Finding Motifs in Time Series

Jessica Lin Eamonn Keogh Stefano Lonardi Pranav Patel

University of California - Riverside
Computer Science & Engineering Department
Riverside, CA 92521, USA

{jessica, eamonn, stelo, prpatel}@cs.ucr.edu

ABSTRACT

The problem of efficiently locating previously known patterns in a time series database (i.e., query by content) has received much attention and may now largely be regarded as a solved problem. However, from a knowledge discovery viewpoint, a more interesting problem is the enumeration of previously unknown, frequently occurring patterns. We call such patterns "motifs," because of their close analogy to their discrete counterparts in computation biology. An efficient motif discovery algorithm for time series would be useful as a tool for summarizing and visualizing massive time series databases. In addition, it could be used as a subroutine in various other data mining tasks, including the discovery of association rules, clustering and classification. In this work we carefully motivate, then introduce, a non-trivial definition of time series motifs. We propose an efficient algorithm to discover them, and we demonstrate the utility and efficiency of our approach on several real world datasets.

1. INTRODUCTION

The problem of efficiently locating previously defined patterns in a time series database (i.e., query by content) has received much attention and may now be essentially regarded as a solved problem [1, 8, 13, 21, 22, 23, 35, 40]. However, from a knowledge discovery viewpoint, a more interesting problem is the detection of previously unknown, frequently occurring patterns. We call such patterns *motifs*, because of their close analogy to their discrete counterparts in computation biology [11, 16, 30, 34, 36]. Figure 1 illustrates an example of a motif discovered in an astronomical database. An efficient motif discovery algorithm for time series would be useful as a tool for summarizing and visualizing massive time series databases. In addition, it could be used as subroutine in various other data mining tasks, for instance:

- The discovery of association rules in time series first requires the discovery of motifs (referred to as "*primitive shapes*" in [9] and "*frequent patterns*" in [18]). However, the current solution to finding the motifs is either high quality and very expensive, or low quality but cheap [9].
- Several researchers have advocated K-means clustering of time series databases [14], without adequately addressing the question of how to seed the initial points, or how to choose K. Motifs could potentially be used to address both problems. In addition, seeding the algorithm with motifs rather than random points could speed up convergence [12].
- Several time series classification algorithms work by constructing typical prototypes of each class [24]. While this approach works for small datasets, the construction of the prototypes (which we see as motifs) requires quadratic time, as is thus unable to scale to massive datasets.

In this work we carefully motivate, then introduce a non-trivial definition of time series motifs. We further introduce an efficient algorithm to discover them.

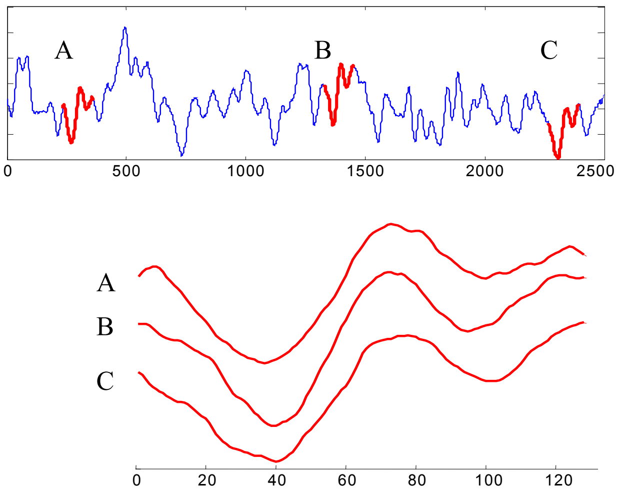

Figure 1: An astronomical time series (above) contains 3 near identical subsequences. A "zoom-in" (below) reveals just how similar to each other the 3 subsequences are.

The main advantage of our algorithm is that of providing fast exact answers, and even faster approximate answers.

The rest of this paper is organized as follows. In Section 2 we formally define the problem at hand. In Section 3 we introduce a novel low-dimensional discrete representation of time series, and prove that it can be used to obtain a lower bound on the true Euclidean distance. Section 4 introduces our motif-finding algorithm, which we experimentally evaluate in Section 5. In Section 6 we consider related work, and finally in Section 7 we draw some conclusions and highlight directions for future work.

2. BACKGROUND

The following section is rather dense on terminology and definitions. These are necessary to concretely define the problem at hand, and to explain our proposed solution. We begin with a definition of our data type of interest, time series:

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

Time series can be very long, sometimes containing billions of observations [15]. We are typically not interested in any of the global properties of a time series; rather, data miners confine their interest to subsections of the time series [1, 20, 23], which are called subsequences.

Definition 2. *Subsequence*: Given a time series T of length m, a subsequence C of T is a sampling of length $n < m$ of contiguous position from T, that is, $C = t_p,\ldots,t_{p+n-1}$ for $1 \le p \le m - n + 1$.

A task associated with subsequences is to determine if a given subsequence is similar to other subsequences [1, 2, 3, 8, 13, 19, 21, 22, 23, 24, 25, 27, 29, 35, 40]. This idea is formalized in the definition of a match.

Definition 3. *Match*: Given a positive real number R (called *range*) and a time series T containing a subsequence C beginning at position p and a subsequence M beginning at q, if $D(C, M) \le R$, then M is called a *matching* subsequence of C.

The first three definitions are summarized in Figure 2, illustrating a time series of length 500, and two subsequences of length 128.

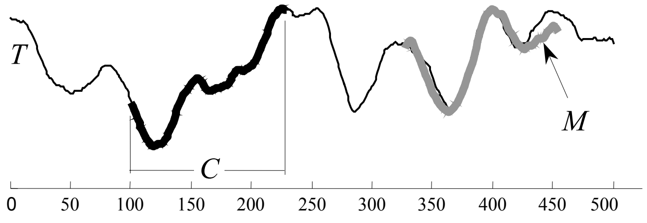

Figure 2: A visual intuition of a time series T (light line), a subsequence C (bold line) and a match M (bold gray line)

For the time being we will ignore the question of what specific distance function to use to determine whether two subsequences match. We will repair this omission in Section 3.3.

The definition of a match is rather obvious and intuitive; but it is needed for the definition of a *trivial match*. One can observe that the best matches to a subsequence (apart from itself) tend to be the subsequences that begin just one or two points to the left or the right of the subsequence in question. Figure 3 illustrates the idea.

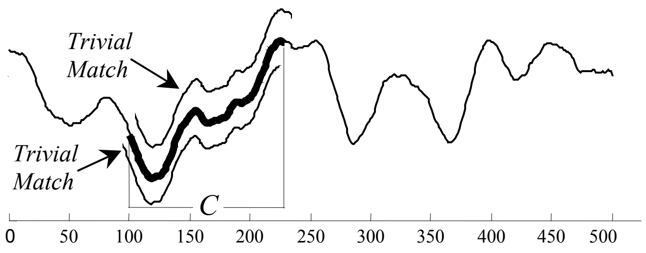

Figure 3: For almost any subsequence C in a time series, the best matches are the trivial subsequences immediately to the left and right of C

Intuitively, any definition of motif should exclude the possibility of over-counting these trivial matches, which we define more concretely below.

Definition 4. *Trivial Match*: Given a time series T, containing a subsequence C beginning at position p and a matching subsequence M beginning at q, we say that M is a *trivial match* to C if either $p = q$ or there does not exist a subsequence M' beginning at q' such that $D(C, M') > R$, and either $q < q' < p$ or $p < q' < q$.

We can now define the problem of enumerating the K most significant motifs in a time series.

Definition 5. *K-Motifs*: Given a time series T, a subsequence length n and a range R, the most significant motif in T (called thereafter *1-Motif*) is the subsequence C_1 that has the highest count of non-trivial matches (ties are broken by choosing the motif whose matches have the lower variance). The K^{th} most significant motif in T (called thereafter *K-Motif*) is the subsequence C_K that has the highest count of non-trivial matches, and satisfies $D(C_K, C_i) > 2R$, for all $1 \le i < K$.

Note that this definition forces the set of subsequences in each motif to be mutually exclusive. This is important because otherwise the two motifs might share the majority of their elements, and thus be essentially the same. Figure 4 illustrates the need for this condition on a simple set of time series projected onto 2-D space.

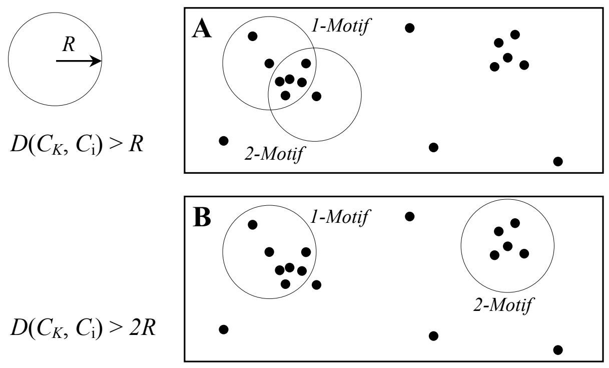

Figure 4: A visual explanation of why the definition of *K-Motif* requires that each motif to be at least *2R* apart. If the motifs are only required to be *R* distance apart as in **A**, then the two motifs may share the majority of their elements. In contrast, **B** illustrates that requiring the centers to be at least *2R* apart insures that the motifs are unique.

Having carefully defined the necessary terminology, we now introduce a brute force algorithm to locate 1-motif. The generalization of this algorithm to finding K-motifs is obvious and omitted for brevity.

```
Algorithm Find-1-Motif-Brute-Force(T,n,R)

best_motif_count_so_far = 0;
best_motif_location_so_far = null;
for i = 1 to length(T)- n + 1
 count    = 0;
 pointers = null;
 for j = 1 to length(T)- n + 1
   if non_trival_match(C[i:i+n-1],C[j:j+n-1],R)
     count = count + 1;
     pointers = append(pointers,j);
   end;
 end;
 if count > best_motif_count_so_far
   best_motif_count_so_far = count;
   best_motif_location_so_far = i;
   motif_matches = pointers;
 end;
end;
```

Table 1: The Find-1-Motif-Brute-Force algorithm

The algorithm requires O(m^2) calls to the distance function. Since the Euclidean distance is symmetric [22], one could theoretically cut in half the CPU time by storing $D(A,B)$ and re-using the value when it is necessary to find $D(B,A)$; however, this would require storing $m(m-1)/2$ values, which is clearly untenable for even moderately sized datasets.

We will introduce our sub-quadratic algorithm for finding motifs in Section 4. Our method requires a discrete representation of the time series that is reduced in dimensionality and upon which a lower bounding distance measure can be defined. Since no representation in the literature fulfills all these criteria, we will introduce such a representation in the next section.

3. DIMENSIONALITY REDUCTION AND DISCRETIZATION

Our discretization technique allows a time series of arbitrary length n to be reduced to a string of arbitrary length w, ($w < n$, typically $w << n$). The alphabet size is also an arbitrary integer a, where $a > 2$. Table 2 summarizes the major notation used in this and subsequent sections.

As an intermediate step between the original time series and our discrete representation of it, we must create a dimensionality-reduced version of the data. We will utilize the Piecewise Aggregate Approximation (PAA) [22, 40], which we review in the next section.

3.1 Dimensionality Reduction

A time series C of length n can be represented in a w-dimensional

C	A time series $C = c_1,\ldots,c_n$
$\bar{C}$	A Piecewise Aggregate Approximation of a time series $\bar{C} = \bar{c}_1,\ldots,\bar{c}_w$
$\hat{C}$	A symbol representation of a time series $\hat{C} = \hat{c}_1,\ldots,\hat{c}_w$
w	The number of raw time series elements represented by a single PAA element or word
a	Alphabet size (e.g., for the alphabet = {**a**,**b**,**c**}, a = 3)

Table 2: A summarization of the notation used in this paper

space by a vector $\bar{C} = \bar{c}_1,\ldots,\bar{c}_w$. The ith element of $\bar{C}$ is calculated by the following equation:

$$\bar{c}_i = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j \qquad (1)$$

Simply stated, to reduce the time series from n dimensions to w dimensions, the data is divided into w equal sized "frames". The mean value of the data falling within a frame is calculated and a vector of these values becomes the data-reduced representation. The representation can be visualized as an attempt to approximate the original time series with a linear combination of box basis functions as shown in Figure 5.

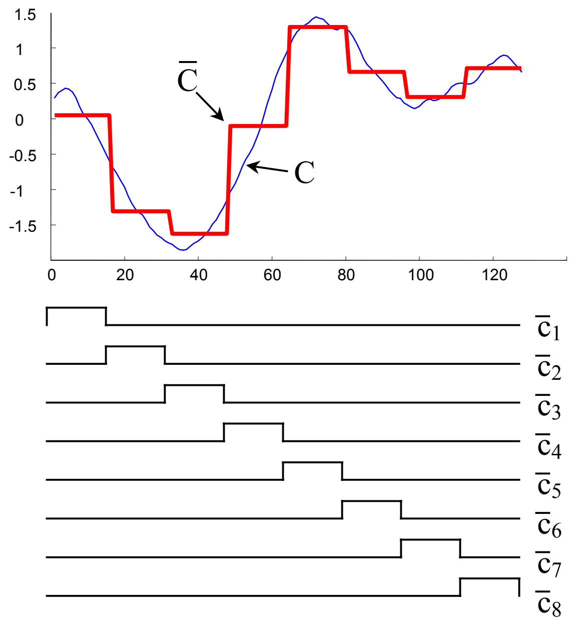

Figure 5: The PAA representation can be readily visualized as an attempt to model a sequence with a linear combination of box basis functions. In this case, a sequence of length 128 is reduced to 8 dimensions

The complicated subscripting in Eq. 1 ensures that the original sequence is divided into the correct number and size of frames.

The PAA dimensionality reduction is intuitive and simple, yet has been shown to rival more sophisticated dimensionality reduction techniques like Fourier transforms and wavelets [8, 22, 40]. In addition, it has several advantages over its rivals, including being much faster to compute, and being able to support many different distance functions, including weighted distance functions [24], arbitrary Minkowski norms [40], and even dynamic time warping [13, 29].

3.2 Discretization

Having transformed a time series database into the PAA we can apply a further transformation to obtain a discrete representation. For reasons that will become apparent in Section 4, we require a discretization technique that will produce symbols with equiprobability. This is easily achieved since normalized time series have a Gaussian distribution. To illustrate this, we extracted subsequences of length 128 from 8 different time series and plotted a normal probability plot of the data as shown in Figure 6.

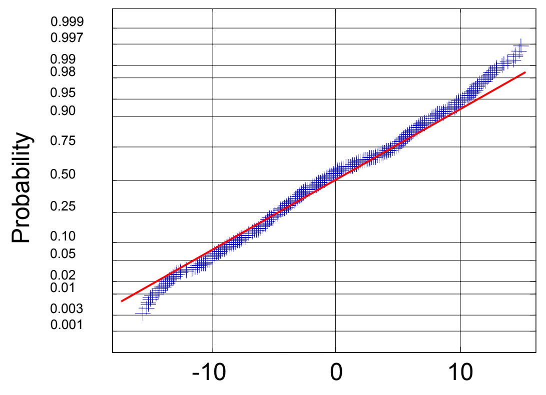

Figure 6: A normal probability plot of the distribution of values from subsequences of length 128 from 8 different datasets. The highly linear nature of the plot strongly suggests that the data came from a Gaussian distribution.

Given that the normalized time series have highly Gaussian distribution, we can simply determine the "breakpoints" that will produce *a* equal-sized areas under Gaussian curve.

Definition 6. *Breakpoints*: breakpoints are a sorted list of numbers $B = \beta_1,\ldots,\beta_{a-1}$ such that the area under a $N(0,1)$ Gaussian curve from β_i to $\beta_{i+1} = 1/a$ (β_0 and β_a are defined as $-\infty$ and ∞, respectively).

These breakpoints may be determined by looking them up in a statistical table. For example, Table 3 gives the breakpoints for values of *a* from 3 to 10.

β_i \ a	3	4	5	6	7	8	9	10
β_1	-0.43	-0.67	-0.84	-0.97	-1.07	-1.15	-1.22	-1.28
β_2	0.43	0	-0.25	-0.43	-0.57	-0.67	-0.76	-0.84
β_3		0.67	0.25	0	-0.18	-0.32	-0.43	-0.52
β_4			0.84	0.43	0.18	0	-0.14	-0.25
β_5				0.97	0.57	0.32	0.14	0
β_6					1.07	0.67	0.43	0.25
β_7						1.15	0.76	0.52
β_8							1.22	0.84
β_9								1.28

Table 3: A lookup table that contains the breakpoints that divide a Gaussian distribution in an arbitrary number (from 3 to 10) of equiprobable regions

Note that in this example the 3 symbols, "**a**", "**b**" and "**c**" are approximately equiprobable as we desired. We call the concatenation of symbols that represent a subsequence a *word*.

Once the breakpoints have been obtained we can discretize a time series in the following manner. We first obtain a PAA of the time series. All PAA coefficients that are below the smallest breakpoint are mapped to the symbol "**a**", all coefficients greater than or equal to the smallest breakpoint and less than the second smallest breakpoint are mapped to the symbol "**b**", etc. Figure 7 illustrates the idea.

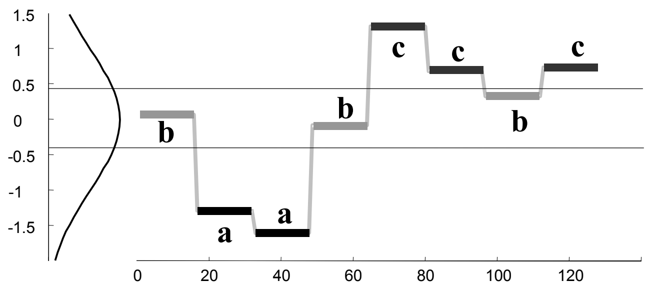

Figure 7: A time series is discretized by first obtaining a PAA approximation and then using predetermined breakpoints to map the PAA coefficients into symbols. In the example above, with *n* = 128, *w* = 8 and *a* = 3, the time series is mapped to the word **baabccbc**

Definition 7. *Word*: A subsequence *C* of length *n* can be represented as a *word* $\hat{C} = \hat{c}_1,\ldots,\hat{c}_w$ as follows. Let alpha$_i$ denote the ith element of the alphabet, i.e., alpha$_1$ = **a** and alpha$_2$ = **b**. Then the mapping from a PAA approximation $\overline{C}$ to a word $\hat{C}$ is obtained as follows:

$$\hat{c}_i = alpha_i, \quad iff \quad \beta_{j-1} < \bar{c}_j \leq \beta_j \qquad (2)$$

We have now defined the two representations required for our motif search algorithm (the PAA representation is merely an intermediate step required to obtain the symbolic representation).

3.3 Distance Measures

Having considered various representations of time series data, we can now define distance measures on them. By far the most common distance measure for time series is the Euclidean distance [8, 22, 23, 32, 40]. Given two time series *Q* and *C* of the same length *n*, Eq. 3 defines their Euclidean distance, and Figure 8.A illustrates a visual intuition of the measure.

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \quad (3)$$

If we transform the original subsequences into PAA representations, $\overline{Q}$ and $\overline{C}$, using Eq. 1, we can then obtain a lower bounding approximation of the Euclidean distance between the original subsequences by:

$$DR(\overline{Q},\overline{C}) \equiv \sqrt{\tfrac{n}{w}}\sqrt{\sum_{i=1}^{w}(\overline{q}_i - \overline{c}_i)^2} \quad (4)$$

This measure is illustrated in Figure 8.B. A proof that $DR(\overline{Q},\overline{C})$ lower bounds the true Euclidean distance appears in [22] (an alterative proof appears in [40]).

If we further transform the data into the symbolic representation, we can define a MINDIST function that returns the minimum distance between the original time series of two words:

$$MINDIST(\hat{Q},\hat{C}) \equiv \sqrt{\tfrac{n}{w}}\sqrt{\sum_{i=1}^{w}\left(dist(\hat{q}_i,\hat{c}_i)\right)^2} \quad (5)$$

The function resembles Eq. 4 except for the fact that the distance between the two PAA coefficients has been replaced with the sub-function *dist*(). The *dist*() function can be implemented using a table lookup as illustrated in Table 4.

	a	b	c
a	0	0	0.86
b	0	0	0
c	0.86	0	0

Table 4: A lookup table used by the MINDIST function. This table is for an alphabet of cardinality, i.e. $a = 3$. The distance between two symbols can be read off by examining the corresponding row and column. For example *dist*(**a**,**b**) = 0 and *dist*(**a**,**c**) = 0.86.

The value in cell (*r,c*) for any lookup table of can be calculated by the following expression.

$$cell_{r,c} = \begin{cases} 0, & if\ |r-c| \le 1 \\ |\beta_i - \beta_{j-1}|, & otherwise \end{cases} \quad (6)$$

For a given value of the alphabet size *a*, the table needs only be calculated once, then stored for fast lookup. The MINDIST function can be visualized is Figure 8.C.

There is one issue we must address if we are to use a symbolic representation of time series. If we wish to approximate a massive dataset in main memory, the parameters *w* and *a* have to be chosen in such a way that the approximation makes the best use of the primary memory available. There is a clear tradeoff between the parameter *w* controlling the number of approximating elements, and the value *a* controlling the granularity of each approximating element.

It is unfeasible to determine the best tradeoff analytically, since it is highly data dependent. We can, however, empirically determine the best values with a simple experiment. Since we wish to achieve the tightest possible lower bounds, we can simply estimate the lower bounds over all possible feasible parameters, and choose the best settings.

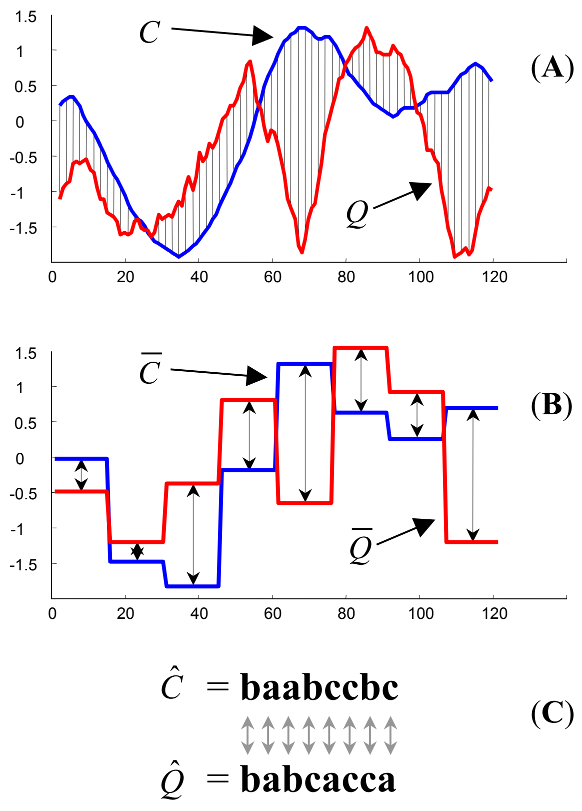

Figure 8: A visual intuition of the three representations discussed in this work, and the distance measures defined on them. **A)** The Euclidean distance between two time series can be visualized as the square root of the sum of the squared differences of each pair of corresponding points. **B)** The distance measure defined for the PAA approximation can be seen as the square root of the sum of the squared differences between each pair of corresponding PAA coefficients, multiplied by the square root of the compression rate. **C)** The distance between two symbolic representations of a time series requires looking up the distances between each pair of symbols, squaring them, summing them, taking the square root and finally multiplying by the square root of the compression rate.

$$Tightness\ of\ Lower\ Bound = \frac{MINDIST(\hat{Q},\hat{C})}{D(Q,C)} \quad (7)$$

We performed such a test with a concatenation of ten time series databases taken from the UCR time series data mining archive. For every combination of parameters we averaged the result of 10,000 experiments on subsequences of length 128. Figure 9 shows the results.

The results suggest that using a low value for *a* results in weak bounds, but that there are diminishing returns for large values of

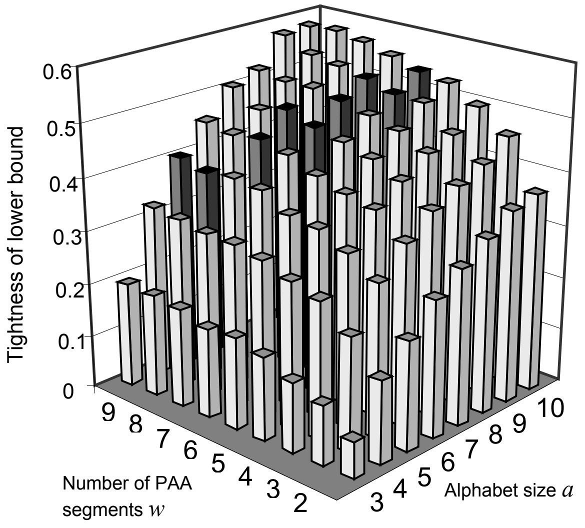

Figure 9: The empirically estimated tightness of lower bounds over the cross product of a = [3…10] and w = [2…9]. The darker histogram bars illustrate combinations of parameters that require approximately equal space to store every possible word (approximately 3.5 megabytes)

a. Based on this experiment, we chose the value of a = 6 for all experiments in this paper.

4. EFFICIENT MOTIF DISCOVERY

Recall that the brute force motif discovery algorithm introduced in Table 1 requires O(m^2) calculations of the distance function. As previously mentioned, the symmetric property of the Euclidean distance measure could be used to half the number of calculations by storing D(Q,C) and re-using the value when it is necessary to find D(C,Q). In fact, further optimizations would be possible under this scenario. We now give an example of such optimization.

Suppose we are in the innermost loop of the algorithm, attempting to enumerate all possible matches within R = 1, to a particular subsequence Q. Further suppose that in previous iterations we had already discovered that $D(C_a,C_b)$ = 2. As we go through the innermost loop we first calculate the distance $D(Q,C_a)$ and discover it to be 7. At this point we should continue on to measure $D(Q,C_b)$, but in fact we don't have to do this calculation! We can use the triangular inequality to discover that $D(Q,C_b)$ could not be a match to Q. The triangular inequality requires that [2, 22, 33]:

$$D(Q,C_a) \leq D(Q,C_b) + D(C_a,C_b) \qquad (8)$$

Filling in the known values give us

$$7 \leq D(Q,C_b) + 2 \qquad (9)$$

Rearranging the terms gives us

$$5 \leq D(Q,C_b) \qquad (10)$$

But since we are only interested in subsequences that are a distance less than 1 unit away, there is no point in determining the exact value of $D(Q,C_b)$, which we now know to be at least 5 units away.

The first formalization of this idea for fast searching of nearest neighbors in matrices is generally credited to Burkhard and Keller [5]. More efficient implementations are possible, for example Shasha and Wang [33], introduced the Approximation Distance Map (ADM) algorithm that takes advantage of an arbitrary set of pre-computed distances instead of using just one randomly chosen reference point.

For the problem at hand, however, the techniques discussed above seem of little utility, since as previously noted, we are unlikely to have O(m^2) space in which to store the entire matrix. We propose to use just such a technique as a subroutine in our motif discovery algorithm. Our idea is to create only a small portion of the matrix at a time, and exploit the techniques above to search it. Our contribution comes from the method we use to construct the small matrix. As we will demonstrate, we can use our MINDIST function to create a matrix, much smaller than O(m^2), which is *guaranteed* to contain all the subsequences which are within R of a promising candidate for a motif.

In addition to all the matching sequences to a promising candidate, the small matrix will generally contain some non-matching subsequences, or "false hits". We use Shasha and Wang's ADM algorithm to efficiently separate the true matches to the false hits.

There is a possibility that a promising candidate for a motif will pan out. That is, after searching the small matrix we will discover that most or all of the subsequences don't match. In this case we will have to construct a new small matrix and continue the search with the next most promising motif. If the new small matrix has any overlap with the previous matrix, we reuse the calculated values rather than recalculating them.

Constructing these small matrices would be of limited utility if their total size added up to O(m^2). While this is possible in pathological cases, we can generally search a space much smaller total size, and still guarantee that we have returned the true best K-Motifs.

This, in essence, is the intuition behind our motif discovery algorithm. We will achieve speed up by:

- Searching a set of smaller matrices, whose total size is much less than the naïve O(m^2) matrix.
- Within the smaller matrices, using ADM to prune away a large fraction of the search space.

We will concretely define our algorithm, which we call EMMA (Enumeration of Motifs through Matrix Approximation), in the next section.

4.1 The EMMA Algorithm

As before, we only discuss the algorithm for finding the 1-Motif. The generalization of the algorithm to finding K-motifs is obvious and omitted for brevity and clarity of presentation. The pseudocode for the algorithm is introduced in Table 5. The line numbers in the table are used in the discussion of the algorithm that follows.

The algorithm begins by sliding a moving window of length n across the time series (line 4). The hash function h() (line 5), normalizes the time series, converts it to the symbolic representation and computes an address:

$$h(C,w,a) = 1 + \sum_{i=1}^{w} (ord(\hat{c}_i) - 1) \times a^{i-1} \qquad (11)$$

Where $ord(\hat{c}_i)$ is the ordinal value of $\hat{c}_i$, i,e., $ord(\mathbf{a}) = 1$, $ord(\mathbf{b}) = 2$, and so on. The hash function computes an integer in the range 1 to w^a, and a pointer to the subsequence is placed in the corresponding bucket (line 6).

```
Algorithm Find-1-Motif-Index(T,n,R,w,a)
 1.  Best_motif_count = 0;
 2.  best_motif_location = null;
 3.  finished = FALSE;
 4.  for i = 1 to length(T)- n + 1   // Hash pointers
 5.     hash_val = h($C_{[i:i+n-1]}$,w,a);   // to subsequences
 6.     append(bucket(hash_val).pointers, i);
 7.  end;
 8.  MPC = address(largest(bucket));   // Find MPC
 9.  neighborhood = bucket(MPC).pointers;
10.  while not(finished)
11.   for i = 1 to $w^a$                  // Build neighborhood
12.    if MINDIST(MPC, bucket(i) ) < R    // around
13.     temp = bucket(i).pointers;        // the MPC
14.     neighborhood = append(neighborhood,temp)
15.    end;
16.   end;                  // Search neighborhood for motifs
17.   [motif_cntr,count]= ADM(T,neighborhood,R);
18.   if count > largest_unexplored_neighborhood
19.     best_motif_location = motif_cntr;
20.     best_motif_count = count;
21.     finished = TRUE;
22.   else          // Create the next neighborhood to search
23.     MPC = address(largest_unexplored(bucket));
24.     neighborhood = bucket(MPC).pointers;
25.   end;
26.  end;
```

Table 5: The Find-1-Motif-Index algorithm

At this point we have simply rearranged the data into a hash table with w^a addresses, and a total of size O(m). This arrangement of the data has the advantage of approximately grouping similar subsequences (more accurately, pointers to similar subsequences) together. We can use this information as a heuristic for motif search, since if there is a truly over-represented pattern in the time series, we should expect that most, if not all, copies of it hashed to the same location. We call the address with the most hits the Most Promising Candidate (MPC) (line 8). We can build a list of all subsequences that mapped to this address (line 9), but it is possible that some subsequences that hashed to different addresses are also within R of the subsequences contained in MPC. We can use the MINDIST function that we defined in Section 3.3 to determine which addresses could possibly contain such subsequences (line 12). All such subsequences are added to the list of subsequences that need to be examined in our small matrix (line 14). We call the contents of a promising address, together with the contents of all the addresses within a MINDIST of R to it, a *neighborhood*.

At this point we can pass the list of similar subsequences into the ADM subroutine (line 17). We will elucidate this algorithm later, in Section 4.2. For the moment we just note that the algorithm will return the best motif from the original MPC subset, with a count of the number of matching subsequences.

If we wish to implement the algorithm as an online algorithm, then at this point we can report the current motif as a tentative answer, before continuing the search. Such "anytime" behavior is very desirable in a data-mining algorithm [7].

Next, a simple test is performed. If the number of matches to the current best-so-far motif is greater than the largest unexplored neighborhood (line 18), we are done. We can record the best so far motif as the true best match (line 19), note the number of matching subsequences (line 20), and then abandon the search (line 21).

If the test fails, however, we must set the most promising candidate to be the next largest bucket (line 23), initialize the new neighborhood with the contents of the bucket (line 24), and loop back to line 11, where the full neighborhood is discovered (lines 13 and 14) and the search continues.

For simplicity the pseudocode for the algorithm ignores the following possible optimization: it is possible (in fact, likely), that the neighborhood in one interaction will overlap with the neighborhood in the next. In this case, we can reuse the subset of calculated values from iteration to iteration.

4.2 The ADM Algorithm

The algorithm we use for searching the small neighborhood matrix is a minor modification of the Shasha and Wang's ADM algorithm [33]. However, we include a brief exposition for completeness. The algorithm is outlined in Table 6.

The algorithm begins by pre-computing an arbitrary set of distances (line 1). Two matrices, ADM and MIN, are used to store the lower bound and upper bound, respectively, distances of any path between two objects. We use the same terminology as introduced in [33] for consistency. Each entry of ADM[i, j] is either the exact distance between i and j (i.e. those that are pre-computed), or the lower bound for the distance between i and j. In other words, if $P_{i,j}$ contains the set of all paths from i to j, then ADM[i, j] is the largest lower bound distance between i and j, obtained from all paths in $P_{i,j}$ using the triangle inequality [2, 22, 33].

As described in [33], we also maintain a matrix MIN because it's impractical to enumerate all paths in $P_{i,j}$ to get the maximum lower bound between i and j. MIN[i,j] stores the minimum distance of any path from i to j (i.e. it's the least upper bound of distance between i and j). ADM and MIN are initialized from line 4 to line 8.

From line 10 to line 22, we construct the matrix ADM and MIN as described in [33]. For each stage k, $1 \le k \le n$, ADM[i, j] is the greatest lower bound of any path from i to j that does not pass through an object numbered greater than k [5]. Similarly, MIN[i, j] is the smallest upper bound of the distance between i and j. Note that we further optimized the algorithm by storing or computing only half of the matrices, due to distance symmetry. However, for simplicity and consistency, we show the algorithm of constructing ADM as was presented in [33].

```
Algorithm ADM(T,neighborhood,R)

1.  Select and pre-compute an arbitrary set of distances;
2.  N = length(neighborhood);
3.
4.  Initialize ADM and MIN: for all cells in ADM
5.  and MIN, if the distances were pre-computed
6.  (from line 1), set them to the true
7.  distances; otherwise set ADM to 0 and MIN to
8.  ∞.
9.
10. for k = 1 to N               // construct ADM
11.   for i = 1 to N
12.     for j = 1 to N
13.       // ADM is the greatest lower bound between
14.       // i, j that does not pass thru k
15.       ADM[i, j] = max(ADM[i, j],
16.                        ADM[i, k] - MIN[k, j],
17.                        ADM[j, k] - MIN[k, i]);
18.       // MIN is the smallest upper bound
19.       MIN[i, j] = min(MIN[i, j], MIN[i, k] + MIN[k, j]);
20.     end;
21.   end;
22. end;
23.
24. Initialize count[1..N] = 0;
25.
26. for i = 1 to N      // compute actual distance
27.   for j = 1 to N    // for lower_bound < R
28.     if ADM[i,j] is a lower-bound && ADM[i,j] < R
29.       compute the actual distance between i,j
30.     end;
31.     if ADM[i,j] < R   // keep track the #items
32.       count[i]++;     // within R
33.     end;
34.   end;
35. end;
36.
37. best_match = max{count[1], count[2], …, count[N]};
38. return neighborhood[best_match], count[best_match];
```

Table 6: The ADM Algorithm

On line 24, we allocate and initialize an array, count, which stores the number of items within R (i.e. number of matching subsequences) for each motif center. From line 26 to line 35, we scan the matrix ADM and compute the actual distance between i and j if ADM[i,j] is a lower bound that is smaller than R (because the true distance might be greater than R). Again, we omit the optimization of distance symmetry for simplicity. At each step, we keep track of the number of items within R (line 32).

Finally, the algorithm returns the best-matching motif (i.e. one with the most items within R), with a count of number of matching subsequences.

5. EXPERIMENTAL EVALUATION

We begin by showing some motifs discovered in a variety of time series. We deliberately consider time series with very different properties of noise, autocorrelation, stationarity etc. Figure 10 shows the 1-Motif discovered in various datasets, together with a much larger subsequence of the time series to give context.

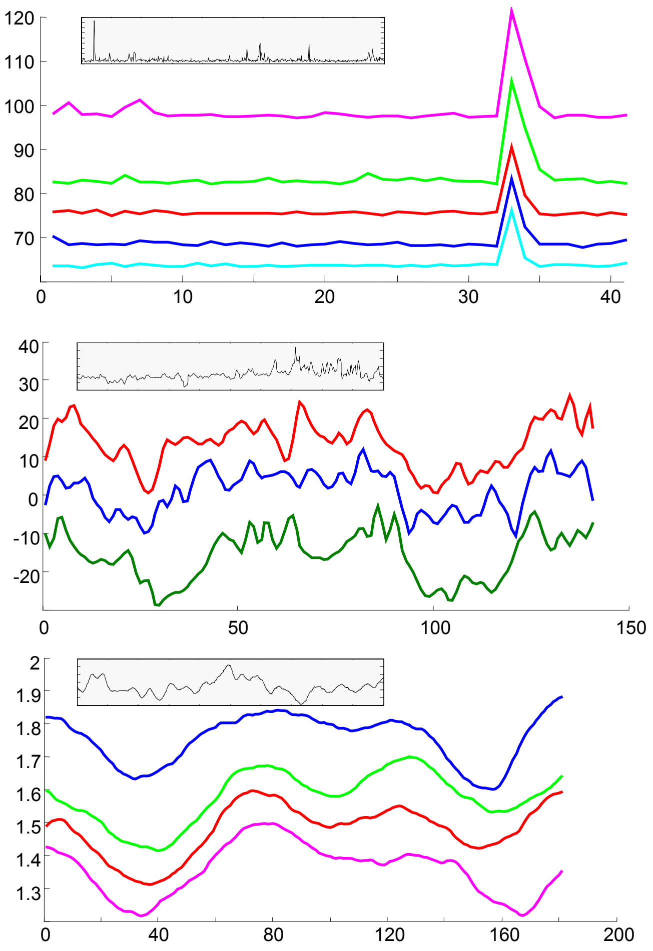

Figure 10: The 1-Motif discovered in various publicly available datasets. From top to bottom, "Network", "Tide" and "Burst". Details about the datasets are available from the UCR time series data mining archive. The small inset boxes show a subsequence of length 500 to give context to the motif

Although the subsequences are normalized [22] before testing to see if they match, we show the unnormalized subsequences for clarity.

We next turn our attention to evaluating the efficiency of the proposed algorithm. For simplicity we have only considered the problem of speeding up motif search when the whole time series fits in main memory (we intend to address efficient disk-based algorithms in future work). So we can evaluate the efficiency of the proposed algorithm by simply considering the ratio of how many times the Euclidean distance function must be evaluated by EMMA, over the number of times it must be evaluated by the brute force algorithm described in Table 1.

$$efficiency = \frac{number\ of\ times\ EMMA\ calls\ Euclidean\ dist}{number\ of\ times\ brute-force\ calls\ Euclidean\ dist} \quad (12)$$

This measure ignores the cost of building the hash table, but this needs to be done only once (even if the user wishes to try several values of R), and is in any case linear to m.

The efficiency depends on the value of R used in the experiments. In the pathological case of $R = \infty$, only one "small" matrix would be created, but it would be O(m^2). Even if we could fit such a large matrix in main memory, the only speed-up would come from ADM algorithm. The other pathological case of $R = 0$ would make our algorithm behave very well, because only a few very small matrices would be created, and the triangular inequality pruning of ADM algorithm would be maximally efficient. Of course, neither of these scenarios is meaningful; the former would result in a Motif with every (non-trivial) subsequence in the time series included, and the latter case would almost certainly result in no motif being found (since we are dealing with real numbers).

In order to test with realistic values of R we will consider the efficiency achieved when using the values used to create the results shown in Figure 10. The results are shown in Table 7.

Dataset	**Network**	**Tide**	**Burst**
efficiency	0.0018	0.0384	0.0192

Table 7: The efficiency of the EMMA algorithm on various datasets

These results indicate a one to two order of magnitude speedup over the brute force algorithm.

6. RELATED WORK

To the best of our knowledge, the problem of finding repeated patterns in time series has not been addressed (or even formulated) in the literature.

Several researchers in data mining have addressed the discovery of reoccurring patterns in event streams [39], although such data sources are often referred to as time series [38]. The critical difference is that event streams are sequentially ordered variables that are nominal (have no natural ordering) and thus these researchers are concerned with similar subsets, not similar subsequences. Research by Indyk et. al. [20] has addressed the problem of finding representative trends in time series. This work is more similar in spirit to our work. However, they only consider trends, not more general patterns, and they only consider locally representative trends, not globally occurring motifs as in our approach.

While there has been enormous interest in efficiently locating *previously known* patterns in time series [1, 2, 3, 8, 13, 19, 22, 23, 24, 27, 29, 32, 35, 40], our focus on the discovery of previously unknown patterns is more similar to (and was inspired by) work in computational biology, which we briefly review below.

In the context of computational biology, "pattern discovery'" refers to the automatic identification of biologically significant patterns (or *motifs*) by statistical methods. The underlying assumption is that biologically significant words show distinctive distribution patterns within the genomes of various organisms, and therefore they can be distinguished from the others. During the evolutionary process, living organisms have accumulated certain biases toward or against some specific motifs in their genomes. For instance, highly recurring oligonucleotides are often found in correspondence to regulatory regions or protein binding sites of genes. Vice versa, rare oligonucleotide motifs may be discriminated against due to structural constraints of genomes or specific reservations for global transcription controls.

Pattern discovery in computational biology originated with the work of Rodger Staten [34]. Along this research line, a multitude of patterns have been variously characterized, and criteria, algorithms and software have been developed in correspondence. We mention a few representatives of this large family of methods, without claiming to be exhaustive: CONSENSUS [16], GIBBS SAMPLER [26], WINNOWER [30], PROJECTION [36], VERBUMCULUS [4, 28] These methods have been studied from a rigorous statistical viewpoint (see, e.g., [31] for a review) and also employed successfully in practice (see, e.g., [17] and references therein).

While there are literally hundreds of papers on discretizing (symbolizing, tokenizing) time series [2, 3, 9, 13, 19, 25, 27] (see [10] for an extensive survey), and dozens of distance measures defined on these representations, none of the techniques allows a distance measure which lower bounds a distance measure defined on the original time series.

7. CONCLUSIONS

We have formalized the problem of finding repeated patterns in time series, and introduced an algorithm to efficiently locate them. In addition, a minor contribution of this paper is to introduce the first discrete representation of time series that allows a lower bounding approximation of the Euclidean distance. This representation may be of independent interest to researchers who use symbolic representations for similarity search [3, 19, 25, 27, 29], change point detection [13], and extracting rules from time series [9, 18].

There are several directions in which we intend to extend this work.

- As previously noted, we only considered the problem of speeding up main memory search. Techniques for dealing with large disk resident data are highly desirable [6].
- On large datasets, the number of returned motifs may be intimidating; we plan to investigate tools for visualizing and navigating the results of a motif search.
- Our motif search algorithm utilizes the Euclidean metric, and can be trivially modified to use any Minkowski metric [40]. However, recent work by several authors has suggested that the Euclidean may be inappropriate in some domains [21, 29]. We hope to generalize our results to work with other more robust distance measures, such as Dynamic Time Warping [29].
- It may be possible to extend our work to multi-dimensional time series (i.e., trajectories) [37]; however, such an extension will compound the need for more research into the best parameters settings for our algorithm.

8. REFERENCES

[1] Agrawal, R., Faloutsos, C. & Swami, A. (1993). Efficient similarity search in sequence databases. In *proceedings of the 4th Int'l Conference on Foundations of Data Organization and Algorithms*. Chicago, IL, Oct 13-15. pp 69-84.

[2] Agrawal, R., Psaila, G., Wimmers, E. L. & Zait, M. (1995). Querying shapes of histories. In *proceedings of the 21st Int'l Conference on Very Large Databases*. Zurich, Switzerland, Sept 11-15. pp 502-514.

[3] André-Jönsson, H. & Badal. D. (1997). Using signature files for querying time-series data. In *proceedings of Principles of Data Mining and Knowledge Discovery*, 1st European Symposium. Trondheim, Norway, Jun 24-27. pp 211-220.

[4] Apostolico, A., Bock, M. E. & Lonardi, S. (2002). Monotony of surprise and large-scale quest for unusual words (extended abstract). Myers, G., Hannenhalli, S., Istrail, S., Pevzner, P. & Waterman, M. editors. In *proceedings of the 6th Int'l Conference on Research in Computational Molecular Biology*. Washington, DC, April 18-21. pp 22-31.

[5] Burkhard, W. A. & Keller, R. M. (1973). Some approaches to best-match file searching. *Commun. ACM*, April. Vol. 16(4), pp 230-236.

[6] Böhm, C., Braunmüller, B., Krebs, F. & Kriegel, H. P. (2002). Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data. In *proceedings of ACM SIGMOD Int. Conf. on Management of Data*, Santa Barbara.

[7] Bradley, P. S., Fayyad, U. M. & Reina, C. A. (1998). Scaling clustering algorithms to large databases. In *proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining*. New York, NY, Aug 27-31. pp 9-15.

[8] Chan, K. & Fu, A. W. (1999). Efficient time series matching by wavelets. In *proceedings of the 15th IEEE Int'l Conference on Data Engineering*. Sydney, Australia, Mar 23-26. pp 126-133.

[9] Das, G., Lin, K., Mannila, H., Renganathan, G. & Smyth, P. (1998). Rule discovery from time series. In *proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining*. New York, NY, Aug 27-31. pp 16-22.

[10] Daw, C. S., Finney, C. E. A. & Tracy, E. R. (2001). Symbolic analysis of experimental data. *Review of Scientific Instruments*. To appear.

[11] Durbin, R., Eddy, S., Krogh, A. & Mitchison, G. (1998). Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge University Press.

[12] Fayyad, U., Reina, C. &. Bradley. P (1998). Initialization of iterative refinement clustering algorithms. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*. New York, NY, Aug 27-31. pp 194-198.

[13] Ge, X. & Smyth, P. (2000). Deformable Markov model templates for time-series pattern matching. In *proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Boston, MA, Aug 20-23. pp 81-90.

[14] Goutte, C. Toft, P., Rostrup, E.,. Nielsen F.Å & Hansen L.K. (1999). On clustering fMRI time series, NeuroImage, 9(3): pp 298-310.

[15] Hegland, M., Clarke, W. & Kahn, M. (2002). Mining the MACHO dataset, *Computer Physics Communications*, Vol 142(1-3), December 15. pp. 22-28.

[16] Hertz, G. & Stormo, G. (1999). Identifying DNA and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics*, Vol. 15, pp 563-577.

[17] van Helden, J., Andre, B., & Collado-Vides, J. (1998) Extracting regulatory sites from the upstream region of the yeast genes by computational analysis of oligonucleotides. *J. Mol. Biol.*, Vol. 281, pp 827-842.

[18] Höppner, F. (2001). Discovery of temporal patterns -- learning rules about the qualitative behavior of time series. In *Proceedings of the 5th European Conference on Principles and Practice of Knowledge Discovery in Databases*. Freiburg, Germany, pp 192-203.

[19] Huang, Y. & Yu, P. S. (1999). Adaptive query processing for time-series data. In *proceedings of the 5th Int'l Conference on Knowledge Discovery and Data Mining*. San Diego, CA, Aug 15-18. pp 282-286.

[20] Indyk, P., Koudas, N. & Muthukrishnan, S. (2000). Identifying representative trends in massive time series data sets using sketches. In *proceedings of the 26th Int'l Conference on Very Large Data Bases*. Cairo, Egypt, Sept 10-14. pp 363-372.

[21] Kalpakis, K., Gada, D. & Puttagunta, V. (2001). Distance measures for effective clustering of ARIMA time-series. In *proceedings of the 2001 IEEE International Conference on Data Mining*, San Jose, CA, Nov 29-Dec 2. pp 273-280.

[22] Keogh, E,. Chakrabarti, K,. Pazzani, M. & Mehrotra (2000). Dimensionality reduction for fast similarity search in large time series databases. *Journal of Knowledge and Information Systems.* pp 263-286.

[23] Keogh, E., Chakrabarti, K., Pazzani, M. & Mehrotra, S. (2001). Locally adaptive dimensionality reduction for indexing large time series databases. In *proceedings of ACM SIGMOD Conference on Management of Data*. Santa Barbara, CA, May 21-24. pp 151-162.

[24] Keogh, E. & Pazzani, M. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *proceedings of the 4th Int'l Conference on Knowledge Discovery and Data Mining*. New York, NY, Aug 27-31. pp 239-241.

[25] Koski, A., Juhola, M. & Meriste, M. (1995). Syntactic recognition of ECG signals by attributed finite automata. *Pattern Recognition*, 28 (12), pp. 1927-1940.

[26] Lawrence, C.E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F. & Wootton, J. C. (1993). Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science*, Oct. Vol. 262, pp 208-214.

[27] Li, C., Yu, P. S. & Castelli, V. (1998). MALM: a framework for mining sequence database at multiple abstraction levels. In *proceedings of the 7th ACM CIKM International Conference on Information and Knowledge Management*. Bethesda, MD. pp 267-272.

[28] Lonardi, S. (2001). Global Detectors of Unusual Words: Design, Implementation, and Applications to Pattern Discovery in Biosequences. PhD thesis, Department of Computer Sciences, Purdue University, August, 2001.

[29] Perng, C., Wang, H., Zhang, S., & Parker, S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. In *proceedings of 16th International Conference on Data Engineering.*

[30] Pevzner, P. A. & Sze, S. H. (2000). Combinatorial approaches to finding subtle signals in DNA sequences. In

proceedings of the 8th International Conference on Intelligent Systems for Molecular Biology. La Jolla, CA, Aug 19-23. pp 269-278.

[31] Reinert, G., Schbath, S. & Waterman, M. S. (2000). Probabilistic and statistical properties of words: An overview. J. *Comput. Bio*., Vol. 7, pp 1-46.

[32] Roddick, J. F., Hornsby, K. & Spiliopoulou, M. (2001). An Updated Bibliography of Temporal, Spatial and Spatio-Temporal Data Mining Research. In *Post-Workshop Proceedings of the International Workshop on Temporal, Spatial and Spatio-Temporal Data Mining*. Berlin, Springer. Lecture Notes in Artificial Intelligence. 2007. Roddick, J. F. and Hornsby, K., Eds. 147-163.

[33] Shasha, D. & Wang, T. (1990). New techniques for best-match retrieval. *ACM Trans. on Information Systems*, Vol. 8(2). pp 140-158.

[34] Staden, R. (1989). Methods for discovering novel motifs in nucleic acid sequences. *Comput. Appl. Biosci*., Vol. 5(5). pp 293-298.

[35] Struzik, Z. R. & Siebes, A. (1999). Measuring time series similarity through large singular features revealed with wavelet transformation. In *proceedings of the 10th International Workshop on Database & Expert Systems Applications*. pp 162-166.

[36] Tompa, M. & Buhler, J. (2001). Finding motifs using random projections. In *proceedings of the 5th Int'l Conference on Computational Molecular Biology*. Montreal, Canada, Apr 22-25. pp 67-74.

[37] Vlachos, M., Kollios, G. & Gunopulos, G. (2002). Discovering similar multidimensional trajectories. In *proceedings of the 18th International Conference on Data Engineering*. pp 673-684.

[38] Wang. W., Yang, J. and Yu., P. (2001). Meta-patterns: revealing hidden periodical patterns. In *Proceedings of the 1st IEEE International Conference on Data Mining*. pp. 550-557.

[39] Yang, J., Yu, P., Wang, W. and Han. J. (2002). Mining long sequential patterns in a noisy environment. In *proceedings SIGMOD International. Conference on Management of Data*. Madison, WI.

[40] Yi, B, K., & Faloutsos, C. (2000). Fast time sequence indexing for arbitrary L*p* norms. In *proceedings of the 26st Intl Conference on Very Large Databases*. pp 385-394.