UNIVERSITY OF CALIFORNIA
RIVERSIDE

Efficient Algorithms for High Dimensional Data Mining

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

Doctor of Philosophy

in

Computer Science

by

Thanawin Rakthanmanon

December 2012

Dissertation Committee:

Dr. Eamonn Keogh, Chairperson
Dr. Stefano Lonardi
Dr. Gianfranco Ciardo

The Dissertation of Thanawin Rakthanmanon is approved:

_____

_____

_____

Committee Chairperson

University of California, Riverside

ACKNOWLEDGEMENTS

I would like to this opportunity to express my sincerest gratitude to my advisor, Dr. Eamonn J. Keogh, for his invaluable guidance, supervision, and generous support during my doctoral study. I am graceful that four years ago he picked up my application from and gave me an invitation letter for studying at University of California at Riverside as one of his graduate students. During my research, he encourages me to solve interesting problems, gives valuable efforts with an insightful advice, and teaches me how to reach the better level of research. It is very fortunate that I have him as my advisor. Thank you very much.

I also would like to thank Dr. Stefano Lonardi and Dr. Gianfranco Ciardo who are my dissertation committees for their valuable comments and suggestions, Dr. Christian Shelton and Dr. Marek Chrobak as my oral-qualification committees and taught me five algorithm classes during my graduate study.

I express my gratitude to my colleagues from our data mining lab, who always help me completing my research: Dr. Gustavo Batista, Dr. Qiang Zhu, Dr. Abdullah Mueen, Dr. Xiaoyue Wang, Bilson Campana, Bing Hu, and Yuan Hao. I really appreciate Bing Hu for having many publications together and a nice memory in Vancouver, Canada, with my family, Dr. Gustavo Batista for deep technical discussion and his professional vision, Yuan Hao and Yoothana Thanmongkhon for helping me delivering my dissertation. I also would like to thank Dr. Surachet Charoenkajonchai who helped me a lot when I first arrived at the United State, Dr. Sira Srinives and Dr. Piyada Juntawong to have some vacations together, and a very long list of Thai people who always gave me some support both physically and mentally.

Moreover, I cannot finish my doctoral program without financial support. Therefore, I would like to thank all of sources of the funding I received here: Thai Government Scholarship, UCR Fellowship, NSF 0803410 and NSF 0808770 for covering all of my tuition fee, travel cost, stipend, and all other stuffs.

Finally, I would like to grateful thank my wonderful wife, Pempeeorn Wangchailert, for being here as my wonderful support and providing constant inspirations, and my 6-month-old daughter, Trin, who is always keeping me and my wife busy and also making this world a better place to live.

ABSTRACT OF THE DISSERTATION

Efficient Algorithms for High Dimensional Data Mining

by

Thanawin Rakthanmanon

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, December 2012
Dr. Eamonn Keogh, Chairperson

Data mining and knowledge discovery has attracted research interest in the last decade. The size and complexity of real world data is dramatically increasing and although new efficient algorithms to deal with such data are constantly being proposed, the mining of high dimensional data still presents a challenge. In this dissertation, several novel algorithms are proposed to handle such datasets. These algorithms are applied to domains as diverse as electrocardiography (ECG), electroencephalography (EEG), human DNA sequencing, protein sequencing, stock market data, gesture recognition data, motion capture data, accelerometer data, audio data, image data, handwritten manuscripts, etc. This dissertation contributes to the data mining community in three ways:

Firstly, we propose a novel algorithm for searching for the nearest neighbor in time series data by using multi-level lower bounding techniques and other speed-up techniques. The proposed algorithm, called *UCRSuite*, is faster than the previous state-of-the-art by several orders of magnitude. Because search algorithms are primitive and a bottleneck in complex data mining algorithms, this contribution is likely to make a significant impact. Secondly, we propose two approximation algorithms to handle the high dimensional data. A fast shapelet discovery algorithm, called *FastShapelet*, has been proposed to discover approximate shapelets, which are as accurate as those found by an exact search. In addition, we show an unsupervised algorithm, called *DocMotif*, which can discover similar figures from given manuscripts. The proposed algorithms are faster than the best known algorithms by two or three orders of magnitude and the

discovered results are not measurably different from the exact algorithm. Moreover, in the second work, a detailed mathematical analysis for bounding an error is provided.

In my final contribution, we show that in order to create a useful clustering of a single time series, an algorithm must have the freedom to ignore some data. We propose a Minimum Description Length based time series clustering algorithm that has this ability. My results demonstrate that not only is the proposed algorithm parameter-free, but it is also efficient and effective for time series clustering.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction to High Dimensional Data Mining

Because data mining has been a hot topic for decades, many algorithms and related applications have been proposed to handle the various types of data such as transactional data, medical data, biological data, multimedia data, streaming data, etc. As a result of the dramatic increase in digital data, efficient algorithms are still needed to handle very large scale and highly complex data. This dissertation proposes several approaches that can handle high dimensional data efficiently and effectively. We demonstrate their usefulness on many real world data sets.

Before we explain further what the challenges of high dimensional data mining are, we would like to point out that high dimensional data is everywhere nowadays and is being created every second.

1.1 Types of High Dimensional Data

We may classify a large portion of high dimensional data as the following.

1.1.1. Time Series Data

Time series data is one of the most well-known types of data. Usually, data is created from multiple sensors to measure real-world measurements in periods of time. For example, the electrocardiogram (ECG), which shows the electrical activity of a subject's heart, is used widely to measure the regularity of heartbeats, the size of the heart and its chambers, and abnormal rhythms; this data can be used to identify the damaged area of the heart or the effects of drugs. In recording the ECG time series, multiple sensors are placed on the subject's heart, hands, and legs. Other examples of time series data are electroencephalography (EEG), which is recorded by attaching 40-100 sensors to the subject's head for recording brain waves; accelerometer data, which is used widely for recording the movement of subjects, etc. If multiple time series are recorded simultaneously, it is called a "multidimensional time series."

In time series data mining, knowledge can be found in many aspects such as repeated pattern or motif discovery, classification, prediction, clustering, data dictionary, data compression, data representation, etc. According to the largest portion of the real-world data, several works in this dissertation focus on solving time series data mining problems.

1.1.2. Streaming Data

Streaming data and time series data are visually similar but conceptually different. Specifically, streaming data may be recorded by multiple sensors at the same as time series data. The main difference is that streaming data could be theoretically recorded forever or for a very long period of time, and thus not all of the data can be stored. The limitations for recording and analyzing the streaming data include small memory/storage and low computational power. Hence, an algorithm for mining streaming data is different and much more difficult than an algorithm for time series data mining because all data can be read only once and only a small portion of the data can be used for analysis. In this dissertation, when we mention streaming data, we mean a stream created from single-dimensional data. However, in general, streaming data can be any kind of data, such as broadcasting audio, streaming video, online transaction, or stock market prices.

1.1.3. Image Data

Another large portion of the digital data consists of images. This kind of data includes not only photos and pictures but also scanned documents and digital books. According to Google 2010, there are more than 130 million books in the world and a portion of them are digitized. Because an image contains a lot of data points, image data analysis is typically complex; scalability of an algorithm is the main concern here. Later in Chapter 4, we show an efficient algorithm for finding similar images from books.

1.1.4. Multimedia Data

Multimedia data is one of the most complex types of data to analyze. It can contain both audio data and video data, so an efficient algorithm is required. Although we do not focus on full multimedia data, audio data is used throughout this dissertation. There are some techniques to convert audio data to a time series. Later in Chapter 5, the Mel-Frequency Cepstrum Coefficients (MFCC) technique is used to create a time series from an audio file.

1.1.5. Sequential Data

Sequential data is primitive and similar to time series data. Time series data contains interval data, such as integers or real numbers. Sequential data can contain any nominal data. The most useful sequences are strings, including binary strings and ASCII texts, biological data including DNA sequences, the human genome, or 3D protein sequences. This kind of data may be one of the most important types of data for mankind and there are more than thousands of algorithms to deal with this kind of data. In this dissertation, we do not focus on mining this kind of data. Later in the next chapter, we show that with an appropriate conversion our proposed methods can support this kind of data efficiently.

1.1.6. Transactional Data

This is the most conservative type of data in the data mining community. This data is varied, consisting of merely single dimensional data, such as items in a shopping cart, or multiple-dimensional data such as customer data, log files, patient records, etc. Transactional data can simply be kept in a relational database management system (RDBMS). Most data mining algorithms can handle this kind of data. However, it may not be appropriate to keep high-dimensional data in an RDBMS because the number of dimensions is possibly larger than the limitations of RDBMS.

1.2 Challenge in high dimensional data mining

The complexity of input data is one of the main challenges in data mining. When data contains more dimensions, the model representing the data is more complicated, the complexity of corresponding algorithms is higher, and the usefulness of the results is degraded. In this section, we discuss the well-known challenges of high dimensional data analysis.

1.2.1. Curse of Dimensionality

When data contains many dimensions such as hundreds or thousands of dimensions, the problem is that the entire space greatly increases and the data becomes sparse inside this huge space [2]. Thus, it is hard to measure the distance between any two data in a useful way because most data are equally far apart from others. Although dissimilar pairs of high dimensional data seem to be useless, the similar pairs are in fact useful, especially in time series data mining [1][3]. There are many approaches that can be used to mitigate this problem, such as feature selection and dimension reduction. The latter approach is used in this dissertation, and we explain more about our methods in Chapters 3 and 4.

1.2.2. Noisy Data

If the data is too noisy, it may be impossible to find a useful pattern inside the data. This depends on the signal-to-noise ratio [4], the ratio between the desired signal and background noise. If the ratio is too low, it means that the noise is significant and may dominate the whole combined data, so any model used to explain the data may be useless because it mainly explains the noise instead of the real or desired signal/data. This idea can be applied to other domains; for example, noise can consist of spam emails, scam websites, background sounds, background colors, noisy signals created by a circuit, the natural frequency of light (50 Hz), etc.

There are several approaches to handling noisy data. The most convenient approaches are smoothing the data to reduce the noise's effects, or assuming the pattern of the noise such as Gaussian, or reducing the cardinality of the data, or converting data into other space, or even learning the noise pattern and removing

it from the data. For example, raw audio data recorded at a frequency of 44.1 MHz is one of the noisiest data and the MFCC technique is a well-known technique used to convert the audio data into simple time series data with less frequency.

1.2.3. High Complexity

The main challenge in mining high dimensional data is the data complexity itself. For the most algorithms, their The complexity of most algorithms will be grows dramatically on according to the number of dimensions. For example, one of the most primitive problems in data mining is to finding the closest pairs in the given data set. The closest pair problem can be solved effectively in a Euclidean Plane or when the dimension is 2 [5] in O(*nlogn*), where n is the number of data. However, the complexity of this problem is grows exponentially based on the number of dimensions and approximation algorithms that are proposed to solve this problem with high dimensional data [6]. Hence, to deal with high-dimensional data, efficient algorithms are necessary; this and it is the main focus of this dissertation. However, most of the algorithms proposed in this dissertation are efficient, effective, and scalable.

1.3 The Proposed Solutions

In this study, we propose several algorithms to solve some problems associated with high dimensional data mining. The proposed algorithms can be grouped into three categories. First, the lower bounding technique can be used to prune some data from further consideration. Second, we propose some approximation algorithms to solve the high dimensional data mining problems. Last but not least, we demonstrate that trying to explain everything in high dimensional data is not appropriate. Some data must be ignored; otherwise, noisy data will be explained or modeled.

1.3.1. Speeding up by Lower Bounding

The lower bounding technique is a well-known technique used to speed up a search algorithm. The mechanism for using lower bounding is simple, as instead of computing costly distance measures, such as

Dynamic Time Warping (DTW), the cheaper lower bound, such as LB_Kim, LB_Yi, or LB_Keogh, can be calculated first. If the lower bounding distance is still farther than the best current search result, there is no need to calculate the costly distance for those corresponding pairs, because the real distance must be larger than its lower bounding. In Chapter 2, we show that our novel search algorithm under DTW distance can be used to find the nearest neighbor subsequences among a trillion subsequences in less than one and a half day. Moreover, our exact search is much faster than the previous state-of-the-art algorithms, including the best approximation algorithms for solving the same problem.

1.3.2. Returning Approximate Results

As mentioned above, some problems such as the closest pair problem cannot be solved in polynomial time, and approximation algorithms have been introduced [6]. In Chapter 3, we propose an approximation algorithm to find an approximate time series shapelet. The proposed algorithm is faster than the state-of-the-art exact algorithm by a few orders of magnitude and the experiment results show that the proposed algorithm is indifferent in term of accuracy. We tested all of the time series datasets in the UCR Time Series Archives. To learn more about time series shapelets, please refer to Chapter 3.

In Chapter 4, we do mining on image datasets. We propose an approximation algorithm for finding repeated patterns from two books, which can be historical manuscripts, hand-written documents, or a set of images. The proposed algorithm is much faster than the best exact algorithm by many orders of magnitude. We demonstrate that the results from our algorithm are visually correct, and a mathematical analysis is provided in the Appendix to show that the error of the proposed algorithm is small and can be bounded.

1.3.3. Trying to Explain Everything Is Wrong

In Chapter 5, we show that it is not appropriate to explain all of the data, especially for time series data. Because having noise in high dimensional data is unavoidable, explaining everything means explaining both the desired data and noise. However, noise is unknown, cannot be explained, and must be

excluded from the model. We demonstrate the problem using time series subsequence clustering problems. In 2005, Keogh and Lin [8] first showed that considering all subsequences will create a useless result for any clustering algorithm. In Chapter 5, we propose a minimum description length (MDL)-based algorithm to cluster the time series subsequences. Because it is hard even for an expert to understand the nature of the data, we apply the MDL technique to help us find the results. Moreover, our algorithm is parameter-free.

The organization of the rest of this dissertation is as follows. As mentioned above, Chapter 2 explains the exact search algorithm; we show that if we have a good enough strategy and many lower bounding techniques, we can speed up the search by many orders of magnitude. In Chapter 3, we show an approximation algorithm to discover time series shapelets by using symbolic representation and hashing techniques to avoid costly calculations. In Chapter 4, we explain an approximate algorithm to find repeated patterns from two digital books. A random projection has been used to achieve a fast approximation algorithm but can guarantee an error. The mathematical proof of the bound of the error and the running time is provided in the Appendix. Chapter 5 shows an algorithm to cluster time series subsequences. The proposed algorithm not only creates meaningful results, but is also parameter-free. The last chapter concludes all of the results from this study.

Chapter 2: Exact Search in Trillions Subsequences

Most time series data mining algorithms use similarity search as a core subroutine, and thus the time taken for similarity search is *the* bottleneck for virtually all time series data mining algorithms. The difficulty of scaling search to large datasets largely explains why most academic work on time series data mining has plateaued at considering a few millions of time series objects, while much of industry and science sits on billions of time series objects waiting to be explored. In this work we show that by using a combination of four novel ideas we can search and mine truly massive time series for the first time. We demonstrate the following extremely unintuitive fact; in large datasets we can exactly search under DTW much more quickly than the current state-of-the-art *Euclidean distance* search algorithms. We demonstrate our work on the largest set of time series experiments ever attempted. In particular, the largest dataset we consider is larger than the combined size of all of the time series datasets considered in all data mining papers ever published. We show that our ideas allow us to solve higher-level time series data mining problem such as motif discovery and clustering at scales that would otherwise be untenable. In addition to mining massive datasets, we will show that our ideas also have implications for real-time monitoring of data streams, allowing us to handle much faster arrival rates and/or use cheaper and lower powered devices than are currently possible.

2.1 Introduction

Time series data is pervasive across almost all human endeavors, including medicine, finance, science and entertainment. As such, it is hardly surprising that time series data mining has attracted significant attention and research effort. Most time series data mining algorithms require similarity comparisons as a subroutine, and in spite of the consideration of dozens of alternatives, there is increasing evidence that the classic Dynamic Time Warping (DTW) measure is the best measure in most domains [13].

It is difficult to overstate the ubiquity of DTW. It has been used in robotics, medicine [12], biometrics, music/speech processing [8][34][48], climatology, aviation, gesture recognition [10][45], user interfaces [23][29][36][45], industrial processing, cryptanalysis [14], mining of historical manuscripts [22], geology, astronomy [27][38], space exploration, wildlife monitoring, etc.

As ubiquitous as DTW is, we believe that there are thousands of research efforts that would like to use DTW, but find it too computationally expensive. For example, consider the following: "Ideally, dynamic time warping would be used to achieve this, but due to time constraints…" [12]. Likewise, [10] bemoans DTW is "still too slow for gesture recognition systems", and [1] notes, even "a 30 fold speed increase may not be sufficient for scaling DTW methods to truly massive databases." As we shall show, our subsequence search suite of four novel ideas (called the UCR suite) removes all of these objections. We can reproduce all the experiments in all these papers in well under a second.

We make an additional claim for our UCR suite which is almost certainly true, but hard to prove, given the variability in how search results are presented in the literature. We believe our exact DTW sequential search is much faster than any current approximate search or exact indexed search. In a handful of papers the authors are explicit enough with their experiments to see this is true. Consider [35], which says it can answer queries of length 1,000 under DTW with 95% accuracy, in a random walk dataset of one million objects in 5.65 seconds. We can exactly search this dataset in 3.8 seconds (on a very similar machine). Likewise, a recent paper that introduced a novel inner product based DTW lower bound greatly speeds up exact subsequence search for a wordspotting task in speech. The authors state: "the new DTW-KNN method takes approximately 2 minutes" [48]; however, we can reproduce their results in less than a second. An influential paper on gesture recognition on multi-touch screens laments that "DTW took 128.26 minutes to run the 14,400 tests for a given subject's 160 gestures" [45]. However, we can reproduce these results in under 3 seconds.

2.1.1. A Brief Discussion of a Trillion

Since we use the word "trillion" in this work and to our knowledge, it has never appeared in a data mining/database paper; we will take the time to briefly discuss this number. By a trillion, we mean the short scale version of the word [21], one million million, or 1012, or 1,000,000,000,000.

If we have a single time series T of length one trillion, and we assume it takes eight bytes to store each value, it will require 7.2 terabytes to store. If we sample a electrocardiogram at 256 Hz, a trillion data points would allow us to record 123 years of data, every single heartbeat of the longest lived human [44].

A time series of length one trillion is a very large data object. In fact, it is more than all of the time series data considered in all papers ever published in all data mining conferences combined. This is easy to see with a quick back-of-the-envelope calculation. Up to 2011 there have been 1,709 KDD/SIGKDD papers (including industrial papers, posters, tutorial/keynote abstracts, etc. [16]). If every such paper was on time series, and each had looked at five hundred million objects, this would still not add up to the size of the data we consider here). However, the largest time series data considered in a SIGKDD paper was a "mere" one hundred million objects [42].

As large as a trillion is, there are thousands of research labs and commercial enterprises that have this much data. For example, many research hospitals have trillions of data points of EEG data, NASA Ames has tens of trillions of data points of telemetry of domestic flights, the Tennessee Valley Authority (a power company) records a trillion data points every four months, etc.

2.1.2. Explicit Statement of our Assumptions

Our work is predicated on several assumptions that we will now enumerate and justify.

A. Time Series Subsequences must be Normalized

In order to make meaningful comparisons between two time series, both must be normalized. While this may seem intuitive, and was explicitly empirically demonstrated a decade ago in a widely cited paper [26], many research efforts do not seem to realize this. This is critical because some speedup techniques

only work on the un-normalized data; thus**,** the contributions of these research efforts may be largely nullified [15][35].

To make this clearer, let us consider the classic Gun/NoGun classification problem which has been in the public domain for nearly a decade. The data, which as shown in Figure 1.*center* is extracted from a video sequence, *was* Z-normalized. The problem has a 50/150 train/test split and a DTW one-nearest-neighbor classifier achieves an error rate of 0.087.

Suppose the data had *not* been normalized. As shown in Figure 1.*left* and Figure 1.*right*, we can simulate this by adding a tiny amount of scaling/offset to the original video. In the first case we randomly change the *offset* of each time series by ± 10%, and in the second case we randomly change the *scale* (amplitude) by ± 10%. The new one-nearest-neighbor classifier error rates, averaged over 1,000 runs, are 0.326 and 0.193, respectively, significantly worse than the normalized case.

Figure 1: Screen captures from the original video from which the Gun/NoGun data was culled. The center frame is the original size; the left and right frames have been scaled by 110% and 90% respectively. While these changes are barely perceptible, they double the error rate if normalization is not used. (Video courtesy of Dr. Ratanamahatana)

It is important to recognize that these tiny changes we made are completely dwarfed by changes we might expect to see in a real world deployment. The apparent *scale* can be changed by the camera zooming, by the actor standing a little closer to the camera, or by an actor of a different height. The apparent *offset* can be changed by this much by the camera tilt angle, or even by the actor wearing different shoes.

While we did this experiment on a visually intuitive example, all forty-five datasets in the UCR archive increase their error rate by at least 50% if we vary the offset and scale by just ± 5%.

It is critical to avoid a common misunderstanding. We must normalize *each* subsequence before making a comparison, it is not sufficient to normalize the entire dataset.

B. *Dynamic Time Warping Is the Best Measure*

It has been suggested many times in the literature that the problem of time series data mining scalability is only due to DTW's oft-touted lethargy, and that we could solve this problem by using some other distance measure. As we shall later show, this is not the case. In fact, as we shall demonstrate, our optimized DTW search is much faster than all current *Euclidean* distance searches. Nevertheless, the question remains, is DTW the right measure to speed up? Dozens of alternative measures have been suggested. However, recent empirical evidence strongly suggests that none of these alternatives routinely beats DTW. When put to the test on a collection of forty datasets, the very *best* of these measures are sometimes a little better than DTW and sometimes a little worse [13]. In general, the results are consistent with these measures being minor variants or "flavors" of DTW (although they are not typically presented this way). In summary, after an exhaustive literature search of more than 800 papers [13], we are not aware of any distance measure that has been shown to outperform DTW by a statistically significant amount on reproducible experiments [13][26]. Thus, DTW is *the* measure to optimize (recall that DTW subsumes Euclidean distance as a special case).

C. *Arbitrary Query Lengths Cannot Be Indexed*

If we know the length of queries ahead of time we can mitigate at least some of the intractability of search by indexing the data [9][18][42]. Although to our knowledge no one has built an index for a trillion real-valued objects (Google only indexed a trillion webpages as recently as 2008), perhaps this could be done.

However, what if we do not know the length of the queries in advance? At least two groups have suggested techniques to index arbitrary length queries [25][30]. Both methods essentially build multiple indexes of various lengths, and at query time search the shorter and longer indexes, "interpolating" the results to produce the nearest neighbor produced by a virtual index of the correct length. This is an interesting idea, but it is hard to imagine it is the answer to our problem. Suppose we want to support

queries in the range of, say, 16 to 4096. We must build indexes that are not too different in size, say MULTINDEX-LENGTHS = {16, 32, 64, .., 1024, 2048, 4096}[1]. However, for time series data the index is typically about one-tenth the size of the data [13][25]. Thus, we have doubled the amount of disk space we need. Moreover, if we are interested in tackling a trillion data objects we clearly cannot fit *any* index in the main memory, much less all of them, or any two of them.

There is an underappreciated reason why this problem is so hard; it is an implication of the need for normalization discussed above. Suppose we have a query Q of length 65, and an index that supports queries of length 64. We search the index for $Q_{[1:64]}$ and find that the best match for it has a distance of, say, 5.17. What can we say about the best match for the full Q? The answer is surprisingly little: 5.17 is neither an upper bound nor a lower bound to the best match for Q. This is because we must renormalize the subsequence when moving from $Q_{[1:64]}$ to the full Q. If we do not normalize any data, the results are meaningless (cf. Section 2.1.2.C.), and the idea *might* be faster than sequential search. However, if we normalize the data we get so little information from indexes of the wrong length that we are no better off than sequential search.

In summary, there are no known techniques to support similarity search of arbitrary lengths once we have datasets in the billions.

D. There Exists Data Mining Problems That We are Willing to Wait Some Hours to Answer

This point is almost self-evident. If a team of entomologists has spent three years gathering 0.2 trillion data points [42], or astronomers have spent billions dollars to launch a satellite to collect one trillion data points of star-light curve data per day [27], or a hospital charges $34,000 for a daylong EEG session to collect 0.3 trillion data points (cf. Section 2.4.2) [33], then it is not unreasonable to expect that these groups would be willing to spend hours of CPU time to glean knowledge from their data.

---

[1] This collection of sizes is very optimistic. The step size should be at most 100, creating two orders of magnitude space overhead.

2.1.3. Related Work

Our review of related work on time series indexing is necessarily superficial, given the vast amount of work on the topic and page limits. Instead, we refer the interested reader to two recent papers [13][35], which have comprehensive reviews of existing work. It has now become common (although not yet *routine*) to see papers indexing/mining datasets with millions of objects. For example, Jegou et al. have demonstrated very fast approximate main memory search of 10 million images [24]. However, this work and much of the current work that addresses multi-million object datasets focus on *approximate* search, whereas we are only considering *exact* search here. Moreover, we are interested in datasets, which are five to six orders of magnitude larger than anything else considered in the literature [13]. Thus, comparisons to related work are very difficult to do meaningfully.

2.2 Background and Notations

2.2.1. Definitions and Notations

We begin by defining the data type of interest, time series:

Definition 1 A Time Series T is an ordered list: $T = t_1,t_2,...,t_m$.

While the source data is one long time series, we ultimately wish to compare it to shorter regions called *subsequences*:

Definition 2 A *subsequence* $T_{i,k}$ of a time series T is a shorter time series of length k which starts from position i. Formally, $T_{i,k} = t_i,t_{i+1},..,t_{i+k-1}$, $1 \leq i \leq m-k+1$.

Where there is no ambiguity, we may refer to subsequence $T_{i,k}$ as C, as in a *C*andidate match to a query Q . We denote $|Q|$ as n.

Definition 3 The Euclidean distance (ED) between Q and C, where $|Q| = |C|$, is defined as:

$$ED(Q,C) = \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2}$$

We illustrate these definitions in Figure 2.

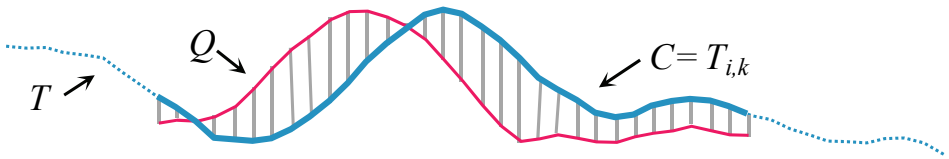

Figure 2: A long time series T can have a subsequence $T_{i,k}$ extracted and compared to a query Q under the Euclidean distance, which is simply the square root of the sum of the squared hatch line lengths

The Euclidean distance, which is a one-to-one mapping of the two sequences, can be seen as a special case of DTW, which allows a one-to-many alignment, as illustrated in Figure 3.

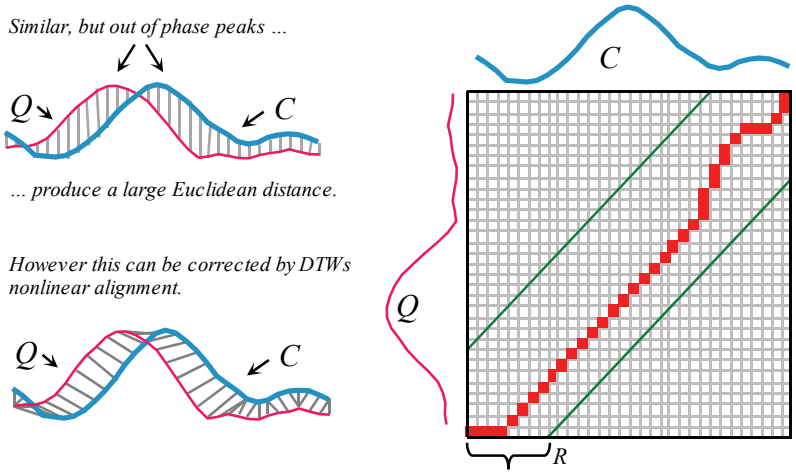

Figure 3: *left*) Two time series which are similar but out of phase. *right*) To align the sequences we construct a warping matrix, and search for the optimal warping path (red/solid squares). Note that Sakoe-Chiba Band with width R is used to constrain the warping path

To align two sequences using DTW, an n-by-n matrix is constructed, with the (i^{th}, j^{th}) element of the matrix being the Euclidean distance $d(q_i, c_j)$ between the points q_i and c_j. A warping path P is a contiguous set of matrix elements that defines a mapping between Q and C. The t^{th} element of P is defined as $p_t = (i, j)_t$ so we have:

$$P = p_1, p_2, \ldots, p_t, \ldots, p_T \quad n \le T \le 2n\text{-}1$$

The warping path that defines the alignment between the two time series is subject to several constraints. For example, the warping path must start and finish in diagonally opposite corner cells of the matrix, the steps in the warping path are restricted to adjacent cells, and the points in the warping path must be monotonically spaced in time. In addition, virtually all practitioners using DTW also constrain the

warping path in a global sense by limiting how far it may stray from the diagonal [13][35]. A typical constraint is the Sakoe-Chiba Band which states that the warping path cannot deviate more than R cells from the diagonal [13][35][39].

2.3 Proposed Algorithms

2.3.1. Known Optimizations

We begin by discussing previously known optimizations of sequential search under ED and/or DTW.

A. Using the Squared Distance

Both DTW and ED have a square root calculation. However, if we omit this step, it does not change the relative rankings of nearest neighbors, since both functions are monotonic and concave. Moreover, the absence of the square root function will make later optimizations possible and easier to explain. Note that this is only an internal change in the code; the user can still issue range queries with the original units, as the code simply internally squares the desired value, does the search, and after finding the qualifying objects, takes the square root of the distances for the qualifying objects and presents the answers to the user.

Where there is no ambiguity below, we will still use 'DTW' and 'ED'; however, the reader may assume we mean the squared versions of them.

B. Lower Bounding

A classic trick to speed up sequential search with an expensive distance measure such as DTW is to use a cheap-to-compute lower bound to prune off unpromising candidates [13][27]. Figure 4 shows two such lower bounds, one of which we have modified.

The original definition of LB_Kim also uses the distances between the maximum values from both time series and the minimum values between both time series in the lower bound, making it $O(n)$. However, for *normalized* time series these two extra values tend to be tiny and it does not pay to compute them, and

ignoring them allows the bound to be $O(1)$, a fact we will exploit below. The LB_Keogh bound is well-documented elsewhere, for brevity we ask the unfamiliar reader to refer to [18][27][13] for a review.

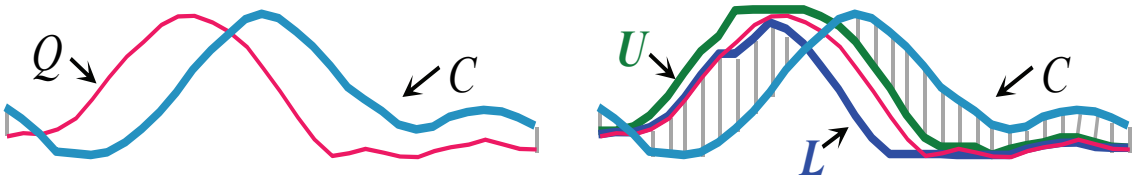

Figure 4: *left*) The LB_KimFL lower bound is $O(1)$ and uses the distances between the **F**irst (**L**ast) pair of points from C and Q as a lower bound. It is a simplification of the original LB_Kim [28]. *right*) The LB_Keogh lower bound is $O(n)$ and uses the Euclidean distance between the candidate sequence C and the closer of $\{U,L\}$ as a lower bound

C. *Early Abandoning of ED and LB_Keogh*

During the computation of the Euclidean distance or the LB_Keogh lower bound, if we note that the current sum of the squared differences between each pair of corresponding data points exceeds the *best-so-far*, then we can stop the calculation, secure in the knowledge that the exact distance or lower bound, had we calculated it, would have exceeded the *best-so-far*, as in Figure 5.

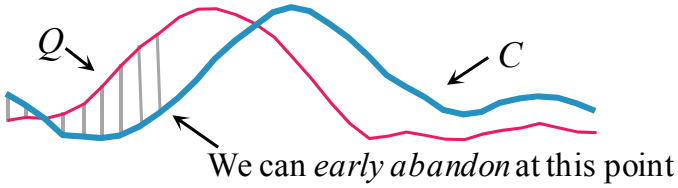

Figure 5: An illustration of ED *early abandoning*. We have a *best-so-far* value of b. After incrementally summing the first nine (of thirty-two) individual contributions to the ED we have exceeded b, thus it is pointless to continue the calculation [27]

D. *Early Abandoning of DTW*

If we have computed a full LB_Keogh lower bound, but we find that we must calculate the full DTW, there is still one trick left up our sleeves. We can incrementally compute the DTW from left to right, and as we incrementally calculate from 1 to K, we can sum the *partial* DTW accumulation with the LB_Keogh contribution from K+1 to n. Figure 6 illustrates this idea.

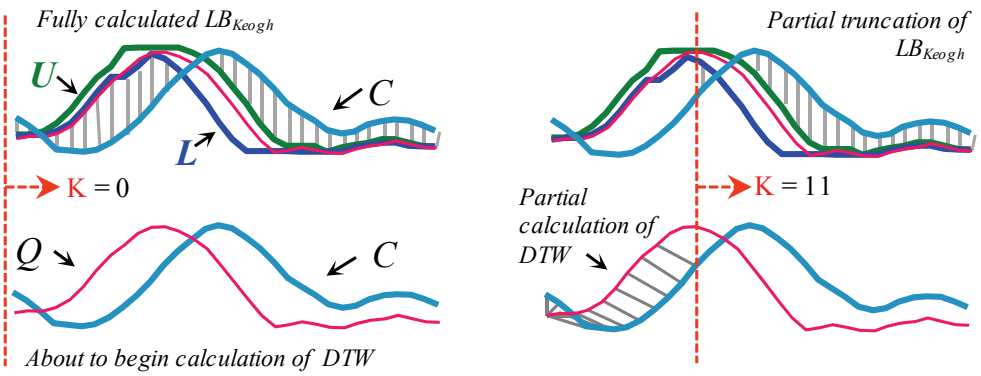

Figure 6: *left*) At the top we see a completed LB_Keogh calculation, and below it we are about to begin a full DTW calculation. *right*) We can imagine the orange/dashed line moving from left to right. If we sum the LB_Keogh contribution from the right of dashed line (*top*) and the partial (incrementally calculated) DTW contribution from the left side of the dashed line (*bottom*), this is will be a lower bound to DTW(Q,C)

This sum of DTW($Q_{1:K}$,$C_{1:K}$) + LB_Keogh($Q_{K+1:n}$,$C_{K+1:n}$) is a lower bound to the true DTW distance (i.e., DTW($Q_{1:n}$,$C_{1:n}$)). Moreover, with careful implementation the overhead costs are negligible. If at any time this lower bound exceeds the *best-so-far* distance, we can admissibly stop the calculation and prune this C.

E. Exploiting Multicores

It is important to note that while we can get essentially linear speedup using multicores, the *software* improvements we will present in the next section completely dwarf the improvements gained by multicores. As a concrete example, a recent paper shows that a search of a time series of length 421,322 under DTW takes "*3 hours and 2 minutes on a single core. The* (8-core version) *was able to complete the computation in 23 minutes*" [41]. However, using our ideas, we can search a dataset of this size in just under one second on a single core. Nevertheless, as it is simple to port to the now ubiquitous multicores, we consider them below.

2.3.2. Novel Optimizations: The UCR Suite

We are finally in a position to introduce our four original optimizations of search under ED and/or DTW.

A. *Early Abandoning Z-Normalization*

To the best of our knowledge, no one has ever considered optimizing the *normalization* step. This is surprising, since it takes slightly longer than computing the Euclidean distance itself. Our insight here is that we can interleave the early abandoning calculations of Euclidean distance (or LB_Keogh) with the online Z-normalization. In other words, as we are incrementally computing the Z-normalization, we can also incrementally compute the Euclidean distance (or LB_Keogh) of the same data point. Thus, if we can early abandon, we are pruning not just distance calculation steps as in Section 2.3.1.C. , but also *normalization* steps.

Recall that the mean and standard deviation of a sample can be computed from the sums of the values and their squares. Therefore, it takes only one scan through the sample to compute the mean and standard deviation, using the equations below.

$$\mu = \frac{1}{m}\sum x_i \qquad \sigma^2 = \frac{1}{m}\sum x_i^2 - \mu^2$$

In similarity search, every subsequence needs to be normalized before it is compared to the query (cf. Section 2.1.2.A.). The mean of the subsequence can be obtained by keeping two running sums of the long time series, which have a lag of exactly m values. The sum of squares of the subsequence can be similarly computed. The formulas are given below for clarity.

$$\mu = \frac{1}{m}\left(\sum_{i=1}^{k} x_i - \sum_{i=1}^{k-m} x_i\right) \qquad \sigma^2 = \frac{1}{m}\left(\sum_{i=1}^{k} x_i^2 - \sum_{i=1}^{k-m} x_i^2\right) - \mu^2$$

The high-level outline of the algorithm is presented in Table 1. In the algorithm, we use a circular buffer (X) to store the current subsequence being compared with the query Q. Note the online normalization in line 11 of the algorithm, which allows the early abandoning of the distance computation in addition to the normalization.

Table 1: Subsequence search with online Z-normalization

Algorithm	Similarity Search
Procedure	[*nn*] = *SimilaritySearch* (T,*Q*)
1	*best-so-far* ← ∞, *count*← 0
2	*Q* ← *z-normalize*(*Q*)
3	**while** !*next*(T)
4	*i*← *mod*(*count*,*m*)
5	X[*i*] ←*next*(T)
6	*ex*← *ex*+X[*i*], *ex*2← *ex*2+X[*i*]^2
7	**if** *count* ≥ *m*-1
8	μ← *ex*/*m*, σ← *sqrt*(*ex2*/*m* - μ^2)
9	*j*← 0, *dist*← 0
10	**while** *j* < *m* **and** *dist* < *best-so-far*
11	*dist*← *dist* + (*Q*[*j*]-(X[*mod*(*i*+1+*j*,*m*)]-μ)/σ)^2
12	*j*← *j*+1
13	**if** *dist* < *best-so-far*
14	*best-so-far* ←*dist*, *nn*← *count*
15	*ex*← *ex*-X[*mod*(*i*+1,*m*)]
16	*ex*2← *ex*2-X[*mod*(*i*+1,*m*)]^2
17	*count* ← *count*+1

One potential problem of this method of maintaining the statistics is the accumulation of the floating-point error [20]. The effect of such error accumulation is more profound if all of the numbers are positive, as in our case with sum of squares. With the "mere" millions of data points the rest of the community has dealt with this effect is negligible, however when dealing with billions of data points it *will* affect the answer. Our simple solution is that once every one million subsequences, we force a complete Z-normalization to "flush out" any accumulated error.

B. Reordering Early Abandoning

In the previous section, we saw that the idea of early abandoning discussed in Section 2.3.1.C. can be generalized to the Z-normalization step. In both cases, we assumed that we incrementally compute the distance/normalization from left to right. Is there a better ordering?

Consider Figure 7.*left*, which shows the normal left-to-right ordering in which the early abandoning calculation proceeds. In this case *nine* of the thirty-two calculations were performed before the accumulated distance exceeded *b* and we could abandon. In contrast, Figure 7.*right* uses a different ordering and was able to abandon earlier, with just *five* of the thirty-two calculations.

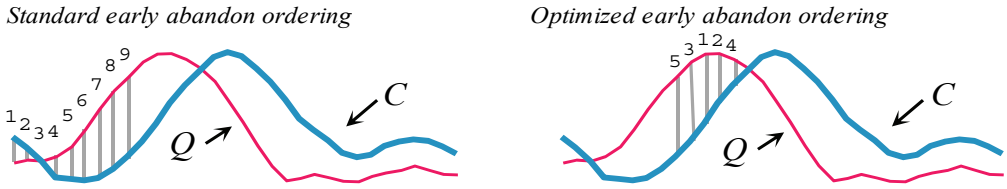

Figure 7: left) ED early abandoning. We have a best-so-far value of b. After incrementally summing the first nine individual contributions to the ED, we have exceeded b; thus, we abandon the calculation. right) A different ordering allows us to abandon after just five calculations

This example shows what is obvious: on a query-by-query basis, different orderings produce different speedups. However, we want to know if there is a *universal* optimal ordering that we can compute *in advance*. This seems like a difficult question because there are $n!$ possible orderings to consider.

We conjecture that the universal optimal ordering is to sort the indices based on the absolute values of the Z-normalized Q. The intuition behind this idea is that the value at Q_i will be compared to many C_i's during a search. However, for subsequence search, with Z-normalized candidates, the distribution of many C_i's will be Gaussian, with a mean of zero. Thus, the sections of the query that are farthest from the mean, zero, will *on average* have the largest contributions to the distance measure.

To see if our conjecture is true we took the heartbeat discussed in Section 2.4.4 and computed its full Euclidean distance to a million other randomly chosen ECG sequences. With the conceit of hindsight we computed what the best ordering *would* have been. For this we simply take each C_i and sort them, largest first, by their sum of their contributions to the Euclidean distance. We compared this *empirically* optimal ordering with our predicted ordering (sorting the indices on the absolute values of Q) and found the rank correlation is 0.999. Note that we can use this trick for both ED and LB_Keogh, and we can use it in conjunction with the early abandoning Z-normalization technique (Section A.).

C. *Reversing the Query/Data Role in LB_Keogh*

Normally the LB_Keogh lower bound discussed in Section 2.3.1.B. builds the envelope around the *query*, a situation we denote $LB\_KeoghEQ$ for concreteness, and illustrate in Figure 8.*left*. This only needs to be done once, and thus saves the time and space overhead that we would need if we built the envelope around each *candidate* instead, a situation we denote $LB\_KeoghEC$.

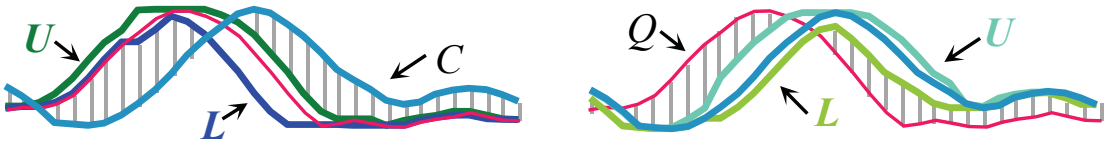

Figure 8: *left*) Normally the $LB\__{Keogh}$ envelope is built around the query (see also Figure 4.*right*), and the distance between *C* and the closer of {*U*,*L*} acts as a lower bound. *right*) However, we can reverse the roles such that the envelope is built around *C* and the distance between *Q* and the closer of {*U*,*L*} is the lower bound

However, as we show in the next section, we can selectively calculate $LB\__{Keogh}EC$ in a "just-in-time" fashion, *only* if all other lower bounds fail to prune. This removes *space* overhead, and as we will see, the *time* overhead pays for itself by pruning more full DTW calculations. Note that in general, $LB\__{Keogh}EQ \neq LB\__{Keogh}EC$ and that on average each one is larger about half the time.

D. *Cascading Lower Bounds*

One of the most useful ways to speed up time series similarity search is the use of lower bounds to admissibly prune off unpromising candidates [13][18]. This has led to a flurry of research on lower bounds, with at least eighteen proposed for DTW [1][13][27][28][40][47][48][49]. In general, it is difficult to state definitively which is the best bound to use, since there is a tradeoff between the tightness of the lower bound and how fast it is to compute. Moreover, different datasets and even different queries can produce slightly different results. However, as a starting point, we implemented all published lower bounds and tested them on fifty different datasets from the UCR archive, plotting the (slightly idealized for visual clarity) results in Figure 9. Following the literature [27], we measured the *tightness* of each lower bound as *LB*(A,B)/*DTW*(A,B) over 100,000 randomly sampled subsequences A and B of length 256.

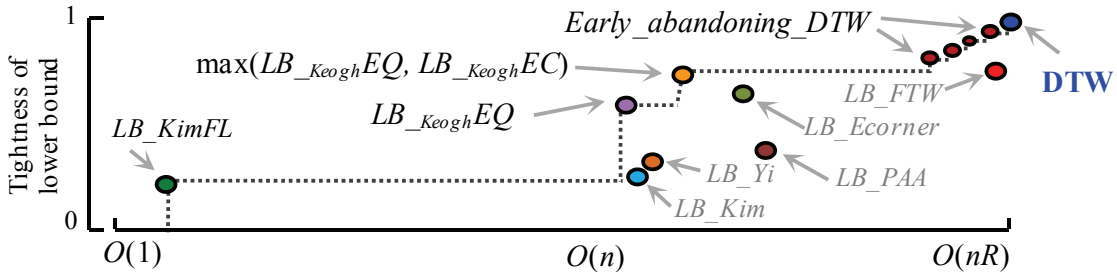

Figure 9: The mean tightness of selected lower bounds from the literature plotted against the time taken to compute them

The reader will appreciate that a *necessary* condition for a lower bound to be useful is for it to appear on the "skyline" shown with a dashed line; otherwise there exists a faster-to-compute bound that is at least as tight, and we should use that instead. Note that the early abandoning DTW discussed in Section 2.3.1.D. is a special case in that it produces a spectrum of bounds, as at every stage of computation it is incrementally computing the DTW until the last computation gives the final true DTW distance.

Which of the lower bounds on the skyline should we use? Our insight is that we should use *all* of them in a cascade. We first use the $O(1)$ $LB\_{Kim}FL$, which while a very weak lower bound prunes many objects. If a candidate is not pruned at this stage we compute the $LB\_{Keogh}EQ$. Note that as discussed in Sections 2.3.1.C. , 2.3.2.A. and 2.3.2.B. , we can incrementally compute this; thus, we may be able to abandon anywhere between $O(1)$ and $O(n)$ time. If we complete this lower bound without exceeding the *best-so-far*, we reverse the query/data role and compute $LB\_{Keogh}EC$ (cf. Section C.). If this bound does not allow us to prune, we then start the early abandoning calculation of DTW (cf. Section 2.3.1.D.).

Space limits preclude detailed analysis of which lower bounds prune how many candidates. Moreover, the ratios depend on the query, data and size of the dataset. However, we note the following: Detailed analysis *is* available at [50], lesion studies tell us that *all* bounds do contribute to speedup; removing any lower bound makes search at least twice as slow; and finally, using this technique we can prune more than 99.9999% of DTW calculations for a large-scale search.

2.4 Experimental Results

We begin by noting that we have taken extraordinary measures to ensure our experiments are reproducible. In particular, all data and code will be available in perpetuity, archived at [50]. Moreover, the site contains several videos, which visualize some of the experiments in real time. We consider the following methods:

- **Naive**: Each subsequence is Z-normalized from scratch. The full Euclidean distance or the DTW is used at each step. Approximately 2/3 of the papers in the literature do (some minor variant of) this.

- **State-of-the-art (SOTA)**: Each sequence is Z-normalized from scratch, early abandoning is used, and the LB_{Keogh} lower bound is used for DTW. Approximately 1/3 of the papers in the literature do (some minor variant of) this.
- **UCR Suite**: We use all of our applicable speedup techniques.

DTW uses $R = 5\%$ unless otherwise noted. For experiments where Naive or SOTA takes more than 24 hours to finish, we terminate the experiments and present the interpolated values, shown in gray. Where appropriate we also compare to an oracle algorithm:

- **GOd's ALgorithm (GOAL)**: An algorithm only maintains the mean and standard deviation using the online O(1) incremental calculations.

It is easy to see that, short of an algorithm that precomputes and stores a *massive* amount of data (quadratic in m), GOAL is a lower bound on the fastest possible algorithm for either ED or DTW subsequence search with unconstrained and unknown length queries. The acronym reminds us that we would like to be as close to this *goal* value as possible.

It is critical to note that our implementations of Naive, SOTA and GOAL are incredibly efficient and tightly optimized, and they are not "crippled" in any way. For example, had we wanted to claim spurious speedup, we could implement SOTA recursively rather than iteratively, and that would make SOTA at least an order of magnitude slower. In particular, the code for Naive, SOTA and GOAL is exactly the same code as the UCR suite, except the relevant speedup techniques have been commented out.

While very detailed spreadsheets of all of our results are archived in perpetuity at [50], we present subsets of our results below. We consider wall clock time on a 2 Intel Xeon Quad-Core E5620 2.40 GHz with 12GB 1333MHz DDR3 ECC Unbuffered RAM (using just one core unless otherwise explicitly stated).

2.4.1. Baseline Tests on Random Walk

We begin with experiments on random walk data. Random walks model financial data very well and are often used to test similarity search schemes. More importantly for us, they allow us to do reproducible

experiments on massive datasets without the need to ship large hard drives to interested parties. We have simply archived the random number generator and the seeds used. We *have* made sure to use a very high quality random number generator that has a period longer than the longest dataset we consider. In Table 2, we show the length of time it takes to search increasingly large datasets with queries of length 128. The numbers are averaged over 1000, 100 and 10 queries, respectively.

Table 2: Time taken to search a random walk dataset with $|Q|$ =128

	Million (*Seconds*)	Billion (*Minutes*)	Trillion (*Hours*)
UCR-ED	0.034	0.22	3.16
SOTA-ED	0.243	2.40	39.80
UCR-DTW	0.159	1.83	34.09
SOTA-DTW	2.447	38.14	472.80

These results show a significant difference between SOTA and UCR suite. However, this is for a very short query; what happens if we consider longer queries? As we show in Figure 10, the ratio of SOTA-DTW over UCR-DTW *improves* for longer queries.

To reduce visual clutter we have only placed one Euclidean distance value on the figure, for queries of length 4,096. Remarkably, UCR-DTW is even faster than SOTA *Euclidean* distance. As we shall see in our EEG and DNA examples below, even though 4,096 is longer than any published query lengths in the literature, there is a need for even *longer* queries.

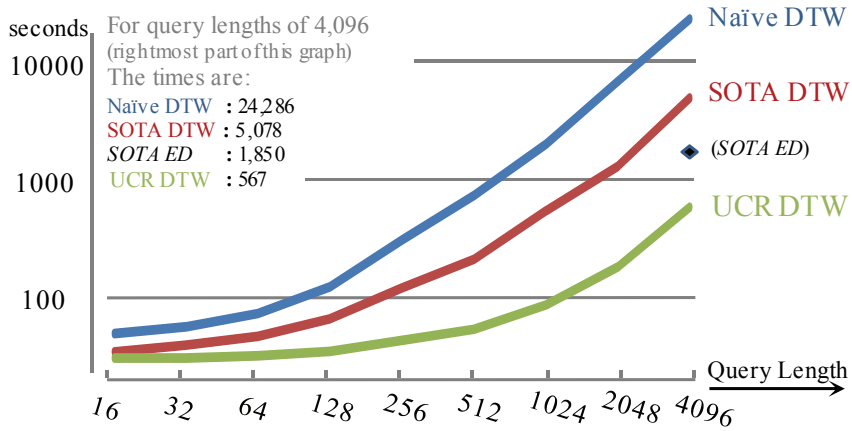

Figure 10: The time taken to search random walks of length 20 million with increasingly long queries, for three variants of DTW. In addition, we include just length 4,096 with SOTA-ED for reference

It is also interesting to consider the results of the 128-length DTW queries as a ratio over GOAL. Recall that the cost for GOAL is independent of query length, and this experiment is just 23.57 seconds. The ratios for Naive, SOTA and UCR suite are 5.27, 2.74 and 1.41, respectively. This suggests that we are asymptomatically closing in on the fastest possible subsequence search algorithm for DTW. Another interesting ratio to consider is the time for UCR-DTW over UCR-ED, which is just 1.18. Thus, the time for DTW is not significantly different from that for ED, an idea which contradicts an assumption made by almost all papers on time series in the last decade (including papers by the current authors).

2.4.2. Supporting Long Queries: EEG

The previous section shows that we gain the greatest speedup for long queries, and here we show that such long queries are really needed. The first user of the UCR suite was Dr. Sydney Cash, who together with Brandon Westover wants to search massive archives of EEG data for examples of epileptic spikes, as shown Figure 11.

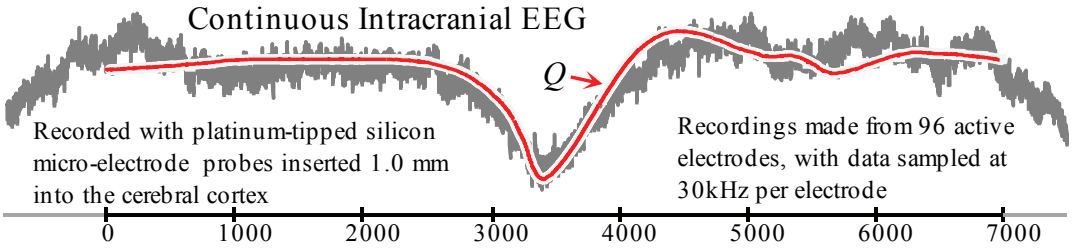

Figure 11: Query *Q* shown with a match from the 0.3 trillion EEG dataset

From a single patient S.C. gathered 0.3 trillion data points and asked us to search for a prototypical epileptic spike *Q* he created by averaging spikes from other patients. The query length was 7,000 points (0.23 seconds). Table 3 shows the results.

Table 3: Time to search 303,523,721,928 EEG data points, $|Q| = 7000$

Note that only ED is considered here because DTW may produce false positives caused by eye blinks		**UCR-ED**	**SOTA-ED**
	EEG	3.4 hours	494.3 hours

This data took multiple sessions over seven days to collect, at a cost of approximately $34,000 [50], so the few hours of CPU time we required to search the data are dwarfed in comparison.

2.4.3. Supporting Very Long Queries: DNA

Most work on time series similarity search (and *all* work on time series *indexing*) has focused on relatively short queries, less than or equal to 1,024 data points in length. Here we show that we can efficiently support queries that are two orders of magnitude longer.

Table 4: An algorithm to convert DNA to time series

$T_1 = 0$,	**for** i = 1 **to** |DNAstring|			
	if $DNAstring_i$ = **A**,	**then** T_{i+1}	$= T_i + 2$
	if $DNAstring_i$ = **G**,	**then** T_{i+1}	$= T_i + 1$
	if $DNAstring_i$ = **C**,	**then** T_{i+1}	$= T_i - 1$
	if $DNAstring_i$ = **T**,	**then** T_{i+1}	$= T_i - 2$

We consider experiments with DNA that has been converted to time series. However, it is important to note that we are not claiming any particular bioinformatics utility for our work; it is simply the case that DNA data is massive, and the ground truth can be obtained through other means. As in [42], we use the algorithm in Table 4 to convert DNA to time series[2].

We chose a section of Human chromosome 2 (H2) to experiment with. We took a subsequence beginning at 5,709,500 and found its nearest neighbor in the genomes of five other primates, clustering the six sequences with single linkage to produce the dendrogram shown in Figure 12.

Pleasingly, the clustering *is* the correct grouping for these primates [31]. Moreover, because Human chromosome 2 is widely accepted to be a result of an end-to-end fusion of two progenitor ancestral chromosomes 2 and 3 [31], we should expect that the nearest neighbors for the non-human apes come from one of these two chromosomes, and that is exactly what we found.

Our query is of length 72,500, and the genome chimp is 2,900,629,179 base pairs in length. The single-core nearest neighbor search in the entire chimp genome took 38.7 days using Naive, 34.6 days using SOTA, but only 14.6 hours using the UCR suite. As impressive as this is, as we shall show in the next section, we can do *even better*.

[2] To preserve the reversible one-to-one mapping between time series and DNA we normalize the offset by subtracting round(mean) and we do not divide by the STD.

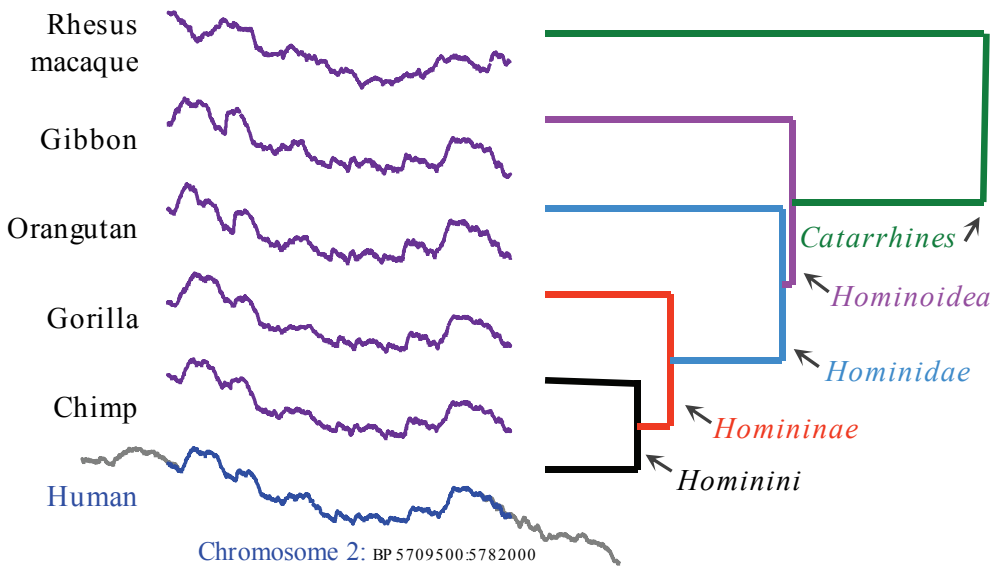

Figure 12: A subsequence of DNA from Human chromosome 2, of length 72,500 beginning at 5,709,500 is clustered using single linkage with its Euclidean distance nearest neighbors from five other primates

A. *Can we do better than UCR Suite?*

We claim that for the problem of exact similarity search with arbitrary length queries, our UCR suite is close to optimal. However, it is instructive to consider an apparent counterexample and its simple "patch".

Consider the search for a query of length 64 considered in Section 2.4.1. Using GOAL took 9.18 seconds, but UCR suite took only a little longer, just 10.64 seconds. Assume that the original query was:

Q = [**2.34**, **2.01**, **1.99**, ...]

But we make it three times longer by padding it like this:

QP = [**2.34**, **2.34**, **2.34**, **2.01**, **2.01**, **2.01**, **1.99**, **1.99**, **1.99**, ...]

Further assume that we do the same to database T, to get TP, which is three times longer. What can we now say about the time taken for the algorithms? GOAL will take exactly three times longer, and Naive takes exactly nine times longer, because each ED calculation takes three times longer and there are three times as many calculations to do. Our UCR suite does not take nine times longer, as it can partly exploit the "smoothness" of the data; however, its overhead *is* greater than three. Clearly, if we had known that the data was contrived in this manner, we could have simply made a one-in-three downsampled version of the

data and query, done the search on this data, and reported the location and distance back in the *TP* space by multiplying each by three.

Of course, this type of pathological contrived data does not occur in nature. However, some datasets are richly *oversampled*, and this has a *very* similar effect. For example, a decade ago, most ECGs were sampled at 256 Hz, and that seems to be adequate for virtually all data analysis applications [11]. However, current machines typically sample at 2,048 Hz which, given the above reasoning, would take up to sixty-four times longer to search ($(2{,}048/256)^2$) with almost certainly identical results.

We believe that oversampled data can be searched more quickly by exploiting a provisional search in a downsampled version of the data that can quickly provide a low *best-so-far*, which, when projected back into the original space can be used to "prime" the search by setting a low *best-so-far* at the beginning of the search, thus allowing the early abandoning techniques to be more efficient.

To test this idea, we repeated the experiment in the previous section, with a one-in-ten downsampled version of the chimp genome / human query. The search took just 475 seconds. We denoted the best matching subsequence distance *rD*. We reran the full resolution search after initializing the *best-so-far* to *rD**10. This time the search fell from 14.64 hours to 4.17 hours, and we found the same answer, as we logically must.

Similar ideas have been proposed under the name of Iterative Deepening DTW [1] or Multi Scale DTW [34][49]; thus, we will not further develop this idea here. We simply caution the reader that oversampled (i.e., "smooth") data may allow more speedup than a direct application of the UCR suite may initially suggest.

2.4.4. Realtime Medical and Gesture Data

The proliferation of inexpensive low-powered sensors has produced an explosion of interest in monitoring real time streams of medical telemetry and/or Body Area Network (BAN) data [29].

There are dozens of research efforts in this domain that explicitly state that while monitoring under DTW is *desirable*, it is impossible [45]. Thus, approximations of, or alternatives to DTW are used. Dozens

of suggested workarounds have been suggested. For example, [23] resorts to only "*dealing with shorter test and class templates, as this is more efficient*"; many research efforts including [43] resort to a low cardinality version of DTW using integers, or DTW approximations that operate on piecewise linear approximations of the signals [27][36], or drastically downsampled versions of the data [19][37]. In spite of some progress from existing ideas such as lower bounding, [10] bemoans DTW is "*still too slow for gesture recognition systems*", [36] laments that the "*problem of searching with DTW (is) intractable*", [19] says "*Clearly (DTW) is unusable for real-time recognition purposes*" and [41] notes "*Processing of one hour of speech using DTW takes a few hours*."

We believe that the UCR suite makes all of these objections moot. DTW *can* be used to spot gestures/brainwaves/musical patterns/anomalous heartbeats in *real-time*, even on low-powered devices, even with multiple channels of data, and even with multiple simultaneous queries.

To see this, we created a dataset of one year of electrocardiograms (ECGs) sampled at 256Hz. We created this data by concatenating the ECGs of more than two hundred people, and thus we have a highly diverse dataset, with 8,518,554,188 data points. We created a query by asking USC cardiologist Dr. Helga Van Herle to produce a query she searches for on a regular basis, she created an idealized Premature Ventricular Contraction (PVC). The results are shown in Table 5. While this was on our multi-core desktop machine, the fact that our results are 29,219 times faster than real-time (256Hz) suggests that real-time DTW is tenable even on low-power devices.

Table 5: Time taken to search one year of ECG data with $|Q| = 421$

	UCR-ED	**SOTA-ED**	**UCR-DTW**	**SOTA-DTW**
ECG	4.1 minutes	66.6 minutes	18.0 minutes	49.2 hours

2.4.5. Speeding up Existing Mining Algorithms

In this section, we demonstrate that we can speed up much of the code in the time series data mining literature with minimal effort, simply by replacing their distance calculation subroutines with the UCR suite. In many cases, the difference is small, because the algorithms in question already typically try to prune as many distance calculations as possible. As an aside, in at least some cases we believe that the

authors could benefit from redesigning the code in light of the drastically reduced cost for similarity search that UCR suite offers. Nevertheless, even though the speedups are relatively small (1.5X to 16X), they are "free", requiring just minutes of cut-and-paste code editing.

Time Series Shapelets have garnered significant interest since their introduction in 2009 [46]. We obtained the original code and tested it on the Face (four) dataset, finding it took 18.9 minutes to finish. After replacing the similarity search routine with UCR suite, it took 12.5 minutes to finish.

Online Time Series Motifs generalize the idea of mining repeated patterns in a batch time series to the streaming case [32]. We obtained the original code and tested it on the EEG dataset used in the original paper. The fastest running time for the code assuming linear space is 436 seconds. After replacing the distance function with UCR suite, it took just 156 seconds.

Classification of Historical Musical Scores [17]**.** This dataset has 4,027 images of musical notes converted to time series. We used the UCR suite to compute the rotation-invariant DTW leave-one-out classification. It took 720.6 minutes. SOTA takes 142.4 hours. Thus, we have a speedup factor of 11.8.

Classification of Ancient Coins [22]. 2,400 irregularly shaped coins are converted to time series of length 256, and rotation-invariant DTW is used to search the database, taking 12.8 seconds per query. Using the UCR suite, this takes 0.8 seconds per query.

Clustering of Star Light Curves is an important problem in astronomy [27], as it can be a preprocessing step in outlier detection [38]. We consider a dataset with 1,000 (purportedly) phase-aligned light curves of length 1,024, whose class has been determined by an expert [38]. Doing spectral clustering on this data with DTW (R = 5%) takes about 23 minutes for all algorithms, and averaged over 100 runs we find the Rand-Index is 0.62. While this time may seem slow, recall that we must do 499,500 DTW calculations with relatively long sequences. As we do not trust the original claim of phase alignment, we further do *rotation-invariant* DTW that dramatically increases Rand-Index to 0.76. Using SOTA, this takes 16.57 days, but if we use the UCR suite, this time falls by an order of magnitude, to just 1.47 days on a single core.

2.5 Discussion and conclusions

While our work has focused on fast *sequential search*, we believe that for DTW, our work is faster than all known *indexing* efforts. Consider [9], which indexes a random walk time series of length 250,000 to support queries of length 256. They built various indexes to support DTW queries, noting that the fastest of the four carefully tuned approaches requires approximately 15,000 pages accesses to answer a query. These disk accesses are necessarily *random* accesses. While they did not give wall clock time, if we assume an HDD spindle speed of 7,200 rpm (average rotational latency = 4.17*ms*), then just the disk I/O time to answer this query must be at least 62.55 seconds. However, as we have shown, we can load all of the data into the main memory with more efficient sequential disk accesses and answer these queries in 0.4 seconds, *including* disk I/O time, on a single core machine.

Note that all experiments in this paper include the time taken to read the data from disk. However, for more than a few million objects this time is inconsequential thus we did not report it separately.

We have made a strong and unintuitive claim in the abstract. We said that our UCR-*DTW* is faster than all current *Euclidean* distance searches. In Table 5, for example, we show that DTW can be three times faster than state-of-the-art ED searching. How is this possible? Recall that all Euclidean searches in the literature require an $O(n)$ data normalizing step to be performed for each subsequence. Thus, no matter how effective the pruning/search strategy used, the amortized time for a single sequence must be at least $O(n)$. In contrast, using the ideas developed in this work, the vast majority of potential DTW calculations are pruned with $O(1)$ work, while some require up to $O(n)$ work, and only a vanishingly small fraction require $O(nR)$ work. The weighted average of these possibilities is less than $O(n)$.

To put our results in perspective, we compare them with a very recent state-of-the art embedding-based DTW search technique, called EBSM (*including* the variant called BSE) [35]. This is an excellent paper to use as a benchmark, as it exhaustively compares to almost all other methods in the literature, and it tests its contributions over different datasets, query lengths, warping widths, etc. In contrast to EBSM:

- Our method is *exact*; EBSM is *approximate*.
- EBSM requires setting some parameters (number of reference sequences, dimensionality, number of split points, etc.). Our method requires *zero* parameters.
- EBSM requires offline preprocessing that takes over 3 hours for just 1 million objects. We have *zero* preprocessing time.
- The EBSM method does not, and cannot, Z-normalize. As noted in Section 2.1.2.A. , we believe that Z-normalizing is critical, and we have shown that failure to do it hurts on 45 out of 45 of the UCR time series classification datasets.
- EBSM can support queries in a predetermined range, which must be predetermined and limited for efficiently. In contrast, we have no minimum/maximum query length.
- We can also handle exact queries under uniform scaling [18].
- Finally, we are simply much faster! (cf. Section 1)

Note, however, that there can be great utility in fast approximate search. There exist data mining algorithms that can use a combination of (hopefully few) exact distance measures and (hopefully much faster) approximate searches to produce overall exact results [42]. However an approximate search method being faster than our approach is a very high threshold to meet.

We have shown our suite of ideas is 2 to 164 times faster than the true state-of-the-art, depending on the query/data. However, based on the quotes from papers that we have sprinkled throughout this work, we are sometimes more than 100,000 times faster than recent papers; how is this possible? The answer seems to be that it is possible to produce very naive implementations of DTW. For example, the recursive version of DTW can be one to three orders of magnitude slower than the iterative version, depending on the computer language and query length. Thus, the contributions of this chapter are twofold. First, we have shown that much of the recent pessimism about using DTW for real-time problems was simply unwarranted [13]. Existing techniques, especially lower bounding, if carefully implemented can make DTW tractable for many problems. Our second contribution is the introduction of the UCR suite of

techniques that make DTW and Euclidean distance subsequence search significantly faster than current state-of-the-art techniques.

We regret that the page limitations preclude full pseudo-code; however, full pseudo-code (and source-code) is available at [50].

In future work we plan to revisit algorithms for time series motif discovery [32][33], anomaly detection [42][38], time series summarization, shapelet extraction [46], clustering, and classification [13] in light of the results presented in this work.

Chapter 3: Fast Shapelet Discovery

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

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

3.1 INTRODUCTION

Shapelets are a recently introduced concept in time series data mining. In essence, shapelets are prototypical time series "snippets" that can be used to classify unlabeled time series that contain an occurrence of the shapelet within some previously learned distance threshold. The utility of shapelets has been confirmed and extended by many independent groups of researchers [51][56][59][60][63][65][66][79][80]. The exploding interest in shapelets can be attributed to the following factors. First, they generalize the *lazy* nearest neighbor algorithm, widely understood to be the state-of-the-art technique for time series [77], to an *eager* decision-tree like classifier, allowing orders of magnitude improvement in classification time. Second, they are *interpretable*, and can give insights as to what defines the differences between two classes [80]. Finally, on some problems, shapelets can be simply more accurate than any known rival method [69][80].

Given these advantages, we have recently seen shapelets (and very similar ideas) applied to creating classifiers in various domains such as gesture recognition [52][67], sensor networks [62], motion capture [76], cardiology, climatology [68], robotics [76], electrical power demand [51][59] and health care [72]. In addition, we have begun to see several generalizations of shapelets, such as *logical shapelets* [69], which classify objects based on conjunctions/disjunctions of shapelets, and *local shapelets* [79], which impose constraints on where valid shapelets may appear within an object.

Before continuing further, we will take the time to develop the reader's intuition for shapelets. Figure 13 shows six examples of reptile skulls [53], and their time series representations. Three of them are horned lizards, (*Phrynosoma coronatum*, *P. braconnieri* and *P. mcallii*), and the other three are turtles, (*Elseya dentate*, *Glyptemys muhlenbergii*, and *Annemys sp*).

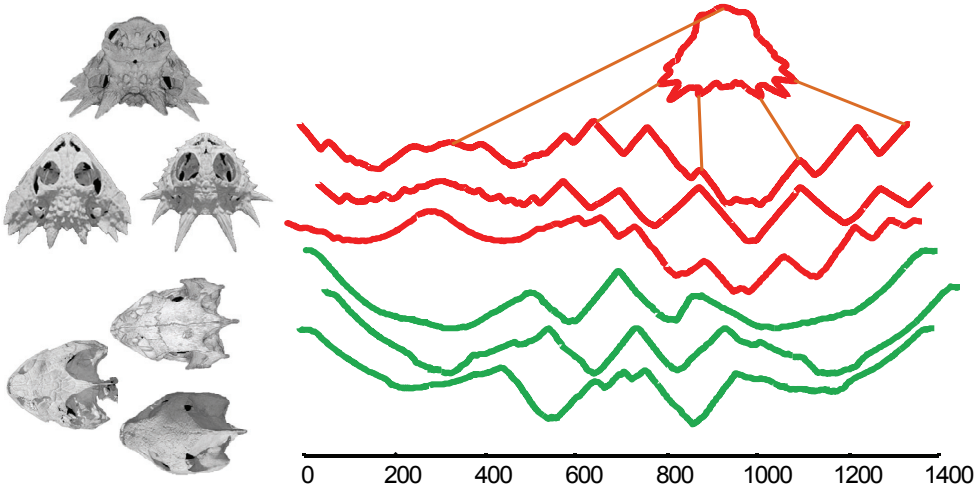

Figure 13: *left*) Skulls of horned lizards and turtles. *right*) the time series representing the images. The 2D shapes are converted to time series using the technique in [64]

For more details on how shapes are converted to time series, we refer the reader to [64][75]; however, Figure 13.*top.right* visually hints at how a shape can be "unwound" into a time series.

As we can see from Figure 13, the two classes here have a lot of intraclass variability, something that does not bode well for traditional classifiers that consider the *entire* time series [77]. Suppose instead we run the shapelet discovery algorithm on this small dataset [80]. Doing so, we find the shapelet that can best distinguish the two types of reptiles. This shapelet is presented in Figure 14 in both the time series space, and "brushed" back onto the original shape space.

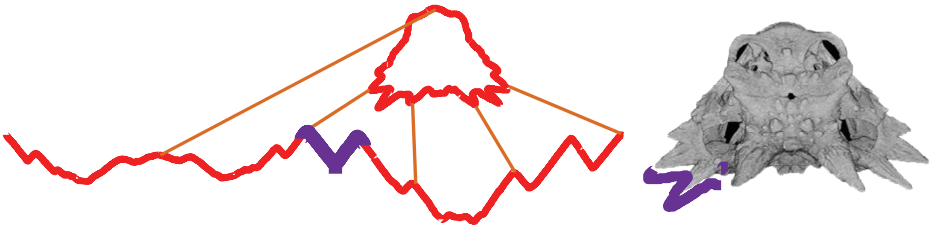

Figure 14: *left*) The shapelet that best distinguishes between skulls of horned lizards and turtles, shown as the purple/bold subsequence. *right*) The shapelet projected back to the original 2D shape space

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

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

The best-known running time for the shapelet discovery algorithm is $O(n^2m^3)$ where n is the number of objects or time series in the dataset, and m is the length of the longest time series. Recently [69] proposed an approach to admissibly prune some candidates and remove redundant calculations to make their algorithm faster than the original approach by a constant factor (of about ten to twenty). To the best of our knowledge, this algorithm is the current state-of-the-art on convention hardware (some speedup attempts exploit exotic hardware [51]).

In this work, we propose an $O(nm^2)$ algorithm for finding shapelets. Our algorithm is *heuristic*, it is not guaranteed to find the same shapelet as [69][80]. We exploit a random projection technique [74][71] on the SAX representation [64][78] to find potential shapelet candidates. However, our experimental results in Section 3.5 demonstrate that the classification accuracy of the proposed algorithm is not significantly different from the accuracy obtained by exact brute-force algorithms [69][80].

The rest of this chapter is organized as follows. In Section 3.2, we introduce definitions and notations. The basic shapelet algorithm is discussed in [80] and the current state-of-the-art algorithm [69] is reviewed in Section 3.3. In Section 3.4, we explain our algorithm in detail. Section 3.5 demonstrates the performance

and accuracy of our algorithm. Case studies to show the advantage of our proposed algorithm are described in Section 3.6 and we offer conclusions in Section 3.7.

3.2 Definitions and Notation

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

Definition 4 A *time series T* is an ordered list of numbers, $T = t_1, t_2, ..., t_m$. Each value t_i can be any finite number and m is the length of time series T.

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

Definition 5 A *time series subsequence S* is a contiguous sequence of a time series. Subsequence S of length l of time series T starting at position i can be written by $S = T_i^l = t_i, t_{i+1}, ..., t_{i+l-1}$.

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

Definition 6 A *dataset D* is a set of pairs of time series, T_i, and its class label, c_i. Formally, $D = \langle T_1,c_1\rangle, \langle T_2,c_2\rangle, \langle T_3,c_3\rangle, ..., \langle T_n,c_n\rangle$. For the rest of the chapter, we use n as the number of time series inside the dataset D. Note that the lengths for each time series are not necessarily equal.

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

Definition 7 The *distance* between subsequence S and $\hat{S}$ of the same length is the length-normalized of Euclidean distance between subsequences S and $\hat{S}$. If both subsequences are Z-normalized with *mean*=0 and *std*=1, the distance is defined as:

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

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

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

Definition 8 The *distance* between subsequence S of length l and time series T is defined as the minimum distance between subsequence S and any subsequence of T of the same length as subsequence S. Formally, $dist(S, T) = min_i\ dist(S, T_i^l)$.

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

Definition 9 An entropy of the dataset D is defined as $E(D) = \sum_{i=1}^{c} p_i log(p_i)$.

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

Definition 10 A *split* is a tuple <*s*,*d*> of a subsequence s and distance threshold d which can separate the dataset into two smaller dataset D_L and D_R are the number of time series in D_L and D_R is n_L and n_R, respectively.

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

Definition 11 The information gain of a split *sp* is:

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

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

Definition 12 A separation gap of a split *sp* is:

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

The definition of *shapelet* will be explained briefly here; however, for a more complete definition of *shapelet*, please refer to [80].

Definition 13 A *shapelet* is a split that separates the dataset into two smaller datasets with maximum information gain; ties are broken by maximizing the separation gap.

We visually summarize the concept of shapelets with our toy example as shown in Figure 15. A candidate (e.g., the highlighted subsequence from Figure 14.*left*) is shown in the box for reference. The distances between the candidate subsequence (i.e., tentative shapelet) and all time series are calculated; all corresponding objects are placed on the *orderline* according to the calculated distance. In Figure 15, three

skulls from horned lizard, whose distances to the shapelet are small, are shown as red rectangles on the left hand side of the orderline. In contrast, the distance between three time series of turtle skulls and the candidate subsequence shown as green triangles on the right hand side of the orderline, because their distances to the candidate shapelet are larger. Note that, in this example, the candidate shapelet corresponds to the horns of a lizard (cf. Figure 14).

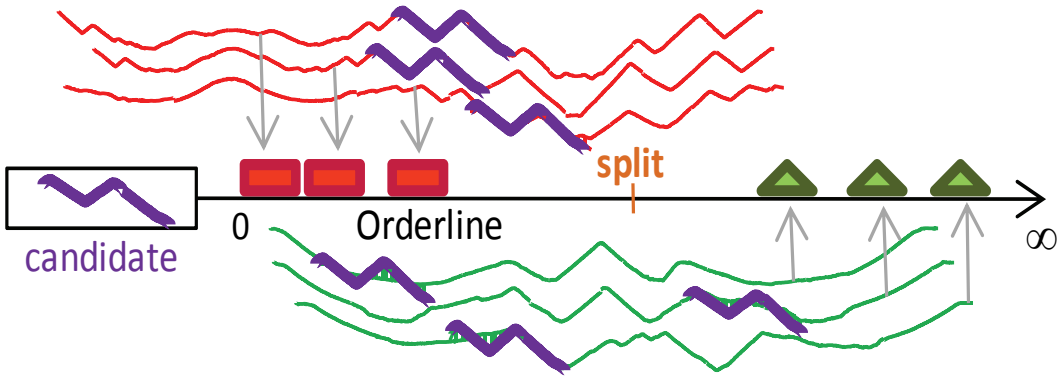

Figure 15: The *orderline* shows the distance between the candidate subsequence and all time series as positions on the x-axis. The three objects on the left hand side of the line correspond to horned lizards and the three objects on the right correspond to turtles

After the orderline is created, we can calculate the split point, the separation gap, and the information gain of the candidate shapelet. When all candidates have been processed in this manner, the best one will be reported as the final shapelet [69][80].

3.3 Related AND BACKGROUND Work

Our work extends the original work by Ye which introduced the concept of shapelets and showed an algorithm to allow their discovery [80]. However the general intuition behind shapelets, the idea of using small sub-patterns to identify the class of a larger object, is known in other domains (esp. bioinformatics) including *class prototypes* [56], *discriminative patterns* [52][55][58], and *predictive motifs* [68], etc.

In the next section, we take a short time to consider the current state-of-the-art shapelet discovery algorithm.

3.3.1. Brute Force Shapelet Discovery

The brute force shapelet discovery algorithm as shown in Table 6 is a simple algorithm that generates and tests all possible candidates and returns the best one.

Table 6: Brute Force Algorithm

Algorithm: *BruteForceShapelet*

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

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

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

Thereafter in line 6, each candidate will be used to compute a split `sp` as explained at Figure 15 in previous section. Finally, the information gain is computed in line 7. The returned shapelet is the candidate with maximum information gain.

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

3.3.2. Current State-of-the-Art

The first improvement of the brute force algorithm was introduced in [80]. They proposed a technique to calculate a cheap-to-compute upper bound of information gain and use it to admissibly prune some candidates.

The current state-of-the-art algorithm is given at [69]. The algorithm can also find the exact shapelet [80], but does so more quickly. The speed-up comes from using a classic pruning technique — triangular inequality — to prune some candidates, and from some caching tricks. The latter idea trades speedup for memory, and may run into space problems for large datasets.

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

To concretely ground this analysis, on the *Wafer dataset* [61], with n=1000 and m=152, the state-of-the-art takes more than 12 hours to find the shapelet. However, the algorithm we will introduce in the next section can find essentially the same shapelet in just 23 seconds.

3.4 Fast Shapelet Discovery

We are finally in a position to explain our algorithm in detail. First, we describe the key ideas using our reptile toy example; then, we give formal details in Section 3.4.2.

3.4.1. Overview of the Algorithm

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

A. *Generating SAX Words*

For each object in the dataset, we transform the time series into a symbolic representation using Symbolic Aggregate approXimation (SAX) [64][78]. For brevity, we assume the reader is familiar with SAX, and refer the interested reader to [64] for additional details.

From our toy example in Section 3.1, a time series from *P.coronatum* is shown once again in Figure 16. The top part of the figure shows an example of a SAX word **adbacc**, created by the first subsequence. Multiple SAX words will be generated for a given time series using the sliding window technique [64]. SAX has two parameters, which are desired (reduced) dimensionality, *d*, and cardinality, *c*. Although some techniques (e.g., [57]) could be applied for setting these parameters, for simplicity, in our implementation, we set cardinality to 4 and word length to 16 so we can represent a SAX word with a simple 4-byte integer.

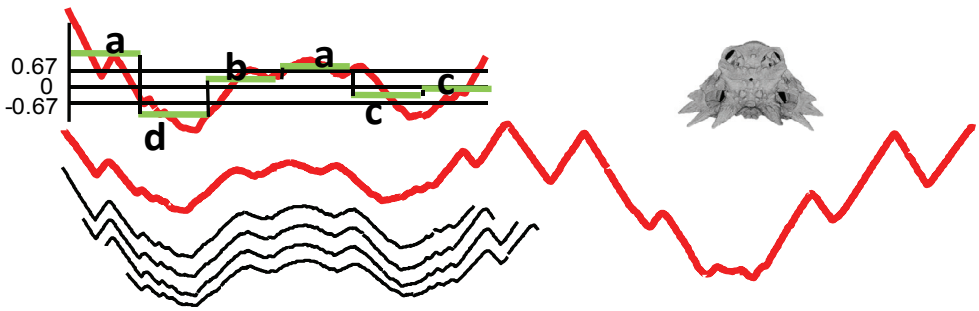

Figure 16: *top.left*) The SAX word `adbacc` created from a subsequence of the time series corresponding to *P. coronatum*. *bottom*) sliding window technique

B. *Random Masking*

It is very important to recall that a *single* time series creates *multiple* SAX words, corresponding to the multiple subsequences we can obtain as we slide the shorter subsequence length across the longer time series. This is hinted at in Figure 16, and shown explicitly in Figure 17.*left*, where each time series (derived from a skull) creates two or three SAX words of length five.

Having created the SAX representation of our data, we have an *apparent* solution to the shapelet discovery problem. We could conduct a brute force search for the shapelets in the SAX space. This would require only slight modifications of the algorithm shown in Table 6, and because of the reduced dimensionality of SAX it would be faster than working with the raw data. There are two problems with this

idea. The search would be faster, but still quadratic in the number of SAX words. However, more importantly, we have the problem of *false dismissals*.

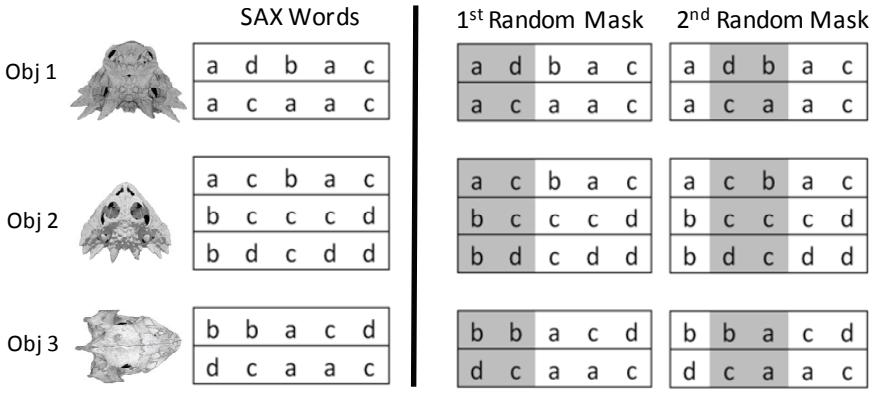

Figure 17: *left*) SAX words of each object. *right*) SAX words after masking two symbols. Note that masking positions are randomly picked

The problem of false dismissals is caused by the fact that two time series that differ only by a tiny epsilon could produce two different SAX words[3]. Thus, it is possible that the best shapelet in the raw data spaces maps to slightly different SAX words, such as the SAX words **adbac** and **acbac**, created by the lizard skulls in Figure 17.*left*.

The solution to this problem is to exploit *random projection*, a mature idea from bioinformatics [74]. The idea is to project all SAX words of high dimensionality to smaller dimensionality. This is illustrated in Figure 17.*right*, where all SAX words of dimensionality five have been randomly masked at two positions, reducing the dimensionally to three. Note that the first such random projection *does* take our two different SAX words, words **adbac** and **acbac**, and makes them identical, **adbac** and **acbac**.

The reader will appreciate two potential problems with the idea of random projection. In our toy example we *contrived* our "random" choice of a mask, we cannot be generally sure that a single projection helps us. The second problem is that if we mask too many locations, our decease in the likelihood of false *dismissals* comes at the cost of an explosion of false *positives*, all of which must be checked. Again we can turn to the bioinformatics literature for the answer [74]. As hinted at by the multiple masks in Figure

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

17.*right*, if we mask conservatively, but do multiple random masks, we can make the probability of false dismissals arbitrarily low while not incurring a measurable increase in false positives [74]. The remainder of Section 4 makes these ideas more concrete.

C. *Counting Similar Objects*

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

Figure 18.A shows that, after hashing, the SAX words **adbac** and **acbac**, share the same signature, ****bac**. All SAX words that have signature **bac** have their counters incremented in the relevant table shown in Figure 18.A.*right*. Similarly, in the second iteration the words **adbac** and **acbac**, once again randomly hash to the same word, this time **a****ac**.

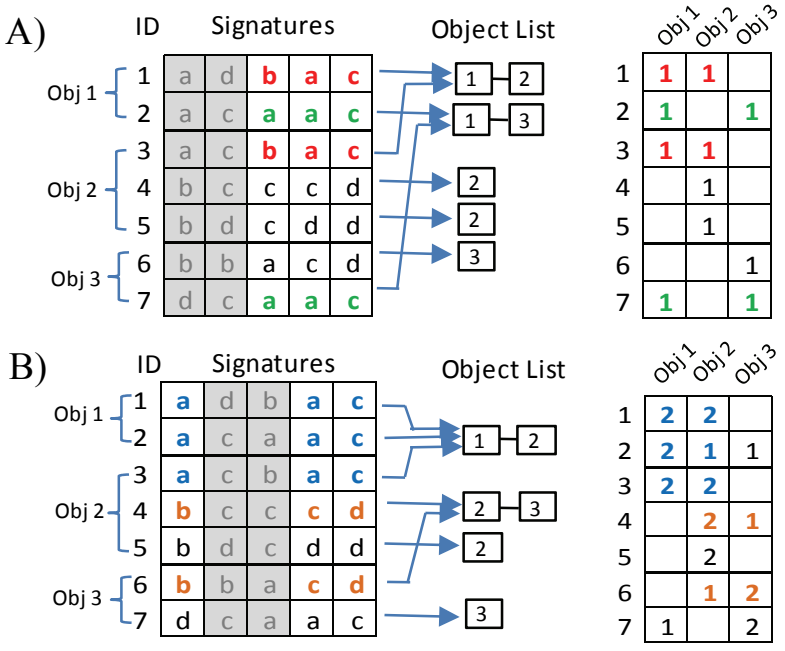

Figure 18: The first (A) and second (B) iterations of the counting process. *left*) Hashing process to match all same signatures. Signatures created by removing marked symbols from SAX words. *right*) Collision tables showing the number of matched objects by each words

Thus, after *r* iterations of random projection, we expect the collision table shown in Figure 18.A.*right* to remain mostly sparse, but to contain some locations that have values that are a significant fraction of *r*. As we shall show in the next section, this information can guide our shapelet search.

D. Finding the Best Candidates

Continuing with our toy example, let us assume that we have done random projections for five iterations and the collision table is shown as in Figure 19.A.

As shown in Figure 19.B we can condense the collision table by summing all the object-based counts to the class-based counts, and creating the complementary data structure in Figure 19.C. From these two tables we can calculate the *distinguishing power* of each SAX word using the simple equation shown in Figure 19.D. Note that *distinguishing power* is high if the reference words appear frequently in one class but rarely in another class.

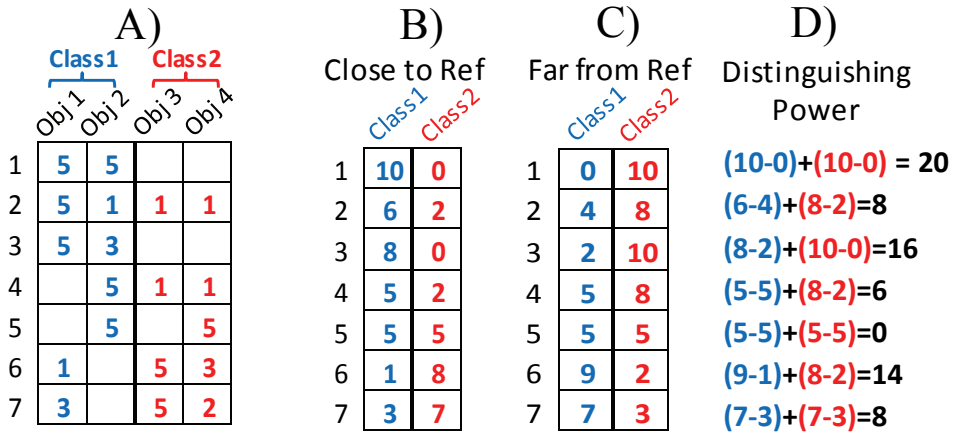

Figure 19: A) The collision table of all words after five iterations. Note that counts show the number of occurrences that an object shares a same signature with the reference word. B) Grouping counting scores from objects in the same class. C) Complement of (B) to show that how many times objects in each class that do not share the same signature with the reference word. D) The distinguishing power of each SAX word

In this example, the highest score is from *word1* because it is close to objects in *class1* 10 times (*obj1* 5 times and *obj2* 5 times) and far from objects in *class2* 10 times; hence, its distinguishing power is 10+10 = 20. In contrast, SAX *word5* receives power score of zero because this reference word is similar to objects from *class1* 5 times but also far from objects in *class1* 5 times, and has the same distribution in *class2*;

hence, the score is (5-5)+(5-5) = 0. This suggests a pattern that is equally frequent in both classes, rather like a "stop-word" in text classification.

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

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

3.4.2. Fast Shapelet Algorithm

Our shapelet discovery algorithm is shown in Table 7. In line 1, we extract all time series with their class labels from the current dataset *D*. Note that the dataset *D* will be iteratively made smaller as we descend deeper into the decision tree.

Table 7: Fast Shapelet Algorithm

Algorithm: *FastShapelet*

Input: *D* : Dataset contain time series and class labels
r : number of random iterations
k : number of SAX to be candidates

Output: *shapelet*: the final shapelet

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

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

To select the candidates, all subsequences of length `len` from all time series are created using sliding window technique, and we create their corresponding SAX word and keep them in `SAXList` (line 3/Figure 17.*left*). After the list of SAX words has been created, we will use these discrete representations to do hashing with `RandProjection()`, by creating a hash signature of each SAX word and give one count for each SAX words based on its signature. Then, we update the total score from multiple iterations (line 7/Figure 18).

Next each SAX word is given a score to show that how many times each word occurs in each object. We then calculate a distinguishing power for each SAX word, and pick top k subsequences that have highest score (line 9/Figure 19). We remapped these SAX words back to their original raw data subsequences (line 10).

Note that we have two parameters r and k here. However, according to our experiments in Section 3.5, our algorithm is not sensitive to them, thus we simply fixed $r = 10$ and $k = 10$ below.

We are now ready for the second phase (line 12-23); we will calculate the information gain for each candidate in the top k list and pick the best one as the shapelet. In detail, each candidate is considered one at a time (line 13). The body of the loop calculates the distance between the candidate subsequences and each time series using the equation in Definition 5. After these calculations, (line 17-21) we will pick the subsequence which has the highest information gain, as the final shapelet, breaking ties (if any) by maximum gap.

This explanation is necessarily terse. We refer the interested reader to [81] for a more detailed line-by-line explanation, and the original (annotated) source code.

3.5 Experimental Results

We begin by noting that *all* the code and data used in this work are available at supporting webpage [81], in addition to numerous additional experiments.

3.5.1. UCR Time Series Dataset

We compared our algorithm with the current state-of-the-art [69]. For fairness, we used the code provided by original authors and set the parameters as they had recommended on their supporting webpage.

We begin by considering the accuracy on 32 datasets from UCR Time Series archives [61] in Figure 20. Note that we attempted tests on all *45* datasets from UCR achieves, however we abandoned the 13 experiments where the state-of-the-art algorithm [69], had not finished after 24 hours.

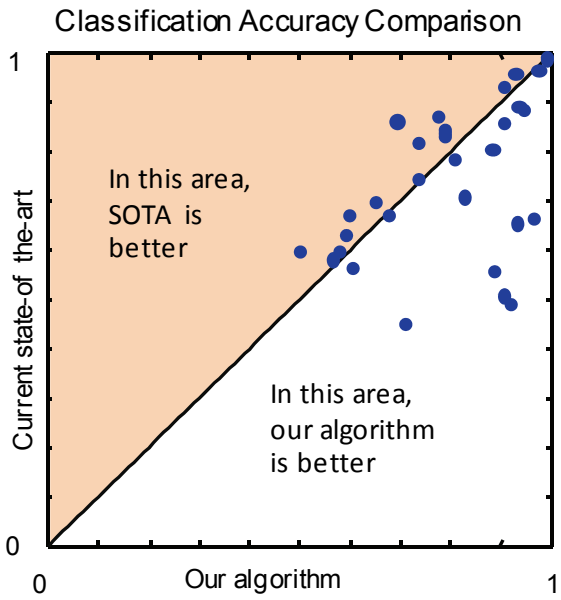

Figure 20: Classification accuracy of our algorithm and the state-of-the-art on 32 datasets from the UCR archive

Visually it is difficult to say which algorithm is better, and counting wins-ties-losses produces similarly ambivalent results. However, the results are *strongly consistent* with our claim that our method is no worse that the state-of-the-art. The only real difference between our algorithms is scalability, as the time comparison shown in Figure 21 illustrates.

Figure 21: Running time comparison between our algorithm and the state-of-the-art on 32 datasets from UCR time series archives

The greatest speedup we achieve on these 32 datasets is 2,030X, and we gain a speedup exceeding 100X for at least 12 other datasets. On *Wafer* dataset, although we gain the most speed up with 2,030X, our method still gets very high accuracy at 99.64%, which appears to be the best known result on this dataset [61][77]. In the next section we explore the issues effecting scalability in more detail.

3.5.2. Scalability

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

Figure 22.*left* shows the result when the number of time series, *n*, is varied from 50 to 800 and the length of all time series, *m*, is fixed at 100. Figure 22.*right* shows the result when *n* is fixed at 100 and *m* is varying from 50 to 800. Note that the maximum size of the experiments we consider here are constrained by implementation of the state-of-the-art algorithm that we received from the original authors. For any experiments larger than the below we get an *out-of-memory* error.

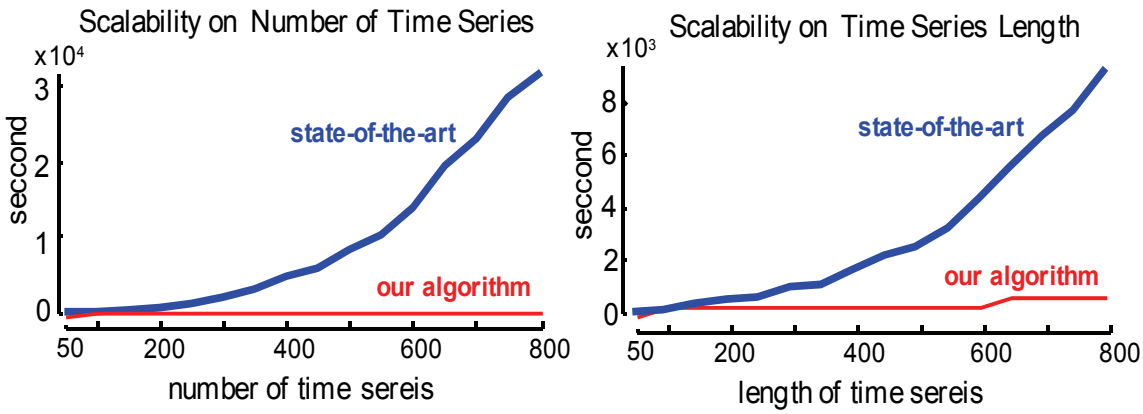

Figure 22: Scalability of our algorithm and the current state-of-the-art on StarlightCurves dataset. *left*) Number of time series in the dataset is varying. *right*) The length of time series is varying

The running time of the current state-of-the-art in Figure 22.*left* is increases from 16 seconds to 8.7 hours from n=50 to n=800 (m is fixed at 100). However, our algorithm is significantly faster; the running time is 0.76 seconds at the beginning and less than 16 seconds when n=800. Thus the speedup factor when n=800 is 1,970X. Figure 22.*right* shows that our algorithm achieves similar speedups when m is increased.

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

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

3.5.3. When to use Shapelet or 1NN

One of the classic questions faced by all data miners is which algorithm to use for a given task. It is well documented in the literature that the classification performance of shapelets are highly variable in the sense that that can be significantly *better*, or significantly *worse* that the only other high performing time series classification method, the nearest neighbor algorithm [59][60][65][80][80].

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

To answer this question, we tested on all 45 datasets from UCR archives using a method recommended by Salzberg [73]. From the training data of all 45 datasets in [61], we split the train data to two equal parts, called **A** and **B**; the test data from archives will be preserved as it is and used as unseen data; we call this test data **C**.

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

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

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

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

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

If a ratio is larger than one, the accuracy of shapelet is higher than the accuracy of 1NN. The plot in Figure 23 shows the comparison between expected ratios and actual ratios of shapelets and 1NN algorithm.

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

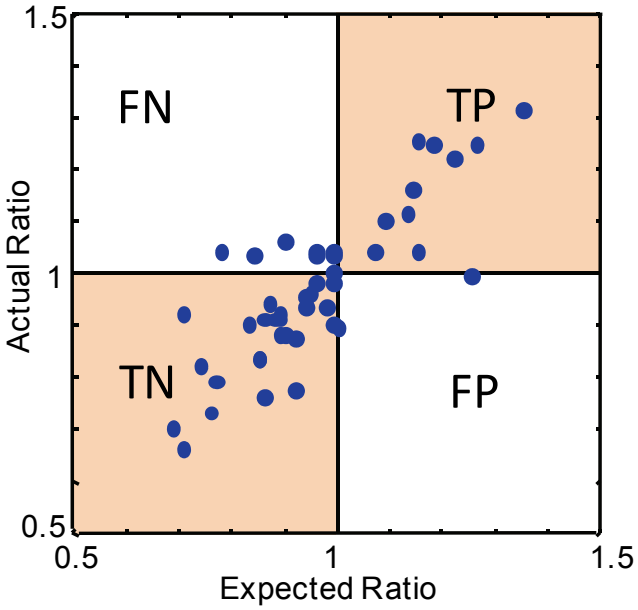

Figure 23: Accuracy ratio between *FastShapelet* algorithm and Euclidean-distance-based one nearest neighbor on 45 datasets from UCR archives

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

3.5.4. Parameter Effects

Our algorithm has two parameters that must be set by the user. They are *r*, the number of iterations of random projections, and *k*, the size of the set of potential candidates that are "promoted" from the SAX space back to the raw data space to be tested (lines 13 to 23 of Table 7).

Intuitively, when the number of iterations of random projections is increased, the process should make the set of potential candidates more tolerant to noise and reduce over-fitting. Likewise, when the set of potential candidates is larger, the quality of final shapelet should be better because we pick from a larger set. These two parameters also (approximately) linearly affect the running time of our algorithm.

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

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

The effect of parameter r and k on all datasets from UCR archives are shown in Figure 24. In this experiment, we vary one of the two parameters r and k from 1 to 50 and fix another one to 10; and we ran the experiments 30 times on all 45 datasets. Although the running is increasing linearly by both parameters, the accuracy is not sensitive to the value of parameters as shown in Figure 24.*top*.

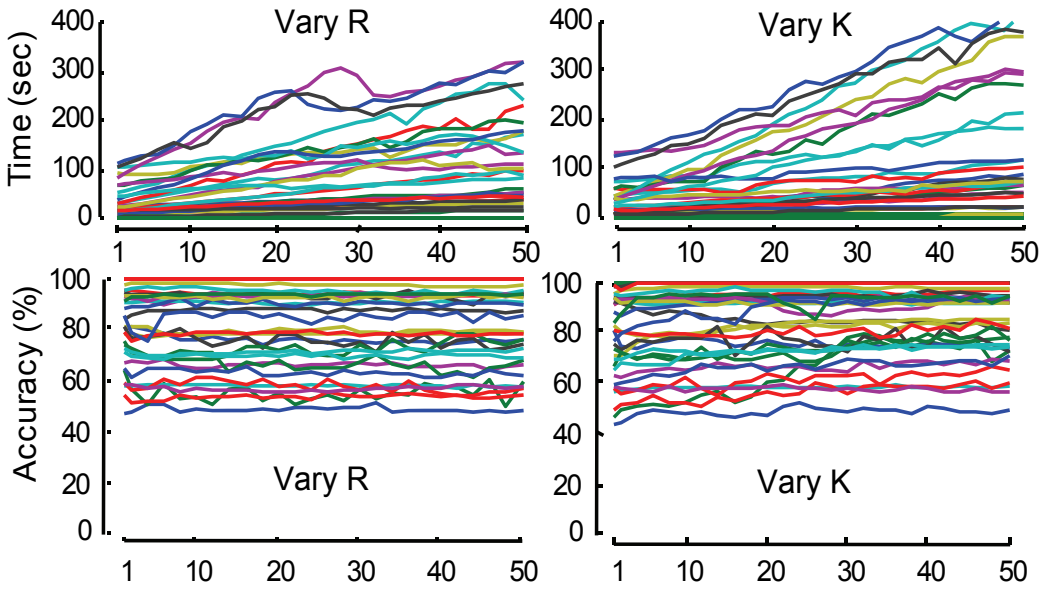

Figure 24: *bottom*) The accuracy of the algorithm is not sensitive for both parameters r and k. *top*) The running time of the algorithm is approximately linear by either parameter. Note that when we vary r (k), we fix k (r) to ten, thus we are changing only one parameter at a time

3.6 Case Studies

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

3.6.1. Starlight Dataset

The *StarlightCurve* dataset is the largest dataset in UCR time series archive. It contains 9,236 starlight curve time series of length 1,024. Three types of star objects that are *Eclipsed Binaries* (*EB*) 2,580 objects, *Cepheids* (*Cep*) 1,329 objects, and *RR Lyrae Variables* (*RR*) 5,236 objects are present. The dataset is divided to train data and test data of size 1,000 and 8,236 objects, respectively. Examples of starlight curves in each class are shown in Figure 25.

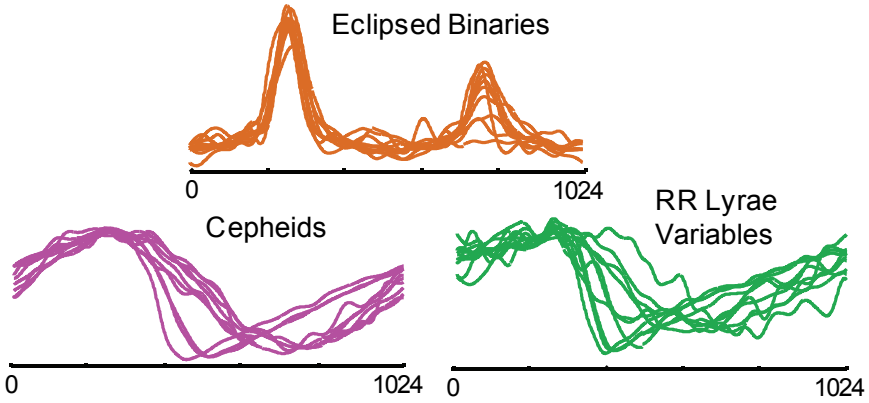

Figure 25: Examples of starlight curves in three classes: *Eclipsed Binaries*, *Cepheis*, and *RR Lyrae Variables*

Because the objects in *Cep* and *RR* are globally similar, these objects are difficult to separate. The accuracy of the one nearest neighbor algorithm using the Euclidean distance and DTW is 84.9% and 90.5%, respectively. However, using a shapelet decision-tree for classification, our *FastShapelet* get an average accuracy at 93.68% from 30 runs. Figure 26 shows a decision tree with three leaf nodes.

To the best of our knowledge this is the highest accuracy every reported on this dataset [54][61]. Moreover, of the hundred-plus papers that cited the UCR archive in the last 3 years (when this particular dataset was added to the Archive) a significant fraction of them do not report *any* results for this dataset, because their algorithm ran out of memory or time.

By interpolation, the current state-of-the-art is expected to take 4.5 months; however, our fast shapelet algorithm can create the decision tree in Figure 26 in 3,150 seconds (just under an hour). Thus the speedup is more than 3,000X on this dataset.

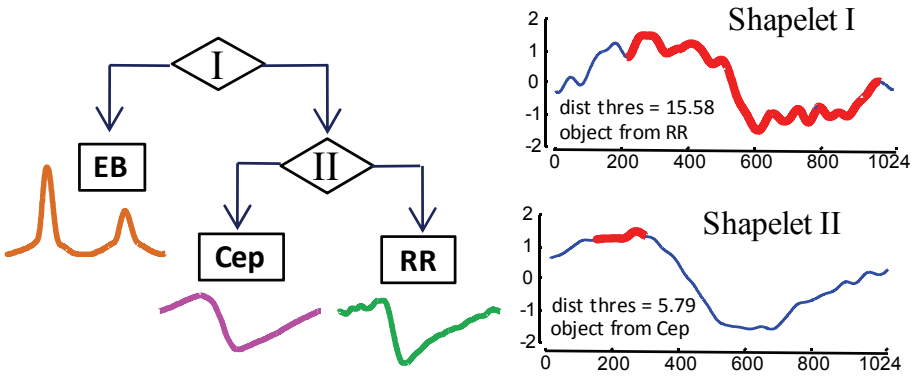

Figure 26: *left*) Decision tree of *StarlightCurve* dataset created by our algorithm. *right*) Two shapelets shown as the red/bold part in time series

3.6.2. Physical Activity Dataset

This section demonstrates that shapelet also can use as a high accuracy classification for activity recognition, an area drawing increasing attention due the increasing availably of inexpensive sensors. The dataset considered is from Physical Activity Monitoring for Aging People (PAMAP) [70].

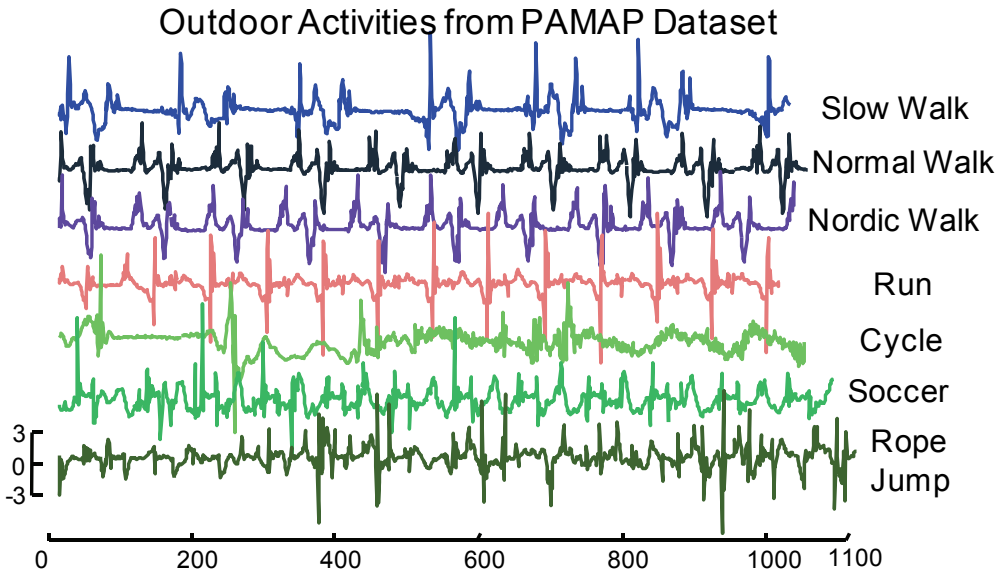

Figure 27: Examples of all outdoor activities from PAMAP dataset. Note that the time series of each activity are generally different lengths

The entire dataset contains 43 time series, which are collected from multiple sensors (e.g., accelerometer, gyroscope, magnetometer, etc.) in various places of body (i.e., hand, chest, and shoe). The

data consists of eight subjects (seven males, one female) taking part in various sporting activities. For simplicity we choose to use only one sensor among all 43 sensors of the original dataset to perform activity classification. Examples of time series generated by a z-accelerometer at hand position of all seven outdoor activities: *Slow Walk*, *Normal Walk*, *Nordic Walk*, *Run*, *Cycle*, *Soccer*, and *Rope Jump*, are shown in Figure 27.

In the original PAMAP dataset, all subjects perform all activities in one long performance so the data is a long time series that contains all (annotated) activities in sequence. We preprocessed the data using a by sliding window technique as a recommended the original authors [72].

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

The original authors of the dataset also created classifiers by using multiple time series [72] from *all* 43 sensors. Beyond the fact that they use all 43 sensors and we only use one, we cannot direct reproduce their experiments because of a lack of explicitness in how the data is processed. Nevertheless, they report that their specialist algorithms can obtain highest accuracy around 80% to 90%. Thus domain-agnostic shapelets using a single time series can be at least competitive with highly tuned, domain-informed specialist methods using all 43 time series.

3.6.3. ECG Dataset

The ECG Five Days dataset from PhysioNet.Org [70] is long ECG time series recorded on two different days with the same patient, a copy of this dataset can also be found at [61]. The dataset contains 890 objects, with 23 objects as train data and the 861 as test data. Example of the time series are shown in Figure 28. The main challenge in classifying this dataset is that the data exhibits linear drift (in medical domains this is called *wandering baseline*) as shown in Figure 28.*top* and the fact that the time series from two different classes are very similar, at least *globally*.

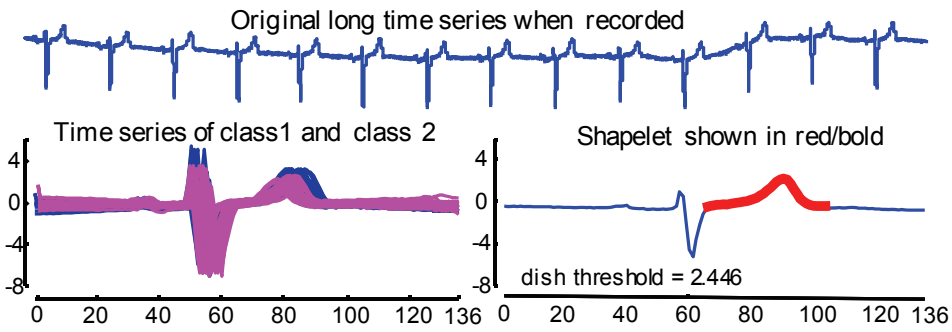

Figure 28: *top*) ECG time series when first recorded. *left*) Time series from two classes are very similar even hard to distinguish by eyes. *right*) the shaplet discovered by our algorithm shown in red/bold

Using one nearest neighbor classification with either Euclidean distance or DTW on this dataset, the accuracy is only 79.7%. However, the shapelet discovered by our proposed algorithm shown in Figure 28.*right* is able to obtain 99.4% accuracy from this dataset. This is, by a large margin, the best result ever reported for this dataset [54][61].

Moreover, the results are quite intuitive according to USC cardiologist Helga Van Herle, focusing our attention on the delayed t-wave which is the only medically significant difference between the two classes.

3.7 Conclusions

We proposed an algorithm for shapelet discovery that is up to three orders of magnitudes faster than the current state-of-the-art and yet has accuracy that does not significantly differ. We have made all our code freely available at [81], and as such hope to expand the scope of problems to which shapelets can be applied.

Chapter 4: Document Motifs

The increasing interest in archiving all of humankind's cultural artifacts has resulted in the digitization of millions of books, and soon a significant fraction of the world's books will be online. Most of the data in historical manuscripts is text, but there is also a significant fraction devoted to images. This fact has driven much of the recent increase in interest in query-by-content systems for images. While querying/indexing

systems can undoubtedly be useful, we believe that the historical manuscript domain is finally ripe for true *unsupervised* discovery of patterns and regularities. To this end, we introduce an efficient and scalable system that can detect approximately repeated occurrences of shape patterns both within and between historical texts. We show that this ability to find repeated shapes allows automatic annotation of manuscripts, and allows users to trace the evolution of ideas. We demonstrate our ideas on datasets of scientific and cultural manuscripts dating back to the fourteenth century.

4.1 Introduction

The world's books and manuscripts are being digitized at an increasing rate, and within a few years, the majority of the world's books will be online. Much of the data will be text, most of which is more or less amiable to optical character recognition. However, in addition, there will be perhaps hundreds of millions of pages that contain one or more images. It is clear that these images will be very difficult to process. Indeed, data mining of *modern* photograph images is challenging, and in the case of images from historical manuscripts the challenges are compounded by the problems of fading, staining, wear, insect damage, abrasions, foxing, pencil annotations, and distortion artifacts from the digitization process, etc. [105][106].

In spite of these challenges, it is clear that the wealth of figures from historical manuscripts offer unique possibilities for data mining of important cultural artifacts. While the completely automated extraction of data from these texts will remain a significant challenge for some time to come, in this work, we introduce a specialized sub-routine that *is* achievable and useful. This sub-routine is the automatic discovery of approximately duplicated figures, both *within* and *between* texts.

Our ideas can best be explained with a simple motivating example. In the early part of the 19th century, relatively inexpensive high-powered microscopes became available for the first time. This initiated an explosion of interest in Diatoms, a major group of eukaryotic algae, whose extraordinary shapes delighted and puzzled Victorian naturalists. Consider the two plates shown in Figure 29. They are typical

examples from the perhaps hundreds of books on Diatoms published during the Victorian era [104][109][113].

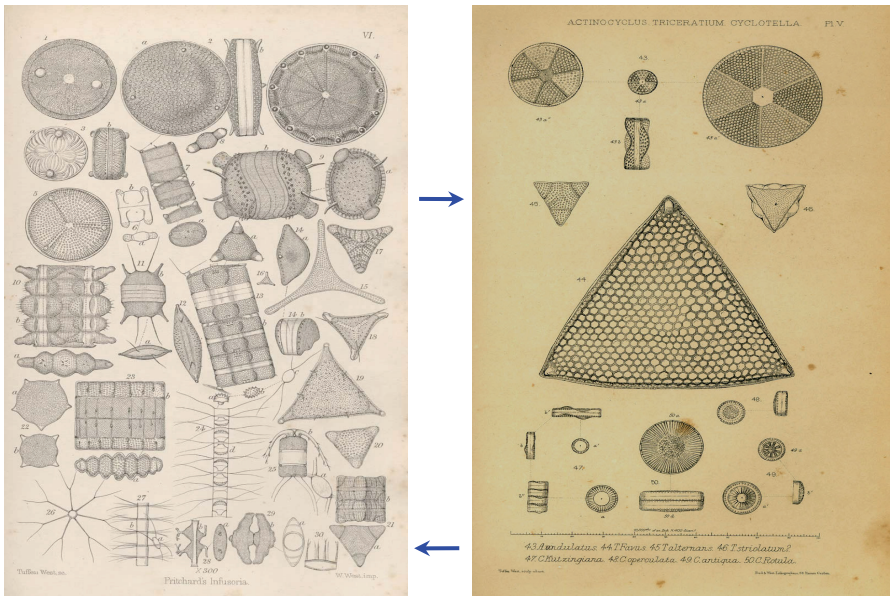

Figure 29: Two plates from 19th-century texts on Diatoms. *left*) Plate 6 of [104] *right*) Plate 5 of [109]. Note that in each plate we point to a triangular specimen, *Biddulphia alternans*

Thanks to efforts by digital archivists, hundreds of these works, representing over one million individual shapes, have been digitized and placed online. Some of them are scholarly classics, such as W. & G.S. West: *A Monograph of the British Desmidiaceae* [113], which is still referenced in modern scientific texts, and some of them are vanity publications by "gentlemen scholars". Figure 30 shows a zoom-in detail from each of the plates in Figure 29.

If we were hosting archives of Diatom images, we might wish to add mutual hyperlinks between each occurrence of *Biddulphi aalternans*, since any researcher with an interest in one, will surely have an interest in the other. Here we show just one pair of shapes deserving of a mutual hyperlink; however, in the domain of Diatoms, there are at least a thousand species defined by a unique shape, which could have all their occurrences linked together.

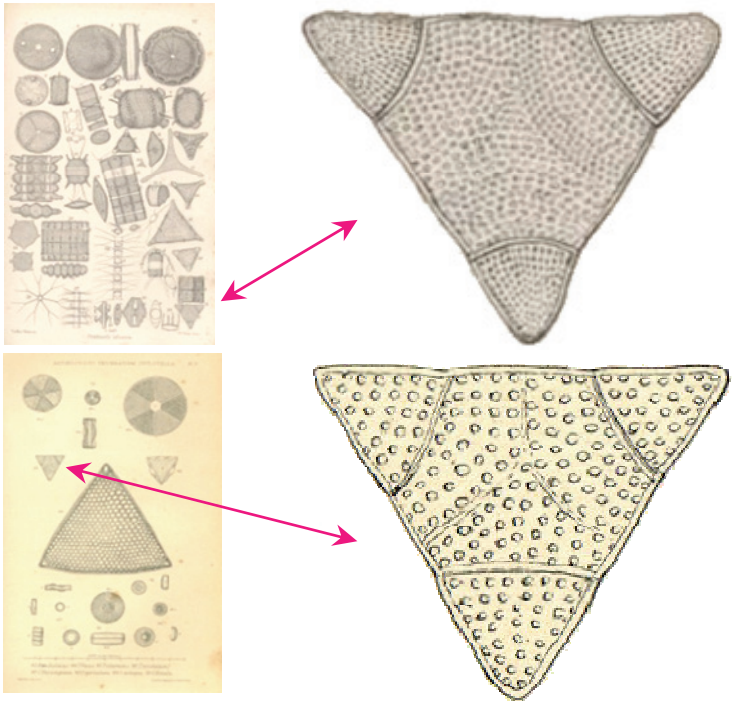

Figure 30: *left*) Two plates in Figure 29. *right*) A zoom-in of the same species, *Biddulphia alternans* appearing in both texts

Figure 31 shows another example where linking between two manuscripts is helpful (figure best viewed in color). The 1915 text (*left*) has much greater information and annotations about a medal, but the image is B/W. In contrast, the older text (*right*) clearly shows that the circular ring is *blue*. In this case, the *combination* of these two figures contains more information than either one on its own.

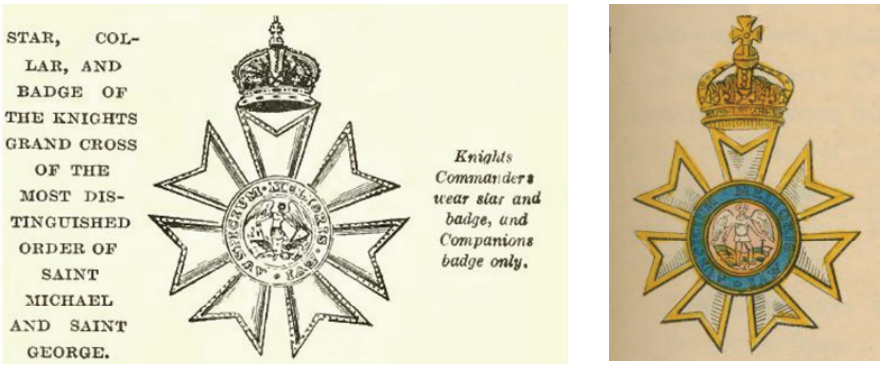

Figure 31: *left*) A figure from page 7 of [91], a 1915 text on peerage. The original text is monochrome. *right*) A figure from page 109 of [85], an 1858 text on honors and decorations

There are many other examples that demonstrate that linking two shapes within or between books can be of help to historians, genealogists or scientists. However, to the best of our knowledge, this problem has never been addressed. In this work, we demonstrate a technique to allow automatic discovery of repeated shapes in historical manuscripts. Beyond the obvious image processing challenges, and the problem of defining an appropriately robust distance measure, the biggest challenge is clearly *scalability*. Even if we have *only* 100,000 shapes in a collection, a brute force "all-to-all" algorithm would require approximately five billion distance calculations, an untenable proposition. Our algorithm is inspired by motif-discovery in bioinformatics [112], as DNA motifs can be considered as near-duplicate *strings*. As we shall show, it can (significantly) adapt bioinformatics algorithms to discover duplicated *shapes* in time linear to the number of "black" pixels in the text.

While there is a significant amount of related work in detection of near-duplicate images, we believe that none of it addresses the task at hand. The vast majority of such work is focused on matching *whole* images, after one of them has been distorted by cropping, resizing, color normalization etc. However, none of these works can locate near-duplicate *sub-images* inside the documents. For example, Chum et al. introduced an efficient method based on the technique of min-Hash to detect near-duplicate images with respect to the query image [87]. *Locality-sensitive hashing* is one of the best-known techniques to use in near-duplicate image detection [86][99], and we were inspired by this idea to develop our technique (Section 4.4). However, most of literature on near-duplicate detection is aimed at locating the *nearest neighbor* of the given queries. However, for our task we are not given a query, instead we want to automatically discovery similar pairs of figures.

By analogy the relationship between these two problems is similar to the much better studied problems in time series data mining [103], of finding the *nearest neighbors* of the query subsequences vs. finding the time series *motif* or the most two similar subsequences. In 2007, Xi et al.[115] proposed a method to find *image motifs* or the most similar pair of images in the image database. They transformed shapes into time series, which is robust, scale invariance and rotation invariance. However, all images are must fully pre-processed including background removing and image segmentation. In contrast, we make no such assumptions.

Image preprocessing such as image binarization [95][98] and image segmentation[84][96] is an important step in document analysis. Images segmentation can distinguish images and text in documents. In 2011, Grana et al. [96] proposed a novel technique to segment the images using *directional histogram* generated by the value of autocorrelation matrix, which is a summation of the relevant directions of the texture inside the block inside the documents. By the way, the quality of the result of our work could be improved by using these image segmentation techniques; however, the longer pre-processing time will be required.

The rest of this chapter is organized as follows. In Section 4.2 we introduce all necessary notations and definitions. We introduce an exact, but untenably slow algorithm in Section 4.3. Then, in Section 4.4 we show a very fast approximate algorithm that exploits novel observations and ideas from bioinformatics and image processing. Section 4.5 sees a detailed empirical study on real data. The theoretical analysis is introduced in Section 4.6. We offer conclusions and directions for future work in Section 4.7.

4.2 Background and Notation

We begin by introducing all necessary notations and definitions. These notations are illustrated in Figure 34.

4.2.1. Definitions and Notation

Whether we are mining archives of postcards, books, maps, etc., we can see our data source as bitmap *documents*:

Definition 1 A *document*, *D*, is a matrix with ternary values which are 1, 0, and -1. For any pixel in the given document that is black or white, we will set the corresponding point to be 1 or 0, respectively. The value -1 is reserved for the null or the area outside the original document. A document size *n* pixel by *m* pixel will be kept by using a matrix of ternary values size $n \times m$.

The historical documents may originally be B/W or color. For our purposes, we are only interested in shape, so as Definition 1 hints, we binarize all images. The binarization of images in the context of

historical manuscripts is a well-studied problem (see [95][98] and references therein) and a relatively easy task for most documents. Nevertheless, we clearly cannot guarantee perfect automatic binarization of large unstructured collections of manuscripts. As we shall show in Section 4.5, our solution to this problem is to use a distance measure and an algorithm that is robust to large amounts of noise and distortions.

Historical documents are often represented by a single surviving instance and many of them have imperfections. As shown in Figure 32, the corners may be burned or worn, or they may have holes due to insect damage [106]. We can support such documents by using null values for the area outside of the document but within the minimum rectangular boundary of document. Most professional scanners of historical manuscripts use a background color or texture that makes the occurrence of "holes" obvious.

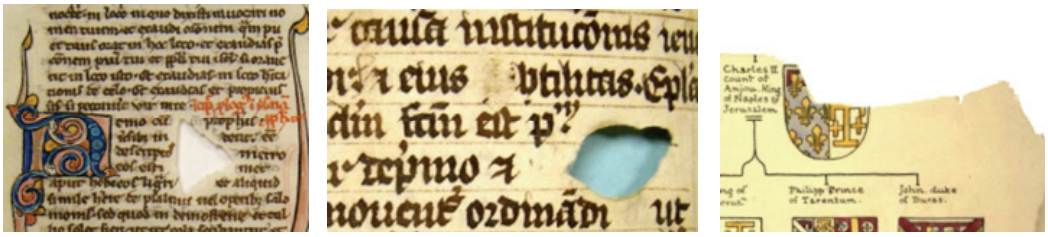

Figure 32: Examples of texts with "holes"

From the definition above, we regard documents as containing only a single "page". To apply our algorithm to many pages of a book or many books, we can simply concatenate each page into a long logical document and use a line of null values to separate one page from the others.

We do not expect (or want) to find globally repeated *pages*, so we confine our attention to small regions of interest within a page; these we call *windows*:

Definition 2 A *window* is a rectangular area inside a document whose size is specified by the user. It is defined by

$$W_{x,y} = \{d_{i,j} \in D \mid i \in [x, x + s_x) \; and \; j \in [y, y + S_y)\}$$

where s_x and s_y are a user-defined width and height.

The data inside the window is a ternary value just like the data in the document. While a document may contain many pages, no window is allowed to span two pages. For the rest of this chapter, we use the

term W_a interchangeably with the term $W_{x,y}$ if the starting position of window W_a is (x,y), i.e., in Figure 34, $W_a=W_{3,2}$.

There are many ways to measure the similarity between given images. However, most techniques make assumptions about the data. For example, geometric hashing [114] assumes the figures have well-defined points of interest, such as intersections, end-points, areas of maximum curvature, etc. However, even if these assumptions are true, this leads to the non-trivial sub-problem of locating the points of interest. This may be very difficult in our domain of interest. Other distance measures assume that the shapes are fully connected [82], or form closed contours [100]. However, as we shall see, neither assumption generally holds in our domain of interest. SIFT and it's variants (G-RIFT, SURF) make less assumptions, but even after tweaking their many parameters, they did not perform well on the problems in Section 4.5.1(to be fair, they are not designed for this domain), so we omit them from further consideration in this work.

Given the above, we need to use a distance measure which is robust to the inevitable noise/distortions we will encounter, and which is general enough to work without any explicit assumptions about the data. Furthermore, as we shall see later, our basic idea to speed up the discovery of repeated figures is to use a "hashing-like" idea from bioinformatics; thus, we need a distance measure that is amenable to hashing.

The recently introduced GHT distance [83][116] is just such a measure. We will explain this in more detail in Section 4.2.4.2.2 and Section 4.5.1. In the meantime, we use it to define the *distance* between any pair of windows as the following:

Definition 3 The *distance* between window W_a and W_b is defined as $dist(W_a,W_b)$. We use the GHT distance as a similarity distance between two windows. Thus, the distance is

$$dist(W_a,W_b) = \mathrm{GHT}(W_a,W_b).$$

As we can see in Figure 33, our distance measure is offset-invariant. As we shall see, this simple fact allows us to greatly speed up our search algorithm (especially for texts with a lot of "white space") by significantly reducing the number of distance comparisons needed. We will expand on this idea in Section 4.4.1.

Figure 33: The distance measure we use is offset-invariant, so the distance between any pair of windows, *left*, *center* or *right* above, is exactly zero. This simple fact can be exploited to greatly reduce the search space of motif discovery. Since a pattern from another book that matches one of the above with a distance X must match all with distance X, we only need to include any one of the above in our search

Recall that our task can be reduced to finding the most similar pair of windows in the document. However, a pathological solution to this would be to have two windows with high overlap, as in the windows W_c and W_d, shown in Figure 34.

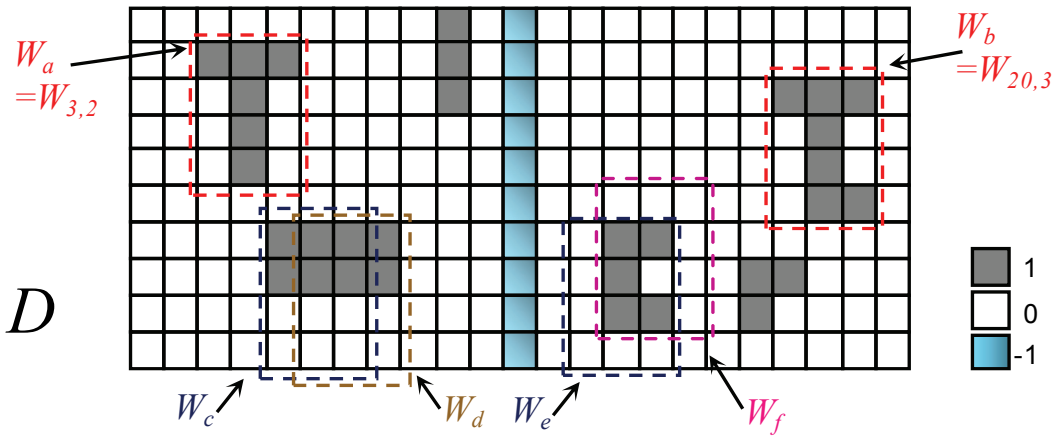

Figure 34: An illustration of our notation. Here the document D consists of two pages, separated by null values. Intuitively we expect the "T" shape in window W_a to match the shape shown in W_b. However, note that the trivial matching pair of W_c and W_d (also pair W_e and W_f) are actually more similar, and need to be excluded to prevent pathological results

We call a pair of windows a *trivial match* if its windows have a high degree of overlap.

Definition 4 A pair of windows $\{W_a, W_b\}$ of size $s_x \times s_y$ is a *trivial match* if its windows are overlapped more than α times the total area of a window ($0 \leq \alpha < 1$), formally,

$$(s_x - |b_x - a_x|) * (s_y - |b_y - a_y|) \geq \alpha * s_x * s_y \text{ and } (s_x - |b_x - a_x|) \geq 0$$

where (a_x,a_y) and (b_x,b_y) are the starting positions of windows W_a and W_b, respectively. If $\{W_a,W_b\}$ is not a trivial match, we call it a *non-trivial* match.

For the special case when α=0, no overlap between two windows of motifs is allowed; in this case, if windows W_a and W_b are share one pixels or more, $\{W_p,W_q\}$ will be a trivial match by the definition. In Figure 34, if we set α=0.5, both $\{W_e,W_f\}$ and $\{W_c,W_d\}$ are trivial matches but $\{W_a,W_b\}$ is a non-trivial match.

Among all possible pairs of windows, we want to find the pair that has the smallest distance between each other and is a non-trivial match. We call this pair the *motif window*:

Definition 5 A *motif window* (or just *motif*) is a non-trivial pair of windows $\{W_a,W_b\}$ such that the distance between windows W_a and W_b is the smallest of all other possible pairs.

$$motif = \{\{W_a,W_b\} \mid \min_{Wa,Wb} \text{dist}(W_a,W_b)\}$$

The definitions above assume that we are looking for exactly one near-duplicated figure. However, we can easily generalize this to allow the discovery of multiple motifs. In order to do so we must eliminate some pathological solutions, as shown in Figure 35.

To avoid redundant solutions in discovering multiple motifs, we will explicitly exclude *insignificant* motifs from our solution. We define an *insignificant* motif as the following:

Definition 6 A motif $\{W_a,W_b\}$ is *insignificant* if another motif $\{W_c,W_d\}$ exists such that

- $\text{dist}(W_a,W_b) \geq \text{dist}(W_c,W_d)$
- $\{W_a,W_c\}$ and/or $\{W_b,W_d\}$ are trivial matches.

The motif windows are insignificant if at least one of their windows shares a large part with other motifs. However, different motifs can share small parts with each other. Next, we define a top-*k* *motif window* as the following:

Definition 7 A top-*k* *motif window* is the set of *k* most similar pairs of windows, none of which is insignificant.

In Figure 35.*bottom*, only one true motif window is discovered, and the other one is insignificant.

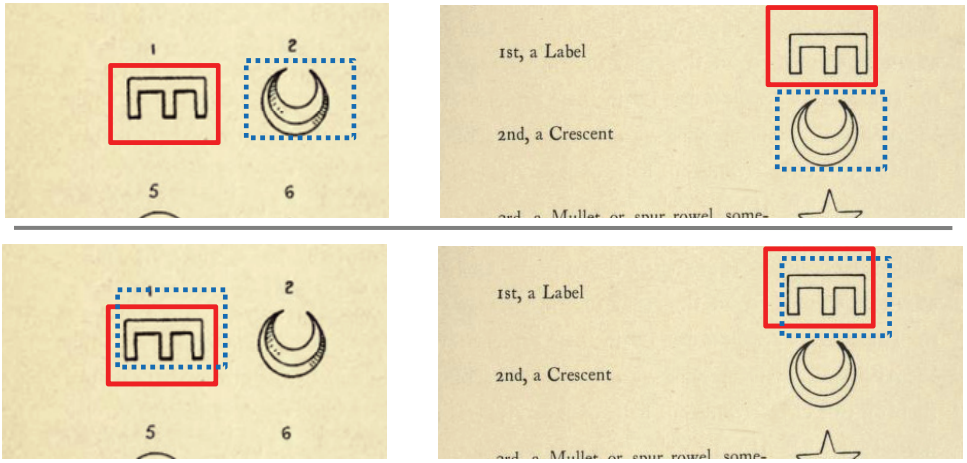

Figure 35: An illustration of a pathological solution to finding the top two motif pairs between two century-old texts. *top*) The desirable solution finds the crescent and label (rotated "E"). *bottom*) A redundant and undesirable solution that we must explicitly exclude is finding one pattern (the label) twice

4.2.2. Generalized Hough Transform

The Hough Transform [97] was introduced by Hough as a tool for finding well-defined geometric shapes (lines, curves, rectangles, etc.) in images [93]. The idea was generalized by many others, including Ballard, who introduced the Generalized Hough Transform to detect arbitrary shapes in images [83]. Computing the GHT distance between pairs of windows is relatively expensive. In particular, the time complexity for each GHT calculation is $O(n_b^2)$, where n_b is the number of black pixels in the window.

However, a recent paper by Zhu et al. [116] shows some computational tricks to reduce the amortized time for a *single* comparison, when a higher level algorithm requires *multiple* comparisons (i.e. clustering or query-by-content). In this work we use the ideas presented by Zhu, but as we shall see, they alone are not sufficient to provide the scalability we require in this domain.

There are two reasons why we chose to use the GHT distance for the problem at hand. Firstly, as shown by Zhu et al. and confirmed by our experiments, the measure is very robust and accurate [116]. Secondly, as we shall see, the method lends itself to being adapted to the random projection framework, which is used to solve the motif discovery problem in bioinformatics [112].

4.3 Exact Algorithm to find Motifs

Given a document D and (user defined) window size s, we want to find the top-k motifs in a given document. For simplicity, in the rest of this chapter we will explain only how to find the top-1 motif because the extension to top-k is trivial.

4.3.1. Brute Force Algorithm

We can easily find the top-1 window motif by comparing the distances from all pairs of windows, as shown in Table 8.

This simple algorithm uses nested loops (lines 3 and 4) to test all possible pairings of motif windows, checking whether or not two particular windows are trivial (cf. Definition 4), recording the one with the smallest distance. Unfortunately, this algorithm has an obvious flaw which makes it untenable for real problems: it will simply take a great deal of time even for a small document.

Table 8: Brute force algorithm

Algorithm: Brute force algorithm to find the top-1 window motif	
Input:	D : document $s_x s_y$: window size
Output:	*motif* : window motif
1	W = set of all windows of size $s_x \times s_y$ in D
2	$bsf = \infty$
3	**for** a = 1 **to** $\lvert W \rvert$
4	**for** $b = a+1$ **to** $\lvert W \rvert$
5	**if**(~IsTrivial(W_a,W_b))&&($dist(W_a,W_b) < bsf$)
6	$bsf = dist(W_a,W_b)$
7	$motif = (W_a,W_b)$
8	**end**
9	**end**
10	**end**

Assume that the document is size $n \times n$ and the user-defined window is size $s \times s$. The brute force algorithm must consider every pair of windows, requiring it to compute GHT distances $O(n^4)$ times. Each GHT calculation takes time $O(n_b^2)$, and n_b, the number of black pixels in a window, can be as large as s^2; it is usually larger than s. Hence, the total running time for the brute force is $O(s^2 n^4)$.

To give concrete numbers, suppose the original document is a B/W image of size 1 megabyte, or 1000×1000 pixel^2. The set of all windows, W, is approximately 10^6 (line 1). To find a motif using the brute

force algorithm, we need to compute the GHT distances about 5×10^{11} times (line 3-4). This would take approximately 10^8 seconds, or 3 years. Because the brute force algorithm cannot find motifs even in a small document in an acceptable amount of time, we will introduce a fast approximate algorithm for this task.

4.4 Our Algorithm

In this section, we introduce a sub-linear time motif discovery algorithm. We begin by giving the intuition behind three ideas that we will exploit to make our algorithm scalable. Later we will show a concrete algorithm that exploits these ideas.

4.4.1. Intuitions Behind Our Algorithm

A. Downsampling Helps Scalability

Our first observation was originally made to help improve scalability, but as a happy side effect, it also *greatly* improves accuracy. Figure 36 shows the effect of downsampling on our data of interest.

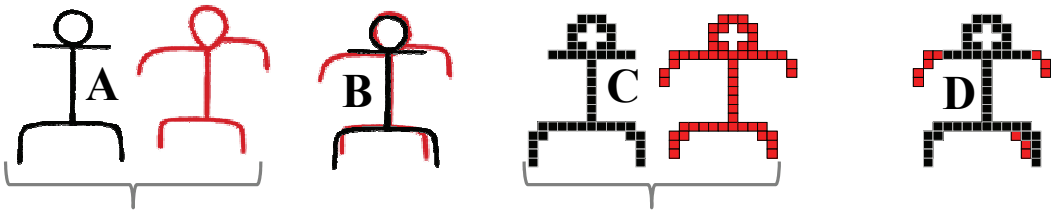

Figure 36: A) Two figures from table 16 of a 1907 text on Native American rock art [101] (one image recolored red for clarity). B) No matter how we shift these two figures, no more than 16% of their pixels overlap. C) Downsampled versions of the figures share 87.2% of their pixels as in (D)

Because downsampling will greatly decrease the number of windows that must be examined, it will clearly improve efficiency. It is natural to ask if this reduction in resolution will reduce the accuracy. The surprising answer is that the opposite is true; downsampling (except when taken to the extreme) actually improves accuracy by eliminating spurious precision and reducing the shape to its bare minimum. We note that we are not claiming this observation as an original contribution; Zhu et al. pointed this out and

demonstrated it with detailed experiments [116]. However, the next two ideas are original and unique to our domain.

B. *Random Projection Further Reveals Similarities*

While the downsampling idea introduced in the last section reduces both the time for a *single* distance calculation and the *number* of distance calculations that must be performed, the number of distance calculations required by the brute force algorithm is still on the order of $O(n^4)$. We have just drastically reduced the value of "n". In order to make significant progress on this problem, we need a much faster way to identify (potentially) very similar shapes. Figure 37 shows the intuition as to how we might achieve this.

Assume we have a pair of windows $\{W_a, W_b\}$ of size 17×14, containing two similar, but not identical figures, whose distance is equal to nineteen (i.e., $dist(W_a, W_b)$=19). For example, in Figure 37 we have two anthropomorphic examples of rock art with this property. Suppose that we randomly choose a single location, `x=randint(1:17)` and `y=randint(1:14)`, and set that pixel to white in both figures. What effect would this have on the distance? There are only two possibilities:

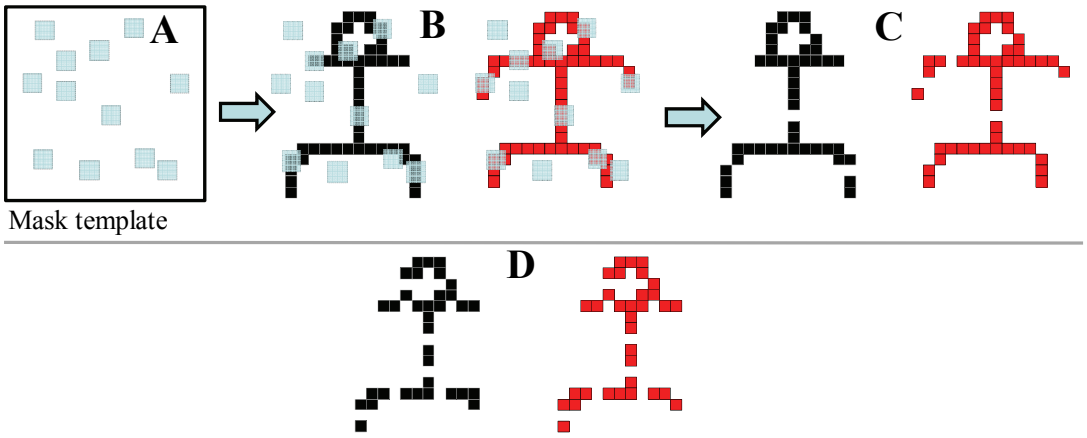

Figure 37: A) If we randomly choose some locations (masks) on the underlying bitmap grid on which the two figures (B) shown in Figure 36 lie, and then remove those pixels from the figures, then the distance between the edited figures (C) can only stay the same or decrease. Several random attempts at removing ¼ of the pixels in the two figures eventually produced two identical edited figures (D)

First, the corresponding pixels in the two windows are already either white or both black. In either case, the distance does not change. Second, exactly one of the corresponding pixels was black, and changing it to white must *decrease* the distance.

From this analysis, we can see that "deleting" black pixels (*randomly projecting* windows to a lower dimensional space [112]) must decrease or hold steady the distance between two objects. This is important, because if we manage to decrease the distance to zero, we can find such zero distance pairs in only linear (in the number of windows) time using *hashing*. This idea is inspired from the well-known hashing technique, min-Hash [87].

In the example shown in Figure 37, ignoring one pixel is clearly not enough to make the two figures identical; we actually need to remove at least 19. Furthermore, we need to remove the *correct* set of 19. A simple combinatorial calculation will convince the reader that this is very unlikely to happen if we choose 19 pixels at random. The obvious solution, to ignore *more* than 19 pixels, say 100, contains a problem. If we ignore too many pixels we will also allow two very different figures to hash together.

To some extent, this is a problem we can live with. Even if two very different figures are projected onto a very low dimensional space where they hash together as a *false positive,* we can later check their distance in the original space. The only danger is that if we have too many false positives, then checking them all may not be much faster than a brute force search.

At first blush the problem may seem insurmountable, because we have the extremely delicate task of making *all* similar things identical, without making (too many) different things identical. Fortunately, there is a solution to a nearly identical problem in DNA motif discovery in bioinformatics, which we can leverage off [112]. The idea (informally stated) is to be conservative in the number of pixels we remove, but to do multiple independent rounds of projection (*hashing*). This increases the number of true positives, while also reducing the number of false positives.

C. Numerosity Reduction Improves Scalability

The final observation we will make has already been hinted at in Figure 33. Even after downsampling, there are many windows that must be explored in order to find a motif (or top-*k* motifs). The number of

windows of size $s_x \times s_y$ in a document of size $n \times m$ is quadratic in terms of the document size, or more precisely, is $(n-s_x+1)(m-s_y+1)$, which is $O(n^2)$ when $n=m$. Naturally, all of these windows have a great deal of redundancy with their neighbors, and many windows may be totally blank or contain only a handful of pixels. Based on this observation, we can reduce the number of windows in the document dramatically by filtering out all but one representative example of a set of redundant windows, and also filter out windows that do not have enough black pixels to form any meaningful shape. We call the remaining windows in the document after this process, *potential windows*:

Definition 8 A *potential window* is a window whose number of black pixels is at least a threshold t and not less than other adjacent windows. Let $sum(\cdot)$ be the total number of black pixels in the particular window: $sum(W_{x,y})$. In the case of a null value, we can set $d_{i,j}$ to either 0 or 1. Then, the set of potential windows P is defined as

$$P = \{W_{x,y} \mid \forall a, b \in \{-1,0,1\}\ sum(W_{x,y}) \geq sum(W_{x+a,y+b}) \geq t\}$$

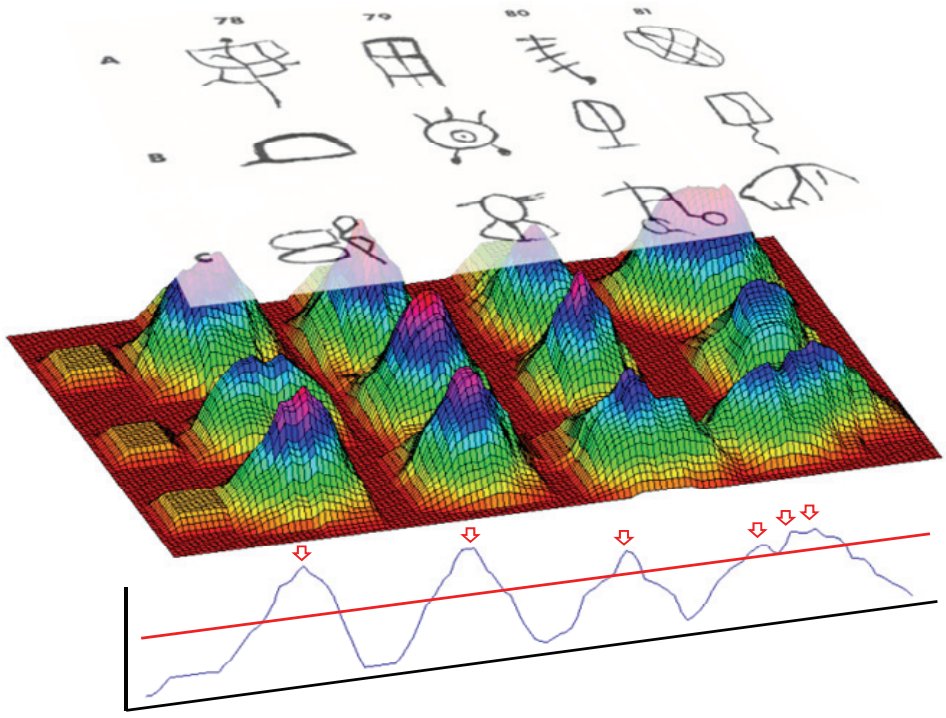

Figure 38: The summation of the number of black pixels in windows. Only windows corresponding to peaks above the threshold (the red line) need to be tested. The arrows show the center position of six potential windows

This idea can be visualized in Figure 38, where potential windows are centered on the peaks of a 3D heatmap. In the equation, we simply set the parameter t to an average number of black pixels in a specific window size, and the results show that our algorithm works well on this default value. Ties can be resolved by selecting just one potential window, changing the definition from "≥" to ">".

We note that this step also has an analogue in bioinformatics algorithms [22]. Many motif discovery algorithms do a preprocessing step of removing regions of low complexity DNA (for example, a long run of a single amino acid) to both speed up search and eliminate pathological solutions.

4.4.2. Document Motif Discovery

We are finally in a position to explain our algorithm and how it exploits the three ideas from the last section. In essence, we *downsample* the original book, extract all *potential windows* and hash them with *random projection*. All pairs that collide are inspected in the original space to see if they are true motifs. Our algorithm is described in Table 9.

Our algorithm uses four more inputs than the brute force algorithm. The first is the downsampling scale, ds, and the other three parameters are used in random projection.

In line 1, we downsample the original document D into the smaller version, DD, with the scale ds. While there are many algorithms for rescaling images, we simply downsample by majority voting the values inside $d_s \times d_s$ pixels in the original document to create a new pixel in the new document, DD. Hence, DD will be smaller than D by a factor of ds^2.

The next step is to locate all windows in the new document in line 2. Note that the total search space is reduced from $O(n^4)$ to $O(n^4/ds^4)$. Further note that in our implementation, we do not set W explicitly. We still need to further reduce the search space, so in line 3 we apply the third idea from the last section. In order to locate the potential windows (cf. Definition 8), for all windows we calculate the total number of black pixels inside that window. We can do this in linear time with respect to the number of pixels in the document. We then locate all potential windows which are at the local maxima of the summation plot, as

visualized in Figure 38. Now our search space is massively reduced; for example, in Figure 38, there are less than 30 potential windows among 22,000 original windows.

Table 9: Proposed Algorithm

Algorithm: *DocMotif*

Input: *D* : document; $s_x s_y$: window size; *ds* : downsampling scale; *it* : number of iteration; *hds* : hash downsampling scale; *mask* : masking ratio

Output: *motif* : window motif

```
1   DD = DownSamplingDoc(D,ds)
2   W = set of all windows of size sx×sy in DD
3   P = LocatePeaks(W)
4   P = AlignCenter(P)
5   HSig = Ø
6   for i = 1 to it
7     hsig = HashSignature(P, hds, mask)
8     HSig = HSig ∪ hsig
9   end
10  cand = CollidedWindow(HSig)
11  best-so-far = ∞
12  for all pair of windows (wa,wb) in cand
13    if(IsTrivial(wa, wb))
14       continue;
15    end
16    if(lb_dist(wa,wb) < best-so-far)
17       if(dist(wa,wb) < best-so-far)
18         best-so-far = dist(wa,wb)
19           motif = (wa,wb)
20       end
21    end
22  end
```

The number of peaks or potential windows is data dependent. It is possible that there are a lot of small peaks in the document. For example, as in Figure 38, there are 3 potential windows on the right which are created from the same symbol. We solve this problem in line 4 with a simple solution. We align every potential window by moving its center of mass to the center of the window. As a result of this process, potential windows may slightly change their position, so if several windows are aligned to the same position, we pick only one window at the position.

After all potential windows have been enumerated, we hash them using random projection (line 5-9). This idea is essentially the same as the one shown in Figure 37; we create a hash signature from each figure by randomly removing some pixels from them (line 7). Then, we find all pairs of windows that "collide"; that is, they share the same signature (line 10). To create the hash signature in line 7, we do two steps. First,

we *further* downsample all potential windows to a smaller size. The parameter controlling this is called *hds*, or hash downsampling scale. Then, we randomly remove some pixels, the number of which is controlled by the parameter *mask*. We do this random projection *it* times to increase the probability that two similar items will collide at least once. Note that only one collision is required to ensure we discover the motif.

In the last step, we create a set of all candidate pairs, *cand* (line 10). We can now calculate the true GHT distance of each pair, and the motif is a pair of windows that has the smallest distance, ignoring as always the trivial pairs (line 13). Lines 12 to 22 are essentially the brute force, but over a relatively tiny subset of the original possibilities. Because the GHT calculation is expensive, we avail of a recently published speed-up trick [116]. We calculate the GHT's *lower bound* first (line 16). If the lower bound is bigger than the *best-so-far*, we do not calculate the expensive GHT distance. Otherwise, we update the *best-so-far* using the GHT distance (line 18-19).

There is one trivial but very useful modification we can make to the algorithm. We can input *two* books instead of one, and insist (by adding an extra test on line 13) that the motif's two occurrences have one representative from each book. This is an idea hinted at in the first three figures in this work. We may see this as a *motif join* between two texts.

4.5 Experimental Results

We have designed all experiments such that they are reproducible, and as such, all data and code are freely available at [118].

In this section we wish to empirically demonstrate the following: that the GHT distance measure, operating on downsampled data, is appropriate for our domain; that our algorithm can find meaningful motifs in real data, and that this information is useful to domain experts; and that our algorithm is scalable enough to allow mining of large texts. Finally, we wish to show that although our algorithm has several parameters, these are easy to set, and their exact value is not critical to efficiency or effectiveness.

4.5.1. Sanity Check for the GHT Measure

While our experimental case studies (cf. Section 4.5.4.5.2) offer compelling anecdotal evidence that our method finds truly similar figures, it would be useful to see *objective* tests. To our knowledge, there is no benchmark dataset to test on; however, we can show the suitability of our ideas for objective tests in very similar domains/problems.

We would like the tests to demonstrate two things: that our choice of the GHT as a similarity measure in this domain is warranted, and that the extreme downsampling we perform to improve scalability does not hurt accuracy. We achieve these aims by testing on datasets of hand-drawn figures. Two of the datasets are from a collection of old music scores (17^{th}-19^{th} centuries) [94][107], and thus are very representative of our domain of interest, and the third one is a modern architectural symbol dataset [102], in which various users hand copied symbols, and is thus also very similar to the task at hand. As Figure 39 shows, these are non-trivial problems. In particular, the symbols in the first two datasets come from degraded texts, written by individuals who may have lived centuries apart and in different countries.

Figure 39: Samples showing the interclass variability in the hand-drawn datasets. *left*) Samples from the music datasets. *right*) Samples from the architectural dataset

Rather than fine-tune our method, we simply hard coded the downsampling to 20×20 for all datasets. Table 10 shows the one-nearest-neighbor leaving-one-out accuracy.

Table 10: The accuracy of GHT on 3 hand-drawn symbol problems

Dataset	# instances	# classes	Accuracy
Clefs	2,128	3	99.58%
Accidentals & Clefs	4,098	7	98.49%
Architectural	7,414	50	99.29%

While others have worked on these datasets, we did not directly compare our results to theirs. The published approaches on these datasets are so slow (an $O(n^3)$ warping method for the music symbols

[94][107], and an $O(n^3)$ adjacency grammar method for the architectural symbols [102]), that in both cases the authors abandoned any attempt at a full leaving-one-out on the entire dataset, and instead created various smaller subsets (hand crafted and thus difficult to meaningfully compare to). However, our accuracies are so close to perfect in every case that our claim is clearly demonstrated: the GHT on downsampled images is an effective distance measure for these kinds of images.

4.5.2. Motifs between Two Manuscripts

While there is undeniable utility in discovering motifs *within* a single text, the real power of motif discovery will undoubtedly come from the linking of two motifs *between* two or more apparently disparate texts.

Taryn Rampley, a Ph.D. student in anthropology at the University of California-Riverside, is interested in correlating DNA studies of peoples from the Americas with studies of cultural artifacts [108]. In particular, she is looking for evidence of cultural transmission from North America to South America prior to contact with Europeans. While this evidence might be found through jewelry, textiles, weapons or language, this researcher is focusing on petroglyphs (rock art), of which there are several million documented examples in the Americas.

This student gave us a classic reference text on Californian petroglyphs [110], which includes a 104 page petroglyph catalog, containing about 2,852 individual examples of petroglyphs. We scanned this text with an off-the-shelf scanner. Figure 40.*left* shows two representative pages.

Thanks to the Google Book Project, the web is replete with possible texts with which to compare. One such text that caught our attention is a 1907 text by the German ethnologist and explorer Theodor Koch-Grünberg (1872–1924) which discusses the origin and significance of rock art in South America [101]. This text contains 233 images of petroglyphs hand-traced by the author. Figure 40.*right* shows two representative pages.

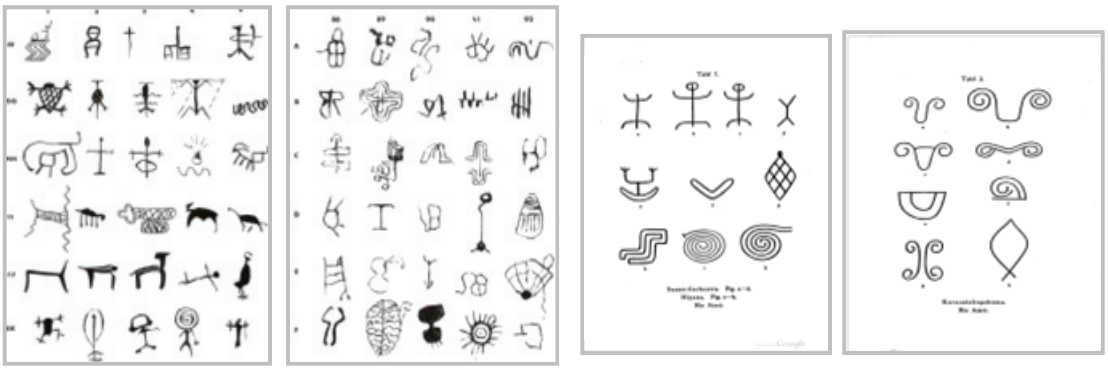

Figure 40: *left*) Two typical pages from Californian petroglyphs [110]. *right*) Two typical pages from [101]. Note that the minor artifacts are from the original Google scanning

We ran our motif join algorithm on these two texts; Figure 41 below shows a selection of the top fifty results.

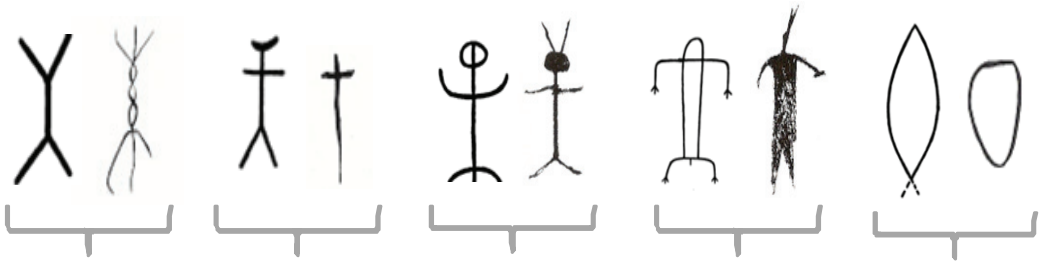

Figure 41:. Five random motif pairs from the top fifty pairs created by joining the two texts [101] and [110]. Note that these results suggest that our algorithm is robust to line thickness, solid vs. hollow shapes, and various other distortions

While the figure pairs are clearly *somewhat* similar, the anthropologist does not feel that this provides evidence of cultural transmission. If we repeat the experiment by comparing the reference text to petroglyphs from Arizona or Utah, the joins are much more similar. Currently, these conclusions are subjective and tentative; in ongoing work we are working with anthropologists to produce a principled theoretical framework for drawing such conclusions. While we defer detailed scalability results to the next section, we note that this join took approximately one minute.

Before moving on, it is worth re-examining Figure 41 to note the invariances that our algorithm has achieved. For example, in the Figure 41.*middle* our algorithm discovered a pair of anthropomorphic figures in spite of the fact that one has a solid head and antenna. To appreciate *why* we can achieve such

invariances we invite the reader to review Figure 36 and Figure 37, whose examples we drew from one of these texts [101].

Such robustness is critical if we are to investigate hand drawn texts in addition to the printed texts we consider next. As part of another project on mining cultural artifacts we are also interested in mining the vast literature on genealogy and heraldry that dates back to the 12th century [89][90]. Figure 42 shows a typical result in this domain.

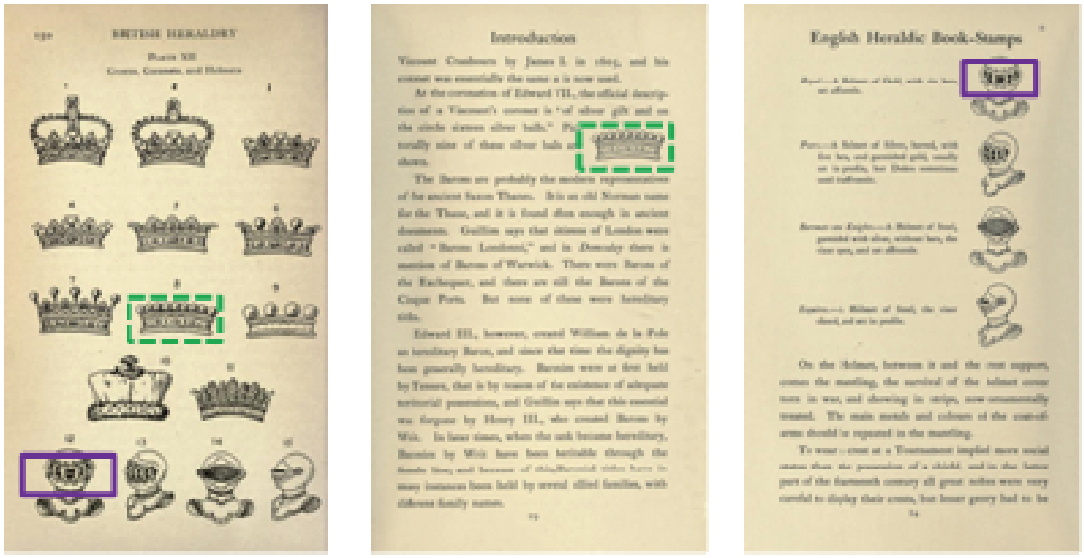

Figure 42: The top two inter-book motifs discovered when linking a 1921 text, British Heraldry [89] (*left*), with a 1909 text, English Heraldic Book-Stamps, Figured and Described [90] (*center*), and (*right*)

In order to make a point about some invariances our distance measure achieved in this domain, Figure 43 shows a zoom-in of the two pairs of discovered motifs shown in Figure 42. Note that in both cases the two members of each motif differ slightly in scale. This is presumably due to differences in the scanning process, since it is likely that the images were produced by the intaglio process, and printed from the same plate. In any case, our method is robust to such minor scale changes.

Note also that the figures are not identical; for example, the helmet in the later text has additional shading on the right side of the dome and under the chin. Again, our method is robust to this issue.

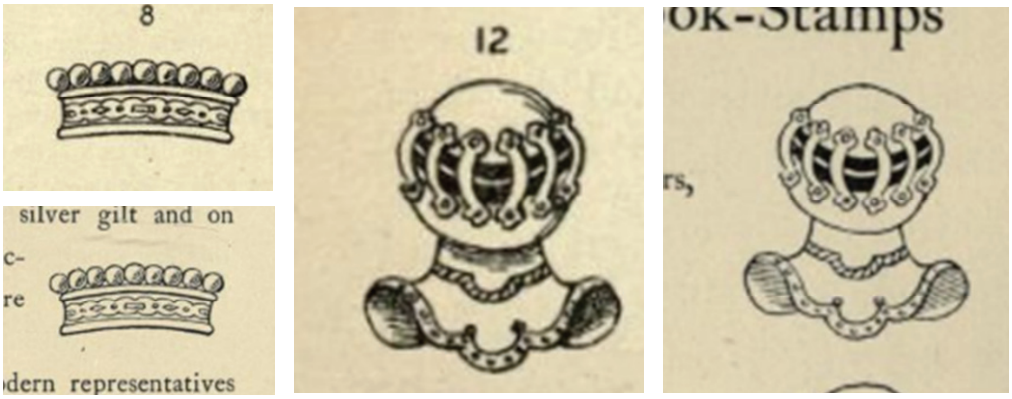

Figure 43: A zoom-in of the motifs discovered in Figure 42. Note that the two helmets differ in size by about 11%, and our algorithm was invarient to this difference

However, the most interesting point about this example is the (relative) invariance to the user-specified size parameter. Note that as shown in Figure 43, we set a window size and aspect ratio that happens to be perfect to enclose the crown. To enclose the helmet, we really need a window size that is about twice as large and with a more vertical aspect ratio. Nevertheless, in spite of a suboptimal window size we still found the helmet motif. This is not a one-off fortuitous occurrence, but generally true (see additional examples at [118]). So long as the user-supplied size is within a factor of two or so of the motif size, we will robustly find it. If the uncertainty in size is greater than a factor of two, our algorithm is efficient enough to allow range-doubling search.

Because our algorithm can discover motifs *between* different books, it is of utility in locating similar patterns in different books, and combining the information between those books such as shape, texture, color, etc., and filling in missing details.

The following example demonstrates that motifs can help us to flesh out some missing data. In 1863, *A Manual of Heraldry, Historical and Popular* [88] showed the heraldic shield of King George III of England after year 1801 and his successors, George IV and William IV, as in Figure 44.*left*. However, later in 1913, *Leopards of England* explained that King George IV and King William had changed the arms a little as shown in Figure 44.*middle*, "*Fourteen years later the Congress of Vienna erected the electorate of Hanover into a kingdom, whereupon the elector's hat was changed into a royal crown*, … *until the death of the last English king of the house of Brunswick in 1837*"[92].

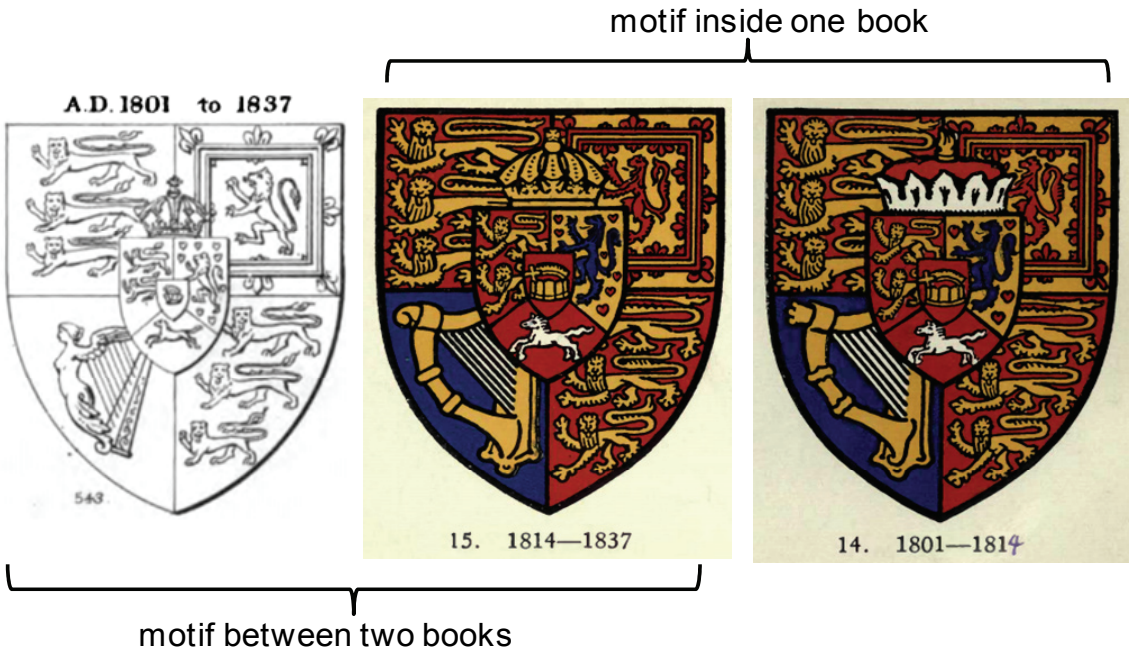

Figure 44: (*left*) Arms of King George III and his successors from *A Manual of Heraldry, Historical and Popular*, 1863 [88]. Two similar arms are explained in *Leopards of England,* 1913 [92]. (middle) Arm of King George IV and his successor's King William IV. (*right*) Arms of King George III after the constitutional change

Thus by finding motifs within one text [88], and between two texts [88] and [92], we can automatically interpolate the missing color information in an monochromic figure.

In the next section, we show that the efficiency and accuracy of our algorithm are largely invariant to parameter choice.

4.5.3. Scalability and Noise Tolerance

Testing the scalability of our approach on real data provides us with significant challenges, since the running time of our algorithm depends on the data. For example, suppose a book has a *perfect* motif on pages 1 and 51, but otherwise there are no significant repeated patterns. The time to search a subset consisting of the first 50 pages would be much *greater* than the time taken to search the first 100 pages, since the latter would encounter a high quality *best-so-far* early on. Given this, we test the scalability on an

artificial book over which we have perfect control. We made every effort to make a realistic book, but when in doubt we made choices designed to strain our algorithm.

We generated an artificial book using the idea of a 14-segment display that be used to create any English alphabetical character or digit. Figure 45.*left* shows some samples. In our artificial book, each page contains a random selection of 100 characters and the size of each page is 1330x1220 pixels, as shown in Figure 45.*middle*. While it is very unlikely that any random character would be created twice, such an occurrence would greatly favor our algorithm. We therefore further distort the book by two methods: adding a random polynomial warping (modeling a distortion caused by non-contact scanning) to the pages and adding some Gaussian noise, as shown in Figure 45.*right*.

Figure 45: *left*) The 14-segment template used to create characters. We can turn on/off each segment independently to generate a vast alphabet. *middle*) An example of a page which is generated from the process. *right*) A page of the book after adding polynomial distortion (*top half*), and Gaussian noise with mean 0 and variance 0.10 (*bottom half*)

In order to set the parameters for our experiments, we did the following: we created a two-page "book" and spent less than five minutes "playing" with it to find reasonable parameter values. Once we had found these values, we fixed them for *all* data sizes up to 2,048 pages.

As we can see in Figure 46, our algorithm can find the top motif in a 128-page book in a minute and in a 2,048-page book in half an hour. Note that these times are close to the time taken to scan (at least valuable) books of this size, so they are not unreasonable.

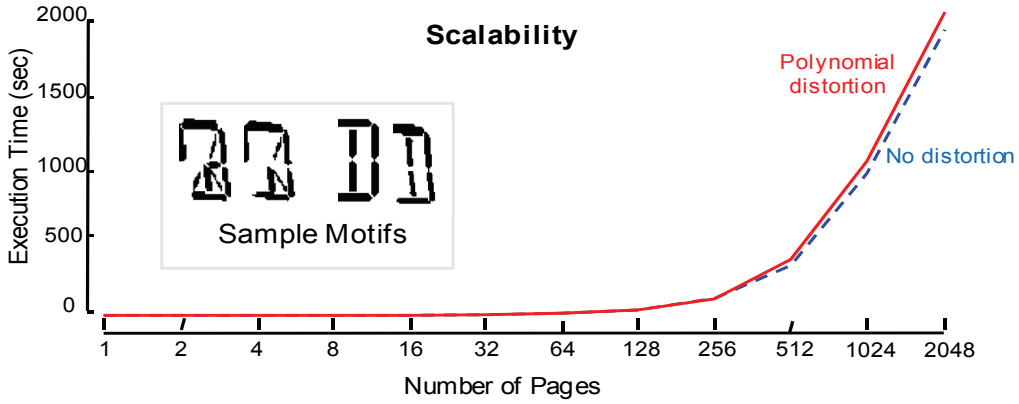

Figure 46: Time to discover motifs in books of increasing size. Our algorithm can find a motif in 512 pages in 5.5 minutes and 2048 pages in 33 minutes. (*inset*) As a sanity check we confirmed that the discovered motifs are plausible, as here (noise removed for clarity)

Note that in this figure and some figures to follow, some lines are difficult to tell apart; however, *this is the point* of these experiments: to show that our algorithm is not sensitive to distortions/noise/parameter choices.

We also test the noise tolerance of our algorithm by generating an artificial book with Gaussian noise added. The mean of the Gaussian noise is set to 0 and its variance is varied from 0 to 0.20. The results in Figure 47 show that our algorithm can tolerate significant noise (*var*=0.15).

When the book contains too much noise (*var*=0.20), the number of potential windows will increase because it is difficult to align all potential windows from a figure into the same position. Hence, the running time increases. However, this case corresponds to a *very* heavily degraded image.

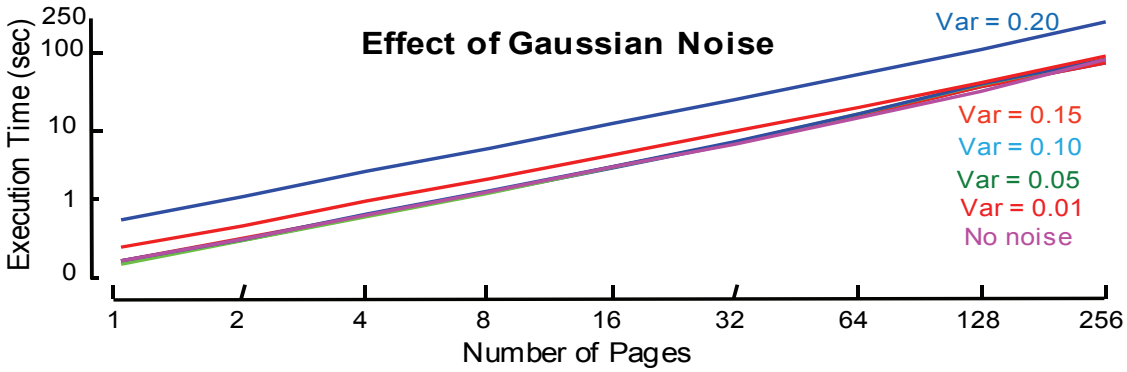

Figure 47: Effect of Gaussian noise. Our algorithm can handle significant amounts of noise. An example of a page containing noise at var=0.10 is shown in Figure 45.*right*

To concretely ground the amount of speed-up our algorithm can achieve we did the following experiment. On a 512-page book, we compared the running times of:

1. Exact motif search over the entire document by applying motif discovery technique in [103]
2. Exact motif search over just the potential windows
3. Our proposed algorithm, *DocMotif*.

The results are shown in Figure 48. We can see that the running time of heuristic search from [103], which is *much faster* than brute force search, rapidly becomes untenable, taking, for example, more than a day for just 8 pages and (an estimated) six months to finish all 512 pages. Our simple trick of only searching over potential windows reduces the search time to just 6.9 hours for the full 512 pages; however, our proposed algorithms take a mere 342.4 seconds.

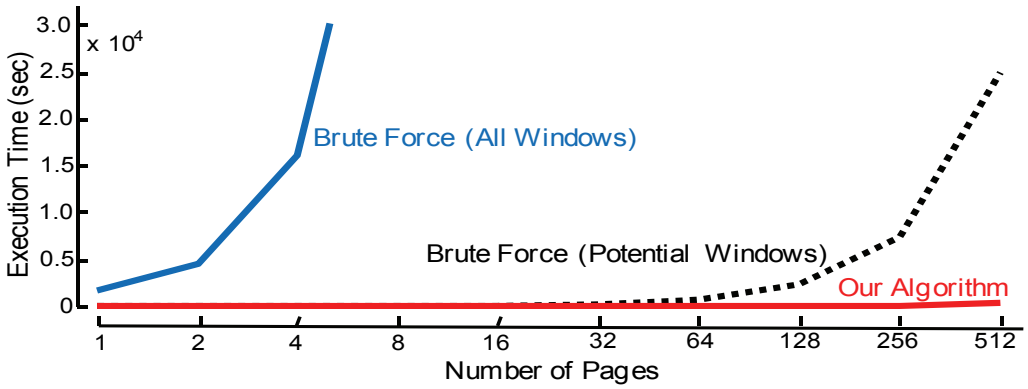

Figure 48: The total execution time of three search algorithms: an exact motif search, an exact motif search on just the potential windows, and our algorithm *DocMotif*

4.5.4. Robustness of Parameters

We have an obligation to explain how the choice of parameters affects the speed of motif discovery and the quality of motifs. As we shall see, our algorithm is *not* particularly sensitive to parameter choice. Recall that in the previous sections we have set the parameters based on a few minutes' experience with a two-page sample. Our simple test for parameter sensitivity is to hold three parameters firm, and adjust the other parameters to higher and lower values, to see what effect this has. Figure 49 tells us that for the most part, the algorithm's performance does not rely critically on parameter choices. Of course, this dramatic

speed-up would be worthless if the faster algorithms produced inferior results. However, as we shall show *empirically* in Section 4.5.4.5.2, the results of all algorithms are virtually identical.

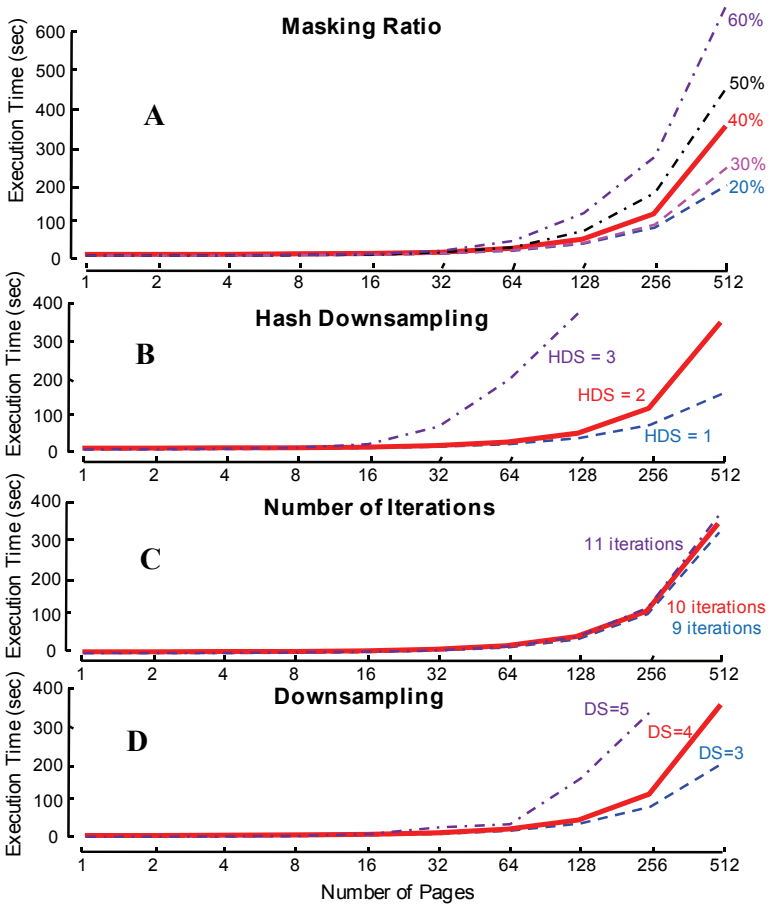

Figure 49: The effect of parameters on our algorithm. We test on artificial books with polynomial distortion and each result is averaged over ten runs. The bold/red line represents the parameters learned from just the first two pages

In the random projection process, the length of the hash signature is affected by two parameters, which are the hash downsampling scale *hds* and masking ratio *mask*. When the signature is shorter, the probability of collisions increases (including false positives that must be checked and dismissed). Thus, when we remove more pixels from a window by increasing *mask*, the running time will increase, as shown in Figure 49.A. As we can easily see in Figure 37, if we continuously remove pixels, eventually all windows will collide to the same shape (with pure white or no content left).

Similar to *mask* which has a linear effect, *hds* has a quadratic effect on the length of the hash signature, so if we change it, the running time may change significantly as shown in Figure 49.B. Recall that *hds*=2,

meaning that we condense 4 pixels to just one. The number of iterations has very little effect (Figure 49.C), which is also true for the bioinformatics algorithm that inspired us [112].

The downsampling scale *ds* parameter (cf. Figure 36) can reduce the search space and increases the quality of the motifs by allowing a greater invariance to noise. Figure 49.D shows that if we fix all other parameters and vary this parameter, the total running time will increase as *ds* increases. When *ds* increases, the downsampled document, *DD* (cf. Table 10), contains fewer pixels and also less information to represent any figure; so after random projection there will be more spurious collisions, increasing the number of false positives that must be checked and thus increasing the running time.

As with any approximate algorithm, the quality of the result is important. Hence, we calculate the *quality* of top-20 motifs by using their total distance. Figure 50 shows the average distance for different parameters values, compared to the exact search algorithm.

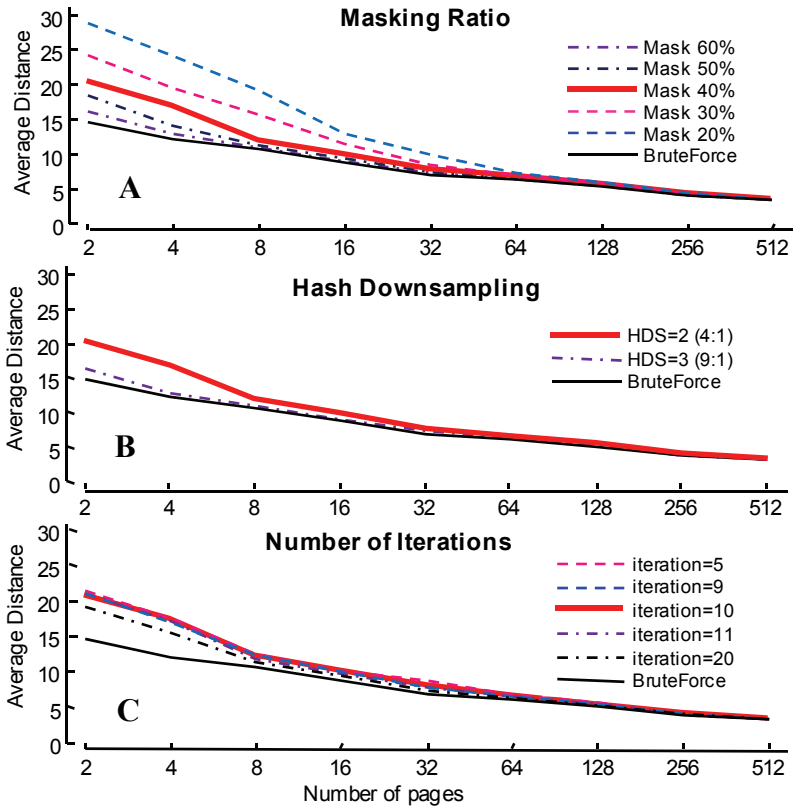

Figure 50: The average distance from top-20 motifs from our algorithm and the exact search algorithm. The bold/red line shows the default parameters. This shows that the quality of motifs is *not* sensitive to different parameter settings and very close to the result from the exact search algorithm

Here the *quality* of the top twenty motifs is simply the sum of all twenty distances of each motif pair (i.e., Definition 3). As Figure 50 shows, the quality of *DocMotif* is very good under *any* parameter setting, even for small books, but as the size of the book increases, the results are essentially indistinguishable from the exact search, which takes about 67,500 times longer.

4.5.5. Data Mining Palm Leaf Manuscripts

We conclude by noting that our algorithm is currently being evaluated for mining massive (four million leafs) archives of palm leaf manuscripts such as the one shown in Figure 51, for medical knowledge.

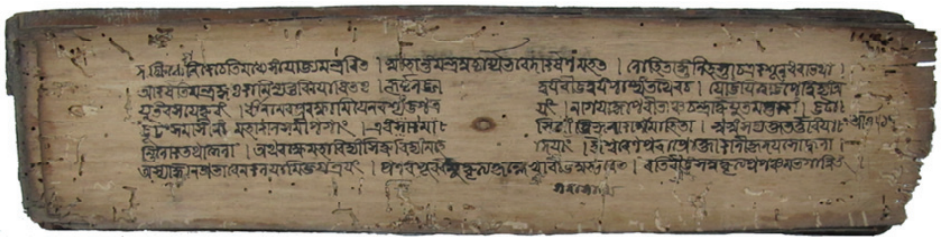

Figure 51: An example of a palm leaf manuscript

Figure 52 shows six motifs discovered from a 52-page palm leaf manuscript. In addition to discover similar *figures* from manuscripts, these examples demonstrate that our algorithm also work well on discovering motifs in *handwritten* documents. Note that we did not do much on image processing such as text line detection or text segmentation, except image binarization. However, the high quality motifs in a handwritten document are achieved as shown in Figure 52.

Figure 52: Six example motifs from a palm leaf manuscript. The window size is set to 30×100 pixel^2

Because it is an interesting and visual example, we present this example as a one-minute long YouTube video [117]. The video demonstrates the speed, robustness and accuracy of our algorithm, even in the face of complex and degraded texts.

4.6 Theoretical Analysis

In this section we briefly introduce some results that make much of the discussion of parameter setting in the previous section moot. In essence, we show that, given some very mild assumptions, we can simply *derive* the best parameters to use, given just the user-required confidence in finding the true motif. Concretely, if an end-user wishes to find the true best motif in a text, with a confidence *conf* (*conf* is the probability that the returned motif is the one the brute force search would have returned), she can use the following results to find the appropriate parameters to use.

In this analysis, we assume that there is only one motif in the document with distance d and mean and standard deviation distribution of the distance between each pair of windows μ and σ, respectively. The window size is $N=s_x \times s_y$. Note that if there is more than one motif in the given dataset, we still get the same result because the content (or location of black pixels) in each non-related window are independent.

Theorem: If two windows of the motif collide with confidence at least *conf*, the probability that any pair of windows will collide at most:

$$it * mask^{u-1} + 2 * it * \sigma^2 mask^{\mu+1} \sum_{i=0}^{2d} 1/(mask^i i^3)$$

where the masking ratio *mask* can be defined by:

$$\text{mask} \geq \frac{1}{N}\left[\left(1 - \ (1 - conf)^{1/it}\right)^{1/d} * (N - d + 1) + d - 1\right]$$

The detail of mathematical proof is provided in Appendix. We can use these results to find the optimal set of parameters in a four dimensional space. To give the visual intuition of this in one dimensional space, we can hold 3 parameters fixed at reasonable values, and use the above theorems to plot the number of false positives created (hence, the *time* taken) vs. the value of the free parameter. In Figure 53.*top* we allow the masking ratio to vary, and in Figure 53.b*ottom* we allow the number of iterations to vary.

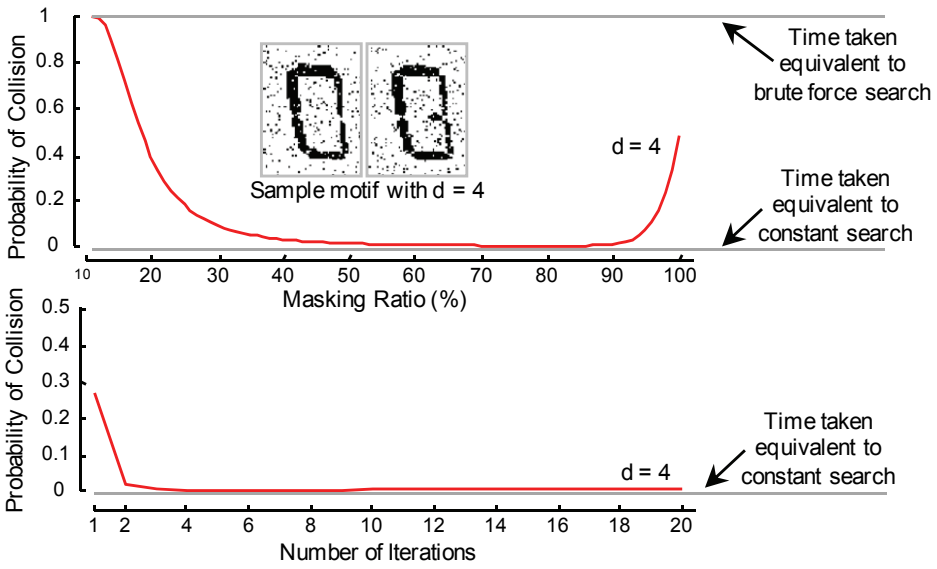

Figure 53: The effects of masking ratio (top) and the number of iterations (bottom) parameters on the spurious collision ratio, Given there is least one motif with a distance d in the data. The figures for other values of d are at [118]. Here we fixed μ=100 and σ=10

These results bode very well for our algorithm. In the first case, they tell us that if the masking ratio is anywhere from about 60% to 90%, our algorithm will produce very few false positives that need to be eliminated, thus giving our algorithm essentially sub-linear time performance.

Note that in the latter case, the minimum value is at 4. After that, the cost very slowly rises, because we have found (with *very* high probability) the true motif, and the additional iterations have a small overhead while contributing nothing to the speedup.

We have tested these theoretical results with experiments, and found that our theoretical model is accurate, but slightly conservative. In other words, setting the parameters is even easier than predicted here.

4.7 Conclusions

We have said little about related work thus far because there is little that does exactly what we propose. Xi et al. do consider "*Finding Motifs in a Database of Shapes*" [115]. However, they make two critical assumptions that are not true in our case (or in general): that the individual shapes can be perfectly

extracted, and that all shapes can be represented by a closed contour. Of course, our work does borrow heavily from the huge literature on motif discovery in bioinformatics; see [112] and the references thereof. Likewise, we have exploited both the classic ideas of the GHT [83], and the recent extensions by [116]. There is an active community working on computerized historical document analyses [95][98][105]: however, while great many papers address *query-by-content*, including [94][102][107] the task of motif discovery in this domain has not been addressed thus far.

We have shown the first general technique for the unsupervised discovery of repeated patterns both within and between texts. We have demonstrated that our algorithm is scalable, it can produce meaningful results that are useful to domain experts, and its parameters are easy to set. In future work we plan to integrate our ideas with (OCRed) text mining algorithms, and leverage off very recent theoretical results in bioinformatics to remove the need to set any parameters in our algorithm.

Chapter 5: Some Data Must Be Ignored

Time series data is pervasive across all human endeavors, and clustering is arguably the most fundamental data mining application. Given this, it is somewhat surprising that the problem of time series clustering from a single stream remains largely unsolved. Most work on time series clustering considers the clustering of *individual* time series that have been carefully extracted from their original context, e.g., gene expression profiles, individual heartbeats or individual gait cycles. The few attempts at clustering time series *streams* have been shown to be objectively incorrect in some cases, and in other cases shown to work only on the most contrived synthetic datasets by carefully adjusting a large set of parameters. In this work, we make two fundamental contributions that allow for the first time, the meaningful clustering of subsequences from a time series stream. First, we show that the problem definition for time series clustering from streams currently used is inherently flawed, and a new definition is necessary. Second, we show that the Minimum Description Length (MDL) framework offers an efficient, effective and essentially parameter-free method for time series clustering. We show that our method produces objectively correct results on a wide variety of datasets from medicine, speech recognition, zoology, gesture recognition, and industrial process analyses.

5.1 Introduction

Time series data is pervasive across almost all human endeavors, including medicine, finance, science, and entertainment. As such it is hardly surprising that it has attracted significant attention in the research community [119][122][143][150]. Given the ubiquity of clustering both as a data mining application in its own right and as a subroutine in other higher-level data mining applications (i.e., summarization, outlier discovery, rule-finding, preprocessing for some classification algorithms etc.), it is surprising that the problem of time series clustering from a *single* time series stream remains largely unsolved, in spite of significant efforts by the community [119][122][150]. Most work on time series clustering considers the clustering of individual time series that have been carefully extracted from their original context, say, gene

expressions or extracted signals such as individual heartbeats. The few attempts at clustering the contents of a *single* time series stream have been shown to be objectively incorrect in some cases [136], and in other cases shown to work only on the most contrived datasets by carefully adjusting a large set of parameters. In this work, we make two fundamental contributions. First, we show that the problem definition for time series clustering from streams currently used is inherently flawed. Any meaningful algorithm must avoid trying to cluster *all* the data. In other words, the subsequences of a time series should only be clustered if they are clusterable. This seems to open up a "chicken and egg" paradox. However, our second contribution is to show that the Minimum Description Length (MDL) framework offers an efficient, effective and essentially parameter-free solution to this problem. MDL has had a significant impact in bioinformatics and data mining of discrete objects such as natural language [133], but has yet failed to have a significant impact on real-valued data mining [132][140][146].

We begin by giving the intuition behind the fundamental observation that motivates and informs our work, that clustering of time series from a single stream of data requires ignoring some of the data.

5.1.1. Why Clustering Time Series Streams requires Ignoring some Data

The observation motivating our efforts to cluster time series is that any attempt that insists on trying to explain *all* the data is doomed to failure. To see this consider one of the most obviously "clusterable" time series data sources: motion-captured sign language, such as American Sign Language (ASL). There has been much recent work on nearest-neighbor classification of such data, with accuracies greater than 90% frequently reported [119]. This suggests that a long data stream of ASL might be amenable to clustering, where each cluster maps to a distinct "word" or "phrase."

However, all such data contains Movement Epenthesis (ME) [126][150]. During the production of a sign language sentence, it is often the case that a movement segment needs to be inserted between two consecutive signs to move the hands from the end of one sign to the beginning of the next. These ME segments can be as long as—or even longer than—the true signs, and are typically not performed with the precision or repeatability of the actual words, since they have no meaning. Recent sophisticated sign

language recognition systems for continuous streams have begun to recognize that "*automated sign recognition systems need a way to ignore or identify and remove the movement epenthesis frames prior to translation of the true signs*" [150].

What we observed about ASL as a concrete and intuitive example matches our experience with dozens of other datasets, and indicates that this is a pervasive phenomenon. We believe that almost all datasets have sections of data that do not represent a discrete underlying behavior, but simply a transition between behaviors or random drifts where no behavior is taking place. In most datasets that we have examined, such sections constitute the *majority* of the data. If we are forced to try to model these in our clusters, they will swamp the true significant clusters. We can best demonstrate this effect, and hint at our proposed solution by an experiment on a discrete analogue of ASL time series, in this case English *text*.

We emphasis that this is *just a expository example*, and if we were really assigned to cluster such text data we could do better than the attempt shown below.

Consider the following string *D*, which from left to right mentions three versions of the name David (English, Persian, Yiddish) and three versions of the name Peter (English, Croatian, Danish). Note that all names have five letters each, making this problem *apparently* simple.

David enjoined Peter who identified Davud son of Petar friend to Dovid and Peder, to do what...

Here the words between the names are *exactly* the epenthesis previously referred to. To make it more like our time series problem, we can strip out the punctuation and spacing, leaving us:

davidenjoinedpeterwhoidentifieddavudsonofpetarfriendtodovidandpedertodowhat

The discrete analogue of the time series clustering algorithm in [128] would begin by extracting all the subsequences of a given fixed length. Let us assume for simplicity the length five is used, and thus the data is transformed into:

david
avide
viden
idenj
...
owhat

In Figure 54 we show representative clusters for two values of K, if we perform partitional clustering as in [128] on this extracted data.

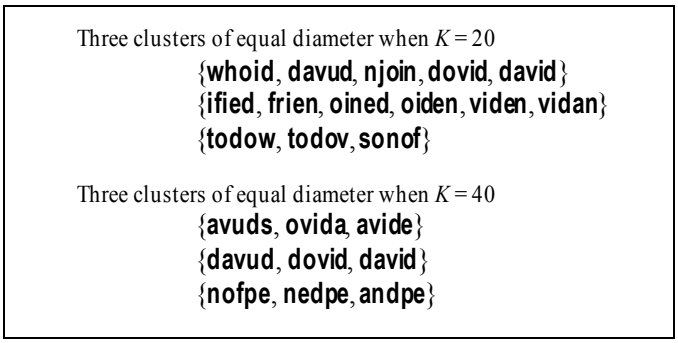

Figure 54: Representative partitional clusters from dataset D for two settings of K

Note that while the cluster of the name variants of David is discovered, we find that under any setting of K there are equally significant *meaningless* clusters, for example {nofpe, nedpe, andpe}. This is in spite of the fact that this can be considered a particularly easy task. Exactly 40% of the signal consists of data we hope to recover, and we deliberately avoided name variants of different lengths (i.e., pieter, pere). In more realistic settings we expect much less of the data to contain meaningful signals. Note also that the problem is not mitigated by using other clustering variants. The problem is inherent in the false assumption that a clustering of a single stream that must explain *all* such data could ever produce meaningful results [136].

5.1.2. How MDL Can Help

In contrast to the previous section, it is instructive to see what our proposed algorithm will do in this case. While the details of our algorithm are not introduced until Section 5.4, we can still outline the basic intuition here.

The original string D has a bit-level representation whose length we denote as $DL(D)$. Our algorithm can be imagined as attempting to losslessly compress the data by finding repeated structure in it. As there is little *exactly* repeated structure, we must find approximately repeated structure and encode the differences. For example, if we find the approximately repeated versions of the name "david", we can think of one version as being a model or *hypotheses* for the data, and encode only the difference between the other occurrences:

H_1 = {1:david}

$_1$____enjoinedpeterwhoidentified$_1$___u_sonofpetarfriendto$_1$_o___andpedertodowhat

In terms of MDL we can see **david** as a partial hypothesis H_1 or description of the data. This model has some size, which is simply the length in bits of the word $DL(H_1) = DL(\textbf{david})$. In addition, the size of the remaining data was both *reduced* by factoring out the common structure and (slightly) *increased* by the overhead of the pointers to the dictionary, etc[4]. When encoded with the hypothesis, the length (in bits) of the description of the data is given as $DL(D \mid H_1)$. The total cost of both the hypothesis and the data encoded using the hypothesis is just $DL(H_1) + DL(D \mid H_1)$.

Because this sum is less than the length of the original data $DL(D)$, we feel that we are making progress. Perhaps, however, there is more structure we can exploit. A brief inspection of the data suggests another model, H_2, that exploits *both* repeated names:

H_2 = {1:david 2:peter}

$_1$___enjoined$_2$___whoidentified$_1$___u_sonof$_2$___a_friendto$_1$_o___and$_2$__d__todowhat

Because $DL(H_2) + DL(D \mid H_2) < DL(H_1) + DL(D \mid H_1)$, we prefer this new hypothesis as a model of the data.

Are we now done? We can try other hypotheses. For example, we could consider the hypothesis H_3 = {1:david 2:peter 3:ono}, attempting to exploit the two occurrences of a pattern "o*o" (i.e.,..sonof.. and ..to do..). However, because this pattern is short, and only has two occurrences, we cannot break even with the cost of the overhead:

$$DL(H_2) + DL(D \mid H_2) < DL(H_3) + DL(D \mid H_3)$$

Because we cannot find any other hypotheses that produce a smaller model, we invoke the MDL principle to claim that H_2 = {1:david 2:peter} is the best model of the data D. Here *best* means something

---

[4] In this toy example, we are deliberately glossing over the concrete details of how the pointers are represented and how the amount compression achieved is measured, etc. [139]. We will formalize these details in Section 5.3.

beyond simply achieving the greatest compression. We can claim that MDL approach has achieved the most parsimonious explanation of the data, recovering the true underlying structure [127][131][134][133][139]. In at least this case, where the sentence was contrived as an excuse to use two names trice, MDL *did* recover the true underlying structure.

Note that while our informally stated algorithm does manage to recover the two embedded clusters, it *does not* attempt to explain all of the data. This is a critical observation, in order to cluster a single stream of data, be it discrete or real-valued, we *must* be able to represent and rank solutions that ignore some of the data.

5.2 Related Work

The tasks of clustering *multiple* time series streams, or many individual time series (i.e., gene expressions) have received significant attention, but the solutions do not inform the problem we consider here, the task of clustering a *single* time series stream. The most commonly referenced technique for clustering a *single* time series stream is presented in [128] as a subroutine for rule discovery in time series. In essence the method slides a fixed length window across the stream, extracting all subsequences which are then clustered with *K*-Means. The reader may have already spotted a flaw here; the algorithm tries to explain *all* the data. In [136] (and follow-up works by more than twenty other authors [123][124][129]), it was shown that this method can only produce cluster centers that are sine waves, and the output of the algorithm is essentially *independent of the input*. Note that even if the algorithm did not have these fatal flaws, it assumes the cluster all have equal length, and that we know the correct value of *K*. As we shall show, our method requires neither assumption.

Since the problem with [128] was pointed out in 2005 [136], at least a dozen solutions have been proposed. In Section 5.5.3 we show that the most referenced of these works [124] does not produce objectively correct results, even after extensive parameter tuning by the original authors on a relatively simple problem.

While there have been some efforts to use MDL with time series [144][147], they all operate on a quantized representation of the data. This has the disadvantage of requiring three parameters (cardinality, dimensionality and window size), eliminating the greatest advantage of MDL, its intrinsically parameter-free nature.

While MDL has had surprisingly little impact in data mining, it is a tool of choice for many bioinformatics problems. For example, working with RNA data, Evans et. al. have proposed a method using data compression and the MDL principle that is capable of identifying motif sequences, some of which were discovered to be miRNA target sites implicated in breast cancer [131]. Moreover, the authors showed the generality of their ideas by applying them, unmodified, to the problem of network traffic anomalies [132]. There is also a significant work on using MDL to mine graphs [134][143], dating back to classic work by Cook et al. [127].

Finally, we note that the task was informed by, and may have implications for many other time series problems, including time series segmentation[5] [122]. To see why, let us revisit the technique of text analogy. It is not obvious how one should segment the three concatenated words "hisabasiais". Perhaps the best we could do is to exploit the known frequencies of bigrams and trigrams, etc. In fact, most time series segmentation algorithms essentially do the real-valued equivalent of this [122]. However, if we see another such triplet of three concatenated words from later in the same stream, for example "withoutabasiais", we can immediately see that "abasia" must be a word[6].

[5] The phrase "time series segmentation" is unfortunately overloaded. It can mean approximating the data with the smallest number of piecewise polynomial segments for a given error threshold, or as here; extracting small, discrete, semantically meaningful segments of data [122].

[6] Abasia is the inability to walk due to impaired muscle coordination.

5.3 Background and Notation

5.3.1. Definitions and Notation

We begin by defining the data type of interest, *time series*:

Definition 1 A *time series* T is an ordered list of numbers. $T = t_1, t_2, ..., t_m$. Each value t_i can be any finite number (e.g., for two-byte values they could be integers in range [-32,768, 32,767]) and m is the length of time series T.

Before continuing, we must make and justify a choice. The MDL technique that is at the heart of our algorithm requires *discrete* data, but most time series datasets use four or eight bytes per value, and are thus *real-valued* [141]. Our solution is simply to cast the *real-valued* numbers into a reduced cardinality version. Does such a reduction lose meaningful information? To test this, we did one nearest-neighbor classification on eighteen public time series datasets, for cardinalities from the original four bytes down to a single bit. Figure 55 shows the results. As we can see, we can drastically reduce cardinality without reducing accuracy. The original four-byte cardinality is typically a by-product of file format convention or hardware specification, and not a claim as to the intrinsic cardinality of the data.

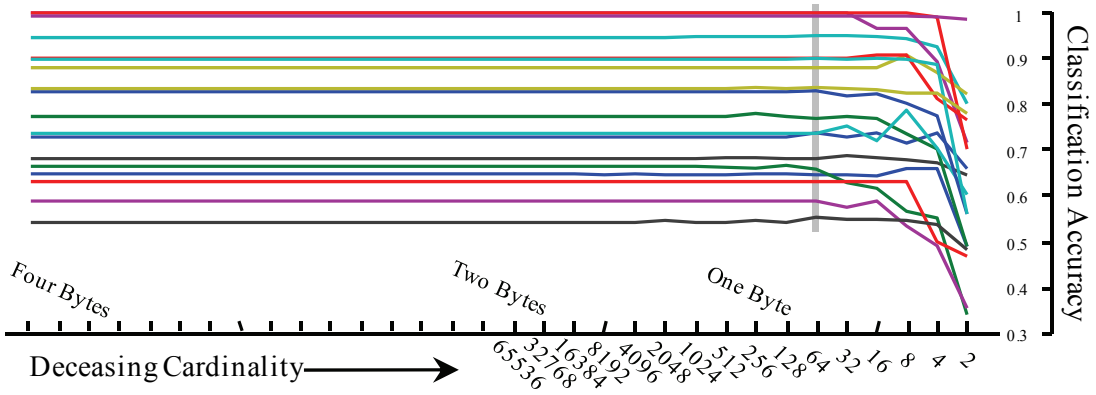

Figure 55: Classification accuracy on 18 time series datasets as a function of the data cardinality. Even if we reduce the cardinality of the data from the original 4,294,967,296 to a mere 64 (vertical bar), the accuracy does not decrease

We note that there may be other things we *could* have done. For example, the MML framework [149] which is closely related to MDL would allows us to work in original continuous space. However, we

choose MDL because it is more familiar and it allows for a more intuitive explanation of our algorithms. Likewise, we have at least a dozen choices of how to discretize the time series (adaptive binning, uniform binning, SAX etc.); however, after testing all published algorithms and finding it made little or no difference, we settled on the simple idea shown below in Definition 3.

Based on the observations in Figure 55, we will simply use 64-value (6-bit) cardinality in the rest of this work.

While the source data is one long time series, we ultimately wish to cluster it into sets of shorter *subsequences*:

Definition 2 A *subsequence* $T_{i,k}$ of a time series T is a short time series of length k which starts from position i. Formally, $T_{i,k} = t_i, t_{i+1}, .., t_{i+k-1}$, $1 \leq i \leq m-k+1$.

As we previously noted, we are working in a space of reduced cardinality. Because comparing time series with different offsets and amplitudes is meaningless [136], we must (slightly) adapt the normalization process for our *discrete* representation:

Definition 3 A *discrete normalization* function *DNorm* is a function to normalize a real-valued subsequence T into b-bit discrete value of range $[1,2^b]$. It is defined as followings:

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

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

Based on the results in Figure 55, b is fixed at 6 for all experiments. We need to define a distance measure; we use the ubiquitous *Euclidean distance* measure:

Definition 4 The distance between two subsequences $T_{i,k}$ and $T_{j,k}$ is the *Euclidean distance* (ED) between $T_{i,k}$ and $T_{j,k}$. Both subsequences must be in the same length. Hence, it is:

$$Dist(T_{i,k}, T_{j,k}) = \sqrt{\sum_{l=0}^{k-1} \left(t_{i+l} - t_{j+l}\right)^2}$$

As we shall see later, the Euclidean distance is not general enough to support clustering from time series streams; nevertheless, it is still a useful subroutine to speed up our more general measures. As generally noted [121][137][148][151], the Euclidean distance is a fast and robust distance measure.

For both the full time series T and any subsequences derived from it, we are interested in knowing how many bits are necessary to represent it. Normally the number of bits depends solely on the data format, which is typically a reflection of some arbitrary choices of hardware and software. In contrast, we are interested in knowing the *minimum* number of bits to exactly represent the data. In the general case, this number is not calculable, as it is the Kolmogorov complexity of the time series [140]. However, there are numerous ways to approximate this, using Huffman coding, Shanon-Fano coding, etc. Because entropy is a lower bound on the average code length from any such encoding, we can use the *entropy* of the time series as its description length:

Definition 5 The *entropy* of a time series T is defined as following. For special case when $P = 0$, $P \log_2 P$ is defined as 0.

$$H(T) = -\sum_t P(T = t) \log_2 P(T = t)$$

We can now define the description length of a time series.

Definition 6 A *description length DL* of a time series T of length m is the total number of bits required to represent it, that is $DL(T) = m * H(T)$.

The *DL* of a time series using entropy clearly depends on the data itself, not just arbitrary representational choices. Figure 56 shows four time series, which all require 250 bytes to characterize in the original representation, but which have differing *entropies* and thus different description lengths.

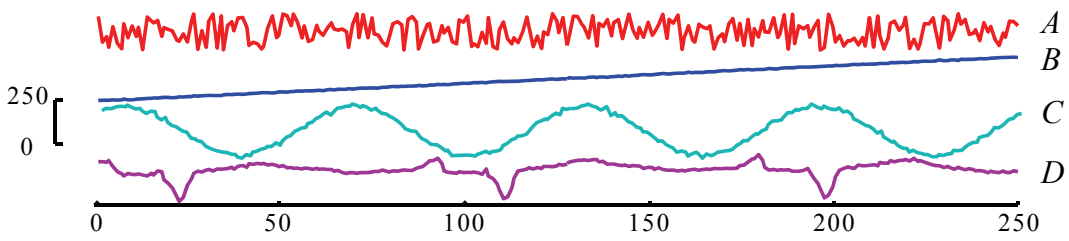

Figure 56: Four time series of length 250 and with a cardinality of 256. Naively all require 250 bytes to represent, but they have different *description lengths*

The reader may have anticipated the following observation. While the (slightly noisy) straight line *B* has high entropy, we would subjectively consider it a simple shape. It *is* simple given our belief (hypothesis) that it is a slightly corrupt version of a straight line. If *H* is this hypothesis, then we can consider instead the entropy of a time series *B′*, which as shown in Figure 57, is simply *B* encoded using *H*, and written as $B' = (B \mid H)$. As a practical matter, to use *H* to encode *B*, we simply subtract *H* from *B* to get a difference vector *B′*, and encode this simpler vector *B′*.

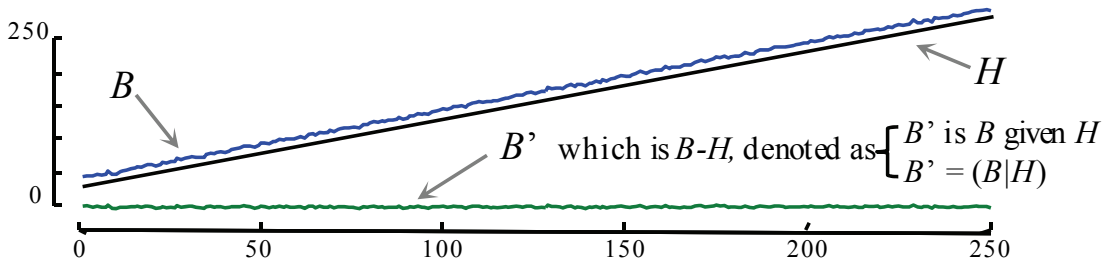

Figure 57: Time series B can be represented exactly as the sum of the straight line *H* and the difference vector *B*'

While the vector *B′* is also of length **250**, it has only 10 unique values, all of which are small in magnitude, thus its entropy rate is only **2.51** bits. In contrast, *B* has 172 unique values and an entropy rate of 7.29. Note that if we are given only *B′*, we cannot reconstruct *B*; we also need to know the slope and mean of the line. Thus, when reporting the overall number of bits for *B′*, we must also consider the number of bits it takes to encode the hypothesis (the line *H*). We can encode the line simply by recording the heights' **two** locations, the first and last points[7], each of which requires a single **byte**. Thus, the number of bits required to represent *B* using our hypothesis is:

$$DL(B) = DL(H) + DL(B \mid H) = (2*8) + (250*2.51) = 643.5 \text{ bits}$$

which is significantly less than the 1,822 bits required for the naive encoding of B without any hypothesis.

[7] If we know the time series is z-normalized, we only need one byte to record the line.

Note the straight line would not help in reducing the number of bits required to represent time series *C*, but using a sine wave as the hypothesis *would* significantly help. This observation inspired one of the principle uses of MDL, model section [146]. Statisticians use this principle to decide if some noisy observations suggest an underlying physical model is produced by, say, a piecewise linear model as opposed to a sinusoidal model. However, our work leverages off a simple but unexploited observation. The hypotheses are not limited to well-defined functions such as sine waves, wavelet basis functions, polynomial models, etc. The hypothesis model can be *any arbitrary time series*. We will see how this observation can be exploited in detail later, but in brief: if *k* subsequences of a stream truly form a cluster, then it should be possible to store them in less space by encoding them as a set of difference vectors to the mean of all of them. Thus, we have a potential test to guide our search for clusters.

Having seen this intuition, we can now formalize the notion of *hypothesis* as it pertains to our problem:

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

As a practical matter, the encoding we use is the one visualized Figure 57, we simply subtract hypothesis *H* from the target subsequence(s) and encoded the difference vector(s). We could encode the difference vector(s) with, say, Huffman encoding, but as we noted in Definition 5, we really only care about the *size* of the encoding, so we simply measure the entropy of the difference vector(s) to get a lower bound of the size of encoding.

A necessary (but not sufficient) condition to place two subsequences *H* and *B* into the same cluster is:

$$DL(B) > DL(B \mid H)$$

This inequality requires that the subsequence *B* takes fewer bits to represent when *H* is used as a basis to encode it, encoding the intuition that the two subsequences are related or similar.

We can hint at the utility of thinking about our data in terms of hypothesis encoding by revisiting our text example. When a clustering text stream, would it be better to merge **A** or **B**?

A = {david, dovid}, **B** = {petersmith, petersmidt}

The first case allows a tight cluster of two short words, is that better than a looser but longer cluster B? The problem is exacerbated when we consider the possibility of clusters with more than two members: how would we rank the relative utility of the tentative cluster **C** = {bob, rob, hob}?

Normally, clustering decisions are made by considering *Euclidean distance* (or its text counterpart, *Hamming distance*); however, Euclidean distance only allows meaningful comparisons when all the subsequences are the same length. The solution for text, to use the length-normalized Hamming distance, *cannot* be generalized here. The reason is subtle and underappreciated, suppose we have two subsequences of length *k* that are distance *d* apart. If we truncate the end points and measure the distance again, we might find it has increased! This is because we should only compare z-normalized time series when using Euclidean distance[8], and after (re)z-normalizing the slightly shorter subsequences, we may find they have grown further apart. Thus, the z-normalized Euclidean distance function is not linear in length and is not even monotonic.

We have already hinted at the fact that the *DL* function can use extra information, by using "*given*", i.e., $DL(B' \mid H)$ is the *DL* of B' given H. We can now formalize this notion:

Definition 8 A *conditional description length* of a subsequence A when a hypothesis H is given is

$$DL(A|H) = DL(A - H)$$

Recall from Figure 56 and Figure 57 that the *DL* of a subsequence depends on the structure of the data. For example, a constant line has a very low *DL*, whereas a random vector has a very high *DL*. If A and H are very similar, their difference (A-H) will be close to a constant line and thus have a tiny *DL*. In essence then, the *DL* function gives us a parameter-free test to see if two subsequences should be clustered together.

[8] The solution of not normalizing the time series would mitigate this problem, but measuring the Euclidean distance between two time series with different offsets or amplitudes produces meaningless results [136].

We generalize the notion of *DL* to multiple sequences next. We can apply the same spirit by using a hypothesis to calculate the minimum number of bits required to keep a cluster. We call this *description length of a cluster*:

Definition 9 A *Description Length of a Cluster* (*DLC*) **C** is the number of bits needed to represent all subsequences in **C**. In this special case, *H* is the center of the cluster. Hence, the description length of cluster **C** is defined as:

$$DLC(\mathbf{C}) = DL(H) - max_{\mathrm{A}\in\mathrm{C}} DL(A|H) + \sum_{A\in\mathbf{C}} DL(A|H)$$

The above *DLC* gives us a primitive to measure the reduction in bits achieved by encoding data with a hypothesis. The two right terms in the equation record the number of bits needed to represent the cluster **C**. Concretely, if the cluster **C** contains *n* subsequences, we need to keep at most *n* subsequences for reconstructing **C**. We choose to keep the hypothesis *H* and the *n*-1 smallest differences. Therefore, the description length of the cluster **C** is the combination of the description length of the hypothesis *H* and the minimum of any *n*-1 conditional description lengths with respect to *H*.

Our clustering algorithm is essentially a search algorithm. Three operators avail of the *DLC* definition to test how many bits a particular choice can save. Thus, these three operators fall under the umbrella definition of *bitsave*:

Definition 10 A *bitsave* is the total number of bits saved after applying an operator that creates a new cluster, adds a subsequence to an existing cluster, or merges two existing clusters together. It is the difference in the number of bits *before* and *after* applying a given action:

$$bitsave = DL(Before) - DL(After)$$

In detail, the *bitsave* for each operator is defined as following:

1) Creating a new cluster **C'** from subsequences *A* and *B*

   $bitsave = DL(A) + DL(B) - DLC(\mathbf{C'})$

2) Adding a subsequence *A* to an existing cluster **C**

   $bitsave = DL(A) + DLC(\mathbf{C}) - DLC(\mathbf{C'})$

where **C'** is the cluster **C** after including subsequence *A*.

3) Merging cluster $\mathbf{C_1}$ and $\mathbf{C_2}$ to a new cluster **C'**.

$bitsave = DLC(\mathbf{C_1}) + DLC(\mathbf{C_2}) - DLC(\mathbf{C'})$

Note that, as we discussed earlier, we do *not* use Euclidean distance to make decisions about which subsequences to place into which clusters. We use only use Euclidean distance in two subroutines: motif discovery and finding the closest subsequence from a given cluster center. Next, we will define closest subsequence or the *nearest neighbor*:

Definition 11 A *nearest neighbor* of the given subsequence *A* is the subsequence *B* such that

$$Dist(A,B) \leq Dist(A,X) \quad \text{for any subsequences } X$$

Another definition, which we borrow from the literature [103], are *time series motifs*.

Definition 12 A *time series motif* is pair of subsequence *A* and *B* such that

$$Dist(A,B) \leq Dist(X,Y) \quad \text{for any subsequences } X \neq Y,\ A \neq B$$

Similar to the nearest neighbor, the time series motif contains two most similar subsequences in the given time series. We use the *simple-but-robust* Euclidean distance as *dist* function (cf. Definition 4) for finding both time series motif and the nearest neighbor of the given subsequence. Note that if subsequence *X* and *Y* are overlapped, it may lead to the discovery of trivial matches. For more details about the time serie*s motif*, we refer the reader to [103]. In next section, we will explain our algorithm in detail.

5.4 Clustering Algorithm

Having introduced the necessary notation, we are finally in a position to introduce our algorithm. We begin by giving a simple text and visual intuition in the next section, and follow by giving detailed and annotated pseudo code in Section 5.4.2.

5.4.1. The Intuition behind Stream Clustering

Recall that our input is a single time series like the one shown in Figure 58.*bottom* and our required output is a set of clusters -- possibly of different lengths and sizes. Recall that the union of all the subsequences in this set of clusters may only cover a fraction of the input time series. Indeed, for pathological cases we are given a pure noise time series, we want our algorithm to return a *null* set of clusters. In Figure 5 we show our running example. It contains the interwoven calls of two very different species of birds.

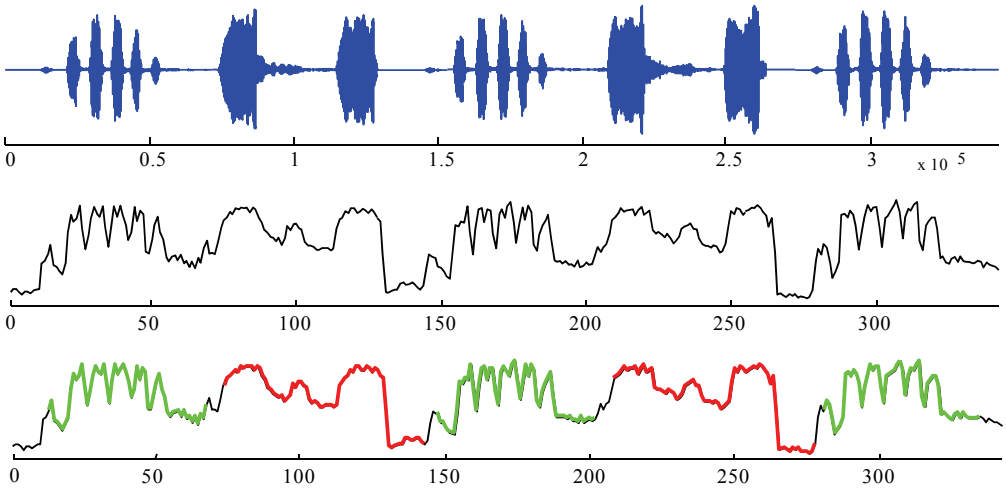

Figure 58: Two interwoven bird calls featuring the *Elf Owl*, and *Pied-billed Grebe* are shown in the original audio space (*top*), and as a time series extracted by using MFCC technique *(middle*) and then clustered by our algorithm (*bottom*).

Our proposed clustering algorithm is a bottom-up greedy search over the space of clusters. For the moment, we will ignore the computational effort that it requires and simply explain *what* is done, leaving the *how* it is (efficiently) done for the next section.

Our algorithm is an iterative merging algorithm similar in spirit to an agglomerative clustering algorithm [135]. However, the differences are telling and worth enumerating:

- Our algorithm typically stops merging before explaining all the data, thus producing a *partitioning* a subset of the data, not producing a *hierarchy* of all the data.

- Agglomerative clustering algorithms are typically implemented such that they require quadratic space; our algorithm has only linear space requirements[9].
- Most critically, agglomerative clustering algorithms assume the *K* items of a *fixed* dimensionally (subsequence length) to be clustered are inputs to the algorithm. However, we do not know how many items will ultimately be clustered, or even how long the items will be.

Similar to agglomerative clustering, we have a search problem that uses operators, in our case, *create*, *add*, and *merge* (Definition 10). When the algorithm begins, only *create* is available to us.

We begin by finding the best initial pair of subsequences to combine so that we may *create* a cluster of two items. To find this best pair, we treat one as a hypothesis and see how well it encodes the other (Definition 8). The pair that reduces the bit cost the most is the pair of choice. This is shown in Figure 59 as **Step 1**.

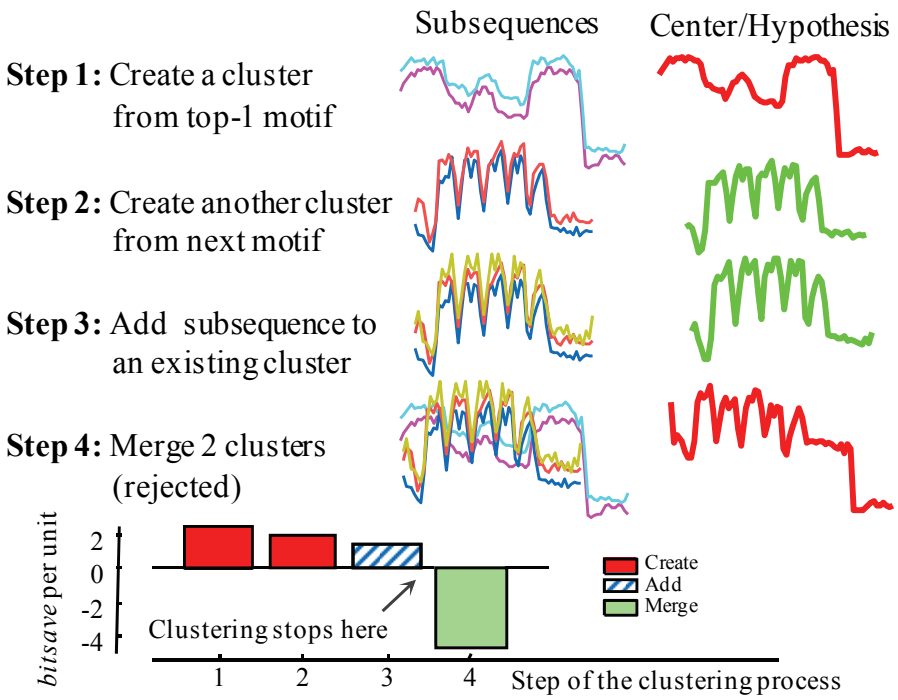

Figure 59: A trace of our algorithm on the bird call data shown in Figure 58.*bottom*

[9] Linear space agglomerative clustering algorithms *do* exist, but require highly multiply redundant calculations to be performed, and are thus rarely used due to their lethargy.

There is a potential problem here. Even if we fix the length of subsequences to consider to a constant s, the number of candidate pairs to consider is quadratic in the length of the time series, approximately $O((m-s)^2/2)$. Furthermore, there are no known shortcuts that let us search this space in sub-quadratic time.

The solution to this problem is to note that Euclidean distance and conditional description length are highly correlated where either of them is small. We can leverage off this fact because there exist very fast algorithms to find the closest pair of subsequences (which are known as *time series motifs* [103]) under Euclidean distance. So rather than a brute force search using the conditional description length, we do a fast motif search and then test the motif pair's conditional description length.

In the next stage of the algorithm, there are two operators available to us. We can either *add* a third item to our existing cluster of size two, or we can *create* a new cluster, possibly of a different length. In Figure 59 in **Step 2**, we can see that in this case our scoring functions suggest creating a new cluster is the better option in.

In the subsequent phase of the algorithm, it happens that *all* operators are available to us: we could try to *create* a new cluster, we could *merge* our two existing clusters, or we could *add* a subsequence to one of our two clusters. As we can see in Figure 59, **Step 3**, the last option is chosen.

In the next iteration, the cheapest operator was to merge our two existing clusters as shown in **Step 4**. However, doing this does not decrease the size of the representation—it *increases* it. As such, our algorithm terminates after returning the two clusters it had created up to **Step 3**. The only other way that our algorithm can terminate is if it simply runs out of data to cluster.

5.4.2. Our algorithm in detail

As we noted in the last section, our algorithm is a bottom-up search algorithm. The input is a single time series, and the output is a set of clusters of subsequences. Our algorithm can cluster subsequences of different lengths, and it does not require the number of clusters to be specified.

There are three operators in our search algorithm: *create*, *add*, and *merge*. In each step, we do all (legal) operations and choose the operator which maximizes the number of bits saved as measure by *bitsave*

(Definition 10). The current clusters are updated with respect to that choice. The algorithm can terminate in just two ways; either the best possible choice cannot save any bits, or all data is used up.

Most attempts to cluster time series [124][128][129] suffer from a surfeit of parameters. Our algorithm allows essentially none. However, if we allow subsequences that are too short, we can get pathological results in some cases. For example, there are only two possible z-normalized subsequences of length two. Moreover, a user may wish to bias the algorithm towards certain clustering. For example, for electrical power demand load we may be interested in weekly *or* daily patterns. Thus, as shown in Table 11, we allow the user the option of suggesting an approximate length *s*.

Table 11: Main time series stream clustering algorithm

Input: *ts* **:** time series,
s **:** approximate length
Output: *cluster* **:** final cluster of subsequences

```
1   cluster = {}
2   while bitsave>0
3     bitsave=-∞, i=0
      // create new cluster
4     for len = s to 2s
5       (A,B)  = MotifDiscovery(ts,len)
6       C'     = CreateCluster(A,B)
7       bs.append(ComputeBitsave(C',A,B))
8       cluster'.append(cluster∪{C'})
9     end for
      // add subsequence to an existing cluster
10    for C ∈ cluster
11      A  = NearestNeighbor(ts,C)
12      C' = AddToCluster(C,A)
13      bs.append(ComputeBitsave(C',C,A))
14      cluster'.append(cluster∪{C'}-{C})
15    end for
      // merge 2 clusters
16    for C1 ∈ cluster
17      for C2 ∈ cluster and C1~=C2
18        C' = MergeClusters(C1,C2)
19        bs.append(ComputeBitsave(C',C1,C2))
20        cluster'.append(cluster∪{C'}-{C1}-{C2})
21      end for
22    end for
      // update the result
23    [bitsave index] = max(bs);
24    cluster = cluster'(index);
25  end while
```

The algorithm begins by initializing the cluster set to *empty*, then it enters a loop until no more bits can be saved (line 2) or it runs out of data. Within each iteration the loop, we perform three operators *create*

(line 4-9), *add* (line10-15), and *merge* (line16-22), and we keep the results of the most parsimonious operator.

For the *create* process, we call a subroutine to find time series motifs under Euclidean distance using the fastest currently known technique [103]. Because we do not know how long the subsequences in the cluster should be, the algorithm runs `MotifDiscovery` multiple times on different lengths of motif (line 5). If the new cluster is created, then the number of bits saved is calculated (line 7). The temporary version of updated clusters are kept (line 8) and used if the algorithm eventually chooses to *create* this cluster (line 24). Recall that the details of function `ComputeBitsave` are provided in Definition 9 and 10.

It is possible to *add* a subsequence into an existing cluster (line 10-15). We first find the most similar subsequence in the input time series with respect to the center of a given cluster (line 11); we can achieve this task by using any nearest-neighbor search algorithm [130], including brute force search. After the search, the cluster is updated to include that nearest subsequence (line 12), the number of bits saved is calculated, and the temporary clusters are recorded (line 13-14).

For our last operator, any pair of clusters is allowed to *merge* (line 18); we then compute the number of bits saved for each pair, and record the temporary cluster.

After the algorithm measures the number of bits saved from all possible choices, the final cluster is updated with respect to the choice that maximizes the number of bits saved (line 23-24).

We have glossed over an important detail: the two items being combined by the merge/add operators may be of different lengths. To allow this critical flexibility, we use a simple data structure to record a cluster. For any given cluster **`C`**, **`C`**`.size` records the number of subsequences in the cluster, **`C`**`.cen` is the center of the cluster, **`C`**.seq is a set of subsequences in the cluster, and **`C`**`.shift` is a set of shift positions (i.e., offsets) of each subsequence in **`C`** when it aligned to the **`C`**.cen. Note that to compute the conditional description length (Definition 8), a subsequence and its hypothesis must be of the same length.

Table 12 shows how a new cluster can be created from two subsequences of the same size. Because those two subsequences are from motif discovery under Euclidean distance, their align position is set to 0 (line 4). The center of new cluster is the average of those two subsequences. In Table 13, when we want to

add a subsequence *A* to an existing cluster **C**, the new center is created by the weighted average of the current center and the subsequence (line 1). Because *A* is the nearest neighbor of **C**.cen, no offset alignment is needed for *A*.

Table 12: *Create* Operator

	Function C = CreateCluster(*A*,*B*)
1	**C**.size = 2;
2	**C**.cen = (*A*+*B*)/2
3	**C**.seq = [*A*; *B*]
4	**C**.shift= [0; 0]

Table 13: *Add* Operator

	Function C = AddToCluster(**C**,*A*)
1	**C**.cen = (**C**.cen**C*.size+*A**1)/(**C**.size+1)
2	**C**.size = **C**.size+1
3	**C**.seq = [**C**.seq; *A*]
4	**C**.shift = [**C**.shift; 0]

Table 14: *Merge* Operator

	Function C' = MergeClusters($\mathbf{C_1}$,$\mathbf{C_2}$)
1	**C'**.seq = [$\mathbf{C_1}$.seq; $\mathbf{C_2}$.seq]
2	**C'**.size = $\mathbf{C_1}$.size+$\mathbf{C_2}$.size
3	n_1=$\mathbf{C_1}$.size, m_1=length($\mathbf{C_1}$.cen)
4	n_2=$\mathbf{C_2}$.size, m_2=length($\mathbf{C_2}$.cen)
5	*i*=0
6	**for** *off* = 0 to m_2
7	*cen*1 = [$\mathbf{C_2}$.cen(1,*off*), $\mathbf{C_1}$.cen]
8	*cen*2 = [$\mathbf{C_2}$.cen, $\mathbf{C_1}$.cen(1,m_1+*off*-m_2)]
9	**C'**.cen = (*cen*$_1$*n_1+*cen*$_2$*n_2)/(n_1+n_2)
10	**C'**.shift = [**C1**.shift+*off*; $\mathbf{C_2}$.shift]
11	*bs*.append(ComputeBitsave(**C'**,$\mathbf{C_1}$,$\mathbf{C_2}$))
12	**Ctmp**.append(**C'**)
13	**end for**
14	**for** *off* = 1 to m_1
15	*cen*$_1$ = [$\mathbf{C_1}$.cen, $\mathbf{C_2}$.cen(1,m_2+*off*-m_1)]
16	*cen*$_2$ = [$\mathbf{C_1}$.cen(1,*off*), $\mathbf{C_2}$.cen]
17	**C'**.cen = (*cen*$_1$*n_1+*cen*$_2$*n_2)/(n_1+n_2)
18	**C'**.shift = [$\mathbf{C_1}$.shift; $\mathbf{C_2}$.shift+*off*]
19	*bs*.append(ComputeBitsave(**C'**,$\mathbf{C_1}$,$\mathbf{C_2}$))
20	**Ctmp**.append(**C'**)
21	**end for**
22	[*bitsave index*] = max(*bs*);
23	**C'** = **Ctmp**(*index*);

Table 14 shows how two clusters of different lengths can be merged into the same cluster. The new cluster contains all subsequences from both clusters (line 1-2). Because two clusters may be different lengths, we need to align them before finding the new center. As the Figure 59 example shows, **Step 4**

merges two clusters from **Step 1** and **Step 3**. The red center in **Step 1** is longer than the green center in **Step 3**. We align the red center at all possible offsets (line 6), and then create a new center by averaging two current centers. Parts of the new centers are created by weighted averaging from all (one *or* two) centers that cover that part (line 7-9). To make a decision among all possible offsets, MDL plays an important role again; at each offset, *bitsave* is calculated (line 11), and we choose the offset which can save the maximum number of bits (line 22-23). Similar to the code in line 6-13, which evaluates all offsets when C_1.cen moves *into* C_2.cen, the code in line 14-21 evaluates the inverse when C_1.cen moves *out of* C_2.cen.

To summarize, our algorithm contains three operators (*create*, *add*, and *merge*), which all use MDL to decide the best choice at each step of the clustering. As there are no known indexing/motif discovery algorithms for MDL, we avail ourselves of two fast external modules that use Euclidean distance for motif discovery [103] and nearest neighbor search [138]. Using Euclidean distance as a fast proxy for MDL is possible because they are highly correlated when both are small.

5.5 Experimental Results

We begin by stating our experimental philosophy. To ensure our experiments are reproducible, all codes/data are available at [152]. In addition, the site contains many more experiments omitted due to space limitations. Furthermore, the website contains video animations of the clustering process for each dataset.

5.5.1. Comparison to Ground Truth

We begin by considering a time series for which we have access to the ground truth (albeit indirectly). Consider the time series shown in Figure 60.*top*. A visual inspection gives a hint of some structure, but even on this tiny example, it is not clear exactly what the clustering should be -- or even what is the natural length for potential clusters. This dataset was obtained by taking an audio snippet of a recording of Edgar Allen Poe's poem "The Bells" and transforming it in to the Mel-Frequency Cepstral Coefficients (MFCC) retaining only the first coefficient.

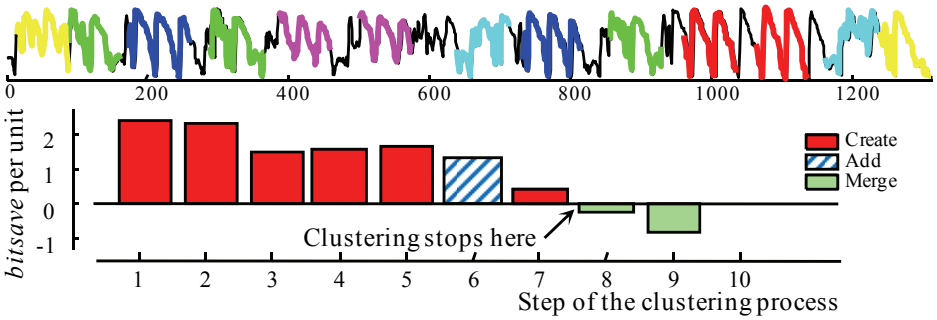

Figure 60: *top*) 29.8 seconds of an audio snippet, represented by the first coefficient in MFCC space, and then annotated with colors to reflect the clusters. *bottom*) A trace of the steps use to produce the clustering

The clustering we obtained looks subjectively intuitive; however, because of the original source material we are in a unique position to do a more objective test. Table 15 shows the original source text brushed with the colors reflecting the clustering obtained.

Table 15: The text corresponding to the time series shown in Figure 60, annotated by color/font

Original Order	Grouped by Clusters
In a sort of Runic rhyme, To the throbbing of the bells-- Of the bells, bells, bells, To the sobbing of the bells; Keeping time, time, time, As he knells, knells, knells, In a happy Runic rhyme, To the rolling of the bells,-- Of the bells, bells, bells-- To the tolling of the bells, Of the bells, bells, bells, bells, Bells, bells, bells,-- To the moaning and the groan- ing of the bells.	bells,bells, bells Bells, bells, bells Of the bells, bells, bells Of the bells, bells, bells the throbbing of the bells the sobbing of the bells the tolling of the bells To the rolling of the bells To the moaning and the time, time, time knells, knells, knells sort of Runic rhyme groaning of the bells.

The results are not perfect with reference to the text version. Recall that we are only considering *one* of the MFCC coefficients, instead of the ten plus typically used in speech processing. This allows some collisions, such as "time" and "knells". However, the structure recovered by our algorithm is significant. Note that our clusters are of different sizes (three items, and two items) and of different lengths (from 55 points to 70 points). Also, note that we could have had a single cluster of eighteen occurrences of the word

"bells." However, that would have obfuscated the information that this word tends to be repeated in this work, as in "bells, bells, bells." These longer clusters are arguably more parsimonious.

5.5.2. Clustering a Noisy Dataset

In Figure 61, we show the results of clustering a noisy industrial dataset. The data comes from an industrial wire winding process. The original data consists of seven dimensions; here we show only the results of clustering the noisiest channel, labeled U1 (the results on the other channels are at [152]). Note that the data has significant non-uniform noise, including spikes and dropouts. While we do not have access to the ground truth here, the clusters, which have different sizes and length, clearly have the property of being similar *within* a cluster and dissimilar *between* clusters. Note that approximately 26% of the data remains unclustered.

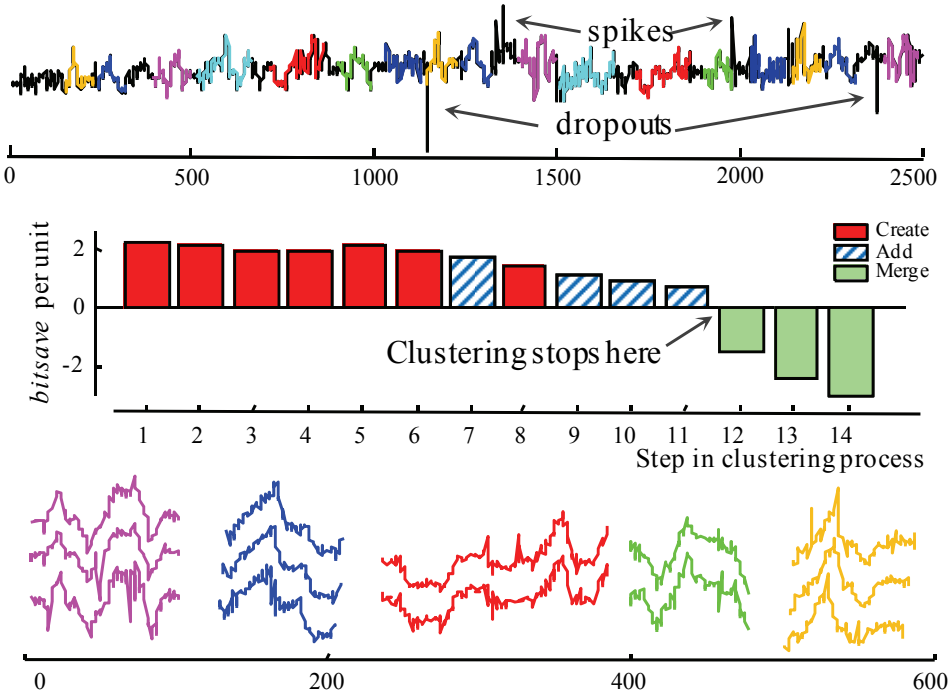

Figure 61: *top*) Dimension U1 of the *Winding* dataset. *middle*) A trace of the clustering steps produced by our algorithm. *bottom*) Representative clusters obtained.

5.5.3. Comparison to other Methods

As we noted above there are few candidate strawmen to compare our work to. Here we compare our work to the most referenced work in the literature. In a sequence of papers, Chen proposes a series of fixes for the stream clustering problem [123][124][125]. He demonstrates his ideas mostly on synthetic data; however, as shown in Figure 62.*right*, he also tests on short section of the Koski heartbeat dataset.

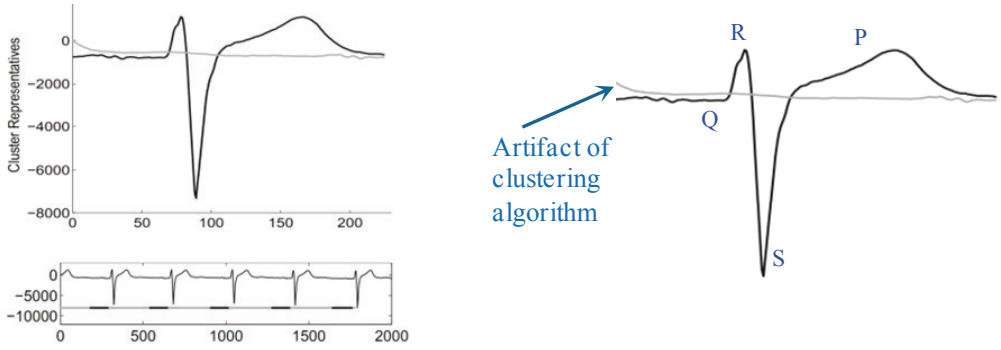

Figure 62: *left*) A screen dump of fig.11 from [124]. The original caption read "TF Clustering: Koski-ECG result". *right*) An annotation of the clusters by a USC cardiologist

While the results are perhaps reasonable, it is not clear why we should have two clusters here since there is clearly just one heartbeat. In addition, there is a subtle artifact noticed by cardiologist, Dr. Helga Van Herle, whom we asked to examine this. The slight slope on the light-gray cluster show in Figure 62.*left* is not in the data; it comes from the fact that the input data is not an integer multiple of beats, instead being roughly 5.2 beats. Since the algorithm is trying to explain *all* the data, it must explain the *extra* P-wave by averaging it into a place where it does not belong. Furthermore, as acknowledged in the original paper, the algorithm requires the setting of several parameters and "magic numbers" (i.e., "*we chose p as the number of points in the time series divided by 15*.."). Finally, we note in passing that the algorithm requires multiple calls to a *quadratic* space and time (in the length of the time series) algorithm, which would make it impractical for many real data mining problems. Our algorithm requires linear space.

In Figure 63, we show the clustering we achieved on *exactly* the same dataset. We believe the results here are intuitively correct, discovering a complete *single* heartbeat as the cluster. Note that our algorithm

explains 87.5% of the data; it does not try to explain the extra P-wave "bump" caused by the fact that we do not have an integer number of heartbeats.

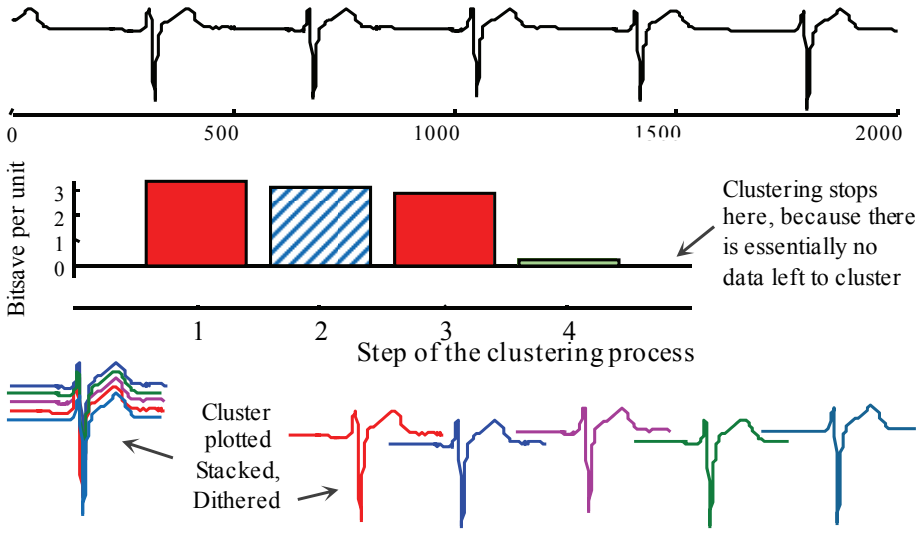

Figure 63: *top*) The same 2,000 data points from Koski-ECG as used in Figure 62. *middle*) A trace of the clustering steps produced by our algorithm. *bottom*) the single cluster discovered has five members

5.5.4. Scalability

From our algorithm in Section 5.4.2, assume that `MotifDiscovery` takes time $O(T)$. In each *create* step, `MotifDiscovery` is called multiple times to find motifs of different length; we run it at most $O(s)$ times. Because each subsequence is of length at least s, there are at most $O(m/s)$ new clusters to be created. This is why the running time for creating new clusters is $O(T*s*m/s) = O(mT)$.

Assume that `NearestNeighbor` can be finished in time $O(ms)$. The maximum number of clusters we can have is $O(m/s)$, and the original time series can be updated only when a new motif is discovered, so the number of clustering steps (cf. line 2 in Table 11) is at most $O(m/s)$. Thus, for add steps we have $O(ms * m/s * m/s) = O(m^3/s)$.

For *merge* steps, if a cluster is created by merging k clusters so far, the number of subsequences in that cluster is at most $O(k)$. The length of its center is at most $O(ks)$; therefore, the number of possible offsets is $O(ks)$, and *bitsave* calculation is finished in time $O(k^2s^2)$. The maximum number of clusters we can have is at most O(m/s), so we can have cluster of size k at most $O(m/sk)$ clusters, and there are at most $O(m/s)$ steps

in our algorithm. This means that the running of *merge* steps is at most $O(m/s*(m/sk)^2*k^2s^2) = O(m^3/s)$. Hence, the total running time of our algorithm is at most $O(mT+m^3/s)$ where T is a running time for a motif discovery. The empirical behavior is shown in Figure 64.

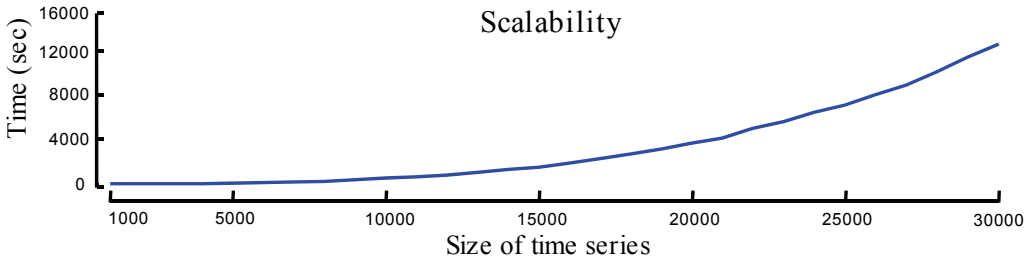

Figure 64: Running time of our algorithm on Koshi data when $s = 350$

To put these results in perspective, the ornithology lab we are working with has spent *months* collecting data in the field (cf. Figure 58), so they are willing to wait the hour we require to cluster several minutes of audio. Nevertheless, we believe that a 100X speedup will soon be possible simply by caching some near redundant motifs calculations.

5.5.5. Disscusion of the MDL Choice

Now that the reader has gleaned some intuition for our algorithm and its utility for clustering data, we will briefly revisit a discussion of why MDL on a discretized time series is our choice of measure to steer the clustering search.

We cannot use Euclidean distance (or the related *correlation* or *Dynamic Time Warping* etc. [130][138]) directly because it does not allow us to compare the relative merits of clusters of different lengths or different sizes. In contrast, MDL does allow such meaningful comparisons. Moreover, in the limited case when MDL and Euclidean distance can be compared (when time series lengths are the same), we find that the two measures are *highly* correlated so long as they are small (if both are destined to be large, it does not really matter how correlated they are). The relationship between Euclidean distance and MDL is shown in Figure 65.

We work in the discrete space rather than the original continuous space because MDL requires it, and because working with the discretized time series makes *no perceptible difference* in classification (as shown in Figure 55) or in similarity search, indexing, motif discovery or outlier discovery.

Because of their relationship especially when the distance is small, to make an intractable problem solvable (in term of acceptable running time), we can apply ED-based techniques to speed up the algorithm in some modules. For example, instead of finding the pair of subsequences whose difference has the smallest MDL, we can use the fast motif discovery algorithm from [103] to find the most similar pair using Euclidean distance. We also can apply some techniques, such as early abandoning and lower bounding, in searching a nearest subsequence of the existing clusters.

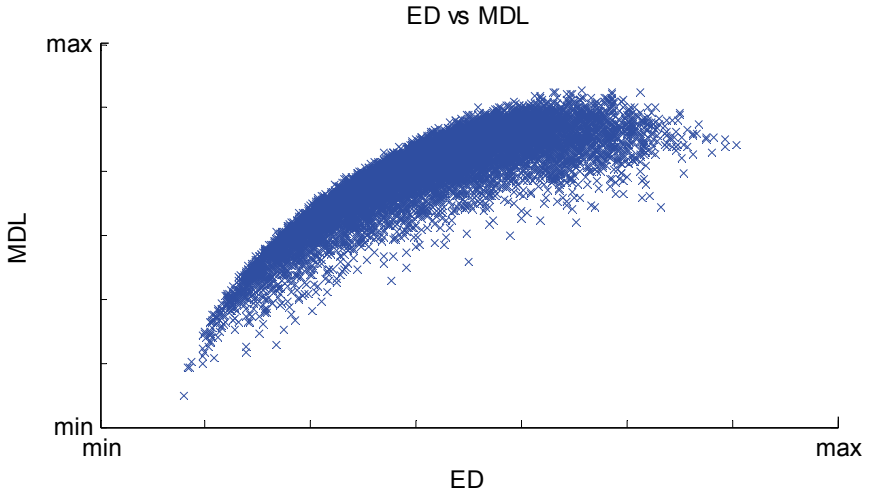

Figure 65: The relationship between Euclidean Distance (ED) of pairs of subsequences in a random walk time series and MDL of their difference. Euclidean distance is calculated in original continuous space but MDL is calculated in discrete space (64 cardinality)

Although Euclidean distance can dramatically speed up the running time of algorithm, sometimes the most similar subsequences using Euclidean distance and MDL are *not* the same. This makes our final score, bitsave, non-monotonically decreasing as the readers may have noticed in Figure 60.*bottom* and in Figure 61.*middle*.

5.6 Multi-dimensional clustering

In additional to a *single* dimensional time series clustering, our algorithm can be extended to cluster multi-dimensional time series data. In many applications that contain more than one feature, the quality of the clustering can be better if we can do clustering across different features or dimensions. For example, in motion capture, some activities have key features only in the upper parts of the body, but lower parts are keys in some activities.

5.6.1. Notation

For the sake of clarity, this section will define some necessary notations related to multi-dimensional time series clustering; however, we not that many of these definitions are obvious analogs of the single-dimensional case discussed in Section 3. We begin by defining *multi-dimensional time series*:

Definition 13 A *d-dimensional time series* T is a d-dimensional ordered list of numbers. $T = <T_1,T_2,\ldots,T_d>$ where a time series in dimension i, $T_i = t_{i,1}, t_{i,2}, \ldots, t_{i,m}$. Each value t_j can be any finite number and m is the length of the time series, which is equal for all dimensions.

When the original time series contain multiple dimensions, its subsequence is called a *d-dimensional subsequence*:

Definition 14 A *d-dimensional subsequence* $T_{i,j,k}$ of a d-dimensional time series T is a short time series of length k of dimension i which starts from position j. Formally, $T_{i,j,k} = t_{i,j}, t_{i,j+1}, .., t_{i,j+k}$ where $1 \le i \le d$ and $1 \le j \le m-k$.

As we mentioned before in previous sections, Euclidean distance and MDL are highly correlated when the distances are small. As in the single version algorithm, many techniques using the Euclidean distance are used for speeding up our algorithm. Hence, we define the *Euclidean distance* for two d-dimensional subsequences as following:

Definition 15 The *Euclidean distance* between two d-dimensional subsequences A and B of the same length, k, is:

$$Dist(A,B) = \sqrt{\sum_{i=1}^{d}\sum_{j=1}^{k}\left(A_{i,j} - B_{i,j}\right)^2}$$

The description length for a multi-dimensional time series is calculated based on the entropy of the given time series. We define *entropy* for *d*-dimensional time series as:

Definition 16 The *entropy* of a *d*-dimensional time series *T* is defined as following equation. For special case when $P = 0$, $P \log_2 P$ is defined as 0.

$$H(T) = -\sum_{\mathrm{i=1}}^{\mathrm{d}}\sum_{\mathrm{t}} P(T_i = t)\log_2 P(T_i = t)$$

For simplicity, we define the entropy of *d*-dimensional time series as the simple summation of the entropy from each dimension. This means we treat each dimension independent from others. While we do not preclude other methods for calculating *d*-dimensional entropy, this method works *very* well empirically.

However, there may be some room here to improve the quality of the clustering if we can exploit use the relationship between different dimensions. We leave this as the future improvement.

Similar as in Definition 6 in Section 5.3.1, the description length of a multi-dimensional time series *T* is defined as $m*H(T)$ where *m* is the length of the time series *T*. We can reuse Definition 8, 9 and 10 in our multi-dimensional clustering algorithm.

We invite the readers to revise some definitions in Section 5.3.1 before we move forward to the algorithm in next section.

5.6.2. Multi-dimensional Clustering Algorithm

In general, the idea of clustering multi-dimensional time series is similar to the idea of clustering a single dimensional time series in Section 5.4.2.

The algorithm composes of three operations – *create*, *add*, and *merge*. The general idea is the same as in the previous algorithm. All possible operators will be considered and their *bitsave* will be computed. Then, the choice whose *bitsave* is maximum will be selected and the algorithm performs the operation

corresponding to that choice. The algorithm for clustering multi-dimensional time series is shown in Table 16.

We allow user to set the approximate length of subsequences, contained inside the final clusters. The set of the final clusters is initialed to an empty set (line 1). The algorithm will be terminated on only two conditions, first, when no possible choices can reduce the number of bits for representing the clusters and, second, when there is no data left and all clusters have been merged.

Table 16: Multidimensional stream clustering algorithm

Input: *ts* **:** multidimensional time series,
s **:** approximate length
Output: *cluster :* final cluster of subsequences

```
1   cluster = {}
2   while bitsave>0
3     bitsave=-∞, i=0
4     // create new cluster
5     for len = s to 2s
6       (A,B)  = MultiDimMotifDiscovery(ts,len)
7       C'     = CreateMultiDimCluster(A,B)
8       bs.append(ComputeBitsave(C',A,B))
9       cluster'.append(cluster U {C'})
10    end for
11    // add subsequence to an existing cluster
12    for C ∈ cluster
13      A  = MultiDimNearestNeighbor(ts,C)
14      C' = AddToMultiDimCluster(C,A)
15      bs.append(ComputeBitsave(C',C,A))
16      cluster'.append(cluster U {C'}-{C})
17    end for
18    // merge 2 clusters
19    for C1 ∈ cluster
20      for C2 ∈ cluster and C1~=C2
21        C' = MergeMultiDimClusters(C1,C2)
22        bs.append(ComputeBitsave(C',C1,C2))
23        cluster'.append(cluster U {C'}-{C1}-{C2})
24      end for
25    end for
26    // update the result
27    [bitsave index] = max(bs)
28    cluster = cluster'(index)
29  end while
```

In the *create* operation (line 4-10), we find a most similar pair of subsequences. The Euclidean distance is used here instead of MDL for speeding up the process and makes this process much faster. A

trivial extension of the motif discovery algorithm from [103] using multi-dimensional Euclidean distance (cf. Definition 15) is used here to find the most similar subsequences in multi-dimensional time series (line 6). Then, a new cluster is created and its *bitsave* is calculated (line 7-8). For convenience, the temporary cluster is collected (line 9).

In *add* process (line 11-17), the nearest neighbor of each existing cluster is discovered (line 13). The rest of the algorithm, including *merge* process (line 18-25), is the same as in the algorithm in Section 5.4.2. After all choices are calculated, the choice, which maximizes *bitsave*, will be selected to perform (line 27-28).

Because each subsequence contains multi-dimensional data, the cluster representative or the cluster's center also contains multi-dimensional data. Table 17 explained how to create the cluster. The new cluster is always created from two subsequences so it will contain only two multi-dimensional subsequences.

Table 17: Multidimensional *Create* Operator

	Function C = CreateMultiDimCluster(*A*,*B*)
1	**C.**size = 2
2	**for** all dimension *d*
3	**C.**cen_d = $(A_d+B_d)/2$
4	**C.**seq_d = $\{A_d, B_d\}$
5	**C.**$shift_d$ = {0 0}
6	**end for**

Similarly, how to add a multi-dimension subsequence to an existing cluster is explained in Table 18. The idea of all three operators is same as in the single-dimension clustering algorithm.

Table 18: Multidimensional *Add* Operator

	Function C = AddToMultiDimCluster(**C**,*A*)
1	**C.**size = **C.**size+1
2	**for** all dimension *d*
3	**C.**cen_d = (**C.**cen_d*(**C.**size-1)+A_d*1)/**C.**size
4	**C.**seq_d.append(A_d)
5	**C.**$shift_d$.append(0)
6	**end for**

Table 19 explains in detail how to *merge* two clusters. As in the single dimension version, when two clusters are being merged, we have to align their center to find the best position, whose *bitsave* is maximized, after merging. We can align the center of the first cluster's center when it moves *into* the center of another cluster (line 10-21), and also when it moves *out of* another center (line 22-33). Each

position in alignment is kept as a local variable inside the corresponding cluster (line 20). However, only the cluster which maximum *bitsave* is return (line 34-35).

Table 19: Multidimensional *Merge* Operator

```
Function C' = MergeMultiDimClusters(C1,C2)
 1  n1=C1.size,     m1=length(C1.cen)
 2  n2=C2.size,     m2=length(C2.cen)
 3  C'.size = n1+n2
 4  bs = {}
 5  Ctmp = {}
 6  for all dimension d
 7      // add all sequences from C1 and C2 to C'
 8      C'.seqd = [C1.seqd ; C2.seqd]
 9  end for

10  // align C1.cen by moving to all positions
11  for off = 0 to m2
12     for all dimension d
13         // create centers of the same length
14         cen1 = [C2.cend(1,off), C1.cend]
15         cen2 = [C2.cend, C1.cend(1,m1+off-m2)]
16         C'.cend = (cen1*n1+cen2*n2)/(n1+n2)
17         C'.shiftd = [C1.shiftd+off; C2.shiftd]
18      end for
19     bs.append(ComputeBitsave(C',C1,C2))
20     Ctmp.append(C')
21  end for

22  // move C1.cend to opposite direction
23  for off = 1 to m1
24     for all dimension d
25         // create centers of the same length
26         cen1 = [C1.cend, C2.cend(1,m2+off-m1)]
27         cen2 = [C1.cend(1,off), C2.cend]
28         C'.cend = (cen1*n1+cen2*n2)/(n1+n2)
29         C'.shift = [C1.shiftd; C2.shiftd+off]
30         end for
31      bs.append(ComputeBitsave(C',C1,C2))
32      Ctmp.append(C')
33  end for
34  [bitsave index] = max(bs);
35  C' = Ctmp(index);
```

In next section, we will demonstrate that our algorithm can handle the time series which contains more than one dimension.

5.6.3. Experimental Results

We demonstrate our multi-dimensional clustering algorithm on a people activity dataset, *Physical Activity Monitoring for Aging People* (PAMAP) from [145]. In this dataset, eight subjects (people) perform

both indoor and outdoor activities such as normal walk, Nordic walk, cycle, run, ascend stairs, and descend stairs. Each subject has sensors placed on three locations on their bodies, which are hand, chest and ankle. We pick one time series from each position; we choose to use only three time series generated by z-accelerometers instead of using all data from all 45 sensors because experience from indexing ASL [119] and motion capture data suggest that three or four time series is enough to represent most activities/behaviors.

A snippet of a three dimensional time series showing an individual walking up and down stairs is shown in Figure 66. Each time series are generated from a z-accelerometer. The data is from subject number 1 of indoor activities in PAMAP dataset [145]. In this example, the subject performed the sequence, walking up stair, walking down stair, and walking up stair.

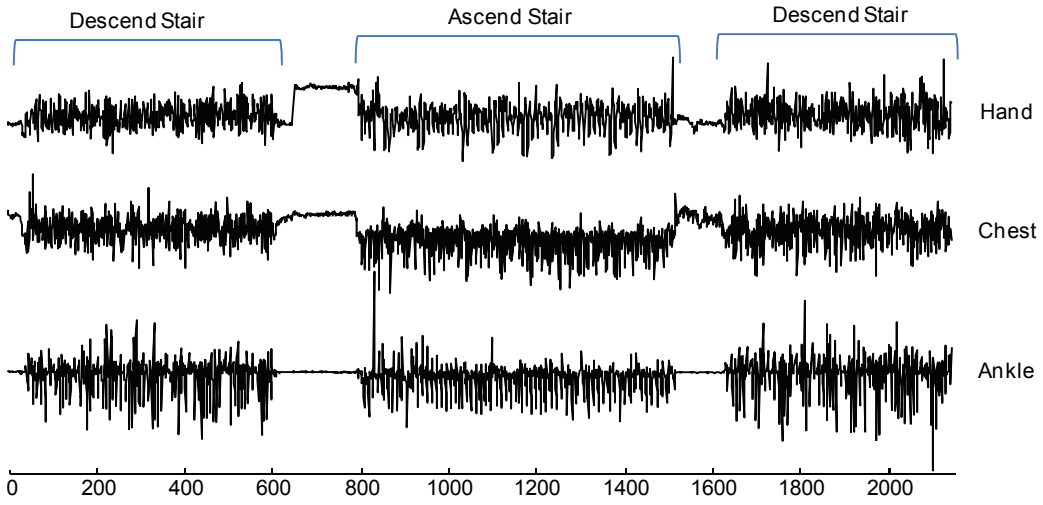

Figure 66: Three time series generated from z-accelerometer of sensors at hand, chest and shoe from PAMAP [145]. The subject performs three activities: descending stairs, ascending stairs, and descending stairs again

The multidimensional clustering result is shown in Figure 67. Each color represents a cluster; hence, the subsequences of the same color are clustered into the same cluster. The result demonstrates that our algorithm can capture the similar actions inside the same activities. For example, three clusters appeared in both side of the time series, which are in the same action, i.e., walking down stair. Some data in the time series has been ignored as we can see in black color. From this example, we conjecture that the subject may

occasionally have a special behavior because it looks like there are patterns inside the clusters. For example, it appear that she initially starts to climb stairs briskly (red), but begins to slow down as she was tired out after a few flights (blue). Likewise, the sequence of three clusters in the same order (pink, cyan, yellow) in *both* descending stairs sections is suggested.

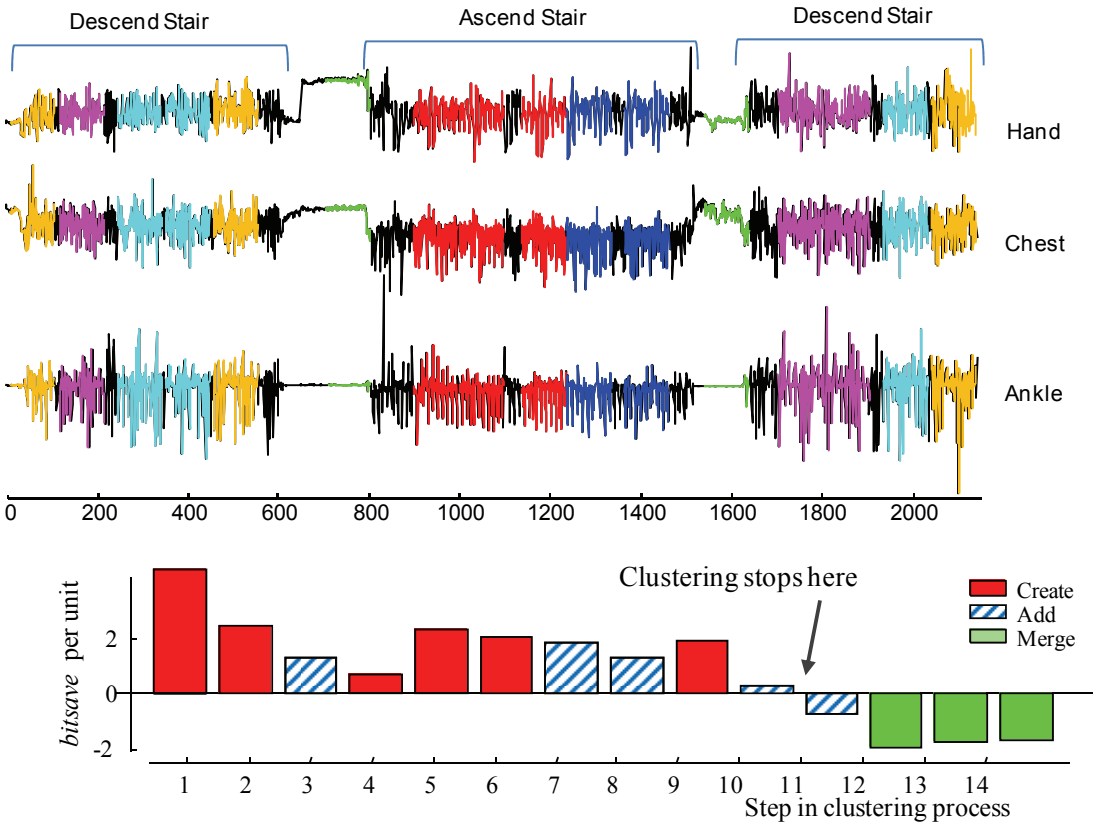

Figure 67: *top*) The multi-dimensional time series clustering result. Two clusters are detected in ascending stair, and three clusters are detected in descending stair. *bottom*) A trace of the multi-dimensional clustering steps produced by our algorithm

We also ran our single dimensional clustering algorithm on each dimension separately. The result shows that there are incorrect clusters across the different activities in the final clusters and, in some dimension, the algorithm is terminated very early and covers less than 25% of the data. Hence, in this example, the clustering result from a single dimension is worse than the result form the multi-dimensional clustering algorithm. We believe that the multi-dimensional time series can create the better clusters than

the clusters created from just one dimension, especially when the dataset contains many related features such as motion capture.

We can speed up the algorithm using Euclidean distance and make the problem solvable in an acceptable time, but we have to trade of the quality of the clustering somehow. Our parameter-lite algorithm can be extended to cluster multi-dimensional time series and it works quite well; however, some readers may notice that the *bitsave* score (in Figure 67.*bottom*) is not much stable as the result of a single-dimensional clustering algorithm in Section 5.5. This is because the Euclidean distance is used instead of MDL in many core modules. By the curse of dimensionality, when the number of dimensionality is increase, the Euclidean distance performs worse and the difference between MDL and Euclidean distance is increase. We believe that if the multi-dimensional time series clustering using MDL has been well study in the future, some researchers will provide the distance which is fast enough and very close to MDL to get the better result.

5.7 Conclusions

In this work, we have shown that any attempt to cluster a single time series stream that insists on explaining *all* the data is almost certainly doomed to failure. We introduced a clustering representation that has the expressive power to ignore some of the data, and can have clusters with different length subsequences. We further showed an efficient and parameter-lite MDL based algorithm to perform the clustering. We have shown on our algorithm is effective on a wide variety of datasets, for both single and multi-dimensional problems.

Currently our algorithm only works on batch time series. In ongoing work, we are attempting to generalize it to the online case.

Chapter 6: Conclusion

This dissertation discusses several interesting issues in high dimensional time series data mining. We show that the time complexity of high dimensional data mining is high and that it is not appropriate to apply the traditional data mining algorithm to solve the high dimensional data mining problem; even the state-of-the-art algorithm cannot obtain the results in an acceptable amount of time. For example, the experimental result in Chapter 4 shows that the state-of-the-art algorithm spends a whole day finding the repeated patterns (near duplicated figures) of the first 8 pages of a book, but our proposed algorithm can finish them in a few seconds.

We believe that this study contributes to the data mining community. From the algorithm perspective, we include many efficient algorithms to mine high dimensional data such as in nearest neighbor searches, motif discovery, shapelet discovery, classification and clustering. We show the drawbacks of the best known algorithms on high dimensional data, and propose new algorithms which are much more efficient than the previous ones. We give some mathematical proofs and demonstrate the usefulness of our algorithms for various kinds of data, such as time series data, streaming data, medical data, biological data, astronomy data, activity data, image data, historical manuscript data, hand-written manuscript data, gesture recognition data, etc. We hope that this study can show the variety of high dimensional data mining in different kinds of datasets.

To deal with high dimensional data, we demonstrate two effective approaches: using lower bounding techniques to prune some data, and solving the problem with approximate algorithms. In Chapter 2, we propose a fast nearest neighbor search algorithm which is on the order of magnitudes faster than the previous state-of-the-art algorithm and can return the exact same results. In Chapter 3, we propose an approximate algorithm for finding time series shapelets which overcome the limitations of both the running time and space of the best known algorithm; this makes the time series shapelet discovery tractable and more useful to the data mining community. In Chapter 4, the approximation algorithm for finding repeated

figures inside manuscripts is proposed. The algorithm returns the approximate results in a very short amount of time. The bound of the error and running of our algorithm is explained in the Appendix.

Moreover, we emphasize to the data mining community that we cannot explain all data; otherwise, we risk explaining some unexplainable data, such as noise. We propose an MDL-based clustering algorithm, which is a parameter-free algorithm. Having the capability to ignore some data is important; otherwise, in many situations the clustering algorithm cannot generate the correct answer.

I would like to share some useful things which I have learned during this dissertation process:

- Lower bounding is a well-known technique which can make algorithms faster. Lower bounding is a basis in heuristic search algorithms. However, in Chapter 2, we have to cleverly choose some lower bounds. Some lower bounds are more useful than others in terms of the running time and pruning ratio. Not only one but also many lower bounds can be applied if they can save some computational power.
- The order of the search is also important. Many problems can be solved by carefully ordering the search space such as in A* search or Dijkstra's algorithm. However, such ordering can be applied to any level of search; as we show in Chapter 2, we can order the calculation and prune further calculations if possible.
- Different datasets have different characteristics and, for learning how good and bad of the proposed algorithms, we have to test them on many datasets. However, honesty is most important, so "test all and report all!" Moreover, one useful dataset can be very important and it can make a good contribution to your papers.
- A fast algorithm is good but not good enough for publication, especially when proposing an approximate algorithm. The usefulness of an approximate algorithm is dependent on both speed and accuracy. At least two orders of magnitude for an approximation algorithm or one order of magnitude for an exact algorithm is needed for publication.
- Parameters can kill the usefulness of an algorithm and reduce the number of citations. To avoid over-fitting, an algorithm should have no parameters or the smallest number of parameters.
- Knowledge from other fields such as bioinformatics can provide some interesting ideas.
- Last but not least, trust and respect your advisor; he is always the best person for you ;-)

Bibliography

Chapter 1

[1] T. Bernecker, M. Houle, H. P. Kriegel, P. Kröger, M. Renz, E. Schubert, and A. Zimek, A. 2011. Quality of similarity rankings in time series. Advances in Spatial and Temporal Databases, 422-440.

[2] R. E. Bellman. 1957. Dynamic programming. Princeton University Press.

[3] E. J. Keogh, A. Mueen. 2010. Curse of Dimensionality. Encyclopedia of Machine Learning. 257-258

[4] E. T. Jaynes .1957. Information theory and statistical mechanics. Physical review, 106(4), 620.

[5] M. I. Shamos, and D. Hoey. 1975. Closest-point problems. In Foundations of Computer Science, 151-162.

[6] J. M. Kleinberg, 1997. Two algorithms for nearest-neighbor search in high dimensions. In Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 599-608.

[7] E. J. Keogh and J. Lin, "Clustering of time-series subsequences is meaningless: implications for previous and future research," Knowl. Inf. Syst., vol. 8, no. 2, 2005, pp. 154-177.

Chapter 2

[8] N. Adams, D. Marquez, and G. Wakefield. 2005. Iterative deepening for melody alignment and retrieval. ISMIR, 199-206.

[9] I. Assent, R. Krieger, F. Afschari, and T. Seidl. 2008. The TS-Tree: efficient time series search and retrieval. EDBT, 252-63.

[10] J. Alon, V. Athitsos, Q. Yuan, and S. Sclaroff. 2009. A unified framework for gesture recognition and spatiotemporal gesture segmentation. IEEE PAMI 31, 9, 1685-1699.

[11] T. Bragge, M.P. Tarvainen, and P. A. Karjalainen. 2004. High-Resolution QRS Detection Algorithm for Sparsely Sampled ECG Recordings. Univ. of Kuopio, Dept. of Applied Physics Report.

[12] N. Chadwick, D. McMeekin, and T. Tan. 2011. Classifying eye and head movement artifacts in EEG Signals. DEST.

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

[14] B. Dupasquier and S. Burschka. 2011. Data mining for hackers – encrypted traffic mining. *The 28th Chaos Comm' Congress*.

[15] Y. Chen, G. Chen, K. Chen, and B. C. Ooi. 2009. Efficient processing of warping time series join of motion capture data. *ICDE*, 1048-1059.

[16] Faceted DBLP. 2011. *http://dblp.l3s.de*

[17] A. Fornés, J. Lladós, and G. Sanchez. 2007. Old handwritten musical symbol classification by a dynamic time warping based method. *Graphics Recognition* 5046, 51-60.

[18] A. Fu, E. Keogh, L. Lau, C. Ratanamahatana, and R. Wong. 2008. Scaling and time warping in time series querying. *VLDB J.* 17, 4, 899-921.

[19] N. Gillian, R. Knapp, and S. O'Modhrain. 2011. Recognition of multivariate temporal musical gestures using n-dimensional dynamic time warping. *Proc of the 11th Int'l conference on New Interfaces for Musical Expression*.

[20] D. Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys* 23, 1.

[21] G. Guitel. 1975. *Histoire comparée des numérations écrites*. Chapter: "Les grands nombres en numération parlée," Paris: Flammarion, 566–574.

[22] R. Huber-Mörk, S. Zambanini, M. Zaharieva, and M. Kampel. 2011. Identification of ancient coins based on fusion of shape and local features. *Mach. Vis. Appl.* 22, 6, 983-994.

[23] M. Hsiao, K. West, and G. Vedatesh. 2005. Online context recognition in multisensor system using dynamic time warping. *ISSNIP*, 283-288.

[24] H. Jegou, M. Douze, C. Schmid, and P. Perez. 2010. Aggregating local descriptors into a compact image representation. *IEEE CVPR*, San Francisco, CA, USA.

[25] T. Kahveci and A. K. Singh. 2004. Optimizing similarity search for arbitrary length time series queries. *IEEE Trans. Knowl. Data Eng.* 16, 4, 418-433.

[26] E. Keogh and S. Kasetty. 2003. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and Knowledge. Discovery* 7, 4, 349-371.

[27] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.H. Lee, and P. Protopapas. 2009. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. *VLDB J.* 18, 3, 611-630.

[28] S. Kim, S Park, and W. Chu. 2001. An index-based approach for similarity search supporting time warping in large sequence databases. *ICDE*, 607–61.

[29] K. Laerhoven, E. Berlin, and B. Schiele. 2009. Enabling efficient time series analysis for wearable activity data. *ICMLA*, 392-397.

[30] S. H. Lim, H. Park, and S. W. Kim. 2007. Using multiple indexes for efficient subsequence matching in time-series databases. *Inf. Sci.* 177, 24, 5691-5706.

[31] D. P. Locke, L. W. Hillier, W. C. Warren, et al. 2011. Comparative and demographic analysis of orangutan genomes. *Nature* 469, 529-533.

[32] A. Mueen and E. Keogh. 2010. Online discovery and maintenance of time series motifs. *KDD*, 1089-1098.

[33] A. Mueen, E. Keogh, Q. Zhu, S. Cash, M. B. Westover, and N. Shamlo. 2011. A disk-aware algorithm for time series motif discovery. *Data Min. Knowl. Discov.* 22, 1-2, 73-105.

[34] M. Muller. 2009. Analysis and retrieval techniques for motion and music data. *EUROGRAPHICS tutorial*.

[35] P. Papapetrou, V. Athitsos, M. Potamias, G. Kollios, and D. Gunopulos. 2011. Embedding-based subsequence matching in time-series databases. *ACM TODS* 36, 3, 17*.

[36] W. Pressly. 2008. TSPad: a Tablet-PC based application for annotation and collaboration on time series data. *ACM Southeast Regional Conference*, 527-52.

[37] B. Raghavendra, D. Bera, A. Bopardikar, and R. Narayanan. 2011. Cardiac arrhythmia detection using dynamic time warping of ECG beats in e-healthcare systems. *WOWMOM*, 1-6.

[38] U. Rebbapragada, P. Protopapas, C. Brodley, and C. Alcock. 2009. Finding anomalous periodic time series. *Machine Learning* 74, 3, 281-313.

[39] Y. Sakurai, C. Faloutsos, and M. Yamamuro. 2007. Stream monitoring under the time warping distance. *ICDE*, 1046-55.

[40] Y. Sakurai, M. Yoshikawa, and C. Faloutsos. 2005. FTW: fast similarity search under the time warping distance. *PODS'05*.

[41] S. Srikanthan, A.Kumar, and R. Gupta. 2011. Implementing the dynamic time warping algorithm in multithreaded environments for real time and unsupervised pattern discovery. *IEEE ICCCT*, 394-398.

[42] J. Shieh and E. J. Keogh. 2008. *i*SAX: indexing and mining terabyte sized time series. *KDD*, 623-631.

[43] T. Stiefmeier, D. Roggen, and G. Tröster. 2007. Gestures are strings: efficient online gesture spotting and classification using string matching. *Proceedings of the ICST 2nd international conference on Body area networks.*

[44] C. R. Whitney. 1997. Jeanne Calment, World's elder, dies at 122. *New York Times* (August 5th, 1997).

[45] J. O. Wobbrock, A. D. Wilson, and Y. Li. 2007. Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. *ACM UIST*, 159-168.

[46] L. Ye and E. Keogh. 2009. Time series shapelets: a new primitive for data mining. *KDD*, 947-956.

[47] B. Yi, H. Jagadish, and C. Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. *ICDE*, 201-208.

[48] Y. Zhang and J. Glass. 2011. An inner-product lower-bound estimate for dynamic time warping. *ICASSP*, 5660-5663.

[49] A. Zinke and D. Mayer. 2006. *Iterative Multi Scale Dynamic Time Warping*. Universität Bonn, Tech Report # CG-2006-1.

[50] Supporting Website: *www.cs.ucr.edu/~eamonn/UCRsuite.html*

Chapter 3

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[81] Supporting Website: *www.cs.ucr.edu/~rakthant/FastShapelet*

Chapter 4

[82] X. Bai, X. Yang, L. Latecki, and W. Liu, "Learning context sensitive shape similarity by graph transduction," IEEE TPAMI, 2009.

[83] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, 1981, pp. 111-22.

[84] S. S. Bukhari, F. Shafait, and T. M. Breuel, "Improved document image segmentation algorithm using multiresolution morphology," *Document Recognition and Retrieval*, 2011, pp. 1-10.

[85] B. J. Burke, Book of Orders of Knighthood and Decorations of Honour of all Nations, London: Hurst and Blackett, 1858.

[86] O. Chum, J. Philbin, M. Isard, and A. Zisserman, "Scalable near identical image and shot detection," *CIVR*, 2007, pp. 549-556.

[87] O. Chum, J. Philbin, and A. Zisserman,"Near Duplicate Image Detection: min-Hash and tf-idf Weighting," *BMVC*, 2008.

[88] C. Boutell, A Manual of Heraldry, Historical and Popular, Winsor and Newton, 1863.

[89] C. Davenport, *British Heraldry*, Methuen, London, 1921.

[90] C. Davenport, *English heraldic book-stamps, figured and described*, London: Archibald Constable. ltd, 1909.

[91] C. R. Dod, and R. P. Dod, *Dod's Peerage, Baronetage and Knightage of Great Britain and Ireland for 1915*, London: Simpkin, Marshall, Hamilton, Kent. ltd, 1915.

[92] E. E. Dorling, *Leopards of England, and other papers on heraldry*, Constable & Company limited, London, 1913.

[93] R. Duda, and P. Hart, "Use of the Hough transform to detect lines and curves in pictures," *Comm. ACM*, vol. 15, 1, 1972, pp. 11-15.

[94] A. Fornés, J. Lladós, and G. Sanchez, "Old Handwritten Musical Symbol Classification by a Dynamic Time Warping Based Method," *Graphics Recognition*, 2007.

[95] B. Gatos, I. Pratikakis, and S. J. Perantonis, "An adaptive binarisation technique for low quality historical documents," *Workshop on Document Analysis Systems*, 2004.

[96] C. Grana, D. Borghesani, and R. Cucchiara. "Automatic segmentation of digitalized historical manuscripts," *Multimedia Tools Applications*, vol. 55, 3, 2011, pp. 483-506.

[97] P. V. C. Hough, Method and mean for recognizing complex pattern. USA patent 3069654, 1996.

[98] E. Kavallieratou and E. Stamatatos, "Adaptive binarization of historical document images," *ICPR*, 2006, pp.742–745.

[99] Y. Ke, R. Sukthankar, L. Huston, "An efficient parts-based near-duplicate and sub-image retrieval system," *ACM Multimedia*, 2004, pp. 869-876.

[100] E. Keogh, L. Wei, X. Xi, M. Vlachos, S. Lee, and P. Protopapas, "Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures," *VLDB Journal*, vol. 18, 3, 2009, pp. 611-30.

[101] T. Koch-Grünberg, *Südamerikanische Felszeichnungen* (South American petroglyphs), Berlin, E. Wasmuth A-G, 1907.

[102] J. Mas, G. Sanchez, and J. Llados, "An Incremental Parser to Recognize Diagram Symbols and Gestures represented by Adjacency Grammars," *Graphics Recognition*, 2006, pp. 252-263.

[103] A. Mueen, E. Keogh, and N. Shamlo, "Finding Time Series Motifs in Disk-Resident Data," *ICDM*, 2009, pp. 367-376.

[104] A. Pritchard, *A history of Infusoria, including Desmidiaceae and Diatomaceae*, British and foreign. Ed. IV. 968. London, 1861.

[105] G. Ramponi, F. Stanco, W. D. Russo, S. Pelusi, and P. Mauro, *Digital automated restoration of manuscripts and antique printed books*, 2005, EVA.

[106] J. V. Richardson Jr., Bookworms: The Most Common Insect Pests of Paper in Archives, Libraries, and Museums.

[107] G. Sanchez, et al., "A platform to extract knowledge from graphic documents. application to an architectural sketch understanding scenario," *DAS*, 2004, pp. 389-400.

[108] K. B. Schroeder, et al. "Haplotypic Background of a Private Allele at High Frequency in the Americas," *Molecular Biology and Evolution*, 2009, pp. 995-1016.

[109] W. Smith, A synopsis of the British Diatomaceae: with remarks on their structure, function and distribution, pp. [V]-XXXIII, pp. 1-89, 31 pls. London, 1853.

[110] Smith, G. A. and Turner, W. G. Indian Rock Art of Southern California with Selected Petroglyph Catalog, San Bernardino County, 1975.

[111] T. Rakthanmanon, Q. Zhu, and E. J. Keogh. "Mining Historical Documents for Near-Duplicate Figures," *ICDM*, 2011, pp. 557-566.

[112] M. Tompa and J. Buhler, "Finding motifs using random projections," *Computational Molecular Biology*, 2001, pp. 67-74.

[113] W. West and G. S. West, *A Monograph of the British Desmidiaceae*, vols. I–V, Ray Soc, London, 1904.

[114] H. J. Wolfson and I. Rigoutsos, "Geometric Hashing: An Overview," *IEEE Computer Science*, vol. 4, 4, 1997.

[115] X. Xi, E. Keogh,L. Wei, and A. Mafra-Neto, "Finding Motifs in a Database of Shapes," *SIAM Conference on Data Mining*, 2007.

[116] Q. Zhu, X. Wang, E. Keogh, and S. H. Lee, "Augmenting the Generalized Hough Transform to Enable the Mining of Petroglyphs," *SIGKDD*, 2009.

[117] Mining Historical Archives for Near-Duplicate Figures 2011 *http://www.youtube.com/watch?v=QYY8A6CwS-A*

[118] Supporting Website: http://www.cs.ucr.edu/~rakthant/DocMotif

Chapter 5

[119] V. Athitsos, H. Wang, and A. Stefan, "A database-based framework for gesture recognition," Personal and Ubiquitous Computing, vol. 14, no. 6, 2010, pp. 511-526.

[120] T. Bastogne, H. Noura, A. Richard, and J. M. Hittinger, "Application of subspace methods to the identification of a winding process," Proc. of the 4th European Control Conference, Brussels, Belgium, 1997.

[121] G. E. A. P. A. Batista, X. Wang, E. J. Keogh, "A Complexity-Invariant Distance Measure for Time Series," SDM, 2011, pp. 699-710.

[122] D. Bouchard and N. I. Badler, "Semantic Segmentation of Motion Capture Using Laban Movement Analysis," IVA, 2007, pp 37-44 .

[123] J. R. Chen. "Making Subsequence Time Series Clustering Meaningful," ICDM, 2005, pp. 114-121.

[124] J. R. Chen., "Useful Clustering Outcomes from Meaningful Time Series Clustering," The Australasian Data Mining Conference. 2007.

[125] J. R. Chen., Making clustering in delay-vector space meaningful. Knowl. Inf. Syst. 11, 3 (2007), 369-385.

[126] Z. J. Chuang, C. H. Wu, and W. S. Chen, "Movement Epenthesis Generation Using NURBS-Based Spatial Interpolation," IEEE Trans. Circuit and Systems for Video Technology, vol. 16, no. 11, Nov. 2006, pp. 1313-1323.

[127] D. J. Cook and L. B. Holder, "Substructure Discovery Using Minimum Description Length and Background Knowledge," J. Artificial Intelligence Research, vol. 1 , 1994, pp. 231-255.

[128] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth, "Rule Discovery from Time Series," Proc. of the 3rd KDD, 1998, pp. 16-22.

[129] A. M. Denton, C. A. Basemann, and D. H. Dorr, "Pattern-based time-series subsequence clustering using radial distribution functions," Knowledge and Information Systems journal, vol. 18, No. 1, Jan. 2009, pp. 1-27.

[130] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh, "Querying and mining of time series data: experimental comparison of representations and distance measures," PVLDB, vol. 1, no. 2, 2008, pp. 1542-1552.

[131] S. C. Evans et. al. "MicroRNA target detection and analysis for genes related to breast cancer using MDLcompress," EURASIP J. Bioinform. Syst. Biol., 2007, pp. 1-16.

[132] S. C. Evans, E. Eiland, T. S. Markham, J. Impson, and A. Laczo, "MDLcompress for Intrusion Detection: Signature Inference and Masquerade Attack," MILCOM, Orlando, Florida, 2007.

[133] P.D. Grünwald, I.J. Myung, and M.A. Pitt, Advances in Minimum Description Length: Theory and Applications, MIT Press, 2005.

[134] I. Jonyer, L. B. Holder, and D. J. Cook, "MDL-based context-free graph grammar induction and applications," Journal on Artificial Intelligence Tools, vol. 13, no. 1, 2004, pp. 65-79.

[135] S. D. Kamvar, D. Klein, and C. D. Manning, "Interpreting and Extending Classical Agglomerative Clustering Algorithms using a Model-Based approach," ICML, 2002, pp. 283-290.

[136] E. J. Keogh and J. Lin, "Clustering of time-series subsequences is meaningless: implications for previous and future research," Knowl. Inf. Syst., vol. 8, no. 2, 2005, pp. 154-177.

[137] E. J. Keogh, J. Lin, S. H. Lee, and H V. Herle, "Finding the most unusual time series subsequence: algorithms and applications," Knowl. Inf. Syst., vol 11, no. 1, 2007, pp. 1-27.

[138] E. J. Keogh and S. Kasetty, "On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration," Data Mining and Knowledge Discovery, vol. 7, no. 4, 2003, pp. 349-371.

[139] H. Li and N. Abe, "Clustering Words with the MDL Principle," Proc. of the 16th Int' Conf' on Computational Linguistics, 1996, pp. 5-9.

[140] M. Li and P. Vitanyi, An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed., Springer Verlag, 1997.

[141] Y.I. Molkov, D. N. Mukhin, E. M. Loskutov, and A. M. Feigin, Using the minimum description length principle for global reconstruction of dynamic systems from noisy time series. Phys. Rev. E 80, 046207, 2009.

[142] A. Mueen, E. J. Keogh, and N. B. Shamlo, "Finding Time Series Motifs in Disk-Resident Data," ICDM, 2009, pp. 367-376.

[143] S. Papadimitriou, J. Sun, C. Faloutsos, and P. S. Yu, "Hierarchical, Parameter-Free Community Discovery," PKDD 2008, pp. 170-187.

[144] E. Pednault, "Some Experiments in Applying Inductive Inference Principles to Surface Reconstruction," IJCAI, 1998, pp. 1603-09.

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

[146] R. A. Stine, "Model Selection Using Information Theory and the MDL Principle," Sociological Methods and Research, vol. 33, no. 2, Nov. 2004, pp. 230-260.

[147] Y. Tanaka, K. Iwamoto, and K. Uehara, K. "Discovery of time-series motif from multi-dimensional data based on MDL principle," Machine Learning, vol. 58, no. 2, 2005.

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

[149] C. S. Wallace and D. M. Boulton, 1968. An information measure for classification. Computer Journal vol. 11, no. 2, Aug. 1968, pp. 185-194.

[150] R. Yang, S. Sarkar, and B. L. Loeding, "Handling Movement Epenthesis and Hand Segmentation Ambiguities in Continuous Sign Language Recognition Using Nested Dynamic Programming," IEEE PAMI, vol. 32, no. 3, 2010, pp. 462-477.

[151] D. Yankov, E. J. Keogh, U. Rebbapragada, "Disk aware discord discovery: finding unusual time series in terabyte sized datasets," Knowl. Inf. Syst., vol 17, no. 2, 2008, pp. 241-262.

[152] Supporting Website. http://www.cs.ucr.edu/~rakthant/TSEpenthesis

Appendix: Mathematical Analysis

According to our document motif discovery algorithm in Chapter 3, there are three main steps make our algorithm ultra fast by reducing the number expensive real distance calculations. Firstly, we locate the potential windows among all windows inside the books. With a good preprocessing, potential windows are not hard to locate as we describe in Section 4.4, and the number of potential windows are, expectedly, in the same order of magnitude as the number of figures inside the books which is depended on window size. Secondly, we apply our hashing technique and then calculate the distances between every window pairs, which share same signatures. While a motif collides, some of other pairs may also collide by coincidence; we call this kind of windows, *non-motif*. It is non-trivial to calculate the expected number of false collisions, which is at the heart of this section. Thirdly, instead of calculating all expansive real distances, we apply the lower bound first introduced by Zhu et al. [116] to reduce the number of real distance calculations.

In this section, we will guarantee the maximum number of false collisions occurred in our random projection process.

Assumptions:

In order to give the number of false collision, our assumptions are:

1. In each image, black pixels are appeared randomly and uniformly.

2. The motif is the pair of windows which has the smallest distance. Thus, in this proof, all other pairs are considered as non-motifs and can only increase the number of false collisions.

Note that in real situation, there are many motifs or similar figures, and the number of false collision will be smaller than the one shown in this section.

3. We know in advance the mean μ and standard deviation σ of the distribution of the distances of all window pairs.

Note that we do not assume that all windows have the same number of black pixels or, even, the distance distribution is Gaussian.

Notations:

For simplicity, we use some new nicknames for some parameters introduced in Section 4.4.

N: user-defined size of image. ($N=s_x*s_y$)

s : masking ratio ($0 \le s \le 1$) or *mask*.

t : number of iteration or *it*.

μ: mean of distance distribution from all window pairs.

σ : standard derivation of the distance distribution.

conf : user-defined confidence which is the probability that at least one iteration the motif will appear in the same bucket.

Lemma1: *Given windows W_a and W_b, if d=dist(W_a,W_b), the probability that W_a and W_b will collide in 1 iteration of random projection is:*

$$s^d > P[W_a \text{ and } W_b \text{ collide in 1 iteration}] > \left(\frac{sN-d+1}{N-d+1}\right)^d$$

Proof:

Because the distance between W_a and W_b is *d*, if the removed pixels cover all of these *d* pixels, W_a and W_b will have the same signature, the remaining pixels. In our hashing process, we randomly remove *sN* pixels from *N*-pixel windows.

Then, the probability of distance *d* will collide is :

$$P_d = P[W_a \text{ and } W_b \text{ collide in 1 iteration}]$$

$$= P[W_a \text{ and } W_b \text{ has same hash signature}]$$

$$= \frac{\#\text{ of ways that all } d \text{ pixels are removed when we remove } sN \text{ pixels}}{\#\text{ of all possible ways to remove } sN \text{ pixels}}$$

$$= \frac{\binom{N-d}{sN-d}}{\binom{N}{sN}} = \frac{(N-d)!}{(sN-d)!\,\cancel{(N-sN)!}} \frac{(sN)!\,\cancel{(N-sN)!}}{N!}$$

$$= \frac{sN(sN-1)(sN-2)\ldots(sN-d+1)}{N\,(N-1)\,(N-2)\,\ldots\,(N-d+1)}$$

Because of $1 \geq s \geq 0$, then

$$\frac{sN}{N} > \frac{sN-1}{N-1} > \frac{sN-2}{N-2} > \cdots > \frac{sN-d+1}{N-d+1}$$

Note that $P_0 = \frac{sN}{N} = s$ and $\forall d > 1, P_d = P_{d\text{-}1} * \left(\frac{sN-d+1}{N-d+1}\right)$. Then, $P_{d\text{-}1} > P_d$. Hence, P_d is monotonic decreasing.

$$s^d = \left(\frac{sN}{N}\right)^d > \frac{sN(sN-1)(sN-2)\ldots(sN-d+1)}{N\,(N-1)\,(N-2)\,\ldots\,(N-d+1)} > \left(\frac{sN-d+1}{N-d+1}\right)^d$$

Therefore, $s^d > P[W_a \text{ and } W_b \text{ collide}] > \left(\frac{sN-d+1}{N-d+1}\right)^d$ □

Lemma2: *For given windows W_a and W_b, if d=dist(W_a,W_b), the probability that they will collide in t iteration is:*

$$P[W_a \text{ and } W_b \text{ collide in } t \text{ iterations}] > 1 - \left(1 - \left(\frac{sN-d+1}{N-d+1}\right)^d\right)^t$$

Proof:

Let $p=P[W_a \text{ and } W_b \text{ collide in 1 iteration}]$. From Lemma 1, $p > \left(\frac{sN-d+1}{N-d+1}\right)^d$.

Then, P[W_a and W_b collide in *t* iterations]=1- $(1- p)^t$. □

Corollary1: *If the motif whose distance is d collides with probability at least user-defined confidence, conf, and the value of the number of iteration t is given, then, the masking ratio s which satisfy that the motif will be collide with confidence conf is:*

$$s \geq \frac{1}{N}\left[\left(1 - (1-conf)^{1/t}\right)^{1/d} * (N-d+1) + d - 1\right]$$

Proof:

By Lemma2, *P*[the motif collides in *t* iterations], $s \geq 1 - \left(1 - \left(\frac{sN-d+1}{N-d+1}\right)^d\right)^t \geq conf$ □

Corollary2: *If the motif whose distance is d collides with probability at least user-defined confidence, conf, and the value of the masking ratio s is given, then, the number of iterations t which satisfy that the motif will be collide with confidence conf is:*

$t \geq log\,(1-conf) \,/\, log\,\left(1 - (\frac{sN-d+1}{N-d+1})^d\right)$

Proof:

By Lemma2, *P*[the motif collides in *t* iterations], $t \geq 1 - \left(1 - \left(\frac{sN-d+1}{N-d+1}\right)^d\right)^t \geq conf$ □

Now we can guarantee the minimum probability that the motif will collide (share same signature) in at least one iteration by Corollary1. For the rest of this section, we assume that the user want to find the motif with probability at least *conf*, i.e., 99%. Thus, we will find the upper bound of the probability that other pairs of windows, non-motifs, will collide after removing some black pixels. Contrary to the motif, increasing number of iterations can increase the number of non-motif collisions.

Chebyshev's Inequality:

Given a distribution X with mean μ and standard deviation σ and $k \geq 0$, then, $P[|x - \mu| \geq k\sigma] \leq \frac{1}{k^2}$

Then, $\frac{1}{k^2} \geq P[|x - \mu| \geq k\sigma] \geq P[\mu - x \geq k\sigma] = P[\mu - k\sigma \geq x]$

Substitute variable by $k = \frac{\mu - d}{\sigma}$; hence, $P[x \leq d] \leq \frac{\sigma^2}{(\mu - d)^2}$

Lemma3: *In one iteration, any pair of windows will collide at most*

$$2\sigma^2 s^{\mu-1} \sum_{i=1}^{\mu-1} 1/(s^i i^3) \; + \; s^{u-1}.$$

Proof:

For any given windows W_c and W_d, by Lemma1, we know that

$P[W_c \text{ and } W_d \text{ collide} \mid dist(W_c, W_d)=d] = P_d < s^d$

P[any pair of windows collides in 1 iteration]

$= \int_0^{\infty} P[dist(W_c, W_d)=x] * P[W_c, W_d \text{ collide} \mid dist(W_c, W_d)=x] \mathrm{d}x$

$= \sum_{d=0}^{\infty} \int_d^{d+1} P[dist(W_c, W_d)=x] * P[W_c, W_d \text{ collide} \mid dist(im_1, im_2)=x] \mathrm{d}x$

From the definition P_d in Lemma1,

$= \sum_{d=0}^{\infty} \int_d^{d+1} P[dist(W_c, W_d)=x] * P_x \, \mathrm{d}x$

Because P_d is monotonic decreasing,

$\leq \sum_{d=0}^{\infty} \int_d^{d+1} P[dist(W_c, W_d)=x] * P_d \, \mathrm{d}x$

$= \sum_{d=0}^{\infty} P_d \int_d^{d+1} P[dist(W_c, W_d)=x] \mathrm{d}x$

From Lemma1,

$$\leq \sum_{d=0}^{\infty} s^d \int_d^{d+1} P[dist(W_c, W_d)=x]\mathrm{d}x$$

$$\leq \sum_{d=0}^{\infty} s^d P[d \leq dist(W_c, W_d) < d+1]$$

$$= \sum_{d=0}^{\mu-2} s^d P[d \leq dist(W_c, W_d) < d+1] \quad + \sum_{d=u-1}^{\infty} s^d P[d \leq dist(W_c, W_d) < d+1]$$

$$\leq \sum_{d=0}^{\mu-2} s^d P[d \leq dist(W_c, W_d) < d+1] \quad + s^{u-1} \sum_{d=u-1}^{\infty} P[d \leq dist(W_c, W_d) < d+1]$$

$$\leq \sum_{d=0}^{\mu-2} s^d P[d \leq dist(W_c, W_d) < d+1] \quad + s^{u-1}(1 - P[\mu\text{-}1 \leq dist(W_c, W_d)])$$

$$\leq \sum_{d=0}^{\mu-2} s^d P[d \leq dist(W_c, W_d) < d+1] \quad + s^{u-1}$$

The maximum is obtained when all Chebyshev's inequalities are tight.

When the inequalities are tight, we have

$P[d \leq dist(im_1, im_2) \leq d+1]$

$$= P[dist(im_1, im_2) \leq d+1] - P[dist(im_1, im_2) \leq d]$$

$$= \frac{\sigma^2}{(\mu-d-1)^2} - \frac{\sigma^2}{(\mu-d)^2}$$

P[any pair of windows collides in 1 iteration]

$$\leq \sum_{d=0}^{\infty} s^d P[d \leq dist(W_c, W_d) < d+1]$$

$$= \sum_{d=0}^{\mu-2} s^d P[d \leq dist(W_c, W_d) < d+1] \quad + \sum_{d=u-1}^{\infty} s^d P[d \leq dist(W_c, W_d) < d+1]$$

$$= \sum_{d=0}^{\infty} s^d \left(\frac{\sigma^2}{(\mu-d-1)^2} - \frac{\sigma^2}{(\mu-d)^2} \right) + s^{u-1}$$

Substitute variable by $d=\mu-x$

$$= \sum_{d=0}^{\infty} s^{\mu-x}\sigma^2\left(\frac{1}{(x-1)^2}-\frac{1}{x^2}\right)+s^{u-1}$$

$$= \sum_{d=0}^{\infty} s^{\mu-x}\sigma^2\left(\frac{2x-1}{(x-1)^2x^2}\right)+s^{u-1}$$

$$\leq \sum_{d=0}^{\infty} s^{\mu-x}\sigma^2\left(\frac{2x}{(x-1)^2x^2}\right)+s^{u-1}$$

$$\leq \sum_{d=0}^{\infty} s^{\mu-x}\sigma^2\left(\frac{2}{(x-1)^3}\right)+s^{u-1}$$

Substitute variable by $x=i+1$

$$= \sum_{d=0}^{\infty} s^{\mu-i+1}\sigma^2\left(\frac{2}{i^3}\right)+s^{u-1}$$

$$= 2\sigma^2 s^{\mu+1}\sum_{d=0}^{\infty} 1/(s^i i^3)+s^{u-1}$$ □

Corollary3: In t iterations, any pair of windows will collide at most

$$1\text{-}(1\text{-}Q)^t \text{ where } Q = 2\sigma^2 s^{\mu+1}\sum_{d=0}^{\infty} 1/(s^i i^3)+s^{u-1}.$$

Proof: Obvious by Lemma3. □

Theorem1: In t iterations, any pair of windows will collide at most

$$ts^{u-1} + 2t\sigma^2 s^{\mu+1}\sum_{d=0}^{\infty} 1/(s^i i^3)$$

Proof: $1-(1-Q)^t = tQ-\binom{t}{2}Q^2+-\binom{t}{3}Q^3-\cdots) \leq tQ$. Then, follow by Corollary 3. □