of d_c. To quantify how well our selection of d_c is, we consider the same

scoring function we used for the warping window. Just like what we did for setting the warping window, we vary d_c and record the score associated. The range of values resulting relatively high scores is the good parameter zone for d_c. For the four datasets shown in Fig. 35, by keeping the warping window size fixed, we find the best d_c values listed as in Table 14.

Table 14: Best Value of d_c for Reasonably Good Warping Window Width

Dataset	Warping Window Width	d_c
Trace	14%	1.7
FaceAll	2%	3.25
FaceFour	6%	4.8
SwedishLeaf	2%	0.65

3.6 Case Studies

3.6.1 Electromagnetic Articulograph Dataset

The Electromagnetic Articulograph (EMA) is a device that is increasingly used for mouth movement studies [99]. 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 [99]. The most common use of articulographs is in speech therapy for a plethora of speech

disorders. However, there is a significant obstacle to EMA use: 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 rather 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 [100]. The duration of utterance of each of the words is ~0.7 seconds. In Fig. 36.*left*), we show the data collection process for one of our subjects. In Fig. 36.*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.
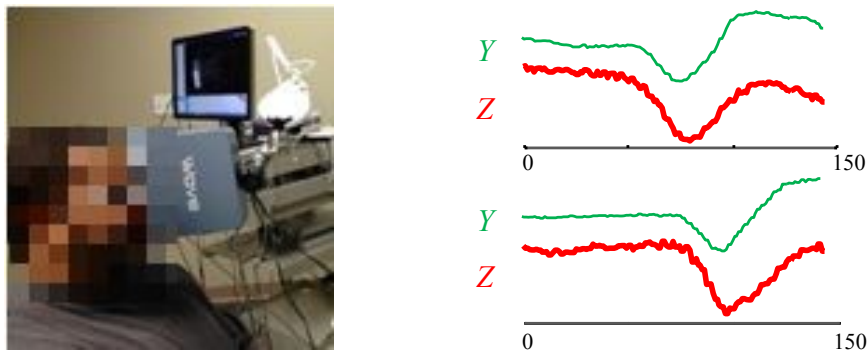
Fig. 36.: *left*) One of our volunteers wearing the articulograph apparatus. *right*) Two examples of the 3D time series produced by enunciating the word "*fate*" show inter-subject warping. The X axis is omitted here, as it only has useful information for patients with facial asymmetry.

We 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

[108]). We can now ask how effective our pruning strategy is when performing this clustering. We show the result in Fig. 37.
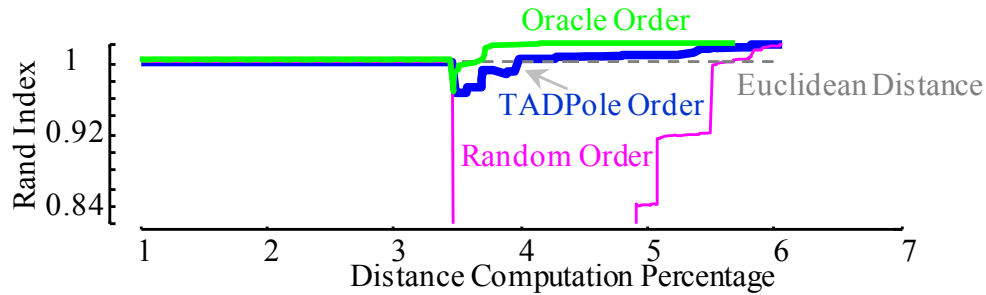
Fig. 37.: The amount of distance pruning achieved by TADPole is ~94% for the Articulographs [99]. Moreover, the ordering heuristic is almost as good as the oracle ordering.

As we can see from Fig. 37, TADPole achieves ~94% pruning, converging almost as quickly as the oracle algorithm. In wall clock terms, TADPole takes only 2.89 seconds, which is fast enough to provide interactive analysis and feedback to the patient.

3.6.2 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 Fig. 38.*top*).

For this case study we consider a dataset of 500 PPGs from two sources: the MIMIC II Waveform Database [60][90] 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 Fig. 38.*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* (Fig. 38.*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.
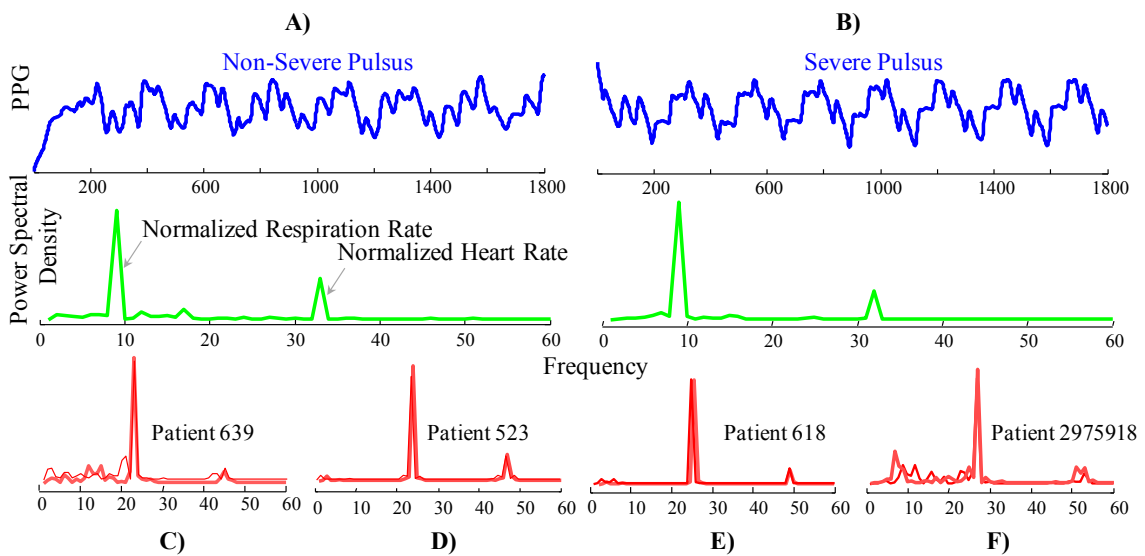
Fig. 38.: *top*) **PPG and Power Spectral Density (PSD) signal of a patient with non-severe Pulsus (*top.left*)) and severe Pulsus (*top.right*)).** *bottom*) **Four PSDs of four patients forming four different clusters within the non-Pulsus objects. From these four clusters, we can see the objects are clearly warped.**

For this dataset, TADPole produced a perfect clustering with a pruning rate of 88%, making it an order of magnitude faster than brute-force. In Fig. 39.*left*), we show the PPG measurement process. From Fig. 39.*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.
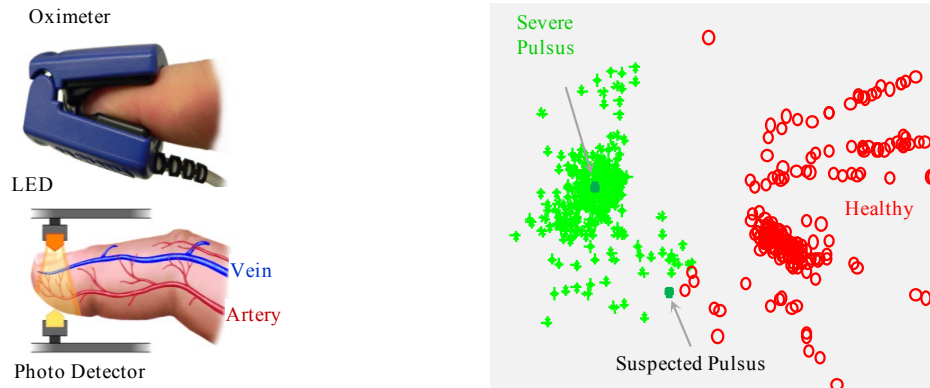
Fig. 39.: *left*) **A PPG apparatus.** *right*) **The Pulsus dataset projected into two dimensions using multidimensional scaling, and color coded by the output of TADPole.**

3.6.3 Person Re-Identification Dataset

Person re-identification is the task of recognizing individuals across spatially disjointed cameras [59], and is an important problem for understanding human behavior in areas covered by surveillance cameras. As shown in Fig. 40, we can extract color histograms from the video, thus treating the problem as a multidimensional time series problem. We considered the PRID dataset [64], 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 [108]). In contrast, Euclidean distance only achieves a Rand Index of 89%.
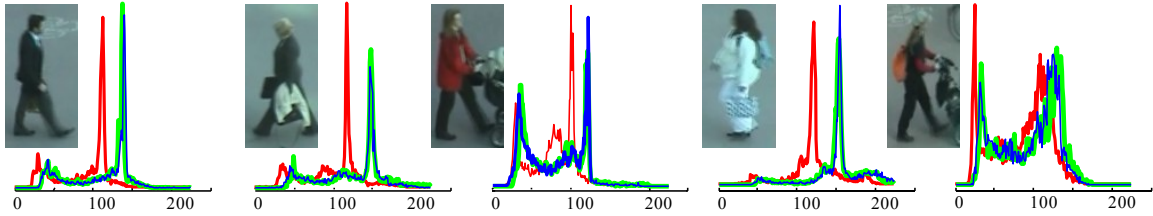
Fig. 40.: Representative images from the dataset [64] with their corresponding color histograms.

3.6.4 Clustering Multidimensional Data

In order to demonstrate TADPole's suitability for multidimensional data, we clustered two real-world datasets - Cricket and uWaveGesture.

3.6.4.1 Cricket Dataset

Cricket is a very popular game around the globe. As part of the refereeing, an umpire uses a fixed set of gestures using his hands and (sometimes) legs to communicate his decisions to a distant scorekeeper. The interested reader may find a list of complete signals in [76].

In this case study, we analyze the utility of TADPole for gesture recognition applications. For this, we use the dataset in [52], where 4 different umpires perform the various signals. On average, each signal is performed 16 times. Two accelerometers were attached to the wrists of the umpires and the data was sampled at 184 Hz. Considering both the left and right hands, this data has 6 dimensions (each accelerometer has X, Y and Z components).

Fig. 41. shows what each of the seven gestures *Last Hour*, *Leg Bye*, *No Ball*, *One Short*, *Out*, *Penalty Runs* and *Six* look like.
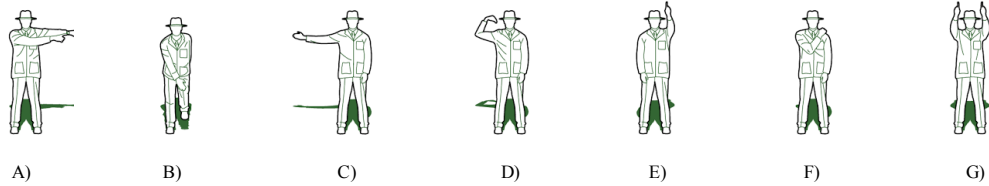
Fig. 41.: The seven cricket game gestures we used for our experiment. A) Last Hour, B) Leg Bye C) No Ball, D) One Short, E) Out, F) Penalty Runs and G) Six.

In this data setting, we picked the tri-axial right hand signals as the three dimensions we wish to cluster. The number of objects in this dataset is 258 with each of the time series being of length 1,155. We used warping window = 5%.

In this dataset, TADPole gives a Rand Index of 0.92 with ~82% pruning. In wall clock terms, TADPole takes only 52 seconds to cluster this 27 minutes long data. That is to say, we can cluster the data about 30 times faster than real time.

3.6.4.2 uWaveGestureLibrary Dataset

For this experiment we consider the three dimensional uWaveGestureLibrary dataset [69]. This data was collected to support efficient personalized gesture recognition on a wide range of electronic and mobile devices. This dataset contains eight simple gestures identified by a Nokia research study. The gestures were collected from eight participants with the Wii remote-based prototype. Fig. 42 shows these gestures as the paths of hand movement [75][82].

Fig. 42.: The gesture vocabulary adopted from [75][82]

In this experiment, as before, we are interested in the actionability of TADPole to cluster human gestures. We use all eight gestures as input to TADPole. The number of objects in this dataset is 896 with each of the time series being of length 315 and a sampling rate of 100 Hz. After we run TADPole on this data, we achieve a Rand Index of 0.93 with ~88% pruning. In wall clock terms, TADPole needs only 56 seconds to cluster this 47 minutes long data. As before, we can cluster the data much faster than real time.

3.6.5 Generalizing TADPole to Cluster Discrete Data

Until now, we have shown that our TADPole framework can efficiently perform exact clustering of *real-valued* time series data. The interested reader might wonder whether this framework is general enough to be extended for handling other types of data. In this section we present a discussion of the generalization of this framework to cluster very long *discrete* biological sequences. We note here that our experiments are not claiming a novel discovery of any biological significance. We are merely demonstrating a

"proof-of-concept," that the TADPole framework can be extended to perform fast clustering of long biological strings.

More generally, we believe that this example shows that the TADPole framework may be useful in scaling up clustering for *any* distance measure for which one can compute both upper and lower bounds. Beyond DTW, this would include most string similarity measures (as we show below), the Earth Movers Distance, Graph Edit Distance etc.

3.6.5.1 Clustering of Protein Sequences with Edit Distance

The identification of biological sequences with similar functionality requires similarity search in large databases, and is a well-known problem in the Bioinformatics community [58][63]. As an example, there exist diseases such as Maple Syrup Urine, Propionic Acidemia, Hurler Syndrome [78], etc. which are caused by the lack of sections of protein sequences in an organism, and can be life-threatening. In order to design drugs for such diseases, biologists produce sequences with the same functionality, and insert these sequences into the organisms to compensate the deficit.

Typically these protein strings are very long, therefore similarity search in such a large sequence database can be computationally challenging [47]. For example, large genomes are usually expressed using Expressed Sequence Tag (EST) databases. In such databases, gene portions are represented by mature mRNA, which typically are 500-800 nucleotides long. Therefore, similarity-based algorithms in such biological databases

require algorithms that can handle large strings efficiently. We see TADPole as a potential general framework for efficiently clustering such biological data.

In order to measure the similarity between two sequences of biological strings, researchers often use Edit Distance (EdD), or one of its many variants [73]. Given two strings A and B, the EdD is defined as the minimum number of deletion(s), insertion(s), and substitution(s) needed to transform A into B. We list the notation of EdD in Table 15:

<p style="text-align:center">Table 15: Symbol Table</p>

Symbol	Explanation
α	Finite alphabet
α^*	Set of all finite sequences of symbols from α
a,b	Variables presenting individual symbols
A,B	Variables denoting sequences from α^*
|A|	Length of sequence A
A(i,j)	Subsequence from the ith to the jth symbol in A
a[*i*]	i^{th} symbol in sequence A

Given the notations above, a substitution operation S(*a*[*i*],*b*[*j*]) is defined as replacing *a*[*i*] in A by *b*[*j*] in B. An insertion operation I(*i, b*[*j*]) is inserting *b*[*j*] in the ith location of A. Deleting the symbol *a*[*i*] in A is denoted by D(*a*[*i*],-). Each of these edit operations is assigned a weight which represent the difference between two symbols or the penalty for insertion/deletion operation. A transformation from A to B, T(A,B) is the set of edit operations applied to A iteratively which transform A to B. The weight of this transformation, w(T(A,B)) is defined as the sum of the weights of the operations in

T(A,B). Given these definitions, the edit distance between A and B, EdD(A,B) is defined as, EdD(A,B) = min(w(t)), $\forall\, t \in$ T(A,B).

As an example, for two strings A = 'INDUSTRY' and B = 'INTEREST', the optimal set of editing operations for T(A,B) are:

Starting string = INDUSTRY, Goal String = INTEREST

  S(A[3],B[3]) = INT(D)USTRY

  S(A[4],B[4]) = INTE(U)STRY

  I(5,B[5]) = INTERSTRY

  I(6,B[6]) = INTERESTRY

  D(A[9],-) = INTEREST-(R)Y

  D(A[9],-) = INTEREST-(Y)

Therefore EdD(A,B) = 6.

Using dynamic programming, EdD(A,B) can be solved in O(nm) time and O($n+m$) space, where n and m are the lengths of A and B respectively. For very large biological sequences, the computation of edit distance can therefore be very time consuming. We propose to exploit our TADPole framework to accelerate edit distance based clustering of long biological strings. Both edit distance and DTW are elastic distance measures [53] and for both of these measures, tight lower and upper bounds are already defined. Therefore, our TADPole framework is a suitable fit to perform fast and exact clustering of long biological strings. For clarity however, we confine our discussion only to the clustering of protein sequences.

We use Edit Distance (EdD) to perform the clustering of long protein sequences with TADPole framework. According to the framework structure, the inputs are the fully computed lower and upper bound matrices of the edit distance of all the objects of the dataset. As the lower bound of the edit distance, we use a notion of the distance that maps the strings of the database onto a multidimensional integer space using a wavelet based method [66]. We use the difference of the length of the strings in the dataset as the upper bound of the edit distance. Tighter bounds are known, but we are only interested in the general principle here.

In order to demonstrate the utility of TADPole in accelerating the clustering of large protein sequences, we use a random chunk of $Dataset_{801}^{1600}$ in [71] of the UniProt dataset [98]. It includes 178 strings defined over an alphabet of 21 letters. The protein strings have variable lengths ranging from 1052 to 1597. We cluster this dataset using TADPole and obtain distance pruning of 28%. Although the speedup we achieve does not look impressive at the first sight, we believe that with the application of tighter upper and lower bound of edit distance, TADPole would further improve.

Given the distance pruning we achieve, it is instructive to see the utility of the clusters returned by TADPole. Out of the eight clusters returned by TADPole, one represents the rpoB gene which encodes the beta subunit of bacterial RNA polymerase. According to the STRING database, the DNA-dependent RNA polymerase "*...catalyzes the transcription of DNA into RNA using the four ribonucleoside triphosphates as substrates.*" [95] Another cluster corresponded to cell division protein mukB, which

"...*plays a central role in chromosome condensation, segregation and cell cycle progression .*"[95]

Considering the fact that the clusters returned by TADPole have some level of biological significance, the readers might wonder how fast the algorithm converges to the ground truth. We show the result in Fig. 43.

From Fig. 43 we can see that even if the amount of pruning achieved by TADPole is not impressive at the very first sight, the quality of the clusters in the earlier stages of the algorithm is *very* impressive, and it is almost as good as the oracle ordering. Therefore, with very small amount of actual edit distance computation, TADPole can produce very high quality cluster results.
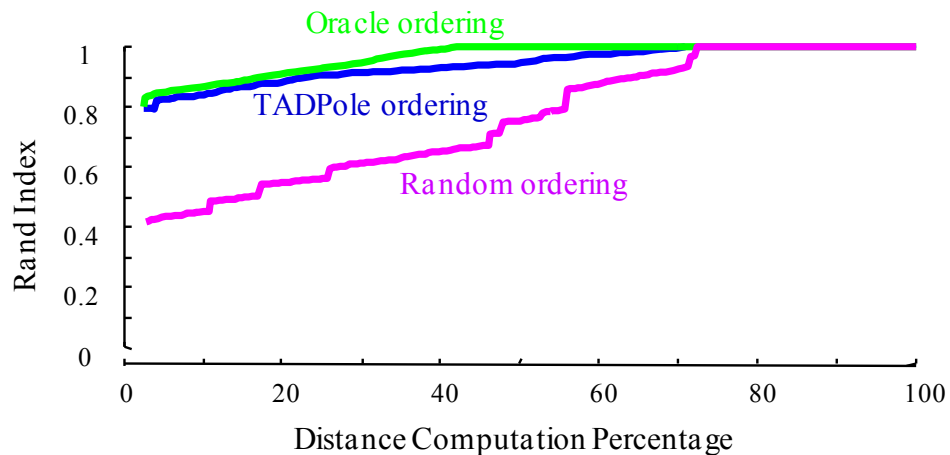
Fig. 43.: The fast convergence of TADPole ordering (blue) to the oracle ordering (green) shows that the quality of the clusters in the earlier iterations is quite good. The cluster quality given by the random ordering (pink), however, is not as good as TADPole.

3.7 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* much faster than the state-of-the-art algorithms, 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 and Pulsus). We have made all our code and data publicly available to allow the community to exploit and extend our ideas.

3.8 Acknowledgements

3.9 Appendix

Proof of Exactness of the TADPole Algorithm

Theorem:

The cluster result obtained by TADPole is exactly the same as the one obtained by DP_{DTW}.

96

Proof:

In order to prove this theorem, we must first consider the following two lemmas:

Lemma 1

The pruned distance computations by TADPole during the density calculation have no effect on the local densities of the objects in the dataset.

Proof (by contradiction):

We prove this lemma in the context of the three cases of distance computation pruning during the local density calculation.

Case A: *Objects i and j are identical*

When i and j are identical, then their actual distance D_{ij} is zero, which means they are definitely within d_c. Therefore, we can safely prune their distance computation. (Proved)

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

In this case our claim is, $D_{ij} \leq d_c$ and we can safely prune the distance computation. For the sake of the proof, let us assume that $D_{ij} > d_c$.

By definition, $LB_{Matrix}(i,j) \leq D_{ij} \leq UB_{Matrix}(i,j)$

Therefore, because $UB_{Matrix}(i,j) < d_c$, and $D_{ij} \leq UB_{Matrix}(i,j)$, $D_{ij} \not> d_c$. Therefore, our claim holds. (Proved)

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

In this case our claim is, $D_{ij} > d_c$ and we can safely prune the distance computation. For the sake of the proof, let us assume that $D_{ij} \leq d_c$.

By definition, $LB_{Matrix}(i,j) \leq D_{ij} \leq UB_{Matrix}(i,j)$

Therefore, because $LB_{Matrix}(i,j) > d_c$, and $D_{ij} >= LB_{Matrix}(i,j)$, $D_{ij} \nleq d_c$. Therefore, our claim holds. (Proved)

TADPole only calculate the distances for the case when $LB_{Matrix}(i,j) < d_c$ and $UB_{Matrix}(i,j) > d_c$. Therefore, lemma 1 holds. (Proved)

Lemma 2

The pruned distance computations by TADPole do not affect the NN distance of any object calculation from its higher density list.

Proof (by contradiction):

We prove this lemma in the context of the two phase pruning during NN distance calculation of an object from its higher density list.

Phase 1: Upper bound calculation

For this phase we need to prove that the NN distance from its higher density list of an object i, $\delta_i <= ub_i$, where ub_i is the upper bound of this NN distance we intend to find.

For the sake of the proof, let us assume $\delta_i > ub_i$.

Assume that the higher density list of i consists of $j_1..j_k$.

Assume that the NN of i from its higher density list is j_n, $1<= n <= k$. Therefore, $D_{ij_n} = \delta_i$

By definition, $ub_i = D_{ij_p}$ or , $ub_i = UB_{Matrix}(i,j_p)$,where $1 <= p <= k$

Therefore, $\delta_i = D_{ij_p}$, when $n = p$ or $\delta_i <= UB_{Matrix}(i,j_p)$, when $n \neq k$.

From this, we can say that $\delta_i \ngtr ub_i$. Therefore, our claim holds. (Proved)

Phase 2: Pruning

For this phase we need to prove that $\forall\ j \in \delta\_list_i(j))$, if $LB_{Matrix}(i, j) > ub_i$, then j is not the NN of i from its higher density list.

For the sake of the proof let us assume j is the NN of i from its higher density list.

From the proof in phase 1, $D_{ij} <= ub_i$. By definition, $LB_{Matrix}(i,j) <= D_{ij} <= UB_{Matrix}(i,j)$

But if $LB_{Matrix}(i, j) > ub_i$, then $D_{ij} > ub_i$. Therefore, $D_{ij} > ub_i$. It means that j cannot be the NN of i. (Proved)

TADPole only prunes the computation of all D_{ij} where j cannot be the NN of i from its higher density list. Therefore, lemma 2 holds. (Proved)

The proof of lemma 1 and 2 states the exactness of our pruning strategy. Therefore, theorem 1 holds. (Proved)

Chapter 4: Rare Time Series Motif Discovery from Unbounded Streams

The detection of *time series motifs*, which are approximately repeated subsequences in time series streams, has been shown to have great utility as a subroutine in many higher-level data mining algorithms. However, this detection becomes much harder in cases where the motifs of interest are vanishingly rare or when faced with a never-ending stream of data. In this work we investigate algorithms to find such rare motifs. We demonstrate that under reasonable assumptions we must abandon any hope of an exact solution to the motif problem as it is normally defined; however, we introduce algorithms that allow us to solve the underlying problem with high probability.

Keywords: Motif Discovery; Time Series; Streaming Data

4.1 Introduction

Time series motifs are approximately repeated patterns found within a longer time series. Since their definition a decade ago [126], dozens of researchers have used them to support higher-level data mining tasks such as clustering, classification, dictionary learning [113], visualization, rule discovery and anomaly detection [133]. While the original time series motif discovery algorithm was an approximate algorithm, the recently introduced MK algorithm provides a scalable exact solution [133]. For a user-defined length of interest, the MK algorithm finds the pair of non-overlapping subsequences that

have the minimal Euclidean distance to each other. The algorithm has been further

generalized to a limited streaming setting; it is now possible to find and maintain the

motif pair in a sliding window, say the last k minutes, of a continuous stream [132].

In spite of recent progress in motif discovery, there are two related situations for

which there are currently no tractable solutions. If the motifs are extremely rare, then it is

extremely unlikely that we will see two of them within k minutes of each other, as

assumed by [132]. Depending on data arrival rates and the hardware platform used, k

might be as large as 20 minutes. However, as shown in Fig. 44, we may be interested in

finding patterns that occur only once a month or less. If we ignore the streaming case and

assume that the m-datapoints are stored in memory, we are still limited by the scalability

of the current fastest known algorithm [131][133], which is O($m$log(m)) with high

constants. For some of the datasets we wish to consider (cf. Section 4.6.3), this would

require decades of CPU time.

In this work we introduce an efficient framework that allows us to solve such

problems with *very* high probability. Our key observation is based on an understanding of

how motif discovery is actually used in domains as diverse as electroencephalography

[133], entomology and nematology [113].
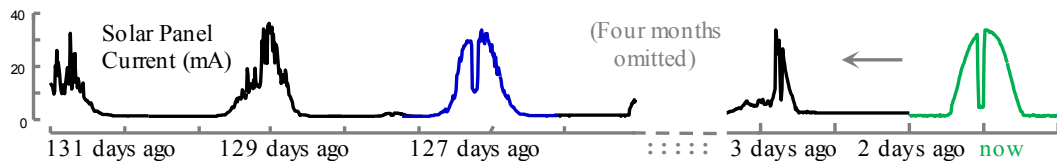
Fig. 44.: A never-ending time series stream from a weather station's solar panel [110], only a fraction of which we can buffer. A pattern we are observing now seems to have also occurred about four months ago.

101

It is important to recognize that the narrowly defined time series motif *pair* discovery problem as defined in [133] is really a proxy problem for the underlying task of finding more generally repeating patterns. For example, suppose that there are ten examples of an unknown repeated behavior in a dataset with ten million items, and we wish to discover them. We could employ the MK algorithm, which would find the pair from the ten patterns that are the *minimum* distance apart, and then use a quick linear scan to find the other eight patterns. Indeed, this is suggested in [133]. Note, however, that it would suffice for our purposes to find *any two* of the ten repeated patterns; we do not really need the smallest distance pair, or any specified pair. This is a simple but important observation, because it allows us to succeed if we find any one of 45 pairs, clearly a much easier task. As we shall show, this slightly relaxed assumption allows us to solve problems that are otherwise intractable, and to discover repeated patterns in unbounded streams. The patterns discovered by our algorithm can serve as an input to higher-order algorithms that do semi-supervised learning [119], or look for changes in the frequencies of the discovered patterns that may signal *anomalous* behavior [126] etc.

4.2 Definitions and Notation

In order to concretely state the problem at hand, we will define the key terms used. We begin with a definition of our data type of interest, *time series*:

Definition 11 *Time Series*: A time series $TS = ts_1, ts_2, ...,ts_m$ is an ordered set of m real-valued variables, where ts_m is the most recent value.

We are only interested in the local properties of a time series; thus, we confine our interest to subsections of the time series, which are called *subsequences*:

Definition 12 *Subsequences*: Given a time series *TS* of length *m*, a subsequence of *TS* is a sampling of length $l \leq m$ of contiguous positions from *TS* starting from position *i*. Formally, $TS_{i,l} = ts_i, ts_{i+1}...ts_{i+l-1}$ for $1 \leq i \leq m-l+1$.

The subsequences from a time series are extracted by the use of a *sliding window:*

Definition 13 *Sliding Window:* For a time series *TS* of length *m*, and a user-defined subsequence of length *n*, all possible subsequences of *TS* can be found by sliding a window of size *n* across *TS*.

It is well understood in the literature that (with very rare and well-defined exceptions) it is meaningless to compare time series unless they are z-normalized [123][126] .

Definition 14 *Z-normalized Subsequence:* If the values of a time series subsequence TS_P have an approximately zero mean with standard deviation (and variance) in a range close to one, then TS_P is called a z-normalized subsequence.

A common task associated with subsequences is to determine if a given subsequence is similar to other subsequences under some distance measure. This notion is formalized in the definition of a *match:*

Definition 15 *Match:* Given a positive real number *T* (called *threshold*) and a time series *TS* containing subsequences TS_P and TS_Q beginning at position *P* and *Q*, respectively, if $D(TS_P, TS_Q) \leq T$, then TS_Q is called a matching subsequence of TS_P.

These notations are summarized in Fig. 45. The obvious definition of a match is necessary to formally define a *trivial match*. Clearly, the best matches to a subsequence tend to be located one or two points to the left or the right of the subsequence in question. Almost all algorithms need to exclude such trivial solutions. The concrete definition is given below:

Definition 16 *Trivial Match:* Given a time series *TS* containing subsequences TS_P and TS_Q beginning at position *P* and *Q*, respectively, TS_Q is a trivial match to TS_P if either *P* = *Q* or a subsequence TS_Q' beginning at *Q'* such that $D(TS_P, TS_Q') > R$ and either *Q* < *Q'* < *P* or *P* < *Q'* < *Q* does not exist.

We are now in a position to define *objects*, which are non-overlapping subsequences from a time series stream:

Definition 17 *Object:* Given a time series $TS = (ts_1, ts_2, ...,ts_m)$ two objects of length *l* are subsequences $(TS_{i,l}, TS_{j,l})$ of *TS* such that $1 \leq i \leq i + l\text{-}1 < j \leq m\text{-}l$.

The reason we consider only non-overlapping subsequences is to avoid trivial matches, which must be excluded to define the success condition of our problem.

To measure the distance between objects, we use the ubiquitous Euclidean distance measure [109][116]:

Definition 18 *Euclidean Distance:* Given two time series (or time series subsequences) TS_P and TS_Q, both of length *m*, the Euclidean distance between them is the square root of the sum of the squared differences between each pair of the corresponding data points: $D(TS_P, TS_Q) \equiv \sqrt{\sum_{i=1}^{m}(TS_{P_i} - TS_{Q_i})^2}$

The reader may imagine that using the Dynamic Time Warping (DTW) distance measure could produce better results for the task at hand. However, this is *not* the case. As shown in [133], for time series objects which are *very* similar (suggestive of these being motifs), the values of Euclidean distance and DTW must be *very* tightly related. Given this, we use the ubiquitous Euclidean distance measure for calculating the similarity between the time series patterns [117][122][130][131]. We are now in a position to give a concrete problem statement.

4.3 Problem Definition

Assume we are given a never-ending time series stream **S** that mostly produces instances of *patternless* data in **R**, and with some low probability p, instances of an *unknown* pattern in **G** (we will define *patternless* later). As shown in our running example in Fig. 45, items in **G** appear visually similar to each other, but are *not* perfectly similar. Our goal is simply to detect an instance in **G** as early as possible.
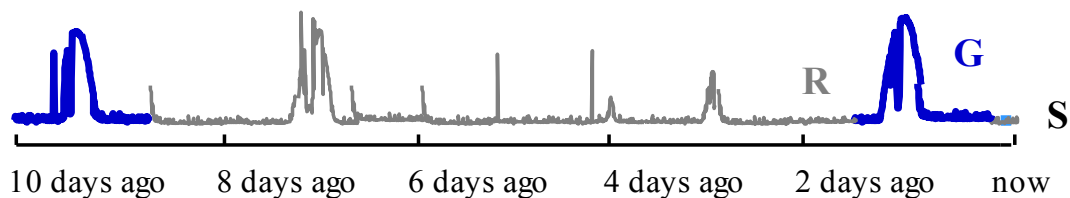
Fig. 45.: A never-ending time series stream S produces mostly patternless data in R, and with *very* low probability, instances in G.

Note that our problem is to *detect* instances in **G** as early as possible; but we are not interested in *testing the significance* of these instances. Our algorithm is designed to be a

subroutine for higher level algorithms, and the ranking of the patterns in **G** in terms of significance can be done by these algorithms [114][115][121]. Further note that we are assuming that we are given the lengths of the patterns-of-interest by domain experts, an assumption which is typical in the data mining community [113][121][130][138]. However, recent work by [131] shows that this assumption can be relaxed with little overhead.

Note that we assume there is no explicit test to check if a pattern is in **G**. Concretely, patterns in **G** have statistical properties that are also typical of data in **R**. Thus, the defining assumption of this work is that the only way we can tell if an item is in **G** is to note that it is sufficiently similar to another item also suspected to be from **G**, which we call a match (cf. Definition 15).

More formally, we assume a distance threshold *T* such that:

- Any two objects, where at least one is *not* in **G**, are unlikely to have a distance < *T*.

- Any two instances in **G** are likely to have a distance < *T*.

The former point is essentially an informal definition of *patternless*, but must come with a caveat. As **R** approaches infinity (recall we have an unbounded stream) and given that we are dealing with z-normalized objects, the space of possible shapes of a time series will eventually be exhausted and **R** must eventually produce two instances that are less than the threshold apart. Thus, the assumption of the existence of a distance threshold *T* discussed above is relative to a sample size of **S** that is approximately *w*, the amount of memory we can devote to this task (measured in the number of instances that can be

106

stored). The threshold T may be given to us in the form of domain knowledge, or we may have to learn it from the data (cf. [141]). This idea is illustrated in Fig. 46.
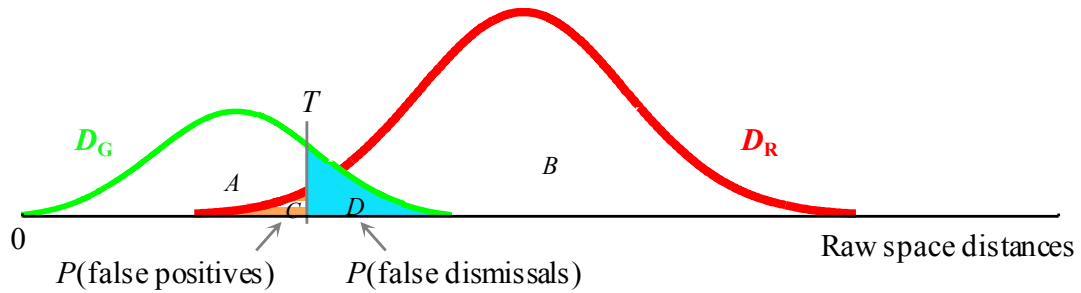
Fig. 46.: D_G and D_R represent the all-pair distance distributions of the patterns in G and R, respectively. The distance threshold T represents the boundary that separates the decision whether the two patterns in question belong to G or not.

Note that the two distributions here are Gaussian. This is *often* the case for real data (cf. Fig. 53 and Fig. 55), and we adopt this assumption for ease of exposition. We note, however, that generalizations to other distributions are trivial. We further note that for illustration purposes, the D_R distribution is only slightly larger than the distribution D_G; however, in the problems we wish to deal with, we expect the prior probability of the patterns in **R** (i.e., 1-p) to be many orders of magnitude larger.

Finally, a critical observation from Fig. 46 is that the two distributions overlap. Thus, no matter what value of T we have, we must deal with some probability of error. In particular, two types of error can occur:

• The region marked C is proportional to the probability that we will falsely believe that two patterns from **R** that happen to be similar are exemplars from **G**.

• The region denoted D is proportional to the probability that two exemplars that are from **G** will not be identified as such because they happen to be farther apart than average.

For real-world problems we expect the area of $C + D$ to be much smaller than that illustrated in Fig. 46. Obviously, we can adjust the threshold based on our relative tolerance for each type of error. This relative tolerance itself may depend on the application [121].

We are finally in a position to formally define the task at hand:

Problem Statement: Given an unbounded time series stream **S** that mostly produces instances in **R**, but with some *very low* probability p, produces instances of an *unknown* pattern in **G**, and a user-defined distance threshold T, detect and return an instance in **G** such that the expected number of instances in **S** seen is minimized.

For example, recall our running example shown in Fig. 44 and Fig. 45. The distinctive pattern (shown *colored/***bold***) consisting of a smooth "dump" with a "dropout" near the center, appears to have been caused by the shadow of a pole falling on the solar panel during a rare cloudless day. Such patterns occur five or six times a year (at that location), so if we wait *long enough* we will probably see two close together, perhaps even on consecutive days. However our task is to discover such patterns as soon as possible, independent of when/how often, they occur.

4.3.1 A Brute Force Algorithm

Given the assumptions above, a trivial algorithm suggests itself. We can simply keep the first k items of **S** in memory, and then, when the $k^{th} + 1$ item arrives, we compare it to all k items currently stored and report "`current item is a member of` **G**" if we find that $D(k + 1, j) < T$ and $j < k + 1$.

Note that this algorithm has some probability of making a type I or type II error, but this is intrinsic to our assumptions, and no algorithm can do better.

This brute force algorithm is clearly optimal, but also clearly untenable. We are assuming that **S** is an infinite stream, and p is a very small number. Thus, we will eventually run out of space to store items, or the time needed to make all the comparisons.

We can at least *mitigate* the time complexity of the naïve algorithm using off-the-shelf dynamic indexing techniques (recall that because we are dealing with *real-valued* high dimensional objects, we cannot avail of the $O(1)$ equality tests available to the *discrete* analogue problems) [118]. Therefore, we concern ourselves here with the more difficult resource limitation: space constraints.

4.3.2 Brute Force with Limited Memory

In our problem setting, we assume that we must work with C, a cache of a fixed size w. Here, w is the number of instances that can be stored in main memory. Note that while taking the distances of the cached patterns, we only consider patterns which are *not* trivial matches (cf. Definition 16) to each other (cf. Section 4.2).

It is clear that the performance of any cache-based algorithm depends critically on the size of the cache. Consider the two following special cases. If $w = \infty$, then the cache-based algorithm is as good as the trivial algorithm discussed above. If $w = 2$, then the probability of an instance in **G** being in the cache as another instance in **G** arrives is just p, and we will typically have to wait a *very* long time before reporting success.

Given this observation, our metric of success can be seen as the expected number of objects seen from **S** before detecting an object in **G**. We are now in a position to give a derivation of this metric, which we discuss in the next section.

4.3.2.1 Derivation of the Success Metric

In order to derive the theoretical model for the success metric, we denote the number of objects that must be seen from **S** before reporting success, Ψ.

By definition, at each time step, **S** produces an instance in **G** with probability p. By the time we detect an instance in **G**, the elements in C may or may not experience cache replacement(s).

Case 1: No cache replacement

Consider the case when there has *not* been any cache replacement by the time we report success. In order to find the probability of success, we denote the number of objects observed from **S** until the *first* and *second* instances in **G** are in C by Ψ_1 and Ψ_2, respectively.

Each Ψ_i is a geometric random variable with success probability p. Because there has not been any cache replacement when we report success, the probability of having the first two instances in **G** be in C is: $p_{\mathbf{G}_C} = p + p = 2p$.

Case 2: Cache replacement

Now consider the case when there has been cache replacement at least once. After discarding an element from the cache, the remaining w-1 elements in the cache can be in either of the following two categories:

Category 1: All of the w-1 patterns in C are instances in R. The probability of this event is: $P(\mathbf{R}_{w-1}) = (1 - p)^{w-1}$.

Category 2: All but one of the w-1 patterns in C are instances in R. The probability of this event is: $P(\mathbf{R}_{w-2}\mathbf{G}) = (1 - p)^{w-2} * p * (w - 1)$.

We can report success if and only if the wth pattern to be inserted into C is in **G**, and the previous w-1 patterns in C belong to Category 2. Therefore, the probability of the success event is: $P(SuccessEvent) = p * P(\mathbf{R}_{w-2}\mathbf{G})$.

If the patterns in C are such that no two of them are instances in **G**, then we call it a *failure event*. We describe the probabilities of this event below:

If the wth pattern to be inserted into C is in **R**, and the previous w-1 patterns in C belong to Category 2, then the probability of the failure event is: $P(FailureEvent_1) = (1 - p) * P(\mathbf{R}_{w-2}\mathbf{G})$.

If the wth pattern to be inserted into C is in **R**, and the previous w-1 patterns in C belong to Category 1, then the probability of the failure event is: $P(FailureEvent_2) = (1 - p) * P(\mathbf{R}_{w-1})$.

If the wth pattern to be inserted into C is in $\mathbf{G}$, and the previous w-1 patterns in C belong to Category 1, then the probability of the failure event is: $P(FailureEvent_3) = p * P(\mathbf{R}_{w-1})$.

Given this analysis, the probability of having the first two instances in $\mathbf{G}$ be in C is:

$$p_{\mathbf{G}_C} = \frac{P(SuccessEvent)}{P(SuccessEvent) + \sum_{i=1}^{3} P(FailureEvent_i)}$$

Based on the analyses above, $p_{\mathbf{G}_C}$ becomes:

$$p_{\mathbf{G}_C} = \begin{cases} 2p & (Case\ 1) \\ \frac{P(SuccessEvent)}{P(SuccessEvent) + \sum_{i=1}^{3} P(FailureEvent_i)} & (Case\ 2) \end{cases} \quad (1)$$

Therefore, after seeing n objects in $\mathbf{S}$, the probability of failing to detect an instance in $\mathbf{G}$ is: $p_{failure} = (1 - p_{\mathbf{G}_C})^n$. Given this, the probability of success after observing n instances in $\mathbf{S}$ is: $p_{success} = 1 - p_{failure}$.

More generally, Fig. 47 shows the relationship between cache size and the number of items seen before detecting an instance in $\mathbf{G}$, under the following concrete assumptions. One in a hundred objects in $\mathbf{S}$ belongs to $\mathbf{G}$; everything else belongs to $\mathbf{R}$. We further assume (just for this toy example) that the moment we have two instances in $\mathbf{G}$ in the cache, we can unambiguously detect that fact.

From Fig. 47 we observe that if $w = 20$, then we have to see about 2,857 objects to have a 0.99 probability of correctly identifying an instance in $\mathbf{G}$. As w gets larger, the number of objects needed to reach this 0.99 probability threshold decreases, but we can clearly see diminishing returns. Using a cache size five times larger only reduces the number of objects we need to see to about 918 objects, and moving to an arbitrarily large

112

cache size ($w = \infty$) further improves this down to about 227. For comparison, with $w = 2$, the 0.99 probability is not reached until we have seen 46,049 objects (this value is truncated from Fig. 47 for clarity).

Note that our toy example considers the case when $p = 0.01$; however, we expect to deal with real-world problems in which p may be several orders of magnitude smaller. Such values of p will "stretch" the x-axis, but our core observations about the diminishing returns properties remain.
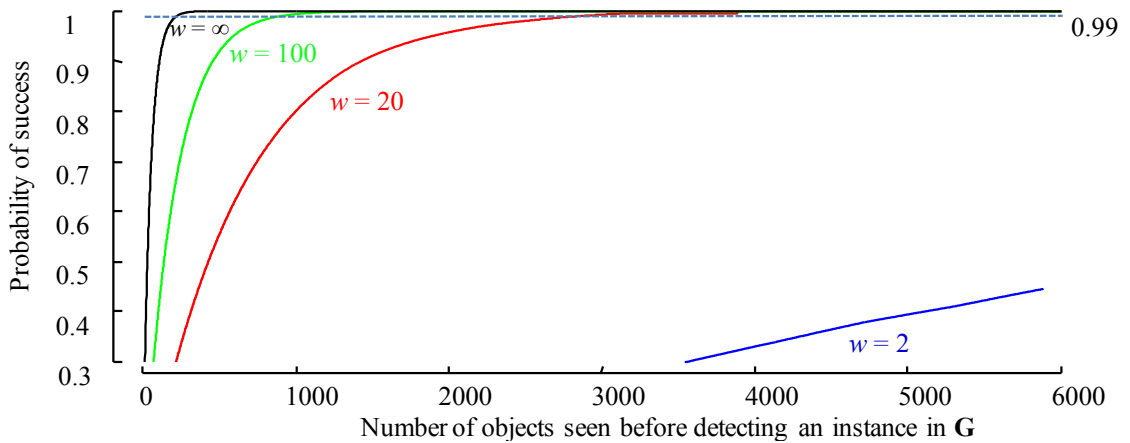
Fig. 47.: The number of objects that must be seen to find a pattern in G, for different cache sizes *w*, with a desired probability of success. The 0.99 probability is highlighted with a horizontal dashed line.

Also note that we have not stated which cache replacement policy we used in Fig. 47. As we shall show below, the two most obvious candidates, First-In-First-Out (FIFO) and Random Replacement (RR), *both* correspond to this analysis.

4.3.3 Analysis of Cache Replacement Policies

Two obvious cache replacement policies are First-In-First-Out (FIFO) and Random Replacement (RR) [136]. As the names suggest, in FIFO, the oldest object in the cache is discarded at each time step, whereas in RR, the object to be discarded is selected randomly. Note that the FIFO replacement policy is vulnerable to an adversarial case [111]. Imagine a version of our problem in which **S** produces instances in **G** at uniform time instances. If the number of objects produced by **S** after an instance in **G** is greater than or equal to *w*-1, then we can *never* detect an instance in **G** using the FIFO cache replacement policy. The obvious solution to mitigate such a problem is to use *randomization*, which is often used in algorithms to avoid such pathological cases [129]. It is important to note that this adversarial worst case is *not* pathologically unlikely. For example, some manufacturing machines may produce a special pattern as the machine is recalibrated during a shift change, every eight hours. A FIFO policy with a cache size of seven hours would never discover this pattern.

Using the analysis in Section 4.3.2, we can more formally define the metric of success for our problem. For the case without any cache replacement, the number of objects observed from **S** until two instances in **G** are in *C*, i.e., Ψ, is a negative binomial random variable. We define Ψ as $\psi = \sum_{i=1}^{2} \psi_i$.

Because the expectation of a sum of random variables is the sum of their expectations, we have: $E(\psi) = E\left(\sum_{i=1}^{2} \psi_i\right) = \sum_{i=1}^{2} E(\psi_i) = \frac{1}{p} + \frac{1}{p} = \frac{2}{p}$. (Recall from

Section 4.3.2, each Ψ_i is a geometric random variable with success probability p).

Therefore, if $w \geq 2/p$, we expect no cache replacement.

For the case with at least one cache replacement, $E(\Psi)$ depends on the number of objects seen from **S** *immediately before* the cache replacement, $\Psi_{replacement'}$, and *after* the cache replacement(s), $\Psi_{replacement}$, until we report finding a pattern in **G**. $\Psi_{replacement'}$ is simply w, and $\Psi_{replacement}$ is a geometric random variable with success probability p_{G_C} as in Equation 1. Therefore, for this case, $E(\psi) = \frac{1}{p_{G_C}} + w$. Similar to our observation about the case without cache replacement, if $w < 2/p$, we expect cache replacement(s) to occur.

In Fig. 48 we show that the expected number of elements seen from **S** before detecting an instance in **G** (our success metric stated in Section 4.3.2) decreases with increasing cache size. In addition to this, Fig. 48 shows how closely the success metric agrees with the empirical results for both of our candidate cache replacement policies.

As Fig. 48 suggests, the larger the cache is, the fewer objects in **S** we can expect to see before detecting a pattern in **G**. This observation does not seem directly exploitable, as the cache size is a domain constraint. However, as we will show in Section 4.4, we can "virtually" increase the cache size by changing the representation of the data.
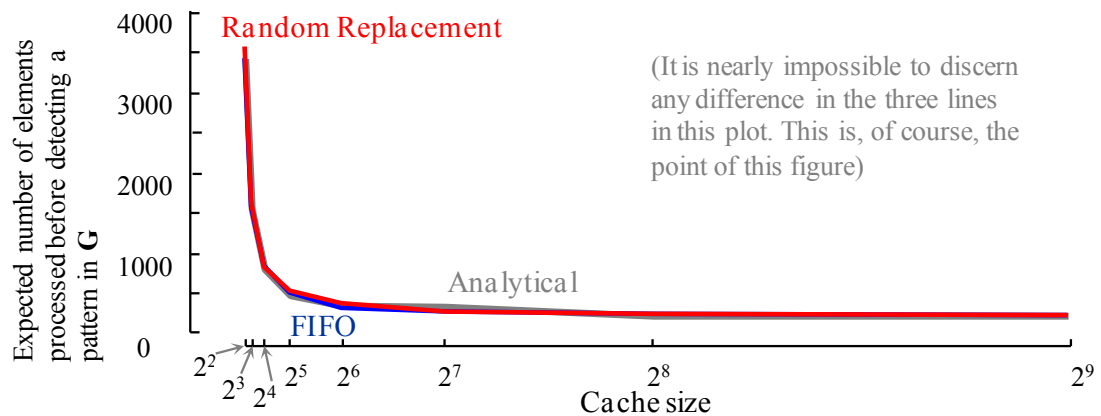
Fig. 48.: The experimentally-determined success metric for FIFO and RR policies closely agrees with the theoretically-derived metric for a wide range of values of *w*.

4.3.4 A Magic Sticky Cache

The analysis in Section 4.3.3 shows that for a fixed *w*, the more objects we see, the higher the probability is of detecting an instance in **G**. Given *w* and *p,* we can predict the performance using the analysis in Section 4.3.3. The results show that a larger cache size helps improve the performance. Beyond making the cache size larger, is there any other way we could improve the performance?

To answer this question, we can perform a *gedankenexperiment*. Imagine for a moment that we could "magically" control the discard probabilities of the patterns in **G** and **R** from the cache, such that patterns from **G** tend to "stick" in the cache for longer.

In particular, imagine we somehow can discard items in **R** with, say, a 50 or 100 times greater probability than items in **G**. This would improve the chances of an item from **G** *remaining* in the cache as a new **G** exemplar arrives, and thus improve Ψ. This intuition is illustrated with an experiment shown in Fig. 49.

116
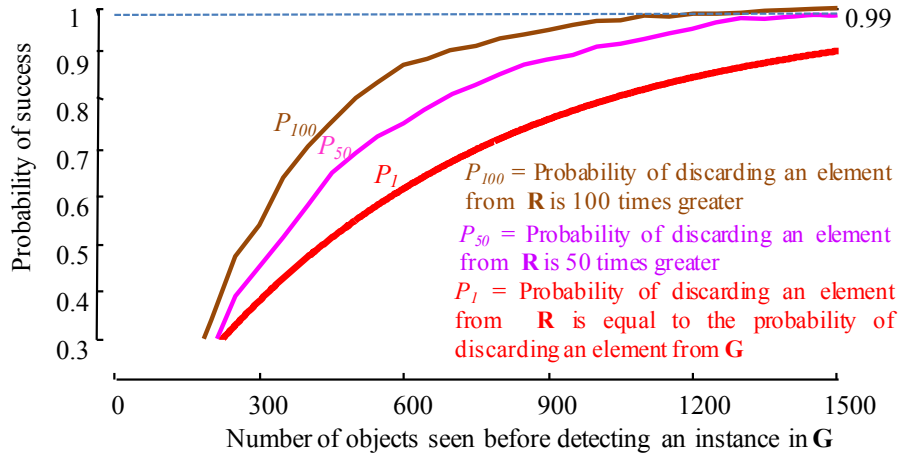
Fig. 49.: The red/bold line here is identical to the $w = 20$ line in Fig. 47. If we could somehow increase the probability of discarding an element from R we would obtain a significant improvement

As we can see, making the cache "stickier" for items that *might* be in **G** significantly improves our chance of detecting an instance in **G**. However, it is not clear yet how we could imbue the cache with this ability.

To summarize: The ideas in this section suggest two possible approaches (which are not necessarily exhaustive or mutually exclusive) that we can consider for improving the performance in our problem:

• Find a cache replacement policy that *minimizes* the probability of discarding instances in **G** as opposed to instances in **R** = **S - G** (cf. Fig. 49);

• Change the representation of the data, such that we can fit more objects into *C*. This implicitly improves the probability of keeping an instance of G in the cache (cf. Fig. 47 and Fig. 48).

In the next section, we explore the latter idea, and in Section 4.5 we discuss the former idea.

4.4 Data Representation Policy

4.4.1   Initial Observation

Recall from Fig. 48 that the larger the cache is, the higher the probability is of detecting an instance in **G**. We can emulate the effect of a larger cache by changing the representation of the data. Compressing or downsampling the data allows more objects to fit in the cache; we can thus expect to detect an instance in **G** *sooner* than in the raw space.

This idea requires some careful consideration. While time series data is typically amiable to lossless compression techniques such as delta encoding or Lempel Ziv, such methods require decompression before the Euclidean distance calculations, and are thus more suited to archiving data. They would clearly introduce an intractable overhead for a fast-moving stream, where we would have to (re)decompress each instance at every cache insertion.

Downsampling (or equivalently, *lossy compression*) avoids this problem, but introduces a new issue we must deal with. From Fig. 50, we can see that changing the representation of the data inevitably changes the distances between objects.
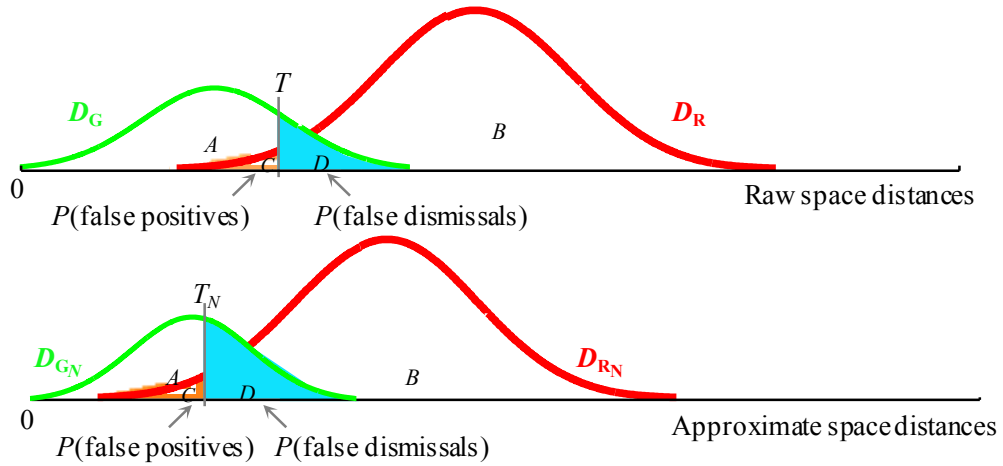
Fig. 50.: *top*) D_G **and** D_R **are identical to the all-pair distance distributions of the patterns in G and R, respectively, in Fig. 46.** *bottom*) **After downsampling the data, the new distance distributions (**D_{GN} **and** D_{RN}**) and the new threshold** T_N **shift left.**

If we consider only orthonormal transformations (DFT, DWT, SVD, etc.), the distances between any pairs of objects can *only* be reduced [109]. Because of this *lower-bounding* property, the distributions in Fig. 50.*top*) can only shift to the left (Fig. 50.*bottom*). As a consequence, the distance threshold T can also only shift left in the approximate space (T_N).

Note, however, that we cannot say how the overlapping area of the distance distributions will change – because *either* of the distribution's standard deviations could increase, decrease, or remain the same (Fig. 50.*bottom*), therefore, the area $C + D$ could also change arbitrarily. In practice, however, this area always increases, and in the next section we show how to incorporate this into our cost model.

4.4.2   Cost Model

It is important to note that in our problem definition, of the two errors we can make while detecting an instance in **G**, one "hurts" more than the other. This is because if we falsely miss two exemplars in **G**, then we still have the *hope* of detecting an instance in **G** in the future. However, if we mistakenly say two exemplars are in **G**, when in fact at least one is in **R**, then we can never detect two instances in **G**, as we immediately stop our search. Of course, we could adapt our definition such that we do not actually stop, perhaps buffering the tentative motif and continuing the search. However, the tentative motif must be examined by a human or algorithm at some point [121]. Thus, these false positives do have an inescapable cost [121].

Given this observation, we can design a cost model in which we fix the probability that we mistakenly claim two patterns to be in **G** (area *C* in Fig. 50). This fixing results in an *increasing tendency* of the probability of falsely dismissing two true patterns in **G** (area *D* in Fig. 50) as we downsample the data. As a consequence, the probability that two patterns in the cache are *believed* to be from **G** *tends to* decrease. We defer the calculations of these probabilities until Section 4.4.2.1. However, Fig. 51 shows an empirical illustration of this observation.

From Fig. 51.*left*) we can see that if we fix the probability that two patterns in *C* are falsely identified to be in **G**, then as we downsample more, the probability of the false dismissals of two cached patterns in **G** tends to increase. As a consequence, the probability that two cached patterns are potential candidates in **G** tends to decrease (Fig. 51.*right*).
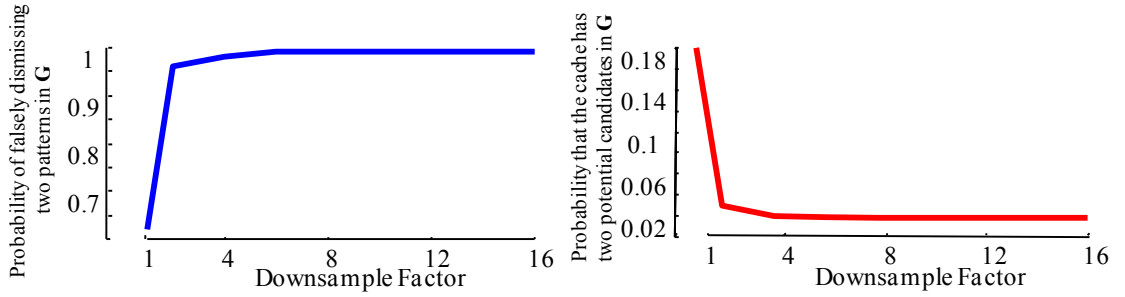
Fig. 51.: *left*) **When downsampling, the probability of falsely dismissing two instances in G tends to increase if we fix the probability that two cache patterns are claimed to be in G by mistake.** *right*) **Consequently, the probability that two patterns in the cache are believed to be from G tends to decrease.**

We are now in a position to describe the derivation of these probabilities and to formally define the 'costs' involved.

4.4.2.1 Cost Calculation

Assume that we know the distance threshold *T* in the raw space, which can come either from user input or from a threshold learning model (cf. [141]). Further assume that we know the means (μ_{D_G}, μ_{D_R}) and standard deviations ($\sigma_{D_G}, \sigma_{D_R}$) of both of the distance distributions in Fig. 50.*top*. Given these, we can calculate the standard score of *T* corresponding to each of these distributions as below:

$$z_{False\ Positive} = (T - \mu_{D_R})/\sigma_{D_R}$$

$$z_{False\ Dismissal} = (T - \mu_{D_G})/\sigma_{D_G}$$

Using the standard normal probability table, we can calculate the area to the left of $z_{False\ Positive}$, which is the probability of mistakenly claiming two patterns to be in **G**. Similarly, we can calculate the area to the right of $z_{False\ Dismissal}$, which is the probability of

falsely dismissing two true patterns in **G**. In Fig. 50.*top*, these errors are represented as shaded areas labeled C and D, respectively. From now on, we will refer to these errors as α and β, respectively.

Given these two error probabilities, Fig. 46 illustrates that the probability that two cached patterns are believed to be exemplars in **G** is:

$$p_G = (A + C)/(A + B + C + D) \qquad (2)$$

As noted above, we fix α to be a constant independent of the downsampling rate. Therefore, the probability of falsely dismissing two true patterns in **G** in this space becomes greater. We illustrate this error as the shaded area D in Fig. 50.*bottom*. We call this error β_N, and calculate it as follows.

From the fixed α, we can calculate $z_{False\ Positive}$. Using this $z_{False\ Positive}$, we can calculate T_N (cf. Fig. 50.*bottom*) as below:

$$T_N = \mu_{D_{R_N}} + (z_{False\ Positive} * \sigma_{D_{R_N}})$$

where $\mu_{D_{R_N}}$ and $\sigma_{D_{R_N}}$ are the mean and standard deviation of the distance distribution D_{R_N} in Fig. 50.*bottom*.

We can calculate the standard score of T_N corresponding to the distribution D_{G_N} in Fig. 50.*bottom* as:

$$z_N = (T_N - \mu_{D_{G_N}})/\sigma_{D_{G_N}}$$

Using the standard normal probability table, we can calculate the area to the right of z_N, which is β_N. β and β_N correspond to the probabilities shown in Fig. 51.*left*) for the original and downsampled spaces, respectively.

From the areas *A, B, C* and *D* in Fig. 50.*bottom),* we can calculate the probability that two cached patterns are believed to be exemplars in **G,** p_{G_N} using equation 2. p_G and p_{G_N} correspond to the probabilities shown in Fig. 51.*right*) for the original and downsampled spaces, respectively.

More generally, Fig. 52 shows the relationship between a 'virtually' large cache made by downsampling the data and the number of objects we expect to see before reporting success, under the following conditions. We make a synthetic dataset of two classes, **G** and **R**, by distorting an instance of Gun point and FaceAll datasets [19], respectively, with some Gaussian noise. We assume the rareness of the patterns in **G** is 1/100 and our cache is allowed to store just two patterns in raw format. Under the assumption that we can unambiguously detect instances in **G**, we show the expected number of objects we need to see before reporting success in Fig. 52.*left*).
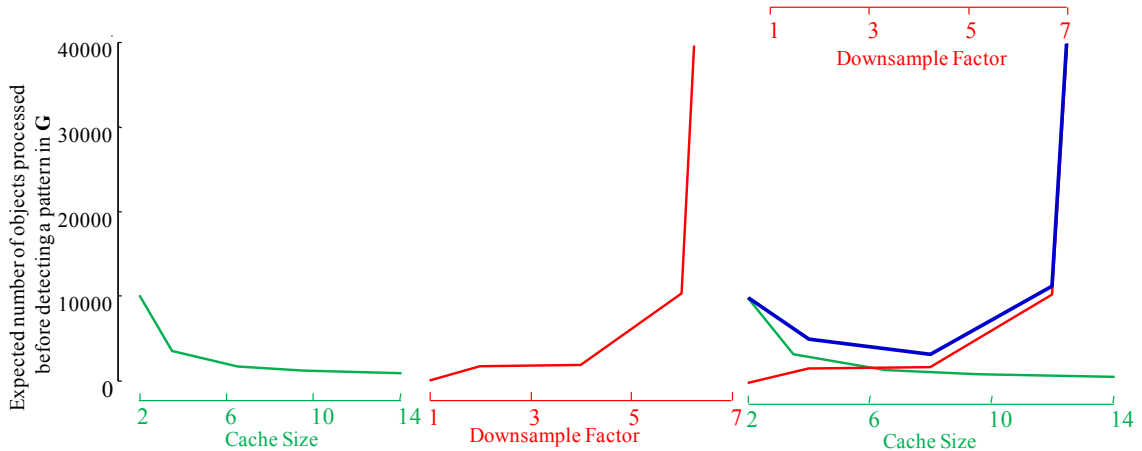
Fig. 52.: *left*) The *help* factor: A larger cache allows faster detection of instances in G. *center*) The *hurt* factor: The greater the downsampling, the slower the detection. *right*) The performance of the overall system is based on the influence of these two factors, with downsample factors of 2 to 4 performing best.

As noted above, due to downsampling the data, we can no longer identify instances in **G** unambiguously; therefore, we detect two instances in **G** with probability $p_\mathbf{G}$. Because we fix the false positive error probability, the probability of falsely dismissing true instances in **G** tends to *increase* as we downsample more (Fig. 51.*left*). This increased probability of false dismissals of the patterns in **G** results in a decreasing tendency of $p_\mathbf{G}$ (Fig. 51.*right*). As a consequence, we expect to see an increase in the expected number of objects in **S** we examine before detecting an instance in **G**. This effect is illustrated in Fig. 52.*center*).

If we review these two observations: *A larger cache allows faster detection of an instance in* **G** and *A "larger" cache* (*emulated by downsampling*) *causes slower detection of an instance in* **G** *given the higher probability of not recognizing a pair from* **G**, they suggest that there must be a cache size that maximizes this tradeoff, i.e., $cost_{final} = cost_{help} + cost_{hurt}$. In Fig. 52.*right*), this cache size is achieved with downsample factors from two to four.

There are two important observations we must make before moving on. First, note that the cache size that maximizes this tradeoff has a fairly wide "valley," suggesting that this parameter is not too sensitive. Second, while the best downsampling factor depends on the data and its sampling rate, it *can* be robustly learned on a small amount of training data. That is to say, if we find that the best downsampling factor for one person's electrooculography data at 1024Hz is about eight to ten, we can expect this to generalize well to other individuals.

4.4.3 Dimensionality Reduction

In our simple analysis in the previous section, we assumed that we placed more objects into the cache by *downsampling*. However, the reader will appreciate that there are more sophisticated dimensionality and cardinality reduction techniques to reduce the size of a time series. Indeed, the literature is replete with dimensionality reduction techniques, such as Singular Value Decomposition (SVD), Discrete Fourier Transform (DFT), Discrete Wavelet Transform (DWT), and Piecewise Aggregate Approximation (PAA) [139]. Here we consider PAA as the dimensionality reduction technique, because it is simple, incrementally computable and has linear time complexity [122]. We note that PAA and DWT are logically identical if both the original and reduced dimensionality are integer powers of two, and nearly identical otherwise [122].

Recall our observation in Fig. 50; in Fig. 53 we demonstrate that we get a similar shifting of the distance distributions under PAA as downsampling.
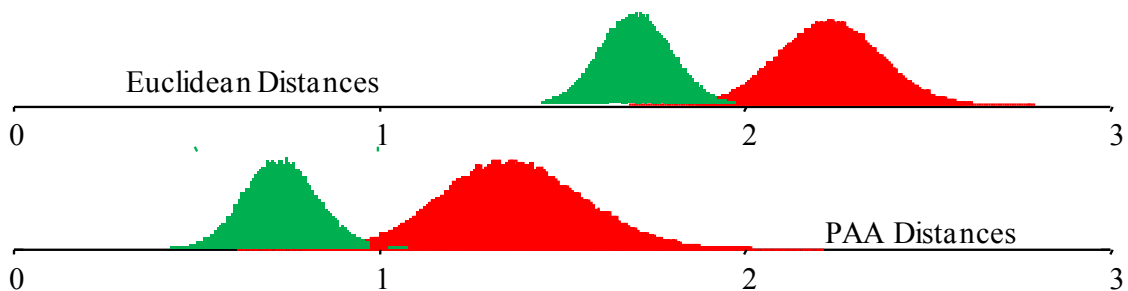
Fig. 53.: *top***) The all-pair distance distributions of the patterns in G and R in the raw space.** *bottom***) After reducing the dimensionality by an integer factor of 5, the new distance distributions shift left.**

125

As noted in Section 4.3.3, a larger cache size allows faster detection of a pattern in **G**, and dimensionality reduction emulates the effect of a larger cache. But as described in Section 4.4.1, working in the dimensionality-reduced space also makes it harder to determine if two objects in the cache belong to **G**. Therefore, we should expect to see a similar "*help and hurt*" effect as illustrated in Fig. 52. To see this, in Fig. 54 we conducted a similar experiment on the same synthetic dataset in Section 4.4.2.1 using PAA instead of downsampling.
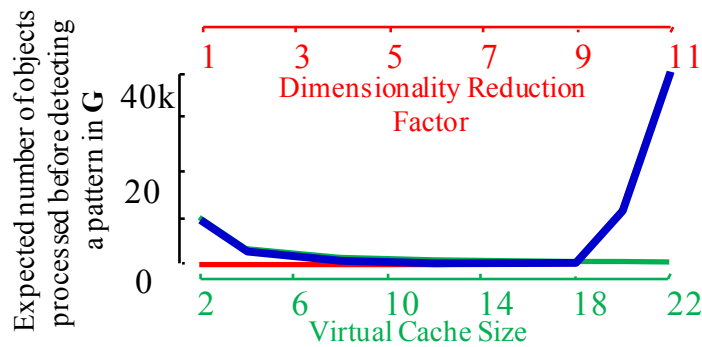
Fig. 54.: *help* **factor vs.** *hurt* **factor using PAA. The tradeoff between both factors suggests that a dimensionality reduction factor of about 10 is best here.**

4.4.4 Cardinality Reduction

We use the SAX (Symbolic Aggregate Approximation) [126] approach for cardinality reduction. This is because SAX is unique in allowing a distance calculation in the symbolic space that is commensurate with the Euclidean distance.

In Fig. 55 we show that if we reduce the volume of time series with *cardinality* reduction rather than *dimensionality* reduction (cf. Fig. 53), the distance distributions in

the approximate space do not shift to the left as much. This is a promising sign that

cardinality reduction might be a better technique for the task at hand.
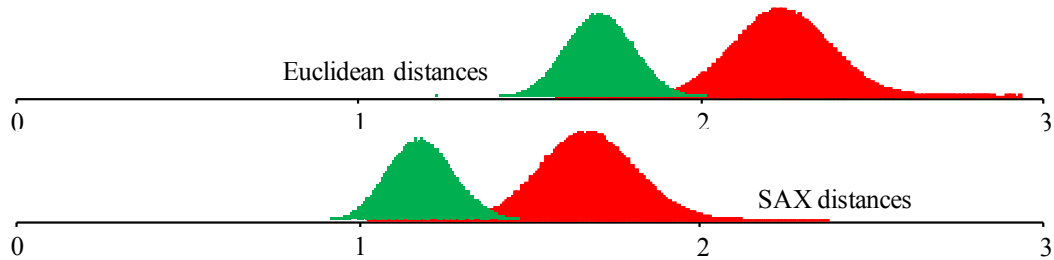
Fig. 55.: *top*) **The all-pair distance distributions of the patterns in G and R in the raw space.** *bottom*) **After reducing only the cardinality of the data by a factor of 5 (6 bits), the new distance distributions shift left (assuming the original data points are 32 bits).**

As hinted at in the observation in Section 4.4.1, we get the similar "*help and hurt*"

behaviors after doing cardinality reduction on the same synthetic dataset in Section

4.4.2.1. We illustrate this in Fig. 56.
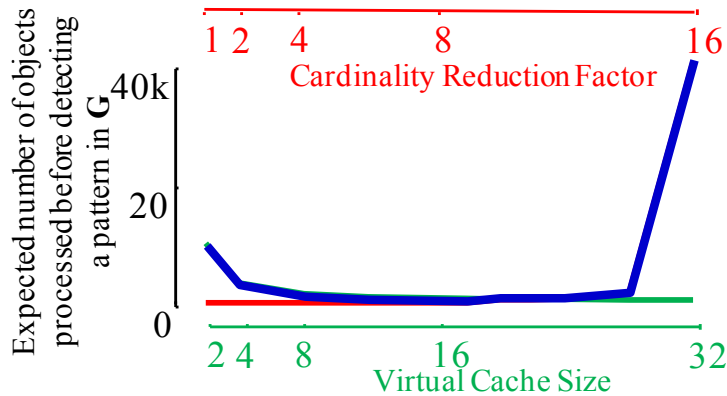
Fig. 56.: **The tradeoff between the *help* and *hurt* factors using SAX suggests that a cardinality reduction factor of about 8 is best.**

We can now answer the following question: Of the three techniques introduced to

emulate a large cache, which is best? To see this, we plot the results of the experiments in

this section on a single commensurate axis in Fig. 57. We can see that cardinality reduction gives the minimum value of the expected number of objects we need to see before we report success. Moreover, cardinality reduction has a very wide flat "valley," meaning a large range of parameter choices produces excellent results.
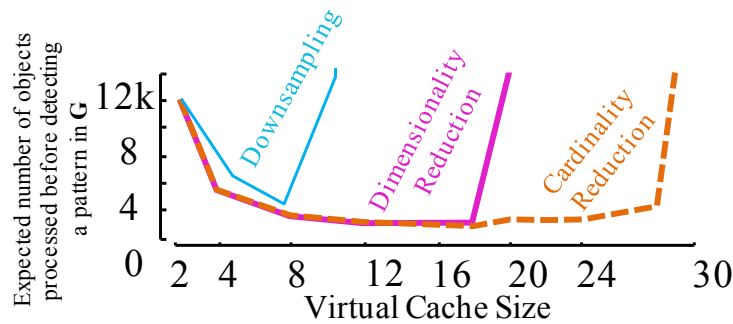
Fig. 57.: Of the three data reduction techniques, cardinality reduction dominates the other two over the entire range of virtual cache sizes.

Note that these results are for a single dataset with a single setting. However, many additional experiments (archived at [141]) confirm this general behavior.

4.5 Sticky Cache Algorithm

As hinted at in the thought experiment in Section 4.3.4, if we had a 'magic' cache in which the *potential* instances in **G** tend to remain for longer, the probability of early detection of an instance in **G** increases significantly. In order to realize this idea, we need to create a biased cache replacement policy that reduces the probability of discarding potential **G** items from the cache as opposed to instances in **R**. This seems to open a

chicken-and-egg paradox, as finding a pair of objects from **G** is our *goal*. In this section we will show how we can use a Bloom filter [112] to resolve this paradox.

A Bloom filter is a space-efficient randomized data structure (bit array) to support membership queries with no false negative and a small false positive probability [112]. It uses k independent hash functions, each of which maps some set element to one of the m bit positions of the array and sets the hashed bit positions. A membership query takes an input element and feeds it to the k hash functions to get k array positions. If any of the bit positions is found to be 0, then the element is *definitely* not a set member; otherwise, the element is *probably* a set member. Bloom filters have been widely used in frequent pattern mining [125][135]. To the best of our knowledge, this work is the pioneer where Bloom filters have been used for *real-valued* time series.

The high-level intuition behind our idea is as follows. For every subsequence we see, we will use a Bloom filter to "remember" seeing (a SAX representation of) it. Before inserting the SAX word corresponding to the subsequence into the Bloom filter, we check to see if we have *already* seen this SAX word. If we have, this is suggestive that the subsequence *may* be from **G**. Given that evidence, we should make sure that it "sticks" in the cache longer than the subsequences we have only seen once.

There are some obvious caveats to this idea. Two SAX words being identical does not guarantee that both original real-valued sequences come from **G**, and two real-valued sequences that come from **G** can map to different SAX words. However, as we shall show, for reasonable SAX parameters false positives and false negatives are rare, and a collision in the SAX space is *really* highly predictive of membership in **G**.

Note that for this idea to work we actually need to see *three* items from **G**. The first is inserted into the Bloom filter as with all items. The second item collides with the first, which tells us to store the *real-valued* version of the second item in the cache, marked with low priority for deletion. Finally, when the third item arrives it will be recognized to be within threshold *T* of the second item, and we can report success.

The success of our method depends on finding reasonable values for the two SAX parameters - word and alphabet size. This is straightforward, so we relegate the discussion to [141].

4.5.1   Setting Appropriate Bloom Filter Size

As noted in Section 4.5, we require a biased cache replacement policy to detect instances in **G** earlier. The Bloom filter tags each instance in **S** as potential members in **G** or **R**. Based on these tags, we make the potential instances in **R** $\sum$ times more likely to be discarded from the cache than that of a potential instance in **G**. It is important to note that $\sum$ is not a critical parameter. Consider the two extreme situations. If $\sum = 1$, then the sticky cache policy degenerates to RR. In contrast, if $\sum = \infty$, the cache gets filled with potential instances in **G**, which results in frequent flushes of the cache, and decreases the probability of detecting instances in **G**. Note that the Bloom filter is not "free"; therefore, we must use up a portion of *C* for it. As the reader will appreciate, the two extreme choices of using almost *all* of *C*, or almost *none* of *C* (thus essentially degenerating to RR), are unlikely to perform well. The following analysis allows us to derive the appropriate size allocation for the Bloom filter in *C*.

Assuming C can hold at most ρ instances of length λ each, then the size of C in bytes is: $w_{cache} = 8\lambda\rho^2$. If we restrict C to hold at most ρ_r instances ($\rho_r \leq \rho$), and allocate the remaining space to the Bloom filter, then the size of the Bloom filter in bytes is: $w_{Bloom} = \lfloor w_{cache} - 8\lambda\rho_r \rfloor$. Therefore, the number of unique elements hashed into the Bloom filter is: $b_{Bloom} = 8w_{Bloom}/bitsPerElement$[3].

Based on the analysis above, we perform an experiment in which we vary the cache /Bloom filter allocation in C to identify the region in which we obtain significant performance improvement over the naive RR policy. We use a fixed buffer which can hold at most 32 patterns each of length 150; thus, $w_{cache} = 38,400$ bytes. We vary the number of cache elements/ Bloom filter allocation. From Fig. 58 we can see that the cache holding 8 to 12 patterns performs best and beyond this allocation, the performance starts degrading.
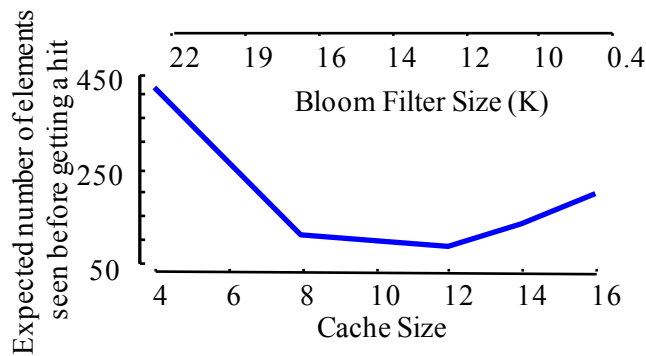
Fig. 58.: For a fixed $w = 38,400$ bytes (32 patterns of length 150), $\sum = 64$, and $p = 1/100$, a cache allocation of 9,600 to 14,400 bytes (i.e., 8 to 12 patterns) performs best.

---

[2] Each value of the time series takes 8 bytes.

[3] 9.58 bits per element for 1% FP probability of the Bloom filter.

The reader will have anticipated the following question: if we exploit the performance improvement of the sticky cache in addition to the cardinality reduction technique (the best technique from Section 4.4.4), can we do even better? To see this, we perform cardinality reduction of the patterns by a factor of 8 (the best reduction factor as of Fig. 56), and redo the sticky cache experiment. We plot the results of this experiment with the other data reduction techniques in Section 4.4 with the same setup and plot the results on a common axis in Fig. 59.
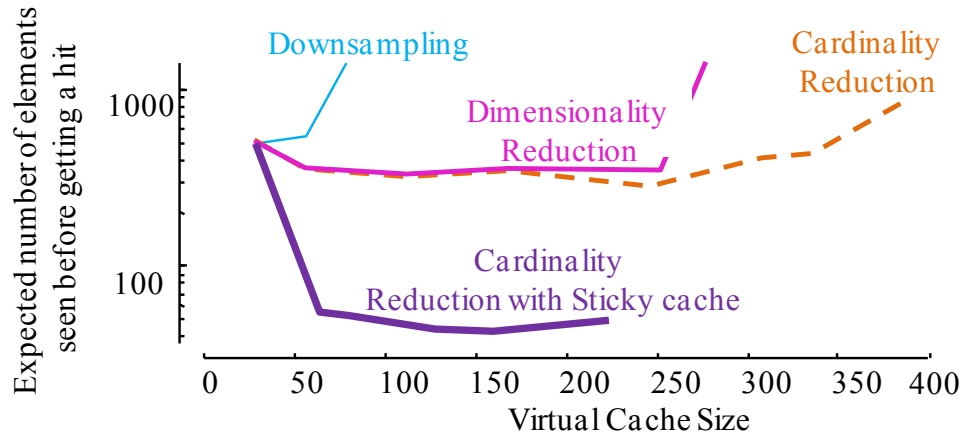
Fig. 59.: Out of all optimization techniques we discuss, the Sticky cache approach with a cardinality reduction factor of eight performs best. See also Fig. 57, which shows a subset of this data.

In the next section we more formally describe the sticky cache algorithm.

4.5.2   Algorithms

In order to elucidate our sticky cache framework, we first explain how we use the Bloom filter in order to identify potential instances in **G**. Later we describe the cache maintenance policy.

4.5.2.1 Detection of Potential Target Instances

For a given alphabet size a and word size ω, we call the probability that two instances in **G** and **R** will map to the same SAX string $p_{\mathbf{G}sameSAXString(a,\omega)}$ and $p_{\mathbf{R}sameSAXString(a,\omega)}$, respectively. As explained in Section 4.5.1, in order to exploit the Bloom filter for detecting potential instances in **G**, we have to determine the approximate SAX parameters so that $p_{\mathbf{G}sameString}$ is maximized and $p_{\mathbf{R}sameString}$ is minimized. In 16, we describe the algorithm.

Recall our assumption of the existence of a distance threshold as a form of domain knowledge in Section 4.3. Based on this domain knowledge, we form a buffer of instances in **G** in line 1. We randomly sample patterns of the length of our interest from our input time series in order to form a buffer of instances in **R** in line 2. From lines 3-9, we determine the appropriate SAX parameters using the scoring function outlined in [141].

Table 16: SAX Parameter Selection Algorithm

Input	**TS**, the input time series **l,** length of the subsequences **A**, predefined set of SAX alphabet sizes **Ω,** predefined set of SAX word sizes
Output	*a*, the appropriate SAX alphabet size *ω*, the appropriate SAX word size

1	**G**Buffer = form**G**buffer(TS,l) //using domain knowledge
2	RBuffer = formRbuffer(TS,l) //by random sampling
3	**for** i = 1 to size(A)
4	**for** j = 1 to size(Ω)
5	$p_{normalized\mathbf{G}sameSAXString}$ =findNormalizedProbability(**G**Buffer,i,j)
6	$p_{normalized\mathbf{R}sameSAXString}$ =findNormalizedProbability(**R**Buffer,i,j)
7	**end**
8	**end**
9	(*a*,ω)=findMaxNormalizedScore($p_{normalized\mathbf{G}sameSAXString}$, $p_{normalized\mathbf{R}sameSAXString}$)

We are now in a position to use the Bloom filter in order to detect potential instances in **G**. In Table 17, we outline the algorithm.

In line 1, we quantize the input subsequence with the appropriate SAX parameters we discovered in 16. We check whether the discretized SAX word exists in the Bloom filter in line 2. If the SAX word exists, then we mark it as a potential instance in **G** and assign weight 1 (line 4). Otherwise, the potential instances in **R** are assigned the weight that determines how likely they are to be discarded from the cache (line 7) and insert the word into the Bloom filter (lines 9 and 13). We check to see whether the Bloom filter is saturated or not (lines 8 and 11). In case the filter *is* saturated, we flush it (line 12).

Table 17: Potential Target Instance Selection Algorithm

Input	**TS**, the input time series **subSeq**, subsequence in question *a*, the appropriate SAX alphabet size *ω*, the appropriate SAX word size $\sum$s, likeliness factor of potential instances in **R** to be discarded from the cache
Output	$\sum$, weight of the potential instances in **R**

1	SAX_word = quantizeBySAX(subseq, *a*, ω);
2	bool exists = existsInBloomFilter(SAX_word)
3	**if** exists == true
4	$\sum$ = 1
5	**return**
6	**else**
7	$\sum$ = $\sum$s;
8	**if** saturatedBloomFilter()== false
9	insertInBloomFilter(SAX_word)
10	**return**;
11	**else**
12	flushBloomFilter()
13	insertInBloomFilter(SAX_word)
14	**return**
15	**endif**
16	**endif**

4.5.2.2 Cache Maintenance

Using the algorithm outlined above, we 'tag' the instances by a user-defined factor, which determines the relative discard probabilities for *potential* instances **R** and **G**. We describe the algorithm in Table 18.

Table 18: Cache Maintenance Algorithm

Input	**subSeq**, subsequence in question *w*, size of the cache **C,** the cache $\sum$, likeliness factor of subSeq to be discarded from the cache **LikelinessBuffer,** the buffer storing $\sum$
Output	**success**, flag indicating successful cache insertion

1	**if** currentCacheElementCount ≤ w //underflow
2	currentCacheElementCount++
3	InsertInCache(subSeq, currentCacheElementCount)
4	//insert $\sum$ in LikelinessBuffer
5	InsertInLikelinessBuffer($\sum$,currentCacheElementCount)
6	success = true
7	**else** //overflow
8	**//Pathological Situation 1**
9	//if all potential instances in **G** are in *C*
10	**if** sum(LikelinessBuffer) == w
11	flushBloomFilter()
12	flushCache()
13	flushLikelinessBuffer()
14	currentCacheElementCount = 0
15	InsertInCache(subSeq,1)//insert in first location
16	InsertInLikelinessBuffer($\sum$,1)
17	SAX_word= quantizeBySAX(subseq, *a*, ω)
18	insertInBloomFilter(SAX_word)
19	currentCacheElementCount++
20	**//Pathological Situation 2**
21	**else if** saturatedBloomFilter()==true
22	flushBloomFilter()
23	currentCacheElementCount++
24	InsertInCache(subSeq, currentCacheElementCount)
25	InsertInLikelinessBuffer($\sum$,currentCacheElementCount)
26	SAX_word= quantizeBySAX(subseq, *a*, ω)
27	insertInBloomFilter(SAX_word)
28	**else**
29	loc = calculateCacheDiscardLocation(LikelinessBuffer)
30	InsertInCache(subSeq, loc)
31	InsertInLikelinessBuffer($\sum$,loc)
32	**endif**
33	success = true

If there is no cache overflow, we insert the subsequence into the next available cache slot (lines 1-6). Otherwise, we check for two pathological situations. If the cache is

full of potential instances in **G** (which is *extremely* unlikely to occur provided that we have set *a*, ω properly), then we flush the cache, the Bloom filter and the buffer storing the likeliness factor of each cache element, and start from scratch (lines 10-19). We also check for saturation of the Bloom filter, and if we find it, we flush the Bloom filter (lines 21-22). Otherwise, we determine the discard location based on the likeliness factors of the elements and insert instances into the cache accordingly (lines 28-32). We describe the cache discard location algorithm in Table 19.

We generate a random index between 1 and the total weight in the LikelinessBuffer (line 1). We calculate the desired discard location so that potential instances in **R** are $\sum$ times more likely to be discarded (lines 2-4), and return.

Table 19: Cache Discard Location Calculation Algorithm

Input	**LikelinessBuffer,** the buffer storing $\sum$
Output	**loc**, cache discard location

1
2
3
4
5
	ind = generateRandomIndex([1,sum(LikelinessBuffer)])
	cumSumArray = cumulativeSum(LikelinessBuffer)
	//find the first smallest index greater than or equal //to ind in cumSumArray
	loc = firstSmallestIndex(cumSumArray,ind)
	return

4.6 Experimental Evaluation

We begin by noting that *all* experiments (including all the figures above) are completely reproducible. All experimental code and data (and additional experiments omitted for brevity) are archived in perpetuity at [141].

The goal of our experiments is to show that our algorithm is more efficient than any obvious strawman technique, and that it is not particularly sensitive to the parameter choices. In addition, we show the utility of our approach with case studies on two real-world datasets.

4.6.1   Rate of Detection

We begin by comparing our algorithm with the naive RR cache replacement policy. RR is a simple, but highly effective algorithm for many problems.

From the UCR archive we take examples of class 1 from the MALLAT dataset [19] and consider them as target motifs. We randomly embed these into a much longer random walk time series with different occurrence probabilities, and our task is to recover at least one matching pair as soon as possible.

We consider the cache size to be a function of the target patterns' occurrence probability. In particular, our cache can buffer at most only 10% of the rareness of the target motifs. In other words, if our target motifs have 1/100 occurrence probability, then our cache can hold at most only 10 patterns. In addition, we use 3.2, 12 and 16 as the values of T, a, and ω, respectively. Furthermore, we will show that the setting of these

parameters is not a black art; there exists a wide range of possible values of the parameters that have no significant impact on the performance of our algorithm.

In order to show how quickly we detect the target patterns, we do the following. We run our sticky cache algorithm and the RR algorithm under the same experimental conditions and record the average number of objects we must process until we get the first true positive (i.e., a pair of patterns *are* in the cache, and our algorithm recognizes this fact).

As we are interested in our algorithm's performance relative to RR, as shown in Fig. 60, we plot the results relative to the *mean* performance of RR, with values less than one indicating that our algorithm offers an improvement. In addition, to show the *quality* of our algorithm, we record the number of false positives we see before we attain success.

From Fig. 60.*top*), we can see that in the worst case, the sticky cache algorithm is faster than the RR policy by a factor of 2. In addition to this, because the median score of the sticky cache algorithm remains almost constant (~0.18) for different rareness factors, we can say that on average, our algorithm is ~6 times faster than the RR policy. In particular, on average, the probability that an object is a false positive before we attain success is at most only ~2% [141]. The results above suggest the utility of an *adaptive T* as a function of the rareness factor. We modified our model to achieve this; however, because this is straightforward, we relegate the discussion to [141].
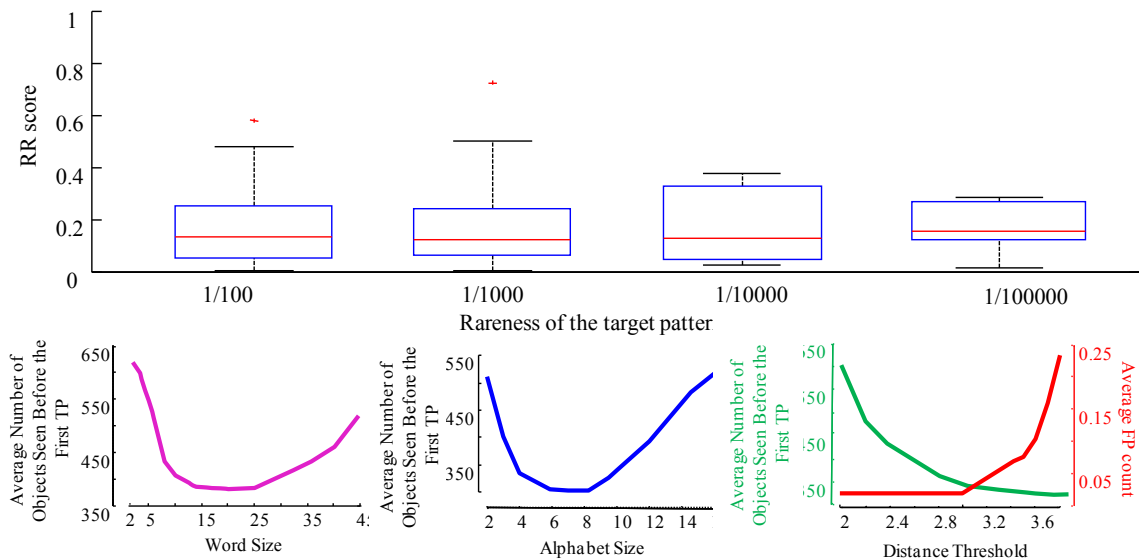
Fig. 60.: *top*) **For the sticky cache algorithm, the average number of objects seen before the first true positive is expressed with respect to the results for RR policy at different rareness factors.** *bottom*) **The parameter robustness of our algorithm offers a wide range of possible values that have no significant impact on performance (from left to right ω, a and T).**

Recall our claim that our algorithm is not sensitive to the parameter choices. In order to show this, we do the following experiment. Assume that our data generator generates the target patterns with a probability of 1/100. Keeping the same experimental setup we discussed above, we vary ω, a and T, and show the result in Fig. 60.*bottom*). The results show too conservative or too liberal values of ω and a result in an increase in the average number of objects we see before the first true positive. In addition, if we make T too liberal, then the average false positive count increases and vice versa. However, the results clearly show the existence of a wide range of choice of these parameters which confirms our algorithm is not parameter-sensitive.

4.6.2 Worst-Case Time Complexity

Consider the worst-case scenario. For each pattern TS_i of length m in question, we make its SAX representation first. This step needs O(m) time. In order to detect whether TS_i is a potential instance in **G** or **R**, we hash it into the Bloom filter. Assume the number of independent hash functions we use in the Bloom filter is h. Given this, querying if the SAX representation of TS_i is in the Bloom filter, and if not, hashing it into the Bloom filter, requires O(h) time. In the worst possible case, the Bloom filter will be saturated, and we flush it. This is a constant-time operation. After this, we insert TS_i into a cache of size w. For an overflowed cache, discarding a cache element requires O(1) time. After we insert TS_i into the cache, we calculate its distance from all cache elements (recall this is the worst-case scenario) until the participating patterns pass the threshold test. This needs O(wm) time. Therefore, the overall worst-case time complexity of our algorithm is O(m) + O(h) + O(wm). In practice, this means our somewhat naive implementation can handle 250Hz under typical parameter settings, and a carefully optimized implementation could easily handle 1,000Hz.

4.6.3 Case Studies

4.6.3.1 Wildlife Monitoring

Wildlife monitoring by examining sensor traces has been shown to be a useful tool for measuring the health of the environment [137]. In some cases, we may have a *known* bird call we would like to monitor, but here we consider the more difficult task of

detecting previously *unknown* calls. Our only assumptions are that the call will be repeated at least once.

Assume we monitor the audio trace of a ten-hour-long night of a forest [134]. Given a data rate of 62 Hz in the Mel-Frequency Cepstral Coefficients (MFCC) space, we will see about 2.2 million data points. Assume we have a fixed-size memory which can buffer at most only 1/4000 of the subsequences that appear on this night. Our final assumption is that we have a predefined distance threshold for detecting a pair of target patterns (which we learned offline on a handful of known bird calls). Given these assumptions, if a bird calls randomly ten times during this night, we can ask the following question: *What fraction of nights can we expect to detect at least one pair (any pair) of bird calls*? Recall that detecting a *single* pair is sufficient for the wildlife monitoring task. In order to answer this question, we perform the following experiment. In a ten-hour-long audio trace of environmental sounds, we randomly insert ten approximately three-second-long calls of a White Crowned Sparrow (*Zonotrichia leucophrys*) [140]. We run our sticky cache algorithm ten times on this dataset and in each run we continue monitoring until we detect the first true positive pairs in the cache.

Our experimental results tell us we can expect to detect the target bird 98 out of 100 nights. Impressive as these results are, they are somewhat pessimistic. After a careful analysis of the results we discovered that the "false positives" are actually true bird sounds. In Fig. 61 we show examples of both the injected bird calls we recovered, and other bird calls (unknown species) we recovered.

We do not report the timing experiments, except to note that we can easily search a dataset with an arrival rate much *faster than real-time* on a laptop, suggesting we could handle real-time even on a resource-limited recording device.
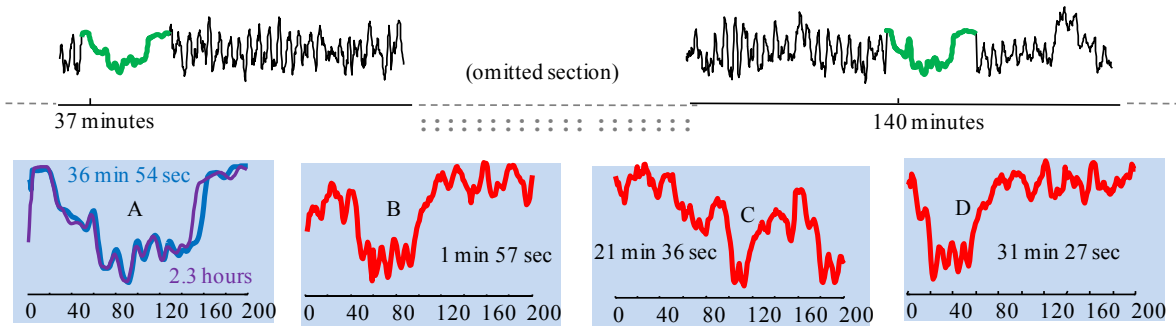
Fig. 61.: *top*) A snippet of the ten-hour-long audio trace in the MFCC space. The injected bird calls are shown in green/bold. *bottom*) A) The detected motif pairs occurring at 37 minutes and 2.3 hours, respectively. B – D) Three examples of *unknown* bird calls discovered.

4.6.3.2 Energy Disaggregation

The problem of reducing energy consumption has attracted increasing interest in recent years. To illustrate how our approach may help in solving energy disaggregation problems, we consider one year of energy usage data, containing 0.5 million points [128]. For simplicity, we consider a meter that monitors the electricity usage of just two appliances– a refrigerator and a dishwasher. From personal experience, we assume the dishwasher cycles are approximately 1.5 hours long. As before, we use a cache which buffers at most 5% of the data, and had 20, 12, and 7.5 as values of ω, a, and T, respectively. In order to show how *effective* our algorithm is, we annotate the ground truth by careful human inspection. As soon as our algorithm detects a target motif pair, we flush the cache and continue scanning the dataset. We show the result in Fig. 62.

143
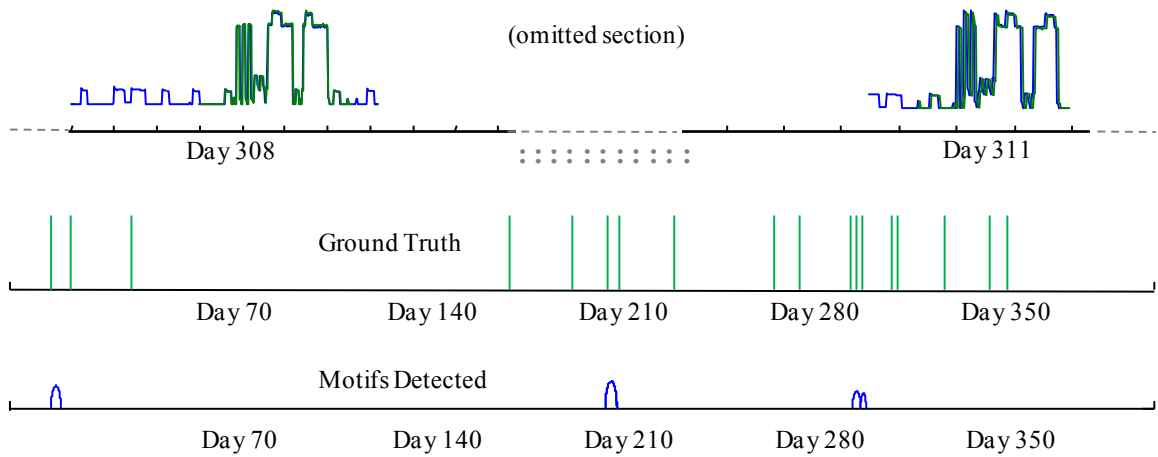
Fig. 62.: *top*) **An excerpt of the electricity usage of a refrigerator and dishwasher. The dishwasher patterns have been marked in green/bold.** *bottom*) **The ground truth locations (dishwasher usage occurrences) are shown as green lines. The locations identified by our algorithm are shown as blue arcs.**

From Fig. 62 we can see that our algorithm detected eight dishwasher motif pairs, most of which are many days apart.

4.7 Related Work

In recent years researchers have devoted significant attention to efficiently discovering motifs in *static* offline databases [117][127][138]. Until [133], all scalable motif discovery algorithms were *approximate*. In [133], the authors proposed an *exact* motif discovery algorithm which was tenable for a database of millions of objects. Although the worst case time complexity of [133] was quadratic, in practice this is a very pessimistic bound and it was shown to be fast enough to serve as a subroutine for summarization, near-duplicate detection, etc.

Because most data sources are not static, and we may need to deal with unbounded streams, the necessity of *online* discovery of time series motifs has been noted [132]. However the only work devoted to this problem limits its consideration to the last *k* minutes, for some small *k* [132]. This means that [132] maintains motifs based on the most recent history of the stream. However, as we noted in our real-world case studies, we may need to find patterns that repeat hours, days or even weeks apart. For such cases, it is very unlikely that motifs will occur in the same window. In addition to this, if we consider the *huge* volume of data that we wish to process, we are bounded by the scalability of the fastest offline algorithm for this problem [131][133]. Our work is different in a sense that we detect *very* sparse motifs using a *very* limited buffer compared to the size of the data with *very* high probability. In the context of the data *volume*, the interested reader might think of [130], which detects motifs from gigabyte-scale databases. However, [130] is a *multi-pass* algorithm, whereas we explicitly address situations where we can scan the data only *once* to detect motifs.

Various discrete analogues of our problem *have* seen significant research; see [118] and the references therein. In the discrete space the ability to directly test for equality and to directly hash that data, makes the "rare pattern" problem significantly easier. However our use of Bloom filters was inspired by this community's literature [118]. Bloom filters have been an area of active research in the database community for the last decade, with research effort in both applying them to various problems and introducing variants such as "*forgetting*" Bloom filters [118].

4.8 Conclusions

We have argued that for most applications, finding the closest pair of subsequences is intractable and *unnecessary*. It suffices to find any pair of repeated patterns. *Any* pair can be used to alert an ornithologist to listen to a snippet of bird calls (cf. Section 4.6.3.1), or allow a technician to build a "dictionary" of electrical demand patterns (cf. 4.6.3.2), etc. Based on this observation, we have introduced the first algorithm that can detect repeated patterns in unbounded *real-valued* time series. We have demonstrated efficiency and effectiveness of this algorithm on both synthetic and real-world datasets.

Chapter 5: Conclusion and Directions to Future Extension

Time series data mining is an active area of research now. With the advancement of sensor technology, these days it is extremely easy obtaining vast amount of time series data. However, for this data to be useful, it has to be collected properly and labeled. In real-life scenario, the data collection process faces failures at different levels. In this circumstance, generating clean data from scratch requires careful cleaning and pre processing. In addition, obtaining human annotation is expensive, and remains the critical bottleneck for the data analysis process. Our core contribution in mining time series data from the perspective of scarce labeled data *and* abundance of noisy data is as follows:

- In Chapter 2 of this dissertation, we propose approaches to mitigate human labor in mining time series data by semi-supervised learning. We proposed a novel method of semi-supervised classification of time series with as few as a single labeled instance. Previous approaches for stopping the classification process required extensive parameter tuning, and hence remain something of a black art. We devised a stopping criterion for the semi-supervised classification based on Minimum Description Length (MDL), which is parameter free, and leverages the intrinsic structure of the data. To our knowledge, this is the first work addressing time series

SSL using MDL. Extensive experiments demonstrate that our method allows us to classify real-world medical datasets with near perfect accuracy.

Our current approach works only on offline time series data; a future direction might be extending it to perform online semi-supervised time series classification. Another potential contribution is to generalize our algorithm in the setting of multiple classes.

- In Chapter 3 of this dissertation, we propose a generalized density based clustering framework for admissible clustering. As we have mentioned before, designing parameter-lite algorithms is one of the key aspects of ensuring the reproducibility of results. In addition, such algorithms are generally less brittle and less ad-hoc. In many situations, clustering the whole dataset might hurt the cluster quality. For example, simple objects tend to be too close from everything. This is why, they act as 'attractors', and tend to swallow almost all the objects in their own cluster. Therefore, it is important that the underlying clustering algorithm is agnostic to such objects which are either not clusterable or hurt the clusterability of other objects.

  Besides being agnostic to noise and outliers, it is necessary that the clustering algorithm provides high quality results as soon as possible. In many state-of-the-art time series works, it has been shown that when the data objects have time lag among them, the Dynamic Time Warping (DTW) distance measure is exceptionally difficult to beat. However, DTW is very slow, for which there is the need for a

clustering algorithm which could accelerate the clustering process by admissibly pruning unnecessary distance computations.

In Chapter 3, we augment the DP (Density Peaks) algorithm to cluster time series data using both the upper and lower bounds of Dynamic Time Warping. We propose a heuristic to order the distance computations in a *best-first* manner, which gives the algorithm a desirable anytime property. In addition, we show the generality of our clustering framework to other domains by efficiently obtaining semantically significant clusters in protein sequences using the *Edit Distance*, the discrete data analogue of DTW.

A future direction to this work is to perform a theoretical analysis on the asymptotic pruning rate. In addition, a concrete analysis on when TADPole does pruning better and when not, is a great direction to look closely into.

- In Chapter 4 of this dissertation, we investigate algorithms to discover rare time series motifs. We argue that that for most applications, rather than finding the closest pair of subsequences, it suffices to find any pair of repeated patterns. *Any* pair can be used to alert an ornithologist, or allow a technician to build a "dictionary" of electrical demand patterns, etc. Based on this observation, we have introduced the first algorithm that can detect repeated patterns in unbounded *real-valued* time series. We designed a rigorous theoretical model to understand the behavior of different cache replacement polices. In addition, we analyzed the cost model showing the independent factors associated on the expected number of

objects before we report success. Extensive empirical results support our claim that our Bloom filter based sticky cache algorithm with changed data representation can typically detect rare motifs with very high probability.

An interesting future direction to this work is to analyze the results of different algorithms when the Bloom filter is full. For now we flush the whole Bloom filter when this situation arises. But with such flushes, we are doing fresh restarts of the algorithm, and we might lose track of some potential motifs already present, but not detected in the cache yet. A future work would be to keep several parallel Bloom filters storing data of one month apart for example. We can think about flushing only the most recently flooded filter. This way we will not lose track of useful patterns already in the cache, and can exploit them better.

Bibliography

Chapter 2

[1] Besemer, J., Lomsadze, A., Borodovsky, M.: GeneMarkS: *A Self-training Method for Prediction of Gene Starts in Microbial Genomes*. Implications for Finding Sequence Motifs in Regulatory Regions. Nucleic Acids Research, 2001.

[2] Blum, A., Mitchell, T.: Combining *Labeled and Unlabeled Data with Co-training*. In Proceedings of the 11th ACM Annual Conference on Computational Learning Theory, pp. 92-100, 1998.

[3] Bouchard, D., Badler, N.: *Semantic Segmentation of Motion Capture Using Laban Movement Analysis*. In Intelligent Virtual Agents, pp. 37-44. Springer Berlin Heidelberg, 2007.

[4] Chapelle, O., Schölkopf, B., Zien, A.: *Semi-Supervised Learning*. Vol. 2. Cambridge, MA:: MIT press, 2006.

[5] Chazal, P. D., O'Dwyer, M., Reilly, R. B.: *Automatic Classification of Heartbeats Using ECG Morphology and Heartbeat Interval Features*. IEEE Transactions on Biomedical Engineering, 2004.

[6] Chen, Y., Hu, B., Keogh, E., Batista, G. E.: *DTW-D: Time Series Semi-supervised Learning from a Single Example*. The 19th ACM SIGKDD, pp. 383-391, 2013.

[7] Demšar, J.: Statistical *Comparisons of Classifiers Over Multiple Data Sets*. The Journal of Machine Learning Research 7: 1-30, 2006.

[8] Druck, G., Pal, C., Zhu, X., McCallum, A.: *Semi-Supervised Classification with Hybrid Generative/Discriminative Methods*. The 13th ACM SIGKDD, 2007.

[9] Florea, F., Müller, H., Rogozan, A., Geissbuhler, A., Darmoni, S.: *Medical image categorization with MedIC and MedGIFT*. Medical Informatics Europe (MIE), 2006.

[10] Geurts, P.: *Pattern Extraction for Time Series Classification*. Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery, 115 – 127, 2001.

[11] Goldberger, A. L. et al.: *PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals*. Circulation, 101(23): pp. 215-220, 2000.

[12] Greenwald, S. D., Patil, R. S., Mark, R. G.: *Improved Detection and Classification of Arrhythmias in Noise-corrupted Electrocardiograms using Contextual Information*. Proc' of IEEE Conf' on Computing in Cardiology, 1990.

[13] Greenwald, S. D.: The *Development and Analysis of a Ventricular Fibrillation Detector*. M.S. thesis, MIT Dept. of Electrical Engineering and Computer Science, 1986.

[14] Grünwald, P.: *A Tutorial Introduction to the Minimum Description Length Principle*, 2005.

[15] Herwig, M.: *Google's Total Library: Putting the World's Books on the Web*, 2007.

[16] Hills, J., Lines, J., Baranauskas, E., Mapp, J., Bagnall, A.: *Classification of Time Series by Shapelet Transformation*. Data Mining and Knowledge Discovery, 1-31, 2013.

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

[18] Jones, P. D., Hulme, M.: *Calculating Regional Climatic Time Series for Temperature and Precipitation: Methods and Illustrations*. International Journal of Climatology 16.4, 361-377, 1996.

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

[20] Li, M., Vitanyi, P.: *An Introduction to Kolmogorov Complexity and Its Applications*. 2nd Ed*, Springer*, 1997.

[21] Maeireizo, B., Litman, D., Hwa, R.: Co-*training for Predicting Emotions with Spoken Dialogue Data*. Proceedings of ACL, 2004.

[22] McClosky, D., Charniak, E., Johnson, M.: *Effective Self-training for Parsing*. In Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics, pp. 152-159, Association for Computational Linguistics, 2006.

[23] Nemenyi, P. B.: *Distribution-free Multiple Comparisons*. PhD Thesis, Princeton University, 1963.

[24] Nguyen, M. N., Li, X. L., Ng, S. K.: *Positive Unlabeled Learning for Time Series Classification*. Proc' of AAAI, 2011.

[25] Nguyen, M. N., Li, X. L., Ng, S. K.: *Ensemble Based Positive Unlabeled Learning for Time Series Classification*. Database Systems for Advanced Applications. Springer Berlin/Heidelberg, 2012.

[26] Ordonez, P., Oates, T., Lombardi, M. E., Hernandez, G., Holmes, K. W., Fackler, J., Lehmann, C. U.: *Visualization of Multivariate Time-series Data in a Neonatal ICU*. IBM Journal of Research and Development 56, no. 5, 2012.

[27] Patton, A. J.: *Copula–based Models for Financial Time Series*. Handbook of financial time series, Springer Berlin Heidelberg, pp. 767-785, 2009.

[28] Philipose, M.: *Large-Scale Human Activity Recognition Using Ultra-Dense Sensing*. The Bridge, National Academy of Engineering 35.4, 2005.

[29] Radovanovic, M., Nanopoulos, A., Ivanovic, M.: *Time-Series Classification in Many Intrinsic Dimensions*. SIAM SDM, 677-688, 2010.

[30] Rakthanmanon, T., Keogh, E., Lonardi, S., Evans, S.: *Time Series Epenthesis: Clustering Time Series Streams Requires Ignoring Some Data*. ICDM, 2011.

[31] Raptis, M., Wnuk, K., Soatto, S.: *Flexible Dictionaries for Action Classification*. The 1st International Workshop on Machine Learning for Vision-based Motion Analysis, 2008.

[32] Ratanamahatana, C. A., Keogh, E., *Making Time-series Classification More Accurate Using Learned Constraints*. SIAM SDM, 2004.

[33] Ratanamahatana, C. A., Wanichsan, D.: *Stopping Criterion Selection for Efficient Semi-supervised Time Series Classification*. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2008.

[34] Rosenberg, C., Hebert, M., Schneiderman, H.: *Semi-supervised Self-training of Object Detection Models*, 2005.

[35] Simon, B. P., Eswaran, C.: *An ECG Classifier Designed Using Modified Decision Based Neural Networks*. Computers and Biomedical Research, 257-272, 1997.

[36] Sun, A., Grishman, R.: *Semi-supervised Semantic Pattern Discovery with Guidance from Unsupervised Pattern Clusters*. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters, pp. 1194-1202, Association for Computational Linguistics, 2010.

[37] Sykacek, P., Roberts, S. J.: *Bayesian Time Series Classification.* Advances in Neural Information Processing Systems, 2002.

[38] Tsumoto, S.: Rule Discovery in Large Time-Series Medical Databases. In Principles of Data Mining and Knowledge Discovery, pp. 23-31. Springer Berlin Heidelberg (1999).

[39] Veeraraghavan, A., Chellappa, R., Srinivasan, M.: *Shape and Behavior Encoded Tracking of Bee Dances*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2008.

[40] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., Keogh, E. J.: *Experimental Comparison of Representation Methods and Distance Measures for Time Series Data*. Data Min. Knowl. Discov. 26(2): 275-309, 2013.

[41] Wei, L., Keogh, E.: *Semi-Supervised Time Series Classification*. SIGKDD, 2006.

[42] Xi, X., Keogh, E., Shelton, C., Wei, L., Ratanamahatana, C. A.: *Fast Time Series Classification Using Numerosity Reduction*. Proceedings of the 23rd ACM International Conference on Machine Learning, pp. 1033-1040, 2006.

[43] Yarowsky, D.: *Unsupervised Word Sense Disambiguation Rivaling Supervised Methods.* Proceedings of ACL, 1995.

[44] Zhu, X.: *Semi-supervised Learning Literature Survey*.Technical Report No. 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[45] Support website: *http://www.cs.ucr.edu/~nbegu001/SSL_myMDL.htm*

Chapter 3

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

[47] Agrawal, R., Faloutsos, C., & Swami, A. *Efficient Similarity Search in Sequence Databases* (pp. 69-84). Springer Berlin Heidelberg, 1993.

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

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

[50] Bradley, P. S., Fayyad, U. M., & Reina, C. *Scaling Clustering Algorithms to Large Databases.* ACM SIGKDD, pp. 9-15, 1998.

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

[52] Chambers, G. S., Venkatesh, S., West, G. A., & Bui, H. H. *Segmentation of Intentional Human Gestures for Sports Video Annotation*. IEEE Multimedia Modelling Conference, 2004.

[53] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. *Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures*. Proceedings of the VLDB Endowment,1(2), 2008.

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

[55] Ding, R., Wang, Q., Dang, Y., Fu, Q., Zhang, H., and Zhang, D. *Evaluation on Real Datasets: YADING: Fast Clustering of Large-scale Time Series Data*. Microsoft Tech Report , 2015.

[56] Ester, M., Kriegel, H. P., Sander, J., & Xu, X. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. ACM SIGKDD, pp. 226-231, 1996.

[57] Ester, M., Kriegel, H.-P., Sander, J., Wimmer, M., and Xu, X. *Incremental Clustering for Mining in A Data Warehousing Environment*. VLDB, vol. 98, pp. 323-333. 1998.

[58] Farré, D., Roset, R., Huerta, M., Adsuara, J. E., Roselló, L., Albà, M. M., & Messeguer, X. *Identification of Patterns in Biological Sequences at the ALGGEN server: PROMO and MALGEN*. Nucleic acids research, 31(13), 3651-3653, 2003.

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

[60] 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.

[61] Guha, S., Rastogi, R., & Shim, K. *CURE: An Efficient Clustering Algorithm for Large Databases*. ACM SIGMOD Record, vol. 27, no. 2, pp. 73-84, 1998.

[62] Hinneburg, A., and Gabriel, H.-H. *Denclue 2.0: Fast Clustering Based on Kernel Density Estimation*. Advances in Intelligent Data Analysis VII, pp. 70-80, 2007.

[63] Hildebrand, J. D., Schaller, M. D., & Parsons, J. T. *Identification of Sequences Required for the Efficient Localization of the Focal Adhesion Kinase, pp125FAK, to Cellular Focal Adhesions*. The Journal of cell biology, 123(4), 993-1005, 1993.

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

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

[66] Kahveci, T., & Singh, A. K. *An Efficient Index Structure for String Databases*. Proceedings of the VLDB Endowment, Vol. 1, pp. 351-360, 2001.

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

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

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

[70] Keogh, E., & Kasetty, S. *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration*. DMKD, 7(4), 349-371, 2003.

[71] Kotsifakos, A., Stefan, A., Athitsos, V., Das, G., & Papapetrou, P. *DRESS: Dimensionality Reduction for Efficient Sequence Search*. DMKD, 1-32, 2015.

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

[73] Levenshtein, V. I. *Binary Codes Capable of Correcting Deletions, Insertions, and Reversals*. In Soviet physics doklady (Vol. 10, No. 8, pp. 707-710), 1966.

[74] Liao, T. W. *Clustering of Time Series Data—A Survey*. Pattern recognition 38, no. 11, 2005.

[75] Liu, J., Zhong, L., Wickramasuriya, J., & Vasudevan, V. *uWave: Accelerometer-based Personalized Gesture Recognition and Its Applications*. Pervasive and Mobile Computing, 2009.

[76] Lord's, The Home of Cricket, *www.lords.org/laws-and-spirit/lawsof-cricket/laws/*, accessed on Sept 17, 2015.

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

[78] Metabolic Diseases Encyclopedia, *http://www.encyclopedia.com/topic/Metabolic_diseases.aspx*

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

[80] 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.

[81] Paparrizos, J., & Gravano, L. *k-Shape: Efficient and Accurate Clustering of Time Series*. ACM SIGMOD, 2015.

[82] Rabiner, L. R., & Juang, B. H. *An Introduction to Hidden Markov Models*. ASSP Magazine, IEEE, 3(1), 1986.

[83] Rabiner, L. R., & Levinson, S. E. *Isolated and Connected Word Recognition--Theory and Selected Applications*. Communications, IEEE Transactions on, 29(5), 621-659, 1981.

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

[85] 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.

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

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

[88] 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.

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

[90] 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.

[91] Sakoe, H., & Chiba, S. *Dynamic Programming Algorithm Optimization for Spoken Word Recognition*. IEEE Transactions on Acoustics, Speech and Signal Processing, 26(1), 43-49, 1978.

[92] 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.

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

[94] Shokoohi-Yekta, M., Hu, B., Jin, H., Wang, J., & Keogh, E. *Generalizing Dynamic Time Warping to the Multi-Dimensional Case Requires an Adaptive Approach*. SDM 2015.

[95] STRING database: http*://string-db.org/*

[96] 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.

[97] Ulanova, L., Hao, Y. and Keogh, E. *Generating Synthetic Data to Allow Learning from a Single Exemplar Per Class*. SISAP 2014.

[98] UniProt dataset: *http://www.ebi.ac.uk/uniprot/*

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

[100] Wang, J., Balasubramanian, A., Mojica de la Vega, L., Green, J. R., Samal, A., & Prabhakaran, B. *Word Recognition from Continuous Articulatory Movement Time-series Data Using Symbolic Representations*, ACL/ISCA Workshop on Speech and Language Processing for Assistive Technologies, 2013.

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

[102] Xu, X., Ester, M., Kriegel, H-P., and Sander, J. *A Distribution-based Clustering Algorithm for Mining in Large Spatial Databases*. ICDE, 1998 .pp. 324-331.

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

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

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

[106] 2009 MTV Video Music Awards *en.wikipedia.org/wiki/2009_MTV_Video_Music_Awards*

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

[108] Supporting Webpage: *http://www.cs.ucr.edu/~nbegu001/TADPoleDMKD*
Password: DMKD2016

Chapter 4

[109] Agrawal, R., Faloutsos, C., and Swami, A. *Efficient Similarity Search in Sequence Databases*. Springer, 1993.

[110] Barrenetxea, G., et al. *Sensorscope: Out-of-the-Box Environmental Monitoring*. IPSN, 2008.

[111] Bhattacharjee, R., Goel, A., and Lotker, Z. *Instability of FIFO at Arbitrarily Low Rates in the Adversarial Queuing Model*. SIAM Journal on Computing 34, no. 2, pp. 318-332, 2005.

[112] Bloom, B. H. *Space/Time Trade-offs in Hash Coding with Allowable Errors*. Communications of the ACM, vol. 13, issue 7, pp. 422 – 426, 1970.

[113] Brown, A. E. X., et. al. *A Dictionary of Behavioral Motifs Reveals Clusters of Genes Affecting Caenorhabditis elegans Locomotion*. Proceedings of the National Academy of Sciences 110, no. 2 pp. 791-796, 2013.

[114] Carton, C. et. al. *Fast Repetition Detection in TV streams Using Duration Patterns*. CBMI, 2013.

[115] Castro, N. C., et al. *Significant Motifs in Time Series*. Statistical Analysis and Data Mining 5, 2012.

[116] Chan, K. P., and Fu, A. W. C. *Efficient Time Series Matching by Wavelets*. Proc' of the 15th IEEE ICDE, 1999.

[117] Chiu, B., Keogh, E., and Lonardi, S. *Probabilistic Discovery of Time Series Motifs*. ACM SIGKDD, 2003.

[118] Cormode, G., and Hadjieleftheriou, M. *Methods for Finding Frequent Items in Data Streams*, VLDB Journal, 19(1), 3-20, 2010.

[119] Dasgupta, D., et. al. *Novelty Detection in Time Series Data Using Ideas from Immunology*. Proc' of the International Conference on Intelligent Systems, 1996.

[120] Hamming, R. W. *Error Detecting and Error Correcting Codes*. Bell System Technical Journal 29, no. 2, 1950.

[121] Hao, Y., Chen, Y., Zakaria, J., Hu, B., Rakthanmanon, T., and Keogh, E. *Towards Never-Ending Learning from Time Series Streams*. Proc' of the 19th ACM SIGKDD, 2013.

[122] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. *Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases*. KAIS 3, pp. 263-86, 2001.

[123] Keogh, E., and Kasetty, S. *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration*. Proc' of the 8th ACM SIGKDD, 2002.

[124] Keogh, E. The UCR Time Series Classification/ Clustering Page: *www.cs.ucr.edu/~eamonn/time_series_data*

[125] Lan, B., Ooi, B. C., and Tan, K. L. *Efficient Indexing Structures for Mining Frequent Patterns*. Proceedings of the IEEE 18th ICDE, pp. 453-462, 2002.

[126] Lin, J., Keogh, E., Wei, L., and Lonardi, S. *Experiencing SAX: A Novel Symbolic Representation of Time Series*. DMKD vol. 15, no. 2, pp. 107 – 144, 2007.

[127] Lin, J., Keogh, E., Lonardi, S., Patel, P. *Finding Motifs in Time Series*. Proc. of the 2nd Workshop on Temporal Data Mining. 2002.

[128] Makonin, S., et. al. *AMPds: A Public Dataset for Load Disaggregation and Eco-Feedback Research*. EPEC, pp. 1-6. 2013.

[129] Mitzenmacher, M., and Upfal, E. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[130] Mueen, A., et al. *A Disk-aware Algorithm for Time Series Motif Discovery*. Data Mining and Knowledge Discovery, 22.1-2, 2011.

[131] Mueen, A. *Enumeration of Time Series Motifs of All Lengths.* Proc'of the IEEE ICDM, pp. 547-556, 2013.

[132] Mueen, A., and Keogh, E. *Online Discovery and Maintenance of Time Series Motifs.* Proc' of the 16th ACM SIGKDD, 2010.

[133] Mueen, A., Keogh, E., Zhu, Q., Cash, S., and Westover, M. B. *Exact Discovery of Time Series Motifs*. Proc'of the 9th SIAM SDM, pp. 473-484, 2009.

[134] URL *http://www.youtube.com/watch?v=ndL6m5vHVhw*

[135] Pietracaprina, A., Riondato, M., Upfal, E., and Vandin, F. *Mining Top-K Frequent Itemsets Through Progressive Sampling*. DMKD, vol 21, no. 2, 2010.

[136] Smith, J. E., and Goodman, J. R. *Instruction Cache Replacement Policies and Organizations*. IEEE Transactions on Computers, 1985.

[137] Trifa, V., et. al. *Automated Wildlife Monitoring Using Self-Configuring Sensor Networks Deployed in Natural Habitats*, 2007.

[138] Vahdatpour, A., et. al. *Toward Unsupervised Activity Discovery Using Multi-Dimensional Motif Detection in Time Series*. IJCAI. Vol. 9. 2009.

[139] Wang, X., et al. *Experimental Comparison of Representation Methods and Distance Measures for Time Series Data*. DMKD, vol. 26, issue 2, pp. 275 – 309, 2013.

[140] Xeno-canto, Sharing Bird Sounds from Around the World, *www.xeno-canto.org/*, accessed on Feb 11, 2014.

[141] Project website: www.*cs.ucr.edu/~nbegu001/RareMotif.htm*