A Novel Bit Level Time Series Representation with Implications for Similarity Search and Clustering

Chotirat Ann Ratanamahatana[1] Eamonn Keogh[1] Anthony J. Bagnall[2] Stefano Lonardi[1]
[1]*Dept. of Computer Science & Engineering*
University of California, Riverside, USA
ratana@cs.ucr.edu eamonn@cs.ucr.edu
[2]*School of Computing Sciences*
University of East Anglia, Norwich, UK
ajb@cmp.uea.ac.uk stelo@cs.ucr.edu

Abstract

Because time series are a ubiquitous and increasingly prevalent type of data, there has been much research effort devoted to time series data mining in recent years. As with all data mining problems, the key to effective and scalable algorithms is choosing the right representation of the data. Many high level representations of time series have been proposed for data mining, including spectral transforms, wavelets, Singular Value Decomposition (SVD), piecewise polynomial models, symbolic models, etc. In this work, we introduce a new technique based on a bit level approximation of the data. The representation has several important advantages over existing techniques. One unique advantage is that it allows raw data to be directly compared to the reduced representation, while still guaranteeing lower bounds to either Euclidean distance or DTW. This fact can be exploited to produce faster exact algorithms for similarly search. In addition, we demonstrate that our new representation allows time series clustering to scale to much larger datasets.

Keywords: Time series representation, similarity search, clustering.

1 Introduction

Time series are a ubiquitous and increasingly prevalent type of data. Because of this fact, there has been much research effort devoted to time series data mining in the last decade [1][7][8][22][39]. As with all data mining problems, the key to effective and scalable algorithms is choosing a suitable representation of the data. Many high level representations of time series have been proposed for data mining, including spectral transforms, wavelets, Singular Value Decomposition (SVD), piecewise polynomial models [39], symbolic models, etc. Figure 1 shows a complete hierarchy of all techniques proposed for data mining (see [20] for a survey). In this work, we introduce a novel technique based on a bit level approximation of the data. As we will show, our *clipped* representation has several important advantages over existing techniques.

Note that the proposed approach is not only a new representation; it is a new *type* of representation. For data adaptive, non-data adaptive, and model-based approaches, the user has a choice (implicit or explicit) of the compression ratio. This allows the user to fine tune the parameters to achieve the ideal compression/ fidelity tradeoff for their particular application.

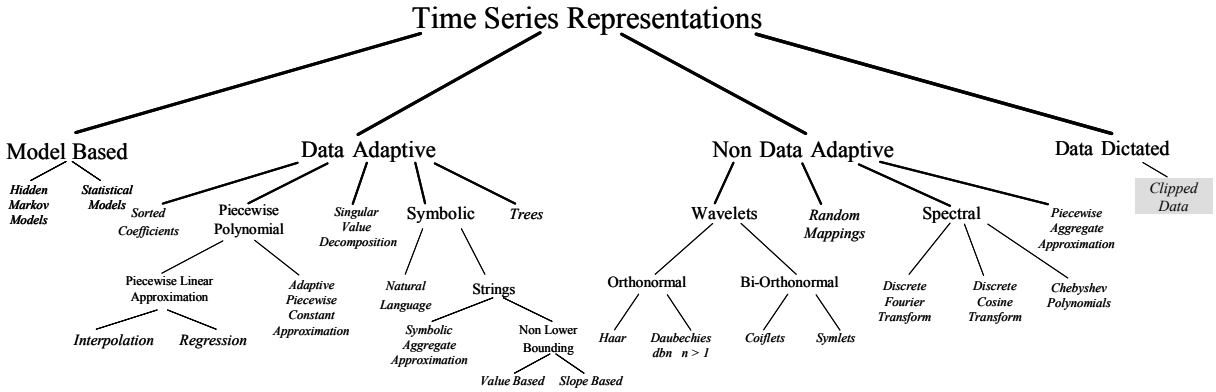

Figure 1. A Hierarchy of all time series representations proposed for data mining. The representation that is the contribution of this work is highlighted

In contrast, with the clipped representation, the data *itself* dictates the compression ratio; the user has no choice to make. This may be seen as somewhat of a disadvantage (although removing parameters from a data mining task is often a good thing [23]). However, this apparent lack of flexibility is counterbalanced by another unique property of the clipped representation. For all other dimensionality reduction approaches, we must transform the query into the same representation as the dimensionality reduced database. This means we have a loss of fidelity for the candidate matches stored in the index *and* a loss of fidelity for the query. The compound fidelity loss combines to produce weak lower bounds, and thus weak pruning power. In contrast, the clipped representation is unique in that the original raw query can be compared directly to the clipped candidate sequences, thus producing tighter lower bounds, greater pruning power and thus faster query by content.

The rest of the paper is organized as follows. In Section 2, we introduce the clipped representation and the distance measures defined on it with some background material. We further expand the reader's appreciation for the clipped representation in Section 3 by showing additional desirable properties, including its ability to support clustering. Section 4 contains a comprehensive empirical evaluation. Finally, in Section 5, we offer some conclusions to this work.

2 The clipped representation

Our proposed representation works by replacing each real valued data point with a single bit. Figure 2 gives the visual intuition.

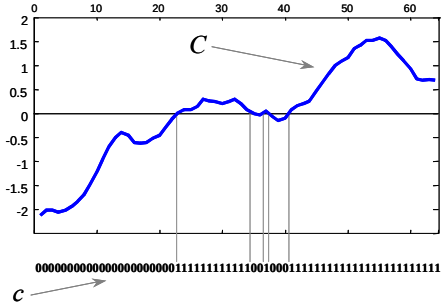

Figure 2. A time series of length 64, denoted *C*, is converted to the clipped representation, denoted *c*, simply by noting the elements of *C* that data points are strictly above zero, and setting the corresponding bits to 1, and setting all other corresponding bits to 0

More formally, we can define c, the clipped representation of C as:

$$c(i) = \begin{cases} 1 & if \quad C(i) > \mu \\ 0 & otherwise \end{cases} \qquad 1)$$

where μ is the mean of C. Since virtually all researchers have noted the importance of normalizing the data before attempting any clustering, classification or indexing [22], we can simply assume μ = 0, without loss of generality for the rest of this work. Note that this representation has been considered before in the statistical community [16][17], but its utility for data mining, in particular the ability to lower bound distance functions, is first documented here.

2.1 Lower bounding Euclidean distance

Before we define the lower bound for the clipped data, we will review some background of the main distance measure used in this paper.

Suppose we have two time series, a query Q, and a candidate match C, of length n, where

$$Q = Q_1, Q_2, \ldots, Q_i, \ldots, Q_n \qquad 2)$$

$$C = C_1, C_2, \ldots, C_j, \ldots, C_n \qquad 3)$$

If we wish to compare the two time series, we can use the ubiquitous Euclidean Distance [19][20]

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n} (Q_i - C_i)^2} \qquad 4)$$

Since the square root function is monotonic and concave, we can remove the square root step to get the squared Euclidean distance that gives identical rankings, clustering, and classifications [22].

$$D(Q,C) \equiv \sum_{i=1}^{n} (Q_i - C_i)^2 \qquad 5)$$

In addition to the utility of slightly speeding up the calculations, working with this latter distance measure also allows other optimizations [22].

Now that the distance measure has been described, if we are given a clipped time series c, and a raw time series Q, we can lower bound the squared Euclidean distance between C and Q, with the following equation (Figure 3 illustrates its visual intuition)

$$LB\_clipped(Q,c) \equiv \sum_{i=1}^{n} \begin{cases} Q_i^2 & if \ (Q_i > 0 \ \ and \ c_i = 0) \ or \\ & (Q_i \le 0 \ \ and \ c_i = 1) \\ 0 & otherwise \end{cases} \qquad 6)$$

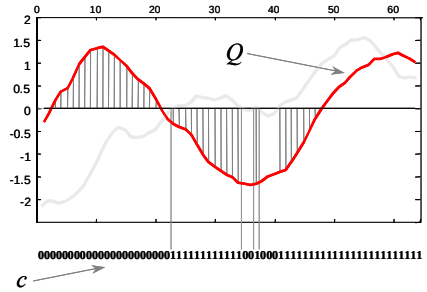

Figure 3. The lower bounding function LB_clipped(*Q*, *c*)

We will now prove the claim of lower bounding of the clipped representation.

Proposition 1: For any two time series Q and C of length n, we have

$$LB\_clipped(Q,c) \le D(Q,C)$$

Proof:

Since the distance between any two points with either measure is non-negative, it is sufficient to show that given any two points $x,y \in \Re$ we have $LB\_clipped(x,y) \le D(x,c)$, where c is the clipped value of y. The result for a summation of n points then naturally follows. Firstly we note that by definitions (4), $D(x,y) \ge 0 \ \forall x,y \in \Re$. We then consider the four possible cases.

(1) $x > 0$ and $y > 0$

(2) $x \le 0$ and $y \le 0$

(3) $x > 0$ and $y \le 0$

(4) $x \le 0$ and $y > 0$

In cases (1) and (2), by equation 6), $LB\_clipped(x,y) = 0$, hence $LB\_clipped(x,y) \le D(x,c)$.

In cases (3) and (4),

$LB\_clipped(x,c) = x^2$. $D(x,y)$ can be rewritten as x^2+y^2-2xy. Since $y^2 \ge 0$ *and* $xy \le 0$, $y^2-2xy \ge 0$. Hence

$$LB\_clipped(x,y) \le D(x,c_y)$$

Thus, for any two real valued points, the clipped distance is less than or equal to the Euclidean distance. Since both distance measures are metrics, the distance between two points is never negative, hence

$$LB\_clipped(Q,c) \le D(Q,C)$$

For any series of real numbers Q and C ■

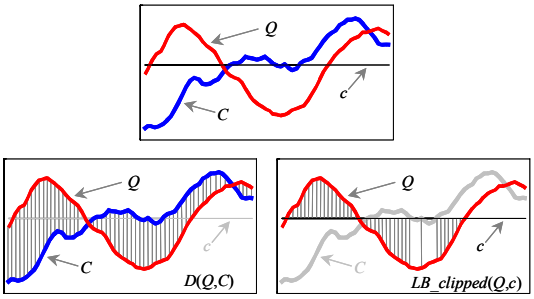

Figure 4. The distance returned by both *LB_clipped(Q,c)* and *D(Q,C)* is the sum of squared lengths of the gray hatch lines. Because every hatch line for *LB_clipped(Q,c)* is matched with corresponding line in *D(Q,C)* which is at least as long, we must have *LB_clipped(Q,c)* ≤ *D(Q,C)*

Figure 4 illustrates the visual intuition of the proof.

The extension of this proof to the Euclidean distance (or any L_p norm [39]) is trivial, and will be omitted.

2.2 Lower bounding Dynamic Time Warping

Recently, there has been increasing evidence that for some problems, the Euclidean distance may be too sensitive to minor distortions in the time axis. It has been forcefully shown that Dynamic Time Warping (DTW) can mitigate this problem [22]. Fortunately, we can also lower bound the DTW distance between Q and C, using Q and c. To do so, we simply need to slightly adapt the enveloping lower bounding function introduced in [18] that is subsequently adapted and extended by many research groups, including [33].

The amount of warping allowed in calculating DTW is determined by a single parameter r. The parameter r is known variously as the warping window width, the warping scope, the warping constraint, etc [18][32]. We can use the term r to define two new sequences, U and L:

$$U_i = \max(Q_{i-r} : Q_{i+r}) \qquad 7)$$

$$L_i = \min(Q_{i-r} : Q_{i+r}) \qquad 8)$$

Where U and L stand for *Upper* and *Lower*, respectively; we can see the reason once we plot them together with the original sequence Q, as seen in Figure 5. They simply form a bounding envelope that encloses Q from above and below.

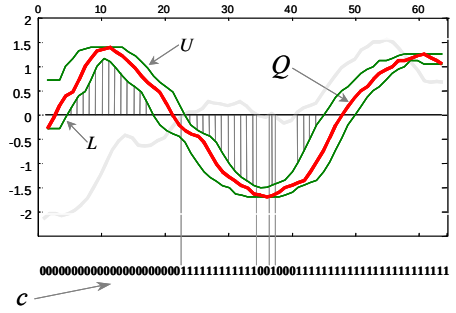

Figure 5. The intuition behind the lower bounding function *LB_Keogh_clipped(Q,c)*, which lower bounds DTW(*Q,C*)

Having defined U and L, we can now define a function $LB\_Keogh\_clipped(Q,c)$, which lower bounds DTW(Q,C):

$$LB\_Keogh\_clipped(Q,c) \equiv \sum_{i=1}^{n} \begin{cases} L_i^2 & if\ Q_i > 0\ and\ c_i = 0 \\ U_i^2 & if\ Q_i \le 0\ and\ c_i = 1 \\ 0 & otherwise \end{cases} \qquad 9)$$

The proof that $LB\_Keogh\_clipped(Q,c) \le$ DTW(Q,C) is a straightforward combination of the proof above and the proof in [18]; we omit it for brevity.

If we assume that each data point in the raw time series requires 4 bytes (a very conservative estimate), then converting to the clipped representation as presented above achieves a 32 to 1 compression ratio. However, as we shall see in the next section, we can use various techniques to achieve further compression.

2.3 Run length encoding

Consider the clipped sequence c, which we have been using as a running example. Its value is 00000000000000000000001111111111110010001111111111111111111111

Note that we could write this as 22#**0**, 11#**1**, 2#**0**, 1#**1**, 3#**0**, 24#**1**, which we can interpret as 22 zeros followed by 12 ones, etc. The shorter format allows us to fit more data in main memory.

In fact, we can be even terser; because we always toggle from zero to one or vice versa, we only need to record the parity of the first bit, giving us 22#**0**, **11**,**2**,**1**,**3**,**24**. This classic lossless compression technique is known as Run Length Encoding (RLE) [12]. To make the representation even shorter, we can represent the parity bits of 0 and 1 with two special characters, e.g. "@" and "!", respectively; our run length encoding now can be represented as @**22**,**11**,**2**,**1**,**3**,**24**. We can use this to further reduce the clipped representation of the data. Note that while the example above illustrates the idea with ASCII characters, we actually do RLE at the bit level.

The following observation motivates a further optimization. Consider the example in Figure 6 below; it shows a subsequence of the power-demand data being extracted in preparation for conversion to the clipped representation.

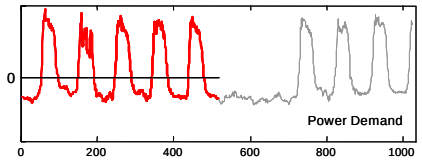

Figure 6. The first 512 data points of the Power Demand dataset just before being converted to the clipped representation

This dataset exhibits the classic structure of a dataset that is highly correlated to a working week, with the first five peaks corresponding to the 9am to 5pm shift on weekdays, and the relatively flat section between 500 and 700 corresponding to a weekend. Note that in this dataset, the run lengths are highly structured. We can see this more clearly in Figure 7, where we plotted the frequency of all sliding windows' run lengths, accumulated over a year's worth of data.

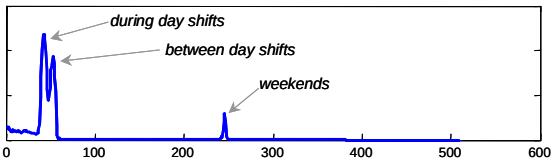

Figure 7. The relative frequency of run lengths for the Power Demand dataset, accumulated over one year

This highly skewed structure immediately suggests a further optimization. We can encode the run lengths with variable length encoding (prefix encoding) [13][35]. For example, out of 514 (with two parity bit symbols included) possible run length symbols, we can encode 43[1], the most common run length with a short symbol such as `0101`, 42, the second most common run length as `0100`, or the next most common run length is 53, which can be coded as an equally terse `0010`.

[1] The data is sampled once every 15 minutes, so a run length of 43 corresponds to 10 and ¾ hours. This is longer than a normal shift because arrival times are staggered, see [36].

Under this scheme, rarely occurring symbols such as 512 (a logical possibility, but never observed in this dataset) will have longer encodings, perhaps `000001010010011001`. However, averaged over the entire dataset, variable length encoding produces significantly smaller files. To create the encoding scheme, we use simple Huffman encoding (the more complex arithmetic encoding, yields only slightly more compact encodings in this case [34]). With on-line implementation of Huffman encoding [24], only a single pass over the data is required. Because of the prefix property of Huffman encoding, the separator ',' between each run length is no longer required, which further improves its compression ratio. In particular, the average Huffman code length obtained for this entire dataset is only 5.8 bits, giving a compression ratio of 281:1. Note that to compute the compression ratio, we use the following formula:

$$CompressionRatio = \frac{\#\,OfBitsUsedIn\_regular\_representation}{\#\,OfBitsUsedIn\_clipped\_representation}$$

We made a very conservative assumption that each number in the original representation is represented in only four bytes (32 bits).

2.4 Numerosity reduction

Even though the run length-encoding scheme itself gives an impressive compression ratio, we can improve it by numerosity reduction on sliding windows. This step is motivated by observing that while applying a sliding window on the streaming data, time series in consecutive sliding windows are very often identical in the clipped representation, except for the first and the last values that are omitted and added, respectively. If the time series in each sliding window has this property, we can exploit this fact and just record the maximum amount of time this property has consecutively been observed, along with a special character, \$, that represents this reduction. Consider the run length encoding from our example in the previous section and let the encoding of the next five sliding windows be:

@**22**,**11**,**2**,**1**,**3**,**24**
@*21*,**11**,**2**,**1**,**3**,*25*
@*20*,**11**,**2**,**1**,**3**,*26*
@*19*,**11**,**2**,**1**,**3**,*27*
@**18**,**22**,**2**,**1**,**3**,**27**,**1**
@*17*,**22**,**2**,**1**,**3**,**27**,*2*

We can readily see that the first four windows are very similar and can be reduced to one since the only values differ from each other are the first and the last (italicized for clarity). However, the fifth window cannot be combined with the previous one since the last bit has changed from 1 to 0, but it can be combined with its next

window. As a result, the final encoding with numerosity reduction becomes

@22,11,2,1,3,24$3@18,22,2,1,3,27,1$1.

As before, although we demonstrate the idea with ASCII text, we actually encode everything at the bit level.

With the Power Demand dataset of size 10,000 data points, numerosity reduction together with Huffman coding yields a huge compression ratio of 1057:1. Note that while the factor of 32 to 1 achieved by clipping is lossy, the remaining factor of approximately 33 to 1 is lossless with respect to the clipped data.

3 Clipped representation with clustering

Clustering time series is a problem that has applications in a wide variety of fields, and has recently attracted a large amount of research. There are three types of objectives when clustering time series. Clusters may reflect similarity in time (i.e. group series that are correlated), similarity in shape (i.e. group series that have similar patterns of change irrespective of time) or similarity in change (i.e. group series that have similar autocorrelation structure). The different objectives are reflected in different distance functions. Thus, for example, Euclidean distance is a metric of similarity in time and is commonly used with the transformations described in Figure 1. Similarity in shape is measured by specific algorithms or by a transformation such as dynamic time warping (DTW). The usual approach to measuring similarity in change is to assume some underlying model form such as hidden Markov models or an ARMA process [38] then cluster based on similarity of fitted models. Given a means of measuring similarity between series, time series may be clustered with any of the numerous techniques available. Clipping the series offers a means of clustering to meet any of the three objectives faster and with less memory and without necessarily decreasing the quality of the clusters obtained.

3.1 Faster clustering with clipped data

Clipped series have been shown to be theoretically and experimentally sufficient to cluster based on similarity of change if the series are long enough, and to produce similar clusterings on real world problems [4][6]. In Section 4.3 we demonstrate that clipped series can form clusters similar to those formed with unclipped data when the objective is to group series based on Euclidean distance.

In addition to the space benefits discussed in Section 2, clustering with clipped data can also allow time improvements for commonly used algorithms. This is because the bit level representation allows for the utilization of bitwise operators. For example, given two clipped series *c* and *d,* the distance calculation given in Equation 5) can be efficiently calculated by finding the *XOR* of *c* and *d* then summing the number of 1s in the result, i.e.

$$D(c,d) \equiv \sum_{i=1}^{n} (c_i \oplus d_i) \qquad 10)$$

If binary series are packed into integers we can find the terms in the summation very quickly using bit operators. We can also speed up the operation to sum the bits. Any algorithm to count the bits is $\Omega(n)$. However, we can improve the constant terms in the time complexity function by using shift operators to evaluate the integer value of each eight-bit sequence then using a lookup table to find the number of bits in that integer. This mechanism makes the distance calculation approximately five times faster even when the series are loaded into main memory.

3.2 Kolmogorov complexity clustering

In a series of recent papers, Ming Li, Paul Vitanyi, and collaborators [26], have championed an interesting new approach to clustering.

The proposed method is inspired by the concept of Kolmogorov complexity, a measure of randomness of strings based on their information content. The Kolmogorov complexity K(*x*) of a string *x* is defined as the length of the shortest program capable of producing *x* on a universal computer — such as a Turing machine. Different programming languages will give rise to distinct values of K(*x*), but one can prove that the differences are only up to a fixed additive constant. Intuitively, K(*x*) is the minimal quantity of information required to generate *x* by an algorithm.

The conditional Kolmogorov complexity K(*x*|*y*) of *x* to *y* is defined as the length of the shortest program that computes *x* when *y* is given as an auxiliary input to the program. The function K(*xy*) is the length of the shortest program that outputs *y* concatenated to *x*.

In using this motivation, the authors consider the distance between two strings *x* and *y,* defined as

$$d_k(x,y) = \frac{K(x \mid y) + K(y \mid x)}{K(xy)} \qquad 11)$$

Kolmogorov complexity is without a doubt the ultimate lower bound among all measures of information content. Unfortunately, it cannot be computed in the general case. As a consequence, one must approximate this distance. Li and Vitanyi suggested approximating this with standard off-the-shelf compression algorithms, such as WinZip or Stuffit. Their distance measure is thus

$$d_c(x,y) = \frac{C(x \mid y) + C(y \mid x)}{C(xy)} \qquad 12)$$

Where $C(x)$ is the size of file *x* after compression, and $C(x|y)$ is the size of file *x* after compressing it with the compression model built for *y*.

In an impressive array of papers, Li and Vitanyi have shown that this distance measure works exceptionally well for clustering DNA strings, MIDI files, natural language text, computer programs, etc.

The above results appear to have little implication for time series, because the approach requires a lossless compression technique where a model (typically a substitution dictionary) is learned from the data. While there are a host of compression algorithms for real-valued time series, virtually all of them are lossy (DFT, DWT, SVD, etc). The handful of lossless techniques (delta encoding, for example) do not produce a compression model.

Our clipped representation offers a unique opportunity to avail of this work, we *do* build a compression model of the data, in particular the run length encoding discussed in Section 2.3. We can therefore define $C(x|y)$ as the size of time series x when compressed with the run length dictionary learned for time series y. As we will empirically show in Section 4.4 This simple parameter free measure can outperform more complex distance measures such as Markov models and ARIMA, on a diverse array of problems.

4 Empirical Evaluation

In this section, we will provide an extensive empirical comparison among the raw and various representations of compressed data in two major data mining tasks, time series indexing and clustering. Twelve datasets were used in our indexing experiments, and two were used for clustering experiments. We also tested on a wide range of both real and synthetic datasets. The datasets range from 66 Kilobytes to 2 Gigabytes in size. We also note that all data used in these experiments are freely available at [21].

4.1 Indexing Experimental methodology

For indexing, we will demonstrate the superiority of our clipped representation in terms of number of disk accesses. We compare our proposed method with the classic Piecewise Aggregate Approximation (PAA) and Discrete Fourier Transform (DFT), all preserving similar compression ratio. However, when managing each representation, we try to best optimize every competing method possible. For example, in PAA representation, if m (the number of reduced section according to the compression ratio) turns out to be a value less than two or does not evenly divide the length of the sliding window, we promote n to two or to the next smallest integer that evenly divides the size of sliding window even if this may largely decrease the compression ratio for that representation. Our justification is that because all the data we use in the experiment is z-normalized to have a zero mean and standard deviation of one, the number of sections (m) fewer than 2 is not a very meaningful representation in that the value representing that entire window will always be zero. We give all the advantage to our competing methods; the smaller is the compression ratio (larger n), the more accurate PAA and DFT will be represented. DFT is treated similarly except that m does not need to evenly divides the window size since it represents the first m DFT coefficients that will be preserved; m could then be any integer larger than 1.

We then demonstrate that clipped series can produce clusters similar to those obtained with the raw data when clustering a very large real world database introduced in section 4.2. We show that clipping performs favorably when compared to clustering with unclipped data since clustering can be done faster and with much less memory requirement.

For similarity search, we performed all experiments over a range of query lengths. Since we want to include PAA in our experiments, the query length is somewhat limited. We therefore consider query lengths of 256 and 512 data points.

We tested our approach on a variety of twelve datasets with various properties within the data, obtained from the UCR Time Series Data Mining Archive [http://www.cs.ucr.edu/~eamonn/TSDMA]. The sizes of the datasets range from 6,875 data points to 198,400 data points. In order to achieve realistic result, we only consider queries that do not have exact matches in the database, but have similar structure. Since our proposed method focuses on streaming data, holdout cross validation method may not be entirely appropriate. Instead, leaving-one-out cross validation is used; on each run, we randomly pick a query from a database and treat that query and some portions before and after as withheld subsections, create a run-length encoding with numerosity reduction for the rest of the data, and determine the resultant compression ratio. We then create PAA and DFT on the same data and with the same compression ratio (or with smaller compression ratio, in favor of PAA and DFT as explained in Section 4 above) then measure the number of random disk accesses for the nearest neighbor queries of all methods. To determine the number of dimensionality reduction (m) in PAA and DFT in these cases, we assume that each value in PAA and DFT can be represented by only *two* bytes (instead of four or eight bytes) to strongly demonstrate that we still have competitive results among all the approaches.

In addition, to avoid any possibility of implementation bias, the number of I/O disk accesses of each method is measured instead of recording the actual running time. This is done by first computing the lower bound distances using LB_clipped (Eq. 6), Euclidean distance, LB_Keogh_clipped (Eq. 9), or LB_ZhuShasha (for PAA and DFT with DTW) [39], as appropriated, between a query and all the sequences in the dataset. Then to retrieve the nearest neighbor, each sequence is visited in

the order according to the lower bound values. We count the number of times the real disk accesses must be made. These numbers also indicates the tightness of the lower bounds for each representation. The results are averaged over 100 separate runs for each dataset. For simplicity, we only report results for one-nearest neighbor queries.

4.2 Indexing results

For completeness, we tested on a variety of time series. Figure 8 shows a small snippet from each.

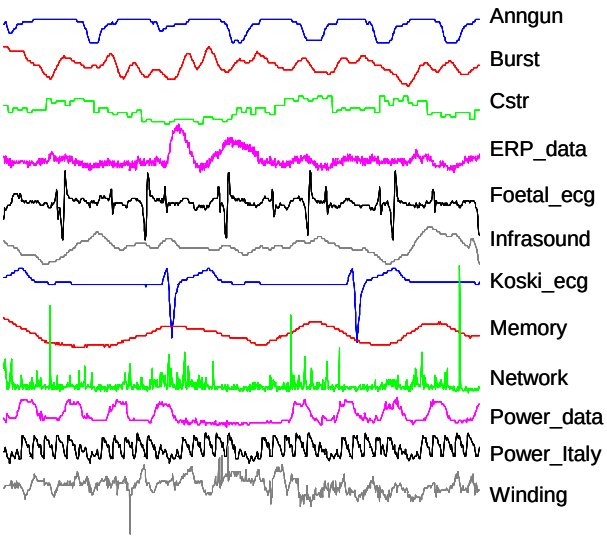

Figure 8. Some snippets (1,000 data points) from each of the twelve datasets used in the experiments

As noted above, the amount of compression is dictated by the data itself. For the twelve datasets considered the compression ratios range between 60.2:1 to 1,089.5:1 (see Appendix A for complete results). As one might expect, higher compression ratios are achieved on smoother datasets or ones with apparent repetitive patterns, such as *Koski_ecg* or *Power_data*.

We compare different representations in terms of I/O random disk accesses during the process of the one-nearest neighbor retrieval of a query time series. In particular, in each run, we reduce the dimensionality of the data from n to m using Clipped, PAA[2], and DFT representations, and build their indices on the reduced spaces based on their lower bounds between each subsection (sliding window) of the time series and the query. To allow a visual comparison, we normalize each experiment on each particular dataset by the worst performing algorithm; the raw numbers are available in Appendix A.

Figure 9 and Figure 10 show the number of disk accesses with lower bounding the Euclidean distance, using the three dimensionality-reduction techniques over the range of query lengths of 256 and 512 data points, respectively.

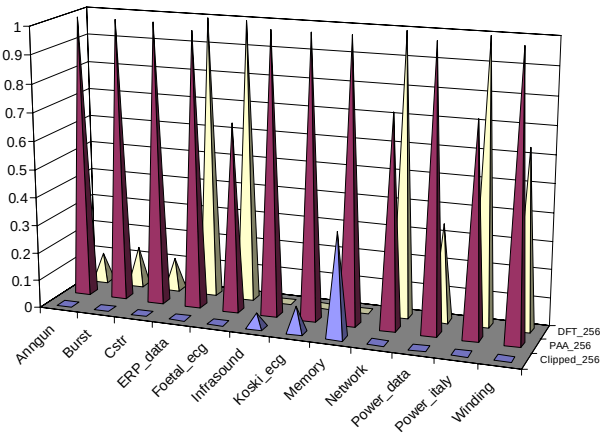

Figure 9. Number of disk accesses with lower bounding of Euclidean distance, normalized by the worst performing approach, using the three representations for the query length of 256 data points

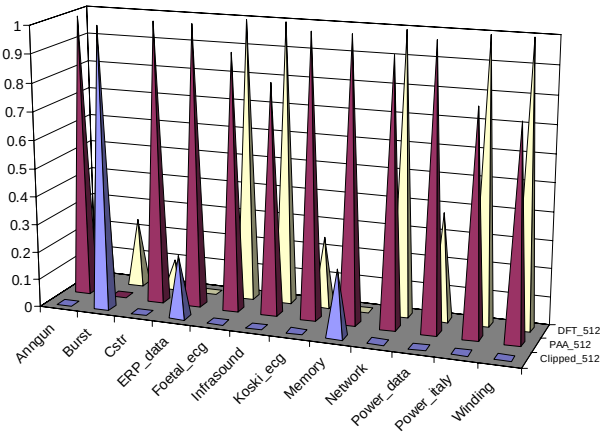

Figure 10. Number of disk accesses with lower bounding of Euclidean distance, normalized by the worst performing approach, using the three representations for the query length of 512 data points

In general, the results show that the clipped representation greatly outperforms or at least is comparable to the other approaches, expressing the superiority in its tightness of the lower bounds. For the *Burst* dataset, the clipped representation works well for shorter queries, but worst for longer sequences. Again, we would like to emphasize that our results here are obtained by conservatively assuming only *two*-byte requirement to represent each number in PAA and DFT, as well as having m promoted as described in Section 4. If we assume 4 or 8 bytes or without m adjusted, the results will be much improved.

Figure 11 and Figure 12 further show the stronger results of the clipped representation in lower bounding the Dynamic Time Warping distance. The experiments were similar to those of Figures 9 and 10, except that the Dynamic Time Warping was used instead of the

[2] The PAA representation has exactly the same indexing power as the Haar wavelet when the length of time series is of power of two [19][39].

Euclidean Distance. The lower bounds for clipped data were calculated using Equation 9); those for PAA and DFT were calculated according to Zhu and Shasha [40].

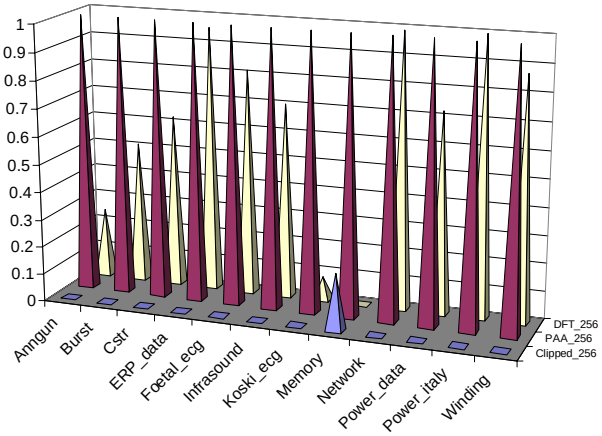

Figure 11. Number of disk accesses with lower bounding of DTW, normalized by the worst performing approach, using the three representations for the query length of 256 data points

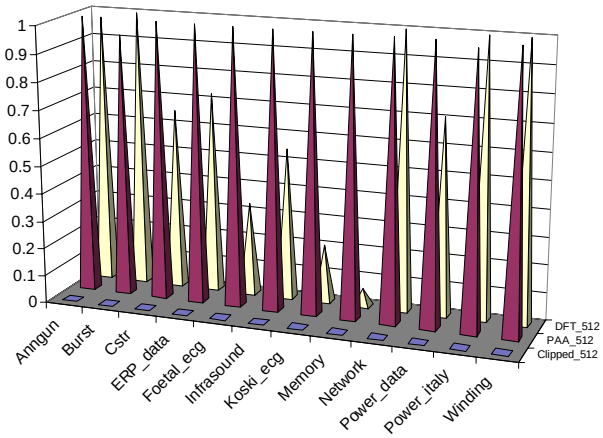

Figure 12. Number of disk accesses with lower bounding of DTW, normalized by the worst performing approach, using the three representations for the query length of 512 data points

These results impressively showed that the clipped representation in lower bounding performed exceptionally well, especially for Dynamic Time Warping. Note that even though the four figures above convey the similar information that clipped data almost always outperforms PAA and DFT, we cannot directly compare between lower boundings of Euclidean and DTW distance since they were normalized by the worst performing approach in each plot. The details of these results appear in Appendix A, in which we could see that the clipped data can outperform the other two approaches by up to 127,000 disk accesses for DTW.

4.3 General compression-based Clustering

We examine a class of problems where a DFT approach should produce good results, and show that clipping does better than the most commonly used DFT approach described in [2].

The class of model we consider is generated from a mixture of Sine waves. A generating model is of the form

$$M(t) = \sum_{j=1}^{r} a_j \cdot \sin(b_j + c_j * t) \qquad 13)$$

The parameters a, b and c control the amplitude, offset and frequency of the curves respectively. Each can take a value on the range [0,1], but they are scaled so that any particular sine wave has a maximum amplitude of 2 and a maximum offset of $n/2$ (where n is the series length). A frequency parameter of 0 means the sine wave completes a single oscillation over the data length, and a frequency parameter of 1 means the curve will complete $n/4$ cycles over the n data points. For a clustering experiment we generate k different models, M_1, M_2, ... M_k, then each time series from cluster c is found by adding Gaussian noise to M_c.

The objective is to cluster series based on similarity in time, hence our benchmark for performance is Euclidean distance on the raw data. Our aim is to evaluate which representation provides clusters as accurate as the baseline results on the class of model described. Following the algorithm described in [2], we cluster with DFT by performing an O($n$log$n$) FFT, retain the first fc coefficients (set to $n/64$ to achieve a compression ratio of 32:1), then use Euclidean distance between the difference of the coefficients as the clustering distance metric. We cluster data from eight clusters, with five series in each cluster, generated from functions of the form given in Equation 13) above with parameters randomly selected in the range [0,1], each model being a combination of three sine waves (i.e. r = 3). We cluster with k-means restarted 100 times at random initial centroids. Each series is 1,024 data points long. An example of the data from 6 series is shown in Figure 13.

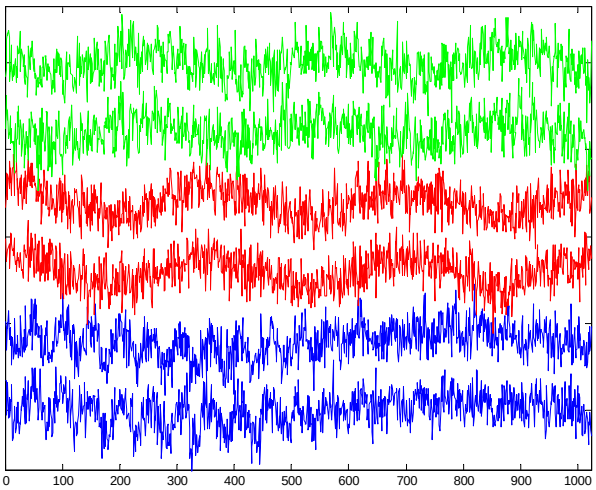

Figure 13. Example of sine wave data series from three separate cluster models.

The results for 20 randomly generated sets of models are shown in Table 1. Accuracies are measured by enumerating all possible cluster labeling and measuring the accuracy against the known correct clustering. The clipped clusters are often identical to those found with the raw data, and the difference in mean accuracy is very small. On 12 out of the 20 runs, the clipped series found completely correct clusters, whereas the DFT approach, in addition to having a significantly lower average accuracy, also only found the correct clustering on a single occasion.

Table 1. The mean accuracies for 20 randomly generated sets of models

	Mean Accuracy	Number of Perfect Clusterings
Raw	96.87%	15
Clipped	94.22%	12
DFT	81.56%	1

We do not claim that clipped clustering will always perform better than a DFT approach for this class of model. One of the reasons the DFT approach does worse in these experiments is that the larger DFT coefficients may be discarded. If we restrict the parameters (a,b,c) to the range [0,0.2], the DFT approach performs at least as well as the other methods. More recent DFT approaches such as those described in [29][37] may perform better on the data used in these experiments. It is also possible to find parameter settings where the approach of keeping the first *fc* coefficients does marginally better than clipping (when the sine waves are all low in frequency and amplitude). However, clipping is a very simple procedure with no need for parameterization. It clusters well in relation to using the raw data for all parameter settings attempted, and has been shown to do well on data where a DFT approach is not appropriate (see [3][4][6]).

To demonstrate how clipping can help with a real world problem, especially in large datasets, we clustered optical recording data from a bee's olfactory system. Analysis of this data can help understand the mechanisms of the olfactory code and the temporal evolution of activity patterns in the antennal lobe. Further information can be found in [10]. The data consists of 980 images, each image containing of 688x520 measurements. If we consider each position in the image as a time series, the data consists of 357,760 time series of length 980. Preliminary analysis has shown that clustering the series based on similarity in time produces results that have a sensible physiological interpretation [10]. We cluster with *k*-means (with *k* set to 16) restarted 50 times from random initial centroids, and take as the best clustering the one with the lowest within-cluster variation. We are assuming the objective is to group together series with similarity in time; hence we use a Euclidean distance metric to measure similarity. Figure 14 shows the clusters produced by the raw data. The dataset size is around 2 gigabytes and clustering the raw data may be prohibitively demanding of time and space resources. A single run of *k*-means takes 50-150 iterations to converge (taking hours to complete), and most machines do not have 2 gigabytes of main memory. Hence, some form of data reduction is appropriate for this problem. We use this data set to demonstrate that compressing the data through clipping can produce clusters more similar to those produced with the whole data (Figure 14) than those found using DFT and PAA.

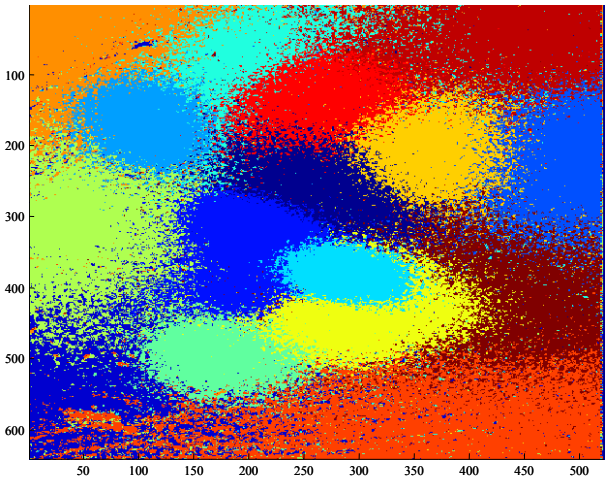

Figure 14. Sixteen clusters produced using the whole 2-gigabyte raw data

For comparison purposes, we assume that clipping provides a compression ratio of 32:1 (in practice, it may be much higher than this) and we set the parameters of DFT and PAA to achieve a similar or smaller ratio to guarantee in giving all the advantages in compression ratio to the competing methods. For PAA, we compress each series into 49 mean values since we mentioned earlier that the number of PAA coefficients must evenly divide the length of each time series, in this case giving a compression ratio of 20:1. We again use Euclidean distance.

Figure 15 to Figure 17 show the spatial clustering co-occurrences found with clipped, DFT, and PAA data. It is clear from these plots that the clipped series are much more similar to those found with the raw data than those found with DFT and PAA. There are structural similarities between all four clusterings, but the clipped clusterings are much cleaner than those found with PAA and DFT. The only major difference between the unclipped and the clipped clusters is an additional central cluster formed around position (325,450) with the unclipped data.

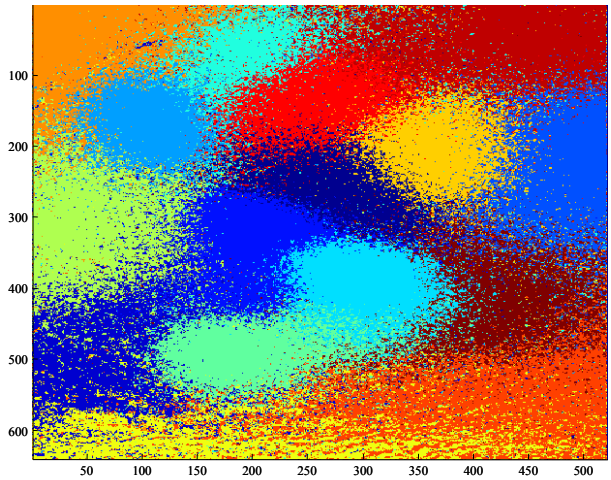

Figure 15. Clusters formed using the clipped data with 32:1 compression ratio. The spatial cluster co-occurrences between this plot and the clusters formed using raw data in Figure 14 shows its effectiveness in the clipped data reduction technique

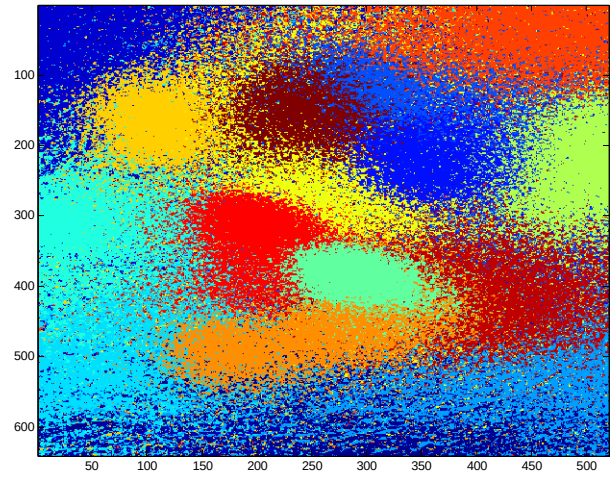

Figure 16. Cluster formed using a piecewise constant approximation (PAA) with 49 coefficients, giving 20:1 compression ratio

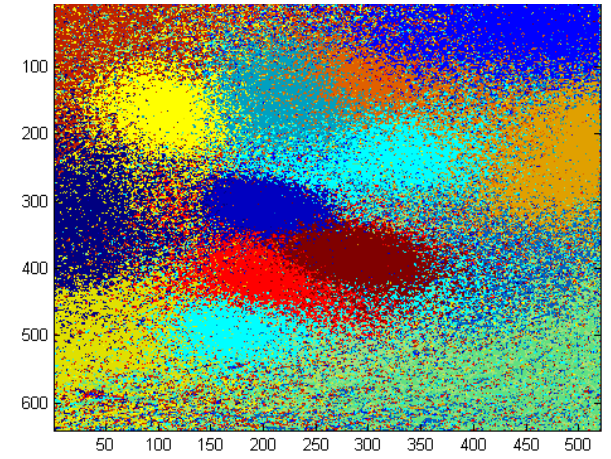

Figure 17. Clusters formed using first 17 Discrete Fourier Transform coefficients, giving 29.7:1 compression ratio

To measure the similarity between the clustering with the raw data and the clusterings with the compressed data, we use three well-known measures based on the count of coincidence of common cluster membership: the Jaccard coefficient [28]; the Rand statistic [31]; and the Folkes and Mallows index [9] (higher values indicate a greater degree of similarity). These statistics are given in Table 2. By all three measures, the clipped clusterings are more similar to the raw clusterings than those found with DFT and PAA.

Table 2. similarity between the best clusters formed by the raw and compressed data

	Jaccard	Rand	Folkes & Mallows
Clipped	0.4843	0.9512	0.6526
DFT	0.4424	0.9476	0.6136
PAA	0.2647	0.9222	0.4188

To demonstrate that for this data clipping gives clusterings significantly closer to those with the whole data, we randomly selected 10 clusterings from the run of 50 for each experiment with clipped, DFT, and PAA compression. For each of these 10 clusterings, we measure the similarity to 10 randomly selected clusterings formed with the complete data, giving us 100 sets of comparisons for each technique. The mean Jaccard, Rand, and Folkes and Mallows statistics, along with the standard deviations, are shown in Table 3.

Table 3. Average similarity and standard deviation between the clusters formed with the raw and compressed data, comparing among 10 clusterings for each technique

	Jaccard	Rand	Folks & Mallows
Clipped	0.4456 ± 0.042	0.9469 ± 0.006	0.6156 ± 0.04
DFT	0.2108 ± 0.017	0.9118 ± 0.012	0.3425 ± 0.093
PAA	0.2761 ± 0.017	0.9227 ± 0.003	0.4327 ± 0.021

For each of the three measures, we can, at the 1% level, reject the null hypothesis that clipped and DFT, and clipped and PAA average similarity are the same in favor of the alternative that clipped similarity is higher (using a one sided t-test for the mean and a Mann-Whittley test for the median). This clearly demonstrates that, when using restarted k-means with an objective to cluster based on similarity in time, clipping is a more appropriate compression than DFT or PAA for this data set.

4.2 Kolmogorov Inspired Clustering

For this experiment we examined the UCR Time Series Archive for datasets that come in pairs. For example, in the **Bouy sensor** dataset, there are two time series, *East* and *North*, and in the **Great Lakes** dataset, there are two time series, *Ontario* and *Erie*. We were able to identify twenty-four such pairs, from a diverse collection of time series covering the domains of finance, science, medicine, industry, etc. Although our method is able to deal with time series of different lengths, we truncated all time series to length 1,000 to enhance the graphical demonstration.

While the correct hierarchical clustering at the top of the tree is somewhat subjective, at the lower level of the tree, we would hope to find a single bifurcation separating each pair in the dataset. Our metric, Q, for the quality of clustering is therefore the number of such correct bifurcations divided by twenty-four, the number of datasets. For a perfect clustering, Q = 1, and because the number of dendrograms of forty-eight objects is greater than 2.2*10106, for a random clustering, we would expect Q = 0. Figure 18 shows the resulting dendrogram for our approach.

The clustering obtained is of very high quality, with Q = 0.875. The minor mistakes made, particularly with the grouping of the **Furnace** and **Evaporator** sequences are reasonable and plausible. Markov models [11] scored Q = 0.458, and ARIMA/ARMA models [25][38] scored Q = 0.625. In the former case, it took considerable parameter tuning to achieve this score, whereas our approach has no parameters.

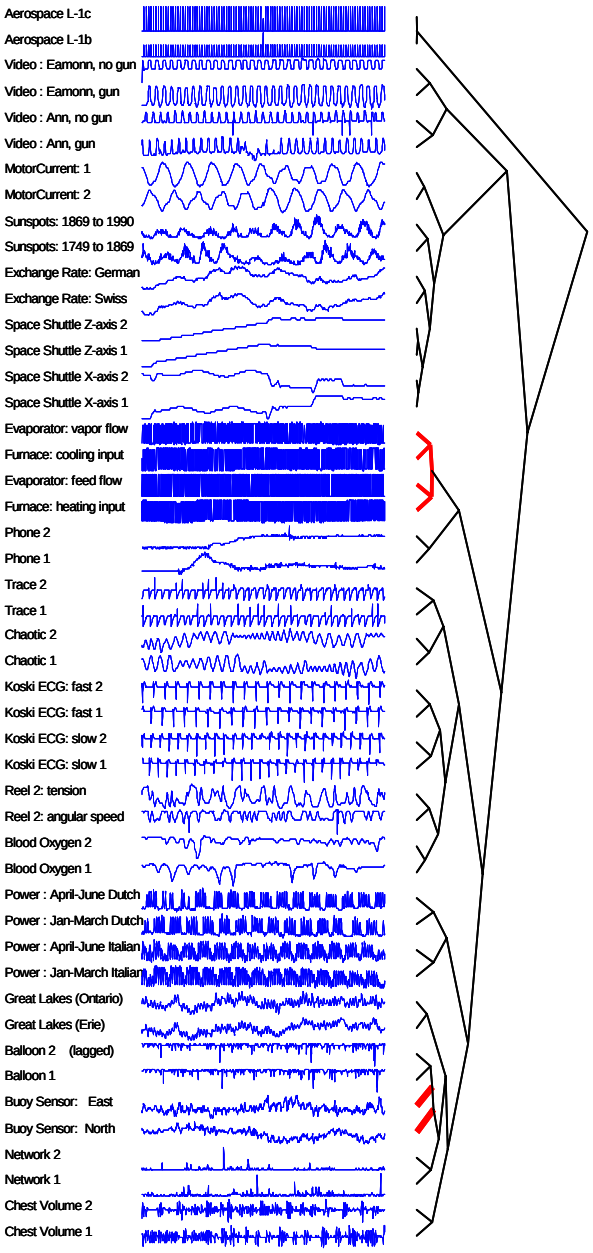

Figure 18. Forty-eight time series (in twenty-four pairs) clustered using the approach proposed in this paper. Bold lines denote incorrect subtrees

5 Conclusions

In this paper, we have shown that a simple dimensionality reduction technique, i.e. the clipped representation, can outperform the more sophisticated technique by a few orders of magnitude. We have shown that our proposed clipped representation can improve the compression ratio by a wide margin, while being able to maintain or increase the tightness of its lower bound, which allows even faster nearest neighbor queries, especially in ones that require Dynamic Time Warping distance measure. Other than producing faster exact algorithms for similarity search, we have also demonstrated that our clipped representation approach can support clustering and scale to much larger datasets.

Acknowledgments: This research was partly funded by the National Science Foundation under grant IIS-0237918.

6 References

[1] J. Aach & G. Church (2001). "Aligning gene expression time series with time warping algorithms. " *Bioinformatics (*17), 495-508.

[2] R. Agrawal, C. Faloutsos, & A.N. Swami (1993). Efficient Similarity Search in Sequence Databases. Proc. of 4th International Conference of Foundations of Data Organization and Algorithms (FODO), pp. 69-84.

[3] Bagnall, A.J. and Janacek, G.J., "Clustering Time Series with Clipped Data", accepted for publication, Machine Learning, 2004/2005

[4] A. J. Bagnall, G. Janacek and B. de la Iglesia and M. Zhang (2003), Clustering Time Series from Mixture Polynomial Models with Discretised Data, *Proc. of 2nd Australasian Data Mining Workshop.*

[5] A.J. Bagnall & G.J. Janacek. Clustering Time Series with Clipped Data. *Information and Software Technology*. To appear 2004/2005.
[6] A. J. Bagnall and G. Janacek (2004), Clustering time series from ARMA models with clipped data, Proc. of ACM SIGKDD.
[7] D. Berndt & J. Clifford (1994). Using dynamic time warping to find patterns in time series. *AAAI-94 Workshop on Knowledge Discovery in Databases*. pp. 229-248.
[8] B. Chiu, E. Keogh, & S. Lonardi (2003). Probabilistic Discovery of Time Series Motifs. In the 9th ACM SIGKDD.
[9] E. Fowlkes & C. Mallows (1983). A method for comparing two hierarchical clusterings. Journal of American Statistical Association, 78:553-569.
[10] R.F. Galan, S. Sachse, C.G. Galizia, A.V.M. Herz (2004) "Odor-driven attractor dynamics in the antennal lobe allow for simple and rapid olfactory pattern classification." *Neural Computation.*
[11] X. Ge & P. Smyth. (2000). Deformable Markov model templates for time-series pattern matching. In proceedings of the 6th ACM SIGKDD. Boston, MA. pp. 81-90.
[12] S.W. Golomb. "Run-length Encodings", *IT(12)*, No. 7, July 1966, pp. 399—401.
[13] D. A. Huffman (1952). A Method for the Construction of Minimum-Redundancy Codes.*Inst. Radio Eng.* 40, 1098-1101.
[14] M. W. Kadous (1999). Learning comprehensible descriptions of multivariate time series. *In Proc. of the 16th International Machine Learning Conference*. pp. 454-463.
[15] K. Kalpakis, D. Gada, & V. Puttagunta. (2001). Distance measures for effective clustering of ARIMA time-series. In proc. Of the IEEE ICDM, pp. 273-280.
[16] B. Kedem, Estimation of the Parameters in Stationary Autoregressive Processes After Hard Limiting, *Journal of the American Statistical Association*, Vol 75, 1980, pp. 146-153.
[17] B. Kedem and E. Slud, On Goodness of Fit of Time Series Models: An Application of Higher Order Crossings, *Biometrika*, Vol 68, pp. 551-556
[18] E. Keogh (2002). Exact indexing of dynamic time warping. In 28th VLDB Intl Conf., pp. 406-417.
[19] E. Keogh, K. Chakrabarti, M. Pazzani & S. Mehrotra (2000). Dimensionality reduction for fast similarity search in large time series databases. KAIS Journal. pp. 263-286.
[20] E. Keogh, K. Chakrabarti, M. Pazzani & S. Mehrotra (2001). Locally adaptive dimensionality reduction for indexing large time series databases. In *Proc. of ACM SIGMOD*, pp. 151-162.
[21] E. Keogh & T. Folias (2002). The UCR time Series Data Mining archive. Riverside, CA. [http://www.cs.ucr.edu/~eamonn/TSDMA].
[22] E. Keogh & Kasetty, S. (2002). On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In the 8th ACM SIGKDD pp. 102-111.
[23] E. Keogh, S. Lonardi, & CA. Ratanamahatana. (2004). Towards Parameter-Free Data Mining. In proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
[24] D.E.Knuth (1985). Dynamic Huffman coding. J. of Algorithms. Vol. 6(2), pp. 163-180.
[25] J. B. Kruskall, & M. Liberman (1983). The symmetric time warping algorithm: From continuous to discrete. Time Warps, String Edits and Macromolecules. Addison-Wesley.
[26] M. Li, X. Chen, X. Li, B. Ma, & P. Vitanyi (2003). The similarity metric. Proceedings of the 14th annual ACM-SIAM symposium on Discrete algorithms. Pp. 863-872.
[27] R. Manmatha, & T.M. Rath (2003). Indexing Handwritten Historical Documents – Recent Progress. In Proc. Of the Symp. on Document Image Understanding.
[28] G.W. Milligan, L.M.Sokol, & S.C.Soon (1983). The effect of cluster size, dimensionality and the number of clusters on recovery of true cluster structure. IEE Trans PAMI, 5(1):40-47.
[29] F. Morchen (2003). Time series feature extraction for data mining using DWT and DFT. Technical Report, Dept. of Mathematics and Computer Science, Philipps-University Marburg.
[30] M. Munich, & P. Perona (1999). Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proc. of 7th Int'l Conf. on Computer Vision*, Korfu, Greece. pp. 108-115.
[31] W.M.Rand (1971). Objective criterion for evaluation of clustering methods. Journal of American Statistical Association, 66: 846-851.
[32] C.A. Ratanamahatana & E. Keogh (2004). Making Time-series Classification More Accurate Using Learned Constraints. In proc. of SDM Int'l Conf., pp. 11-22
[33] T. Rath & R. Manmatha (2002). Word image matching using dynamic time warping, *Tec Report MM-38*. Center for Intelligent Inf. Retrieval, Univ. of Massachusetts Amherst.
[34] J. Rissanen and G.G.Langdon, Jr.(1979) "Arithmetic coding," IBM J. of Res. and Dev. 23(2), pp. 149—162.
[35] E. S. Schwarz "An Optimum Encoding with Minimum Longest Code and Total Number of Digits." *Inf. and Control* 7, 37-44, 1964.
[36] J. J. van Wijk & E. R. van Selow (1999). Cluster and calendarbased visualization of time series data. In Proc. IEEE Symp on Inf. Visualization, pp. 4-9.
[37] M. Vlachose, C. Meet, & Z. Vagena (2004). Identifying similarities, periodicities and bursts for online search queries. Proc. of the ACM SIGMOD International Conference on Management of Data.
[38] Y. Xiong and D.-Y. Yeung, "Mixtures of ARMA models for model-based time series clustering", *ICDM'02.*
[39] B K. Yi & C. Faloutsos.(2000). Fast time sequence indexing for arbitrary L*p* norms. VLDB. pp. 385-394.
[40] Y. Zhu & D. Shasha (2003). Warping Indexes with Envelope Transforms for Query by Humming. SIGMOD, pp.181-192.

Appendix A. Experiment results (Section 4.2)

Dataset	Size	Query Size: **EUCLIDEAN DISTANCE**							
		256				512			
		Ratio:1	Clipped	PAA	DFT	Ratio:1	Clipped	PAA	DFT
Anngun	10,001	691.2	**982**	5478	1433	677.6	**1053**	5614	4656
Burst	9,382	381.7	**2298**	4785	2644	498.9	5481	4086	**4424**
Cstr	22,500	464	**1885**	6704	2433	608.8	**2445**	7533	2978
ERP_data	198,400	239.8	**4897**	17438	17701	321	8763	33210	**1860**
Foetal_ecg	20,000	121.5	**3034**	7802	10141	157.3	**3507**	15641	16843
Infrasound	8,192	390.8	1826	3778	**1722**	379.1	**2474**	3119	3263
Koski_ecg	144,002	419.2	3024	10098	**2314**	628.5	**2931**	28829	9386
Memory	6,875	396.9	973	1882	**439**	715	1513	2739	**1134**
Network	18,000	60.2	**13421**	16174	17084	62.4	**12305**	17074	17384
Power_data	35,040	437.3	**1636**	9071	4207	1089.5	**708**	9949	4214
Power_Italy	29,931	209.9	**651**	2919	3666	286	**771**	13430	16765
Winding	17500	152.3	**2090**	5433	4214	176.1	**2684**	5257	6101

Dataset	Size	Query Size: **DTW DISTANCE**							
		256				512			
		Ratio:1	Clipped	PAA	DFT	Ratio:1	Clipped	PAA	DFT
Anngun	10,001	714.2	**995**	9411	3074	694.6	**2**	9382	9164
Burst	9,382	390.3	**1832**	6607	4269	511.79	**795**	6248	6572
Cstr	22,500	474.4	**614**	12291	7904	612.7	**28**	16160	10573
ERP_data	198,400	204.2	**6**	127520	122025	321.5	**1**	127423	92590
Foetal_ecg	20,000	121.6	**2**	14394	11757	159.3	**2**	19082	6352
Infrasound	8,192	401.1	**1314**	6262	4796	387.7	**72**	1704	968
Koski_ecg	144,002	419.8	**370**	40955	3759	629.5	**711**	128576	27215
Memory	6,875	396.9	1271	2528	**953**	715.8	**2293**	4822	2454
Network	18,000	60.3	**434**	17583	17441	63	**91**	17378	17388
Power_data	35,040	440.8	**42**	21260	15401	1103.9	**6**	27037	19174
Power_Italy	29,931	210	**2**	29534	29575	286.2	**7**	28412	28946
Winding	17500	154.3	**1**	16464	14367	178.2	**1**	16869	16864