Enumeration of Time Series Motifs of All Lengths

Abdullah Mueen
University of New Mexico
mueen@unm.edu

***Abstract*— Time series motifs are repeated patterns in long and noisy time series. Motifs are typically used to understand the dynamics of the source because repeated patterns with high similarity evidentially rule out the presence of noise. Recently, time series motifs have also been used for clustering, summarization, rule discovery and compression as features. For all such purposes, many high quality motifs of various lengths are desirable and thus, originates the problem of enumerating motifs for a wide range of lengths.**

Existing algorithms find motifs for a given length. A trivial way to enumerate motifs is to run one of the algorithms for the whole range of lengths. However, such parameter sweep is computationally infeasible for large real datasets. In this paper, we describe an *exact* algorithm, called $MOEN$, to enumerate motifs. The algorithm is an order of magnitude faster than the naive algorithm. The algorithm frees us from re-discovering the same motif at different lengths and tuning multiple data-dependent parameters. The speedup comes from using a novel bound on the similarity function across lengths and the algorithm uses only linear space unlike other motif discovery algorithms. We describe three case studies in entomology and activity recognition where $MOEN$ enumerates several high quality motifs.

I. Introduction

Time series motifs are repeated patterns in a noisy and long time series data. Typical use of time series motifs is in unsupervised data exploration or analytics. Figure 1 shows two real examples where enumeration of multiple repetitions or motifs from a time series are used for knowledge discovery. First example is in understanding clusters of genes affecting the locomotion of a nematode (*Caenorhabditis elegans*) [3] and the second example is in understanding the repetitive structure of human brain activity recorded through *Electroencephalography* (EEG) [13]. In both of the examples, scientists have run a motif discovery algorithm numerous times to find the repeated patterns for a range of lengths and process the patterns to mine the data.

We refer the problem of finding repetitions at different lengths as enumeration of motifs in a time series. The enumerated motifs can be used in many data mining tasks. The set of motifs work as a summarization of the data [4], when the time series data contains a lot of noise. Clustering the enumerated motifs is shown to be better than clustering all of the subsequences in time series [20]. Enumerated motifs can be used to compress time series data tightly [12]. Finally, all of these applications work better if we can find a comprehensive enumeration over a range of lengths.

There has been a considerable amount of work on time series motif discovery that successfully find motifs from a variety of time series data. Exact algorithms find motifs for a given length [12][13] and are believed to be intractable for enumeration because of the massive computation time required

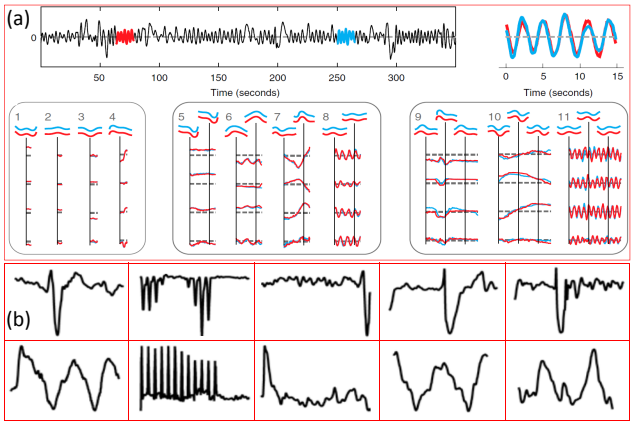

Fig. 1. Two examples of enumeration of time series motifs. (a) In the top, a sample time series projected from moving *Caenorhabditis elegans* and a sample motif in the time series are shown. Eleven of the fourteen motifs are shown in three of the four clusters found. The shapes of the worm are shown at the top of the bars and the motifs in the four channels are shown below them [3]. (b) A set of repeated EEG patterns collected to study the hierarchical structure of human EEG data [13]. There are approximately 114,000 distinct motifs found so far, spanning a wide range of lengths.

to find motifs for all lengths. For example, to find all the motifs in Figure 1(a), it takes about fifteen hours on a four way down-sampled data. A number of approximate algorithms exist that project the data in lower dimensions and find approximate motifs in the lower dimensional space [10][15]. *None* of them guarantees finding motifs of various lengths and *all* of them require a set of data-dependent parameters. Multiple trials are needed to tune those parameters for arbitrary data and often times, the process is equally as time consuming as running the exact methods. In the worst case, different segments of the time series may require different sets of parameters and thus, making the search for the best parameters infeasible.

In this paper, we describe an *exact* algorithm to enumerate motifs of all lengths. The algorithm searches for the entire range of motif-lengths and outputs only maximally covering motifs. Figure 2 shows a set of maximally covering motifs of various lengths discovered by our tool from an EEG trace. The algorithm is significantly faster than running the existing exact algorithms multiple times and an order of magnitude faster than the naive solution. We demonstrate the application of the algorithm to discover high quality motifs in entomological exploration and activity/dance pattern recognition.

The key feature of the algorithm is a novel bound on the similarity function to prune most of the similarity computations for successive lengths once the motifs for the first length is found. The algorithm is free of data-dependent parameter

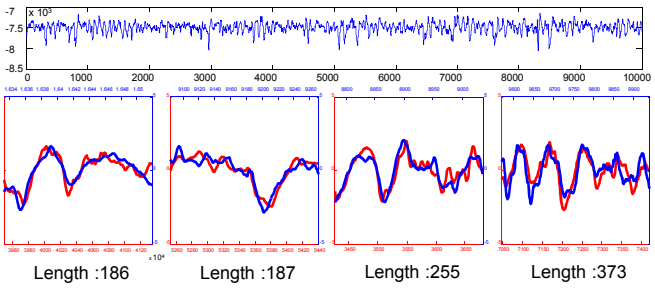

Fig. 2. An EEG trace is shown at the top. Four different motifs of different lengths are shown at the bottom.

and uses only linear space, just enough to store the time series and some additional statistics. We add a knob to configure the number of motifs to output for iterative exploration.

We organize the rest of the paper in traditional fashion starting with some background on existing work (Section II). We define the necessary notations in Section III. In Section IV we give the trivial algorithm first and then provide an intuitive description of the method and finally describe the algorithm and its variants. In Section V we show experimental results and show the case studies in Section VI.

II. Background and Related Work

Time series motif discovery was introduced in 2002 [10] with a hash based technique to find repeated patterns. Since then numerous algorithms have been proposed focusing on many applications [22][17] [5][23][15]. Many of the methods for motif discovery are based on searching a discrete approximation of the time series, inspired by and leveraging off the rich literature of motif discovery in discrete data such as DNA sequences.

Several of the algorithms are based on *SAX* (Symbolic Aggregate approXimation) representation that discretizes both time and y-values and represents the time series with symbols [24][17][5]. The quality of the discretization process depends on two parameters w and α. For example, large w on a noisy time series can be massively lossy. Some algorithms are based on locality sensitive hashing and depends on optimal setting of at least four independent parameters. Beside the undesirable list of parameters, the algorithms also have some complexity in space usage. They extract all the contiguous subsequences (n) for a specific length (m) and convert all of them to SAX representation. Therefore, the space requirement is $O(n\frac{m}{w})$. In motif enumeration, the maximum length of a motif can be arbitrarily large limiting the largest time series we can input.

It has long been held that the exact motif discovery is intractable even for datasets residing in main memory. In a recent work the current authors have shown that motif discovery is tractable for large in-core datasets [13]; however, the algorithm we proposed (called MK) also suffers from a very demanding parameter m and $O(nm)$ space requirement. Enumerating motifs of all lengths can be done by repeatedly running MK but, very quickly becomes intractable for moderate sized datasets.

There are algorithms to discover variable-length motifs [9][16][23]. The method in [9] uses SAX representation and thus, the method is an approximate one. The method in [23] doesn't even normalize the subsequences and therefore, can't detect motifs with a shift and scaling.

III. Definition

In this section, we define the problem and the other notations used in the paper.

Definition 1: A *Time Series* T is a sequence of real numbers $t_1, t_2, \ldots, t_n$. A time series *subsequence* $T_{i,m} = t_i, t_{i+1}, \ldots, t_{i+m-1}$ is a continuous subsequence of T starting at position i and length m.

A time series of length n can have $\frac{n(n+1)}{2}$ subsequences of all possible lengths from one to n.

If we are given two time series X and Y of same length m, we can use the euclidean norm of their difference (i.e. $X - Y$) as the similarity function. To achieve scale and offset invariance, we must normalize the individual time series using *z-normalization* before the actual distance is computed. This is a critical step; even tiny differences in scale and offset rapidly swamp any similarity in shape [7]. The normalized euclidean distance is generally computed by the formula $\sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$ where $x_i = \frac{1}{\sigma_x}(X_i - \mu_x)$ and $y_i = \frac{1}{\sigma_y}(Y_i - \mu_y)$. Thus, computing a distance value requires time linear on the length of the time series.

In contrast, we can compute the normalized Euclidean distance between X and Y using five numbers derived from X and Y. These numbers are denoted as sufficient statistics in [21]. The numbers are $\sum x$, $\sum y$, $\sum x^2$,$\sum y^2$ and $\sum xy$. It may appear that we are doing more work than necessary; however, as we make clear in section IV-A, computing the distance in this manner enables us to reuse computations and reduce the amortized time complexity from *linear* to *constant*.

The sample mean and standard deviation can be computed from these statistics as $\mu_x = \frac{1}{m}\sum x$ and $\sigma_x^2 = \frac{1}{m}\sum x^2 - \mu_x^2$, respectively. The positive correlation and the normalized euclidean distance between X and Y can then be expressed as below [14].

$$C(x,y) = \frac{\sum xy - m\mu_x\mu_y}{m\sigma_x\sigma_y}$$

$$dist(x,y) = \sqrt{2m(1 - C(x,y))}$$

Now we define motif and the enumeration problem.

Definition 2: A *motif* ($T_{i,m}$,$T_{j,m}$) is the most similar pair of non-overlapping subsequences of length m where $i < j$.

To extend the above definition for a range of lengths is not as straight forward as making the length of the motifs (m) variable. In Figure 3, we illustrate the locations of different motifs for four hundred different lengths i.e. [100 500]. Let's take an example to describe the *covering* relationship between motifs. The first locations of the motifs (see Figure 3) at lengths 250 and 175 are identical (100% overlap) and therefore, we say 250 *covers* 175. The number of such covering motifs can vary in different time series. We see, in Figure 3, there are only ten different motifs present among the 400 different lengths. For most smooth time series, only a few motifs cover the entire

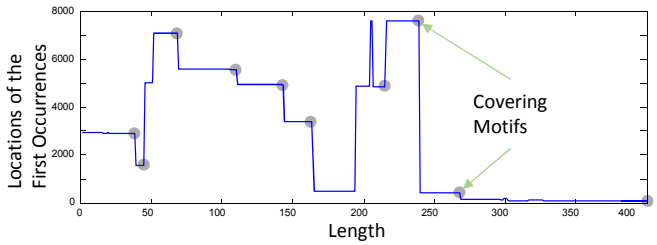

Fig. 3. Locations of the motifs' first occurrences for 400 different lengths. There are only a few *different* motifs that show up. In this example we have only ten covering motifs among the 400 lengths.

range. Therefore we need a way to remove this redundancy by showing only the covering motifs.

Definition 3: A subsequence $T_{i,m}$ *covers* another subsequence $T_{u,p}$, $p \leq m$, if the overlap between $T_{i,m}$ and $T_{u,p}$ is more than a threshold $c\%$. More mathematically, any of the two conditions is true.

$$i - u \geq 0 \text{ and } i - u \leq (1 - c\%) * p$$
$$i - u < 0 \text{ and } m + i - u \geq c\% * p$$

The conditions cover both complete and partial overlaps. One point is worth mentioning here. The definition of motif does not allow any overlap between the subsequences. Whereas the definition of cover does allow $c\%$ overlap.

Definition 4: A motif, ($T_{i,m}$,$T_{j,m}$), of length m *covers* a motif of length $p \leq m$, ($T_{u,p}$,$T_{v,p}$), if any of ($T_{i,m}$,$T_{j,m}$) covers any of ($T_{u,p}$,$T_{v,p}$). A motif is *maximally covering* when no motif covers it.

Problem 1 (Motif Enumeration): Given a time series T, find all the maximally covering motifs.

We would like to solve the above problem *exactly*. Recall, the motivation for finding motifs is to retrieve the most similar segments that, by definiton, are the least noisy segments for further investigation by human. Therefore our algorithm keeps minimizing similarity as the primary goal while finds as many motifs as possible for different lengths.

Note that the definition is free from any parameter. Reader may think c as a parameter of the enumeration algorithm. It is a configuration parameter for the output. We don't use this parameter in speeding up the enumeration algorithm.

A. Why the Problem is Difficult?

Finding motifs for all possible lengths is an inherently difficult problem. Here we describe why. Let's assume we have two time series subsequences $T_{i,5}$ = [3 7 3 5 3] and $T_{k,5}$ = [10 6 9 8 9]. Let's also consider that z-normalization is not required. The distance between the subsequences $T_{i,4}$ and $T_{k,4}$ is $d = \sqrt{7^2 + 1^2 + 6^2 + 3^2}$ = 9.75. Computing the distance between $T_{i,5}$ and $T_{k,5}$ is now a straight forward process, which is adding the squared difference of the fifth elements to d^2. Therefore, $dist(T_{i,5}, T_{k,5}) = \sqrt{d^2 + 6^2} = 11.45$ (see Figure 4(a)).

When we compute z-normalized euclidean distance the situation becomes difficult. The primary reason is that we cannot use the overlap between $T_{i,4}$ and $T_{i,5}$ while computing $dist(T_{i,5}, T_{k,5})$. The normalized version of $T_{i,4}$ and $T_{i,5}$, usually, do not share any common subsequence. In Figure 4(b) the normalized $T_{i,4}$ and $T_{k,4}$ are shown by the dotted lines and the normalized $T_{i,5}$ and $T_{k,5}$ are shown by the solid lines. Clearly, the values of the first four elements change with the length of the subsequences and we cannot just add the squared difference of the fifth elements to get $dist(T_{i,5}, T_{k,5})$. Therefore, both $dist(T_{i,4}, T_{k,4})$ and $dist(T_{i,5}, T_{k,5})$ require adding four and five squared differences, respectively. And more importantly, $dist(T_{i,5}, T_{k,5})$ can be smaller than $dist(T_{i,4}, T_{k,4})$.

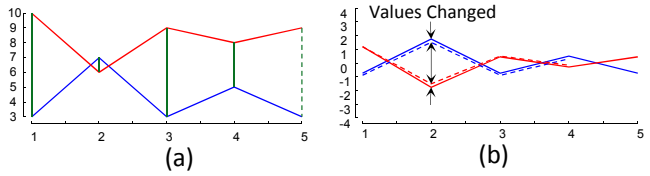

Fig. 4. (a) If we do not require normalization, the distance between slightly longer subsequences can reuse the distance between shorter subsequences. (b) If we normalize, the overlapping values change and we *cannot* reuse the distance.

In this paper, we show a way to use the overlaps between subsequences to create bounds on the distances of the longer subsequences. We use the bounds in speeding up the motif enumeration algorithm.

IV. Motif Enumeration

We start with the smartest trivial algorithm that can be designed using known optimization techniques. We then describe our novel bound on distance function and give a faster algorithm to enumerate motif.

A. Smart Brute Force

A brute force algorithm compares all possible pairs of subsequences of all possible lengths. If the length of the time series is n then there are $O(n^3)$ comparisons each taking $O(n)$ time. Thus a *naive* brute force algorithm takes $O(n^4)$ computation time.

A *smart* brute force algorithm can use the known techniques described in [19]. Precisely, we can cache and reuse the dot products of a subsequence with all other subsequences in the entire time series. It costs only linear space and enables a way to compute distances in constant time. Thus the additional linear space saves us the $O(n)$ factor of computation required for distance computation in the naive brute force.

The Algorithm 1 describes the smart brute force algorithm. It takes the time series T and the minimum (m) and maximum (mx) length of the motifs as input. Note that, the parameters m and mx can easily be set to 0 and $n/2$ to enumerate all motifs of all lengths although it is not meaningful to do so. The parameters are more amenable for exploratory analysis than exactly one length m for the motif. They are not data dependent and can be set flexibly to cover a wide range of meaningful lengths.

The lines 3-5 computes the cumulative sum of the time series values and the square of the values. In lines 6-9 the dot

Algorithm 1 $SmartBruteForce(T, m, mx)$

Require: $T \leftarrow$ a time series, $m, mx \leftarrow$ minimum and maximum length of a time series motif
Ensure: Output motifs for lengths m to mx and return a $List$ of distances computed

1: $n \leftarrow |T|$
2: $x_0 \leftarrow 0, xx_0 \leftarrow 0$
3: **for** $i \leftarrow 1$ to $|T|$ **do**
4: $\quad x_i \leftarrow x_{i-1} + t_i$
5: $\quad xx_i \leftarrow xx_{i-1} + t_i^2$
6: **for** $i \leftarrow 1$ to $n - m$ **do**
7: $\quad xy_i \leftarrow 0$
8: $\quad$ **for** $j \leftarrow 1$ to m **do**
9: $\quad\quad xy_i \leftarrow xy_i + t_{i+j-1}t_j$
10: $prevXY \leftarrow xy$
11: **for** $i \leftarrow 1$ to $n - m$ **do**
12: $\quad$ **if** $i > 1$ **then**
13: $\quad\quad$ **for** $k \leftarrow i + m$ to $n - m$ **do**
14: $\quad\quad\quad xy_k \leftarrow prevXY_{k-1} - t_{k-1}t_{i-1} + t_{k+m-1}t_{i+m-1}$
15: $\quad\quad prevXY \leftarrow xy$
16: $\quad$ **for** $j \leftarrow m$ to mx **do**
17: $\quad\quad$ **for** $k \leftarrow i + j$ to $n - j$ **do**
18: $\quad\quad\quad$ **if** $j > m$ **then**
19: $\quad\quad\quad\quad xy_k \leftarrow t_{k+j-1}t_{i+j-1}$
20: $\quad\quad\quad d \leftarrow ctDistance(T, i, k, j, x, xx, xy)$
21: $\quad\quad\quad List.Add(d, i, k)$
22: $\quad\quad$ Update the best pair for length j if necessary
23: $\quad$ Output the best pair for all the lengths
24: **return** $List$

product of $T_{1,m}$ with $T_{i,m}$ is computed and stored in xy_i. A copy of this dot product array is stored in $prevXY$ to use later in line 10. This dot product could have been computed in $O(n \log n)$ time using fast Fourier transform. We skip this efficient method for simplicity and it doesn't hurt the overall complexity of the algorithm.

Line 11,16,17 shows the main three loops of the algorithm. Logically, the loop at line 17 finds the nearest neighbor of $T_{i,j}$. Lines 12-15 computes the dot products for $T_{i,j}$ given we have the dot products for $T_{i-1,j}$ in $prevXY$ and store that back in $prevXY$ for future use (line 15). Lines 18-19 compute the dot products for the next length i.e. from $T_{i,j}$ to $T_{i,j+1}$. In aggregate, lines 12-15 and 18-19 maintain the xy array for constant time distance computation and it is done at line 20. Note that, all the arrays are of the same size as the time series. Line 21 adds the distance values of the pairs in a list to return. This is not a mandatory statement for the smart brute force algorithm. We use the $List$ data structure later as we will be using the Algorithm 1 as a subroutine in our proposed algorithm.

At line 22, after the nearest neighbor of $T_{i,j}$ is found, we can update the best pair for the length of j. And finally when all of the loops are done, we can output the bests for all lengths.

B. MOEN: Efficient Enumeration of Motifs

The smartest brute force algorithm for motif enumeration described in the previous section runs in $O(n^3)$ time. In this section we describe an order of magnitude faster algorithm. The algorithm is called **MOEN** (i.e. **MO**tif **EN**umerator). The algorithm is based on a novel bound on normalized Euclidean distance for longer subsequences.

Algorithm 2 $ctDistance(T, i, k, j, x, xx, xy)$

Ensure: Return the distance between $T_{i,j}$ and $T_{k,j}$

1: $sumX \leftarrow x_{i+j-1} - x_{i-1}$
2: $sumX2 \leftarrow xx_{i+j-1} - xx_{i-1}$
3: $sumY \leftarrow x_{k+j-1} - x_{k-1}$
4: $sumY2 \leftarrow xx_{k+j-1} - xx_{k-1}$
5: $\mu_{i,j} \leftarrow sumX/j$
6: $\mu_{k,j} \leftarrow sumY/j$
7: $\sigma_{i,j} \leftarrow \sqrt{sumX2/j - \mu_{i,j}^2}$
8: $\sigma_{k,j} \leftarrow \sqrt{sumY2/j - \mu_{k,j}^2}$
9: $C \leftarrow (xy_k - (j * \mu_{i,j} * \mu_{k,j}))/(j * \sigma_{i,j} * \sigma_{k,j})$
10: $d \leftarrow \sqrt{2 * j * (1 - C)}$

*1) **Bounding Distances for Longer Lengths**:* We gave an example in Section III-A to demonstrate the difficulty of using the overlap between successive subsequences while computing the distance metric. Since the values of the overlapping portion change after normalization we cannot reuse computation directly. Instead we can build bounds for distances of the longer subsequences using the distances between shorter subsequences. More formally, we want to find upper and lower bound of the distance between $T_{i,j}$ and $T_{k,j}$ given that $d = dist(T_{i,j-1}, T_{k,j-1})$. We use the lower bounds to speed-up the algorithm. We don't use the upper bounds, yet we show the derivation for completeness.

Let's assume x and y are two normalized sequences of length $j - 1$ where $x_r = (t_{i+r} - \mu_{i,j-1})/\sigma_{i,j-1}$ and $y_r = (t_{k+r} - \mu_{k,j-1})/\sigma_{k,j-1}$ for $r = 0, 1, \ldots, j - 2$. We know the following tautologies.

$$\sum_{r=0}^{j-2} x_r = \sum_{r=0}^{j-2} y_r = 0$$
$$\sum_{r=0}^{j-2} x_r^2 = \sum_{r=0}^{j-2} y_r^2 = j - 1$$

Let the distance between $T_{i,j}$ and $T_{k,j}$ be d_{next}. We would like to append the values of t_{i+j-1} and t_{k+j-1} to the end of x and y. For that we have to first normalize them using the means and variances of x and y. Therefore we must add $x_{j-1} = (t_{i+j-1} - \mu_{i,j-1})/\sigma_{i,j-1}$ and $y_{j-1} = (t_{k+j-1} - \mu_{k,j-1})/\sigma_{k,j-1}$ and let the new subsequences after appending x_{j-1} and y_{j-1} are $\hat{x}$ and $\hat{y}$. The new mean and variance of $\hat{x}$, after the increase in length, can be computed as follows. The mean $\hat{\mu}_y$ and variance $\hat{\sigma}_y^2$ can also be computed in the same manner.

$$\hat{\mu}_x = \tfrac{1}{j}\sum_{r=0}^{j-1} x_r = \tfrac{1}{j}\sum_{r=0}^{j-2} x_r + x_{j-1} = \tfrac{x_{j-1}}{j}$$

$$\begin{aligned}\hat{\sigma}_x^2 &= \tfrac{1}{j}\sum_{r=0}^{j-1} x_r^2 - (\tfrac{1}{j}\sum_{r=0}^{j-1} x_r)^2 \\ &= \tfrac{1}{j}(\sum_{r=0}^{j-2} x_r^2 + x_{j-1}^2) - \tfrac{1}{j^2}(\sum_{r=0}^{j-2} x_r + x_{j-1})^2 \\ &= \tfrac{j-1}{j} + \tfrac{1}{j}x_{j-1}^2 - \tfrac{1}{j^2}x_{j-1}^2 \\ &= \tfrac{j-1}{j} + \tfrac{j-1}{j^2}x_{j-1}^2\end{aligned}$$

Thus the distance d_{next} can be computed by computing the sum of squared error of the new normalized subsequences.

$$d_{next} = dist(\hat{x}, \hat{y}) = \sum_{r=0}^{j-1} ((x_r - \hat{\mu}_x)/\hat{\sigma_x} - (y_r - \hat{\mu}_y)/\hat{\sigma}_y)^2$$

At this point, we are in position to define the lower and upper bounds for d_{next}.

Lower Bound: We start with the lower bound. Imagine we do not know the values of t_{i+j-1} and t_{k+j-1}. We would like to choose the values of x_{j-1} and y_{j-1} as such we achieve the smallest possible d_{next}.

We argue that we achieve a lower bound for d_{next} if we set $z = x_{j-1} = y_{j-1}$. Let's verify why it is true. If $z = x_{j-1} = y_{j-1}$, then means and variances of $\hat{x}$ and $\hat{y}$ become equal.

$$\hat{\mu}_x = \hat{\mu}_y = \frac{z}{j}$$
$$\hat{\sigma}_x^2 = \hat{\sigma}_y^2 = \frac{j-1}{j} + \frac{j-1}{j^2} z^2$$

Clearly, the contribution of x_{j-1} and y_{j-1} to d_{next} becomes empty. Depending on the value of z the contribution of the first $j-1$ values also change. The larger the value of z, the larger the means and variances are and the smaller the values of $(x_r - \hat{\mu}_x)/\hat{\sigma_x}$ are for $r = 0, 1, \ldots, j-2$. If we set z to the maximum value of t_i normalized by the means and standard deviations of the immediately preceding subsequence of length $j-1$, we are guaranteed to have the smallest value for d_{next} (see Figure 5).

More formally, we get the lower bound d_{LB} for d_{next} when we set the following.

$$z = x_{j-1} = y_{j-1} = \max_i(t_i - \mu_{i-j+1,j-1})/\sigma_{i-j+1,j-1}$$

The lower bound can then be computed as below.

$$\begin{aligned} d_{LB}^2 &= \sum_{r=0}^{j-1} ((x_r - \hat{\mu}_x)/\hat{\sigma_x} - (y_r - \hat{\mu}_y)/\hat{\sigma}_y)^2 \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-1} ((x_r - y_r) - (\hat{\mu}_x - \hat{\mu}_y))^2 \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-1} (x_r - y_r)^2 \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-2} (x_r - y_r)^2 + (x_{j-1} - y_{j-1})^2 \\ &= (\frac{j-1}{j} + \frac{j-1}{j^2} z^2)^{-1} d^2 \end{aligned}$$

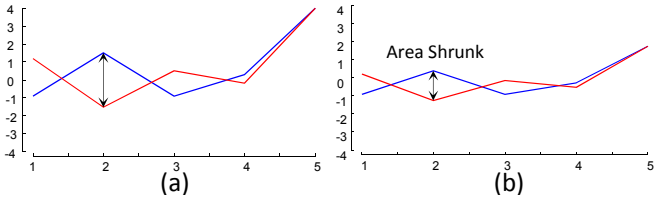

Fig. 5. (a) Before and (b) After re-normalization when the maximum value is appended to a pair of subsequences. Note that the scales in the y-axes are aligned and the area in between the subsequences shrunk as we re-normalize the subsequences.

The bounds use the maximum value of a sample (i.e. z) which can be efficiently computed from the time series and the algorithm for that is described in the next section.

Upper Bound: The upper bound d_{UB} can also be computed as above except we set the following to ensure that x_{j-1} and y_{j-1} are opposite to each other.

$$z = x_{j-1} = -y_{j-1} = \max_i(t_i - \mu_{i-j+1,j-1})/\sigma_{i-j+1,j-1}$$

Note that, the above assignment doesn't change the variances while the means have opposite signs. The upper bound can then be computed as below.

$$\begin{aligned} d_{UB}^2 &= \sum_{r=0}^{j-1} ((x_r - \hat{\mu}_x)/\hat{\sigma_x} - (y_r - \hat{\mu}_y)/\hat{\sigma}_y)^2 \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-1} ((x_r - y_r) - (\hat{\mu}_x - \hat{\mu}_y))^2 \\ &\leq \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-1} ((x_r - y_r) + 2\frac{z}{j})^2 \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-1} ((x_r - y_r)^2 + 2(x_r - y_r)2\frac{z}{j} + 4\frac{z^2}{j^2}) \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-1} ((x_r - y_r)^2 + 8\frac{z^2}{j} + 4\frac{z^2}{j^2}) \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-2} (x_r - y_r)^2 + (x_{j-1} - y_{j-1})^2 + 8\frac{z^2}{j} + 4\frac{z^2}{j^2}) \\ &= \frac{1}{\hat{\sigma}_x^2} \sum_{r=0}^{j-2} (x_r - y_r)^2 + 4z^2 + 8\frac{z^2}{j} + 4\frac{z^2}{j^2}) \\ &= (\frac{j-1}{j} + \frac{j-1}{j^2} z^2)^{-1} (d^2 + 4z^2(1 + \frac{1}{j})^2) \end{aligned}$$

Example: Let's take the example of Section III-A. $T_{i,5}$ = [3 7 3 5 3] and $T_{k,5}$ = [10 6 9 8 9]. We know the normalized distance between $T_{i,4}$ = [3 7 3 5] and $T_{k,4}$ = [10 6 9 8] which is 3.97 (rounded for simplicity). The normalized sequences are x_r = [-0.90 1.51 -0.90 0.30] and y_r = 1.183 -1.52 0.51 -0.17]. Let's assume we have computed the maximum normalized value for z = 4;

Then the hypothetical sequences are x_r = [-0.90 1.51 -0.90 0.30 4] and y_r = 1.183 -1.52 0.51 -0.17 4]. The means and variances are $\hat{\mu}_x = \hat{\mu}_y = 0.8$ and $\hat{\sigma}_x^2 = \hat{\sigma}_y^2 = 3.36$. Therefore $d_{LB} = \frac{d}{\hat{\sigma}_x} = 2.16$ and d_{UB} is 5.667. Since d_{next} = 4.43, the bounding inequalities $d_{LB} \leq d_{next} \leq d_{UB}$ hold.

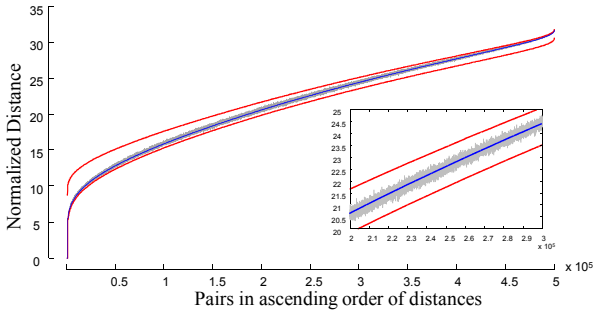

Fig. 6. The pairwise distances of 1000 random walks of length 255 in ascending order shown in blue. After increasing the lengths of the random walks by one, the bounds are shown in red and the true distances are shown in gray. A zoomed-in segment is shown in the inset.

Before we end the section, we demonstrate the goodness of the bounds. We perform an experiment on 1000 random walks of lengths 255. We compute the normalized distances between all possible pairs of the random walks and plot them in ascending order in Figure 6. Then we increase the length of the random walks by one and compute the d_{LB}, d_{UB} and the d_{next} for all possible pairs. We plot the them together to see the tightness of the bound.

As depicted in the Figure 6, the bounds are really tight and the biggest advantage is that the bounds are *constant factors* away from the distances at length 255. Therefore the orders of the pairs are the same. More formally, if $d(x, y) \leq d(p, q)$ then $d_{LB}(x, y) \leq d_{LB}(p, q)$. We use this fact in our algorithm rigorously.

Algorithm 3 $ComputeMax(T, m, mx)$

Require: $T \leftarrow$ a time series, $m, mx \leftarrow$ minimum and maximum length of a time series motif

Ensure: Return the maximums of the absolute values for every length after normalization by the immediate preceding subsequence

1: **for** $j \leftarrow 0$ to mx **do**
2: $\quad Max_j \leftarrow 0$
3: **for** $j \leftarrow m+1$ to mx **do**
4: $\quad$ **for** $i \leftarrow 1$ to $n-j$ **do**
5: $\quad\quad Y \leftarrow (t_{i+j-1} - \mu_{i,j-1})/\sigma_{i,j-1}$
6: $\quad\quad$ **if** $|Y| > Max_j$ **then**
7: $\quad\quad\quad Max_j \leftarrow |Y|$
8: **return** Max

*2) **Computing the** Max **Array**:* As described in the previous subsection, we use the maximum value of a sample, t_i, relative to its immediately preceding subsequence while bounding the Euclidean distance. In effect, we want to compute an array Max where

$$Max_j \leftarrow \max_i(t_i - \mu_{i-j+1,j-1})/\sigma_{i-j+1,j-1} \text{ for } j = m \text{ to } mx$$

Algorithm 3 describes how to compute the Max array. The algorithm assumes the sum of values and sum of squared values are accessible and therefore, means and variances of arbitrary subsequences are ready to use. For every length j, the algorithm loops through every position of the time series and finds the one which is the most displaced form the mean of the immediately preceding subsequence of length $j-1$. The process takes $O(nm)$ time and has no impact on the end-to-end complexity of the algorithm.

*3) **Computing the Maximal Motif**:* In the last two sections we show the ways to bound the distances of longer subsequences using the distances of the shorter ones in constant time by using only one factor. In this section, we use the lower bounds to speed-up the motif enumeration algorithm. The algorithm is shown in Algorithm 4.

The algorithm first computes the Max array in line 1 and then computes the motif pair and a sorted list of pairs in ascending order of the distances in line 2. Note the call to the Algorithm 1 by passing m as both the minimum and maximum length to search. Thus the brute force algorithm runs once for the length m and in return, sends a list of triplets $< dist, i, k >$ sorted by the $dist$ field in ascending order.

If we wanted to return all the pairwise distances in the sorted list, it would have required quadratic space. Instead, we use only linear space to store the best $O(n)$ pairwise distances. The key idea is that there can be $O(n)$ maximally covering motifs among the $O(n^3)$ possible pairs and the covering motifs roughly remain the same for segments of lengths. In our implementation, we store exactly n pairs in the $List$.

After the list is generated, the algorithm takes the maximum distance in the list and computes the lower bound by multiplying the magic constant. This lower bound represents the whole set of pairs that we *did* not keep in the $List$ because *none* of those pairs could have a smaller distance than LB for length $m+1$. Thus LB helps the algorithm to confine the search space for the next motif pair of length $m+1$ within the pairs in the $List$.

Let's now look at the behavior of the algorithm for $j = m+1$ at line 5. The algorithm loops through the pairs in the $List$ and compute the distances of all the pairs in it at line 8. The $distance$ function is not a constant time function as the $ctDistance$ used in Algorithm 1. It normalizes the sequences and computes the distance and therefore, requires at least one full scan of the sequences. By using more space and caching techniques it can be optimized but, not required.

When the loop at line 6 is finished execution, we have all the new distances for the pairs in the $List$ stored in the $NewList$ and at line 9 we find the minimum of the $NewList$ to get the $best$ pair of length $m+1$ among the $O(n)$ pairs of length m from $List$.

At line 11, the algorithm makes a key decision. If the $best$ distance is less than the LB then we know that $none$ of the pairs that are not in the list can be smaller than the $best$ distance and therefore, the algorithm has found the motif for length $m+1$. The algorithm updates the LB for the next iteration at line 13. Since the lower bounds are constant factor approximation of the truth, we can cascade the factor to find the LB for length $j+1 = m+2$. The hypothesis remains the same for the next iteration: *none* of the skipped pairs can have a smaller distance than LB for length $j+1$. Figure **??**(a) shows how the cascade drops the the LB down to smaller values.

The algorithm stores the locations of the motif pair in two parallel arrays $L1$ and $L2$ at line 14. These arrays will be used to find the maximally covering set. The algorithm outputs the best pair for the current iteration at line 15.

As the LB gets smaller and goes below $best$ for some length j, LB loses its pruning power. At this point, the algorithm is no more confirm about finding the smallest distance for length $j+1$ and therefore, calls the $SmartBruteForce$ algorithm for length j. Thereafter, the algorithm has a new LB (line 18) with a new $List$ (line 17) of pairs that have more pruning power and continues for $j+1$.

One minor implementation detail is worth mentioning here. The $List$ is best implemented as a *max-heap* when we insert and find the maximum in the $SmartBruteForce$ algorithm. In the $MOEN$ algorithm we iterate and find the minimum in the $List$ and therefore, it is best implemented as a *sorted array*. To facilitate the implementation, we change the data structure of the $List$ in $MOEN$ algorithm after lines 2 and 17 from max-heap to sorted array.

*4) **Finding the Covering Motifs**:* In Algorithm 6, we describe how to find the set of covering motifs given the locations of the motifs for the whole range of lengths. The algorithm uses the fact that $L1_i$ is always less than $L2_i$ and it is ensured by both $SmartBruteForce$ and $MOEN$. The algorithm starts with the smallest length. For each length i, it searches the motifs of length $i+1$ to mx to see if any motif covers the ith one. Line 4 calls the $isCovering$ function described in the Algorithm 5 which returns true if the longer subsequence covers the shorter. Whenever a motif is covered, the algorithm *marks* it. At the end, the motifs without mark are collected as the covering motifs.

Algorithm 4 $MOEN(T, m, mx)$

Require: $T \leftarrow$ a time series, $m, mx \leftarrow$ minimum and maximum length of a time series motif
Ensure: Output all the motifs and $L1, L2 \leftarrow$ Locations of the motif pairs for all lengths

1: $Max \leftarrow ComputeMax(T, m, m_x)$
2: $List \leftarrow SmartBruteForce(T, m, m)$
3: $z \leftarrow Max_{m+1}$
4: $LB^2 \leftarrow List.Max^2 * (\frac{m+1}{m} + \frac{m}{(m+1)^2} z^2)^{-1}$
5: **for** $j \leftarrow m+1$ to mx **do**
6: **for** $p \leftarrow 1$ to $List.Count$ **do**
7: $i \leftarrow List(P).i, k \leftarrow List(P).k$
8: $d \leftarrow distance(T_{i,j}, T_{k,j})$
9: $NewList.Add(d, i, k)$
10: $best \leftarrow NewList.Min$
11: **if** $best \leq LB$ **then**
12: $z \leftarrow Max_j$
13: $LB^2 \leftarrow LB^2 * (\frac{j}{j-1} + \frac{(j-1)}{j^2} z^2)^{-1}$
14: $L1_j \leftarrow i$, $L2_j \leftarrow k$
15: output $List.Min$ for length j
16: **else**
17: $List \leftarrow SmartBruteForce(T, j, j)$
18: $LB^2 \leftarrow List.Max^2 * (\frac{j}{j-1} + \frac{(j-1)}{j^2} z^2)^{-1}$

Algorithm 5 $isCovering(i, m, u, p)$

Require: i, m and u, p for two subsequences

1: **if** $i - u \geq 0$ *and* $i - u \leq (1 - c\%) * p$ **then**
2: **return true**
3: **else if** $i - u < 0$ *and* $m + i - u \geq c\% * p$ **then**
4: **return true**
5: **else**
6: **return false**

*5) **Enumerating More Motifs**:* It is possible that we find only one motif that covers all the other motifs of smaller lengths. The reason is that the longer motif is too similar to each other and its shorter versions become motifs for smaller lengths also. If we want more motifs, an obvious solution is to remove the maximally covering motif from the data and run the $MOEN$ algorithm again to find more motif. A more interesting approach would be to generate more motifs without using additional space and time. We can maintain more (e.g. K) pairs for every length and thus, increase the size of the enumeration output. K motif has been studied previously in [8][24] for a given length. We use them here for enumeration purposes.

Definition 5: A *K-motif* is the K-most similar pairs of non-over-lapping subsequences of length m such that none of the K pairs cover each other.

It is easy to find the K-motifs given the $List$ data structure. We do it with a linear scan. Let's look at an example for a length m. Imagine we have the pairs of subsequences in the ascending order of the distances stored in the $List$ at line 2 of the Algorithm 4. Recall they are all of the same length. Imagine $List(1)$ covers $List(2)$, $List(2)$ covers $List(3)$, $List(3)$ covers $List(4)$ and none covers $List(5)$. Then 3-motif for length m consists of $List(1)$, $List(3)$ and $List(5)$. We skip the description of the algorithm for lack of space.

Algorithm 6 $CoveringMotifs(L1, L2)$

Require: $L1, L2 \leftarrow$ Locations of the motif pairs for all lengths
Ensure: Output the covering motif pairs marked

1: **for** $i \leftarrow m$ to mx **do**
2: $Mark_i \leftarrow 0$
3: **for** $j \leftarrow i+1$ to mx **do**
4: **if** $\begin{pmatrix} isCovering(L1_j, j, L1_i, i) \text{ or} \\ isCovering(L2_j, j, L1_i, i) \text{ or} \\ isCovering(L1_j, j, L2_i, i) \text{ or} \\ isCovering(L2_j, j, L2_i, i) \end{pmatrix}$ **then**
5: $Mark_i \leftarrow 1$
6: **break**
7: **return** $Mark$

As we find the K-motifs for each length, we can use the same definitions of cover and maximally covering motifs as in Section III to define K-motif enumeration problem. Note that the definition of motif in Section III is for 1-motif.

Problem 2 (K-Motif Enumeration): Given a time series T, find the maximally covering K-motifs.

Now that we have defined the K-motif enumeration problem, we show the changes required in the $MOEN$ algorithm. At line 2, after getting the $List$ the algorithm finds the K-motif of length m and stores them in $L1$ and $L2$. Then it enters the loops and at line 10, instead of the Min, the algorithm uses the distance of the K-th motif. This guarantees that LB is good for the K-motif enumeration problem.

Since $L1$ and $L2$ are now having multiple motifs of the same length, there is a change needed in the $CoveringMotifs$ algorithm. We have to add a loop to *look for* the covering motif for each of the K motifs of length i. We also need to add loop to *try* each of the K motifs of length j to see if it covers the motif of length i. The addition of these loops doesn't change the overall complexity of the algorithm as they run after $MOEN$ outputs.

Note that the value of K doesn't necessarily guarantee there will be at least K motifs enumerated unless $c = 0$. By changing K and c, we can only change the number of output motifs flexibly without taking additional runs.

V. Experiments

We have evaluated the $MOEN$ algorithm experimentally. We implemented the algorithm in C# language and run our experiment on a commodity desktop computer. We argue that the experiments are absolutely reproducible given the detailed description in the paper and the free datasets used. The code, the slides and raw numbers of the experiments of this paper are available at [2].

We start with a sanity check by a "planted motif" problem. We then show the scalability experiments.

A. The Planted Motif

We design a "planted motif" problem for motif enumeration. We insert three patterns; a sinusoid wave, a square wave and a sawtooth wave into a sequence of white noise having a ultra low frequency trend. We insert two copies of the same pattern with different scaling and noise addition. The resulting time series is shown in Figure 7(a). The enumerator is expected to find all of the three waves.

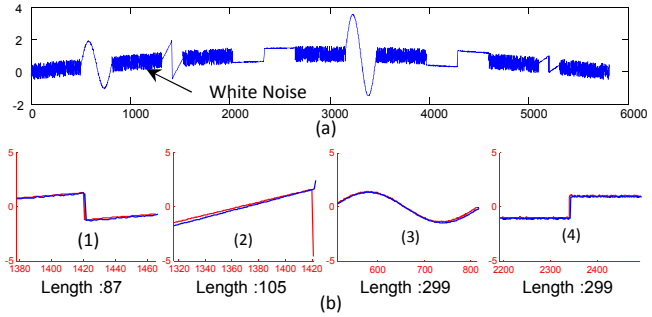

Fig. 7. (a) The time series after plantation. A white noise series is used as the ground. We plant three waveforms of different lengths, scales and noise content. (b) The motifs we found.

In our experiment we indeed find the patterns. The success in this experiment strongly depends on the scale of the waves, the wandering baseline and the scale of the added noise. For example, the plantation shown in Figure 7(a) is a difficult one and the 1-motif enumerator fails to identify all of the three waves. More precisely, 1-motif finds only motif 2 and 3 shown in Figure 7(b).

At this point we employ our K-motif enumeration algorithm and use K = 3 and successfully discover the four waves shown in Figure 7(b). Note that the segments in motif 1 and 2 have $\frac{1420-1378}{87}$ = 48.27% overlap which is less than the $c = 80\%$ threshold set for the experiment.

B. Scalability

We use three datasets to compare the scalability against two algorithms. **EEG**: An EEG trace of 180,000 samples [13]. **Random Walk**: A synthetic random walk data to model financial time series. **EOG**: An Electro-Oculogram of 8 million samples [6].

We use the $SmartBruteForce$ algorithm and $IterativeMK$ algorithm to compare with the $MOEN$ algorithm. $IterativeMK$ repeatedly uses the existing exact algorithm (MK) [13] for finding motifs of all lengths. MK algorithm uses pruning techniques after dimensionality reduction to reduce the number of distance computations and guarantees no false negatives. We choose *iterative* MK because, it has been used in several works [3][13][4][20] to iteratively enumerate motifs.

Note that, $MOEN$ neither uses any dimensionality reduction technique nor it uses quadratic space. And more importantly, there is an *opportunity* to use both of these techniques in $MOEN$ for further improvement. We can easily optimize the periodic calls to the $SmartBruteForce$ algorithm in $MOEN$ by these techniques for more speedup. We leave such improvement for future work.

Increasing the Data Size: Our first experiment is to empirically show the growth of the execution time as the size of the datasets grow. In Figure 8, we show the execution time of MOEN for the three datasets mentioned above. Since the smart brute force algorithm takes identical time for all of the datasets, we show the curve once. We show the best timing for $IterativeMK$. As claimed, we see an order of magnitude speedup for all of the datasets from both of the $SmartBruteForce$ and $IterativeMK$.

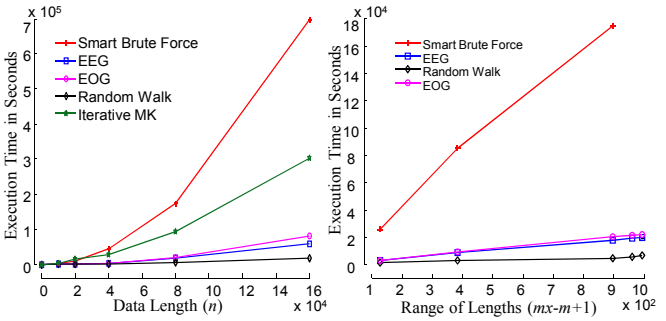

Fig. 8. Execution times of 1-motif, c = 80%, (left) for different size of the datasets for the range 128 to 256 and (right) for different range of lengths for n = 180k.

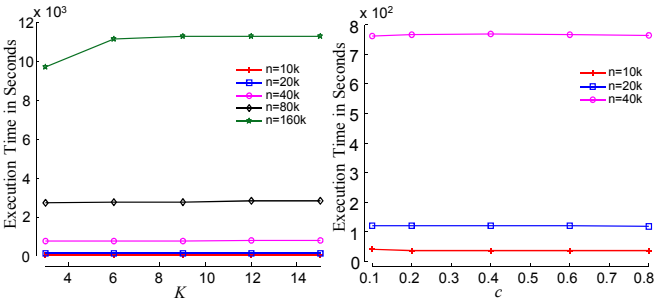

Fig. 9. Execution times for (left) varying K, c = 80% and (right) varying c, K = 15. Here the range is 128 to 256.

Increasing the Range of Lengths: We experiment on the effects of larger range of lengths. The relationship is linear as expected from the complexity expression and holds for all the datasets and for both of the algorithms.

Changing K and c: K and c are two parameters to control the number of outputs of the enumerator without additional running time. The larger the K or the smaller the c, the more the number of motifs are while the execution time does not change significantly as we go to extreme values of K (=200) and c (=0).

We experiment to see the change in execution time for increasing K and decreasing c. As shown in Figure 9, the execution time remains flat for larger/smaller values of K/c which confirms the algorithm is independent of the K/c.

VI. Case Studies

The $MOEN$ algorithm is directly motivated from real scientific applications [3]. The case studies shown here, are not designed to claim results in their respective domains, rather, they demonstrate how the enumeration of time series motifs across lengths can provide more *precision* in discovering repetitions. We use $MOEN$ on three different signals; Accelerometer, Sound pressure and EPG (Electrical Penetration Graph).

A. Activity Recognition: Sound Pressure

In this case study we use one signal (Environment:SoundPressure:Modest) from a benchmark dataset of context recognition [11][1]. There are 5 activity scenarios performed by two users. Each scenario has around 50 trace instances performed by the two users. An example scenario is given in TABLE I. All the instances are randomly concatenated to form a time series of length 46,045. A segment of one such concatenation is shown in Figure 10(a).

Our goal is to discover motif in this randomly concatenated time series and see if the motifs come from the same the scenarios. This task is inherently difficult because of two reasons. First, we are using only one signal out of the 28 available signals in the dataset. Second, the scenarios have many common activities (such as walking in the street) to create confusion.

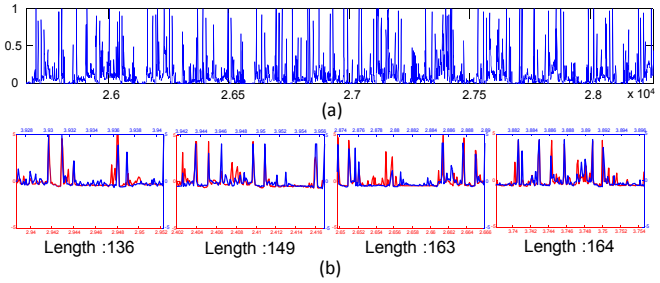

Fig. 10. (a) Activity scenarios. (b) Four motifs found in the sound pressure signal.

We use $MOEN$ to find repetitions of length 128 to 256. We performed ten different random concatenations and find twelve motifs on average with 88.15% accuracy in finding motifs from identical scenarios. Figure 10(b) shows four of the motifs found. We further perform a similarity search for all of these twelve motifs. We find 34 other occurrences with a correlation threshold of 0.55 that yield an accuracy of 91.28% of being generated from identical scenarios.

B. Activity Recognition: Accelerometer

In this case study, we have used a set of accelerometer signals from [18]. In the original data, there were four participants who danced with a remix of Lady Gaga's song "Just Dance." The song has 119 beats per minute and it is 4:54 minutes long. The participants were equipped with four accelerometer devices (with x, y and z axis in each) in four of their body parts; hand, arm, hip and leg. The sampling rate of the accelerometer was good enough to collect at about 10Hz rate and we received a set of time series of length 29,700.

TABLE I
A Scenario

Activity	Location
start	office
walking	corridor
walking	stairs
walking	lobby
walking	street
walking	lobby
walking	corridor
walking	stairs
stop	office

TABLE II
Dance Steps

Step	Action
A	Side steps with no arm movement
B	Rock steps sideways without arm movement
C	Rock steps sideways with arm movement
D	Side steps with arm movement
E	Side steps with arms up in the air
F	Standing still with head bopping

Six different dance-steps (shown in TABLE II) were defined and assembled into a choreography for the study (shown in the top of Figure 11).

Although there are six different dance-steps, individual body parts have two or three distinct movements. We show the motion transitions of the body parts in the Figure 11 by thick waveforms.

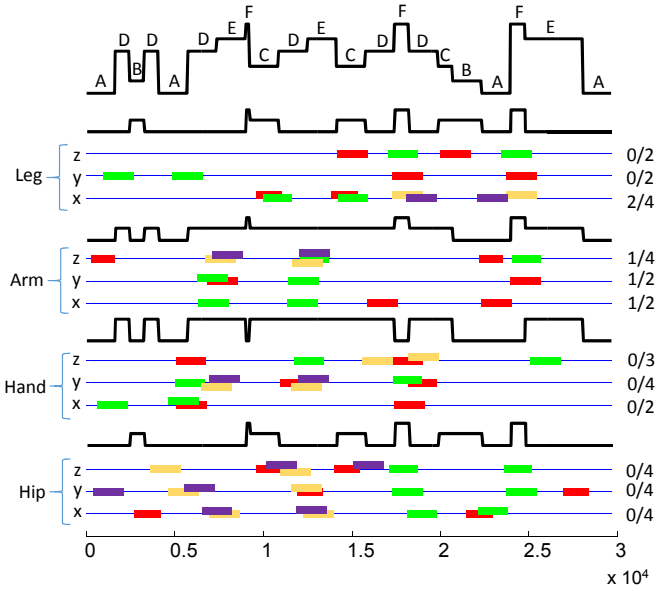

Fig. 11. The motifs found in the twelve accelerometer signals. Same colored bars represents sequences of a motif pair. The four thick waveforms show the state transitions of each of the body parts according to the choreography. The number of motif-pairs spanning different motions are counted on the right side of each axis line.

Our goal is to enumerate a set of maximally covering motifs for a range of lengths and verify if the motifs align with identical dance steps or step-transitions. We use our algorithm to find 3-motifs for all the twelve accelerometer signals of *one* participant. In Figure 11, we show all of the motifs aligned to the known choreography. The motifs are grouped by the body-parts carrying the sensors and acceleration axes are marked on the left. For example, the green motif in the x-axis of the hand are aligned with A to D transitions in the choreography. Similarly, the red motif in the z-axis of the leg is aligned with the motion "Rock steps sideways" although the dance-steps are

different (i.e. C and B). There are 37 motifs detected which covers almost the entire song and only 5 of them span different motions of their respective body parts. This corresponds to about 86.5% accuracy.

C. Data Understanding: EPG

We use the EPG trace of the beet leafhopper [13] to enumerate motifs. The motifs found are shown in Figure 12.

There are roughly two distinct shapes in the motifs. The noise pattern (N) and the spikes (S) as shown in the figure. We can use the pattern identifiers to describe the motifs. For example, the motif 334 can be written as NSN. Similarly the motif 384 can be written as NNS. As we go on with the next motifs, we find other valid combinations of the two distinct shapes. For example, the 799 motifs have NSNNSNN and SSSNSSN patterns.

Entomologists confirm us the patterns N represent certain excretion process of the insect [13]. With the longer motifs, we now know that three successive excretions can happen. Similarly, the motif 595 evidently says that two spikes can happen successively. Thus enumerating more motifs of larger lengths gives us more information about the possible arrangements of the patterns and helps understanding the sequential structure of the data.

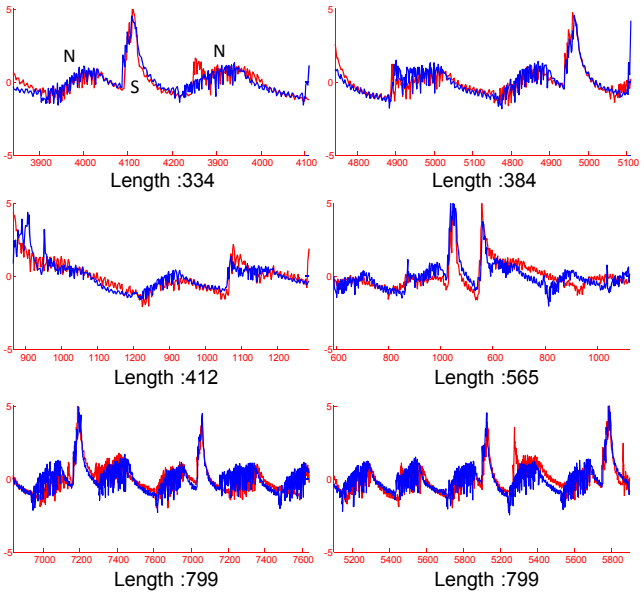

Fig. 12. Six motifs of different lengths found in the EPG trace. The motifs reveal a hierarchy of structure.

VII. Conclusion

Enumerating motifs of a large unstructured time series can potentially retrieve valuable structural information about the data. In this paper, we have described the first exact algorithm to find maximally covering motifs of all lengths. The algorithm is upto 23x faster than the naive method and produces the same results. We have empirically verified the speedup and showed cases of motif discovery in entomology and activity recognition. In future, we plan to investigate new time series data using our enumerator including system performance counters and resource usages.

References

[1] Webpage to download the benchmark dataset for context recognition. `http://cis.legacy.ics.tkk.fi/jhimberg/contextdata/scenarios.html`.

[2] Webpage to download the code, dataset, slides, excel sheet and the paper itself. `http://www.cs.unm.edu/~mueen/Projects/MOEN/index.html`.

[3] A. E. X. Brown, E. I. Yemini, L. J. Grundy, T. Jucikas, and W. R. Schafer. A dictionary of behavioral motifs reveals clusters of genes affecting caenorhabditis elegans locomotion. *Proceedings of the National Academy of Sciences*, 110(2):791–796, 2013.

[4] C. Cassisi, M. Aliotta, A. Cannata, P. Montalto, D. Patan, A. Pulvirenti, and L. Spampinato. Motif discovery on seismic amplitude time series: The case study of mt etna 2011 eruptive activity. *Pure and Applied Geophysics*, pages 1–17, 2012.

[5] N. Castro and P. J. Azevedo. Multiresolution motif discovery in time series. In *SDM*, pages 665–676, 2010.

[6] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet : Components of a new research resource for complex physiologic signals. *Circulation*, 101(23):e215–e220, 2000.

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

[8] H. T. Lam, N. D. Pham, and T. Calders. Online discovery of top-k similar motifs in time series data. *SIAM Conference on Data Mining, SDM '11*, 2011.

[9] Y. Li, J. Lin, and T. Oates. Visualizing variable-length time series motifs. pages 895–906, 2012.

[10] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *Proc. of 2nd Workshop on Temporal Data Mining at KDD*, pages 53–68, 2002.

[11] J. Mäntyjärvi, J. Himberg, P. Kangas, U. Tuomela, and P. Huuskonen. Sensor signal data set for exploring context recognition of mobile devices. In *Workshop "Benchmarks and a database for context recognition" in conjuction with the 2nd Int. Conf. on Pervasive Computing (PERVASIVE 2004)*, 2004.

[12] A. Mueen and E. J. Keogh. Online discovery and maintenance of time series motifs. In *KDD*, pages 1089–1098, 2010.

[13] A. Mueen, E. J. Keogh, Q. Z. 0002, S. Cash, and M. B. Westover. Exact discovery of time series motifs. In *SDM*, pages 473–484, 2009.

[14] A. Mueen, S. Nath, and J. Liu. Fast approximate correlation for massive time-series data. In *SIGMOD Conference*, pages 171–182, 2010.

[15] A. Narang and S. Bhattcherjee. Real-time approximate range motif discovery & data redundancy removal algorithm. EDBT/ICDT '11, pages 485–496, 2011.

[16] P. Nunthanid, V. Niennattrakul, and C. Ratanamahatana. Discovery of variable length time series motif. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2011 8th International Conference on*, pages 472 –475, 2011.

[17] P. Patel, E. Keogh, J. Lin, and S. Lonardi. Mining motifs in massive time series databases. In *Proceedings of the IEEE International Conference on Data Mining*, ICDM, 2002.

[18] H. Pohl and A. Hadjakos. Dance pattern recognition using dynamic time warping. In *SMC 2010 Proceedings*, pages 183–190, 2010.

[19] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. KDD '12, pages 262–270, 2012.

[20] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans. Time series epenthesis: Clustering time series streams requires ignoring some data. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining*, ICDM '11, pages 547–556, 2011.

[21] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD Conference*, pages 599–610, 2005.

[22] Y. Tanaka, K. Iwamoto, and K. Uehara. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Mach. Learn.*, 58:269–300, 2005.

[23] H. Tang and S. S. Liao. Discovering original motifs with different lengths from time series. *Know.-Based Syst.*, 21:666–671, 2008.

[24] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, pages 844–853, 2007.