Online Discovery of Top-k Similar Motifs in Time Series Data

Hoang Thanh Lam[1], Ninh Dang Pham[2] and Toon Calders[1]

[1] Department of Math. and Computer Science

TU Eindhoven The Netherlands {t.l.hoang,t.calders}@tue.nl

[2] University of Technology Ho Chi Minh City, Vietnam, ninhpham@contentinterface.com

Abstract

A motif is a pair of non-overlapping sequences with very similar shapes in a time series. We study the online *top-k* most similar motif discovery problem. A special case of this problem corresponding to $k = 1$ was investigated in the literature by *Mueen and Keogh* [2]. We generalize the problem to any k and propose space-efficient algorithms for solving it. We show that our algorithms are optimal in term of space. In the particular case when $k = 1$, our algorithms achieve better performance both in terms of space and time consumption than the algorithm of *Mueen and Keogh*. We demonstrate our results by both theoretical analysis and extensive experiments with both synthetic and real-life data. We also show possible application of the *top-k* similar motifs discovery problem.

1 Introduction

Usually, time series data is available in abundance because they are easy to collect from many sources. For instance, a set of sensors mounted on a single bridge in a city in the Netherlands produces GBs of data per day [1]. Online managing such amounts of data is very challenging. Typically, such data is processed online either at the distributed devices themselves, or at a designated and centralized system. In both cases, the limitations of system resources such as processing power and memory challenges any data mining task. We study the online motif discovery problem under such a context.

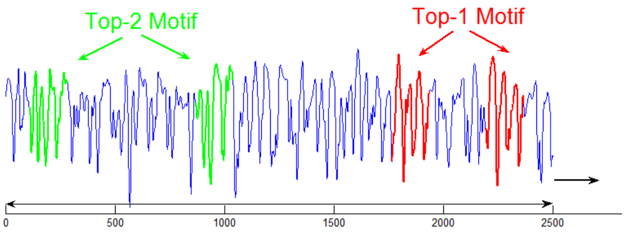


Figure 1: An Example of motifs in brain activity data. Two non-overlapping subsequences with similar shapes are defined as a motif. Picture looks better in color.

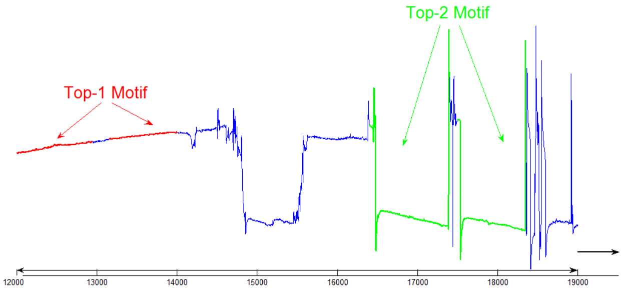


Figure 2: An Example of motifs in the insect data set. The motif in red likely corresponds to a steady state and hence does not present useful information. The second motif in green, however, may correspond to an important event. Figure looks better in color.

Time series motifs are repeating sequences which have different meanings depending on the application. For example, Figure 1 shows an example of one-second long motifs discovered by our algorithm in a time series corresponding to brain activity. These motifs reflect the repetition of the same sequence of actions in a human brain and can be useful in predicting the seizure period during an epileptic attack [7]. Besides, motif discovery was shown to be useful in many other applications as well, including time series data compression, insect time series management and even in the robotics domain [2].

In [2], *Mueen and Keogh* proposed several methods for summarizing the time series effectively and answering the motif query exactly at any time point in the sliding window setting. Their work, however, only focuses on the discovery of a single motif. Therefore, some interesting motifs could be missed accidently. For example, Figure 2 shows two motifs in the *Insect* time series. The first one in red likely corresponds to an idle or steady state of the device; this type of motif is probably not interesting. The second motif in green, however, is more interesting because it is likely that some important events happen at these time intervals. Therefore, an inherent advantage of the *top-k* motifs discovery for $k > 1$ is that the users are allowed to choose the meaningful motifs for their application and remove the meaningless ones while this is not allowed in the special case $k = 1$. Motivated by this reason we extend the prior work to the general case for any k. In summary, our contributions to this work are as follows:

- We extend the exact motif discovery problem to the *top-k* motifs problem and motivate this extension.
- We derive a theoretical lower-bound on the memory usage for any exact algorithm solving the *top-k* motifs discovery problem. We show that the obtained lower-bound is tight by showing a trivial algorithm that achieves the bound.
- As the trivial algorithm is not very efficient at query time, we propose two new algorithms, *nMotif* and its refinement *kMotif*. Both proposed algorithms are not only close to optimal in terms of memory usage but also fast in comparison to the existing algorithm *oMotif* for $k = 1$ [2].
- We demonstrate the significance of our work not only with theoretical analysis but also with extensive experiments with real-life and synthetic data.

The organization of the paper is as follows. Section 2 discusses the related work. Section 3 revisits some background knowledge and states the problem formally. Section 4 presents the theoretical memory lower-bound necessary for any exact algorithm to solve the *top-k* motifs discovery problem. In that section, we also see a simple approach achieving the given lower-bound. In Section 5, we introduce a more complicated algorithm which is more practical for very large-scale applications. Experimental results conducted with real-life and synthetic datasets are shown in section 6. Section 7 describes possible applications of the *top-k* motifs discovery problem. Finally, we make some conclusions about our work in section 8 and sketch some future work.

2 Related Work

The importance of motif discovery in time series has been addressed in several communities, including motion-capture, telemedicine and severe weather prediction. Several algorithms tackling this problem are based on searching a discrete approximation of the time series [3, 4, 5, 6, 7, 10]. *Based on discrete representations of real-valued data, these methods have introduced some levels of approximation in motif discovery* [9].

In recent years, some algorithms have been proposed for finding exact motifs directly in the raw time series data. The approach of [9] is the first tractable exact motif discovery algorithm based on the combination of early abandoning the Euclidean distance calculation and a heuristic search guided by the linear ordering of data. The authors also introduced for the first time a disk-aware algorithm for exact motif discovery for massive disk-resident datasets [11].

Table 1: Summary of Notations

Notations	Descriptions
r_i	i^{th} element of a time series
S_t	time series at time point t
m	motif length
w	number of m-dimensional points in the window
W	number of float values in the window
P_i	the m-dimensional point ending on time-stamp i
$d(P_i, P_j)$	Euclidean distance from P_i to P_j
L_i	forward nearest neighbor list or promising list

Although there has been significant research effort spent on efficiently discovering time series motifs, most of the literature has focused on fast and scalable approximate or exact algorithms for finding motifs in static offline databases. In fact, for many domains including online compression, robotics and sensor-networks, *the inherent streaming nature of time series requires online discovery and maintenance of time series motifs* [2]. In [2], A. Mueen et al. therefore propose an algorithm for solving the *exact motif discovery problem*. Their approach is denoted as *oMotif* in our work. The key idea behind the algorithm *oMotif* is to use a nearest neighbors data structure to answer the motif query fast. Although this approach demonstrates a space-efficient algorithm for exact motif discovery in real time, in several real applications with large sliding window sizes, this algorithm is very space demanding. Furthermore, in many real applications, the discovery of *top-k* motifs may be more useful and meaningful than a single motif. It is non-trivial to modify the *oMotif* algorithm to the problem of online *top-k* motifs discovery.

3 Problem Definition

Let $S_t = r_1, r_2, \cdots, r_t$ be a time series, in which all r_i are float values. In our theoretical analyses, we will assume that storing a float value requires one memory unit. When the time series S_t is evolving its length also evolves. In many applications, due to the limitation of the system's memory only a window of the W most recent float values in the time series, corresponding to the values $r_{t-W+1}, \ldots, r_t$, are kept in memory while the less recent ones are discarded. When a new value appears in the time series, the window is shifted one step further; the new item is appended to the sliding window, and the oldest one is removed. People usually refer to this as the *sliding windows model.*

Given a window of length W, a sequence of m consecutive float values in this window can be seen

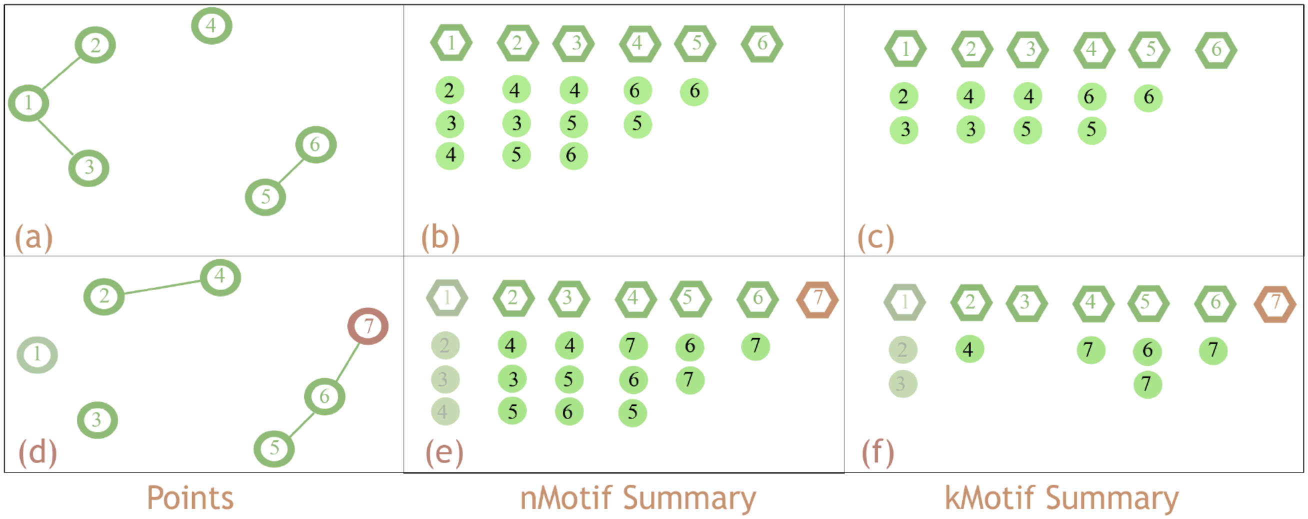


Figure 3: A toy example shows a set of 6 2-dimensional points and the data summaries used in algorithm *nMotif* and *kMotif* when $k = 3$ and $w = 6$.

as a point in a m-dimensional space. Hence, W float values in the window form a set of $w := W - m + 1$, m-dimensional points, which will be denoted as P_{t-w+1}, $P_{t-w+2}, \cdots, P_t$. Notice that W refers to the number of float values in the window and w refers to the number of m-dimensional points in the window. Each point P_i is associated with the time-stamp i of its last coordinate. The smaller the time-stamp, the earlier the point is. In [2], Mueen et Keogh defined a motif as the pair of points (P_i, P_j), $(i < j)$ which are closest to each other according to the Euclidean distance. We extend this notion and define the *top-k* motifs as the set of k pairs such that they are amongst the *top-k* closest pairs according to Euclidean distance. Sometimes we will require that points in a *top-k* pair are non-overlapping, i.e. $j - i \geq m$ [2].

For instance, assume $m = 2$, $k = 3$ and $w = 6$. Figure 3.a shows a set of 2-dimensional points. The current window contains 6 points P_1—P_6 and the *top-k* motifs are $\{(P_1, P_2), (P_5, P_6), (P_1, P_3)\}$. Consider now the case in Figure 3.d where a new point P_7 enters the time series. The window is shifted one point and consists of points P_2—P_7. Since point P_1 is expired, we get a new set of *top-k* motifs, which is $\{(P_5, P_6), (P_6, P_7), (P_2, P_4)\}$. Notice that in this toy example for simplicity we did not consider the constraint that points (P_i, P_j) in a motif pair should be non-overlapping $(j - i \geq m)$. If we do so the *top-k* motifs in Figure 3.a become $\{(P_1, P_3), (P_2, P_4), (P_4, P_6)\}$.The *top-k* similar motifs problem is defined as follows:

Definition 3.1. (*Top-k* Motifs Problem) *Given a time series S_t, a window length w, a motif length m and a parameter k, at any time point t, maintain a summary of the time series from which we can answer the query for the top-k motifs exactly.*

The *Top-k Motifs problem* is a generalization of the *Exact Motif Discovery Problem* of Mueen and Keogh [2], which corresponds to the case $k = 1$. We propose space-efficient algorithms for the generalized problem. First, we will derive a memory lower-bound for any exact algorithm solving the *top-k* motifs problem. Next, we show that the given lower-bound is tight by showing a trivial algorithm that consumes an amount of memory deviating only a constant factor from the lower-bound. This algorithms, however, does not scale up with large-scale applications monitoring thousands of time series, such as, e.g., sensor network. Therefore, we will propose other algorithms which are almost as space-efficient and achieve much better update times, and even outperform *oMotif* when ran with parameter k set to 1.

The comparison of the theoretical complexity the algorithms is summarized in Table 2. In all cells of the table we show the worst case complexity of the algorithms except the places where we denote *avg.* as

Table 2: Theoretical Complexity Comparison

	Alg.	**Space (floats)**	**Time per Update**
$k = 1$	*oMotif*	$\mathbf{O}(w^{\frac{3}{2}})$ amo.	$\mathbf{O}(wm)$
	nMotif	$\mathbf{O}(w)$	$\mathbf{O}(wm)$
$k > 1$	*nMotif*	$\mathbf{O}(kw)$	$\mathbf{O}(wm + w \log k)$
	kMotif	$\mathbf{O}(w)$ avg.	$\mathbf{O}(wm + \lvert V\rvert(k + \log \lvert V\rvert))$ *

*V denotes the so-called *promising points* and has size $2k \log(w)$ on average

for on average and *amo.* as for amortized. Notice that in this table we show the space complexity in term of the number of floats while the *lower-bound* in lemma 1 is based on the number bits.

4 Memory Lower-bound

In this section, we will derive a lower-bound on the memory requirement in order to summarize the time series and answer the *top-k* motifs query exactly. We further show that this lower-bound is tight by introducing several algorithms achieving the given bound.

4.1 Memory Lower-bound Deriving memory lower-bound introduces an insight into a research problem. Sometimes this step is very helpful in designing an optimal algorithm. The worst-case memory lower-bound of any exact algorithm solving the *top-k* similar motifs discovery problem is derived as follows:

Lemma 1. (Lower-bound) *Given a time series S_t, a window of size w, a motif length m and a parameter k. Any algorithm being able to answer the top-k motifs query exactly at any time point requires at least $\Omega(w\log(w))$ bits to summarize the time series.*

Proof. We will prove the lemma in the special case when $k = 1$ from which the correctness of the lemma with $k > 1$ will automatically hold. Consider a time series of length $w-1$: $S_{w-1} := r_1 r_2 \dots r_{w-1}$ in which all r_i are distinct. We will show that a summary of S_{w-1} denoted as SUM_{w-1} must allow for reconstructing perfectly all $w-1$ entries of S_{w-1}. Notice that since the window length is $W = w+m-1$ when we extend S_{w-1} with m values $y_1, y_2, \cdots, y_m$ no element of S_{w-1} is expired.

As the summary of S_{w-1} represents the full state of the machine recording the stream, the summary SUM_{w+m-1} of $S_{w+m-1} := r_1 \dots r_{w-1} y_1 \dots y_m$ only depends on SUM_{w-1} and $y_1, \dots, y_m$. Furthermore, SUM_{w+m-1} contains enough information for answering the top-1 query for S_{w+m-1}.

Let i be any number from 1 to $w-m$. Assume now that $y_1 = r_i$, $y_2 = r_{i+1}$, $\dots$, $y_m = r_{i+m-1}$, then the answer to the top-1 query for S_{w+m-1} will correspond to the subsequences $r_i \dots r_{i+m-1}$ and $y_1 \dots y_m$ with distance 0. As i is any arbitrary number from 1 to $w-m$, this shows that SUM_{w-1} contains enough information to reconstruct the sequence $r_1 \dots r_{w-1}$. Since, according to information theory, storing the order between $w-1$ different numbers requires at least $(w-1)\log(w-1)$ bits, this proves the lemma □.

Let us consider a very simple approach achieving the lower-bound in lemma 1. The algorithm stores the entire window as the summary which requires $\mathbf{O}(wb)$ bits where b is the maximal number of bits representing any float value r_i in the stream. The value of b can be arbitrarily large depending on how large r_i are. Let us make an optimistic assumption that b is upper-bounded by $\log(w)$, the number of bits minimally needed to store w different numbers. Consequently, the summary is as large as $w\log(w)$ bits, matching the lower-bound of Lemma 1. The lemma actually shows that there is no hope w.r.t. memory consumption, of doing better than just storing the complete window. The *top-k* query processing for such a summary, however, is very time demanding as we need to compute the Euclidean distances between all pairs of points, which results in a quadratic complexity. Therefore, the given approach is impractical for online applications which require not only a compact summary but also a quick response time to the query. The naive algorithm described in the following subsection will take into account both time and space efficiency.

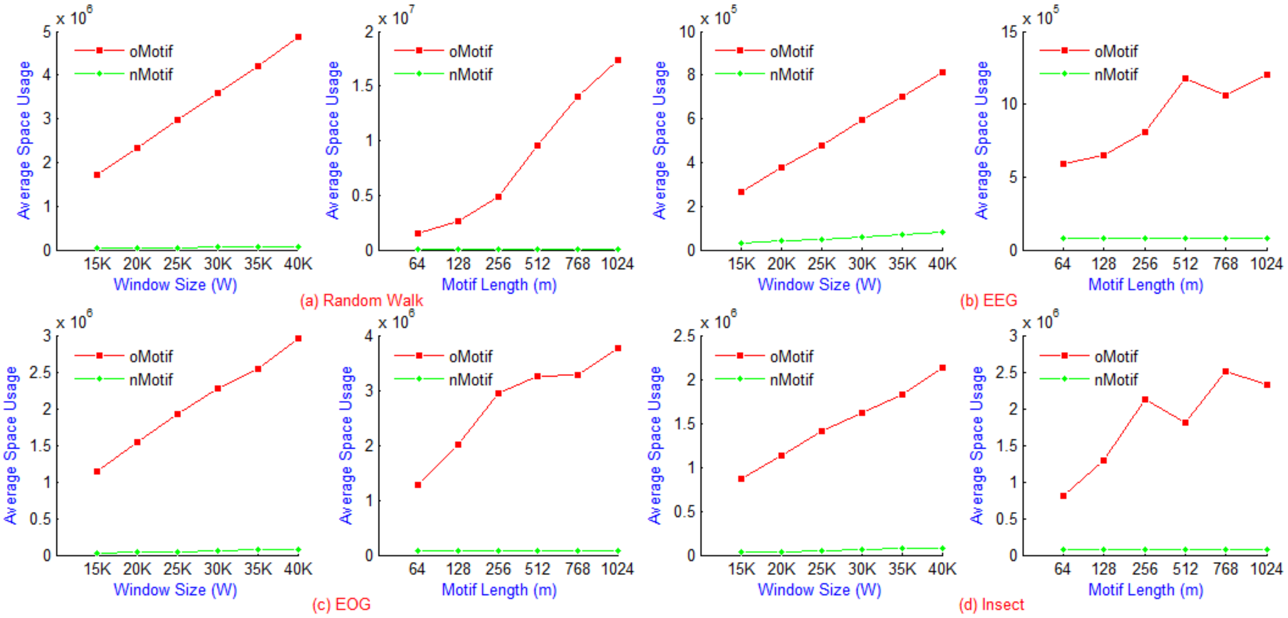
Algorithm 1 nMotif(w, m, k)

```
$\mathcal{L} \longleftarrow \emptyset$
while new point $P_t$ arrives do
    $\mathcal{L}$.PopFront()
    $L_t \leftarrow \emptyset$
    $\mathcal{L}$.PushBack($L_t$)
    for $i = t - w + 1$ to t-1 do
        Let $P_l$ be the last element of $L_i$
        upper-bound$\leftarrow d^2(P_i, P_l)$
        $d \leftarrow$ distance($P_t, P_i$,upper-bound)
        if $d < d(P_i, P_l)$ then
            Insert $P_t$ to list $L_i$ such that $L_i$ is sorted
            increasingly according to the distance to $P_i$
            if $L_i$.Size()$> k$ then
                $L_i$.PopBack()
            end if
        end if
    end for
end while
```

Algorithm 2 distance(X, Y,upper-bound)

```
$i \longleftarrow 0$
sum $\longleftarrow 0$
while sum $<$ upper-bound and $i < m$ do
    sum $\leftarrow$ sum $+(X[i] - Y[i])^2$
    $i = i + 1$
end while
return sqrt(sum)
```

4.2 A Naive Algorithm An online motif discovery algorithm should consider the trade-off between mem-



(a) Random Walk

(b) EEG

(c) EOG

(d) Insect

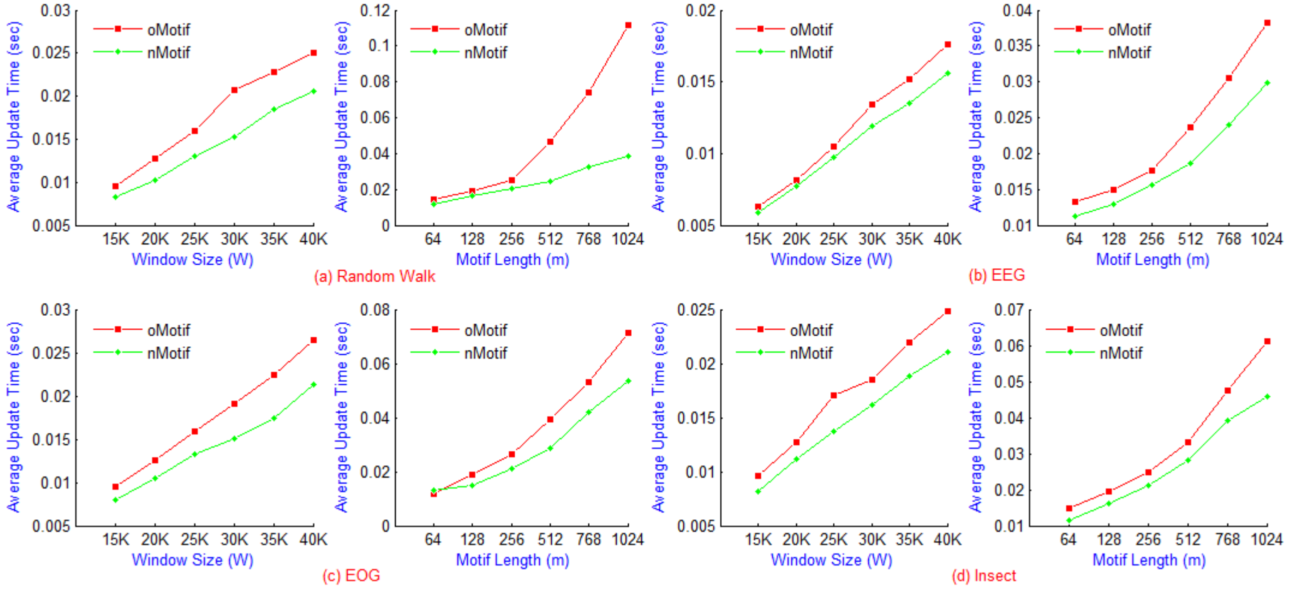
Figure 4: A comparison of the average space usage of two algorithms nMotif and oMotif in the special case $k = 1$.

ory usage and processing time. As time series data arrive continuously and unpredictably, summary update should be done as fast as possible while the summary should be as small as possible. In addition to that the query time also should be fast for online decision. However, usually it is the case that we have to trade processing time for smaller space usage and vice versa. We will now first introduce the *nMotif* algorithm, which has similarities with a first version of the *oMotif* algorithm of Mueen and Keogh [2]. Nevertheless, we will show both theoretically and empirically that *nMotif* with k set to 1 outperforms *oMotif* w.r.t. both space usage and query performance. Then we will further improve the *nMotif* algorithm to arrive at the more efficient *kMotif* algorithm for any $k > 1$.

The *oMotif* algorithm of Mueen and Keogh [2] roughly works as follows: instead of only storing the window, also for every point in the window its so-called *backward nearest neighbor list* (BNN) is stored. The BNN of a point P_i stores the time-stamp of all points with an earlier timestamp than P_i, ordered w.r.t. increasing distance to P_i. When a new point enters the window and hence the oldest point of the window is discarded, the lists are updated. Mueen and Keogh [2] discuss several optimizations and show that the size of the BNNs is $\sqrt{w}$ with an amortized analysis, leading to a summary of size $w^{3/2}$. For details we refer to [2]. In contrast to *oMotif*, our *nMotif* algorithm will store a *forward top-k nearest neighbor list* instead of a BNN. This seemingly trivial difference, however, has huge implications, as we will see.

The pseudo-code in Algorithm 1 shows how *nMotif* works. It maintains a list $\mathcal{L} = L_{t-w+1}L_{t-w+2}\cdots L_t$ where each entry L_i corresponds to a point P_i in the current time series window (lines 3-5). Each list L_i contains the set of the k points closest to P_i among the points with later time-stamp than i. For this reason we call L_i a *forward top-k nearest neighbor list*. In order to update these forward nearest neighbor lists when a new point P_t arrives we need to calculate the distance from P_t to any P_i (lines 8-9). Subsequently, P_t is inserted into the forward nearest neighbor list of P_i if P_t is closer to P_i than point P_l is, where P_l is the furthest point to P_i in the current list L_i (lines 10-15).

For instance, Figure 3.b shows this data structure for the points in Figure 3.a for $k = 3$. Corresponding to point P_1, the forward nearest neighbors are P_2, P_3 and P_4 sorted increasingly according to their distance to P_1. Similarly, the forward nearest neighbor list associated with P_2 contains P_4, P_3 and P_5 in that order. Recall that each list L_i associated with point P_i only contains the forward nearest neighbors. Therefore, although point P_1 is the nearest neighbor of point P_2 it is not contained in L_2 because P_1 appears earlier than P_2 in the time series. Figure 3.e shows how the data structure is updated when P_1 is expired and point P_7 arrives. In particular, we delete the entry corresponding to point P_1 because it has been expired. Then, the Euclidean distance from P_7 to every point P_i $(1 < i < 7)$ is calculated and the forward nearest neighbor list L_i is



(a) Random Walk

(b) EEG

(c) EOG

(d) Insect

Figure 5: A comparison of the average update time of two algorithms nMotif and oMotif in the special case $k = 1$.

updated subsequently.

Euclidean distance calculation: when the motif length is large, the update time is dominated by the Euclidean distance calculation process. In order to reduce the computational effort of this process an early termination technique was introduced in A. Mueen and Keogh [2]. The pseudo-code describing the Euclidean distance computation with early termination is shown in Algorithm 2. An extra parameter *upper-bound* is passed to the algorithm, reflecting the current "distance to beat" to be added in the forward *top-k* nearest neighbor list of X. As soon as in the distance computation it becomes clear that the distance between X and Y will exceed this bound, the distance computation is aborted.

Given the forward nearest neighbor data structure, the query phase is performed by first sorting all the pairs of points resident in the forward nearest neighbor structure increasingly according to Euclidean distance. The *top-k* pairs whose Euclidean distance are shortest are returned as answer to the *top-k* motifs query.

Lemma 2. (Correctness of *nMotif*) *In the forward nearest neighbor data structure handled by algorithm nMotif, we can find the true top-k motifs.*

4.3 Comparing *nMotif* to *oMotif*

First recall that *oMotif* only finds the *top-1* motif in the current window. Therefore, *oMotif* should be compared to *nMotif* with parameter k set to 1. Concerning space consumption, *nMotif* uses $\mathbf{O}(kw)$ float numbers in order to store the entire forward nearest neighbor structure, which is much better than the $\mathbf{O}(w^2)$ float numbers in the worst case and $\mathbf{O}(w^{\frac{3}{2}})$ float numbers in the amortized analysis needed by *oMotif*. Furthermore, every update operation of algorithm *nMotif* consists of the Euclidean distance computation cost ($\mathbf{O}(wm)$) and the summary update cost ($\mathbf{O}(w\log(k))$). Overall, the aggregated cost for each update operation is $\mathbf{O}(w(m + \log(k)))$ which is comparable to that of *oMotif* when $k = 1$. Hence, besides being more space-efficient *nMotif* is theoretically as time-efficient as the *oMotif* approach.

In order to illustrate the significance of our algorithm with $k = 1$ in comparison to algorithm *oMotif*, we plot the time and space consumption of the two algorithms in Figure 4 and Figure 5. In particular, Figure 4 compares the average space usage of two algorithms in terms of the number of elements stored in the data summary plus the window size. The comparison is done on 4 different datasets (see section 6 for the description of the datasets). We observe in all datasets that *nMotif* is significantly more space-efficient than *oMotif*. Besides, Figure 5 shows that *nMotif* is faster than *oMotif* although theoretical analysis shows that two algorithms are comparable in term of update time.

The reason for this are two folds. Firstly, the algorithm *nMotif* uses tighter bounds in the Euclidian distance computation, since there are less points in the summary. Secondly, the summary of *nMotif* is much smaller allowing efficient access and update. This analysis shows how a small change to the algorithm—

maintaining the forward nearest neighbor instead of all backward nearest neighbors—has a huge impact on the efficiency of the algorithm. In the following we will only compare to *nMotif* as it extends to arbitrary values of k and outperforms *oMotif* anyway.

5 A Space Efficient Approach

In this section, we introduce a space-efficient algorithm. The new algorithm is called *kMotif* and uses $\mathbf{O}(w)$ float numbers on average. Compared to *nMotif*, algorithm *kMotif* is about k times more space-efficient on average, and has, in worst case, still the same space and similar time complexity.

5.1 Algorithm *kMotif* We describe the main idea behind *kMotif* with a simple example. First let us define a couple of points (P_i, P_j) with $(i < j)$ a *promising pair* if it belongs to the *top-k* motifs of the window starting from point P_i. In the *kMotif* algorithm, we will store for every point P_i not its complete forward *top-k* nearest neighbor list, but instead only those timestamps j such that (P_i, P_j) forms a promising pair. This list is always a subset of the list maintained by *nMotif.* For example, consider again the illustration in Figure 3.a. In the window starting from P_1 and finishing at point P_6 the *top-3* motifs are (P_1, P_2), (P_1, P_3) and (P_5, P_6). Thus, in the forward nearest neighbor list of point P_1 (Figure 3.b), P_4 is redundant because (P_1, P_4) is not a promising pair. This pair will never become a *top-3* motif in any sliding window starting from P_1. As a result, we can safely remove point P_4 from the forward nearest neighbor list of point P_1 without any effect on the accuracy of the algorithm.

Similarly, if we consider the window starting from P_2 and finishing at point P_6, the pairs (P_2, P_4), (P_2, P_3) and (P_5, P_6) are the *top-3* motifs. Hence, point P_5 can be excluded from the forward nearest neighbor list of P_2. If we repeat this action for the other windows we will end up with the very compact list shown in Figure 3.c. This list has an important property that only promising pairs are present in it. Therefore, we call this data structure the *promising neighbors structure.*

The pseudo-code in Algorithm 3 describes how *kMotif* works. It always maintains and updates a promising neighbors list $\mathcal{L}$. In particular, when a new point P_t occurs, we traverse $\mathcal{L}$ backwards from the end to the beginning to check whether all pairs remain promising and remove unpromising ones (lines 7-21).

Lines 8-15 check whether an existing pair (P_l, P_i), where $P_l \in L_i$, is promising. In particular, we need to compare the value $d(P_l, P_i)$ to the distance between the current k^{th} closest pair, e.g $V[k-1]$ (line 10). If (P_l, P_i) is promising, i.e. $d(P_l, P_i) < V[k-1]$ the value $d(P_l, P_i)$ is inserted into a sorted dequeue V (line 11). Otherwise, we remove P_l from L_i (line 13).

Algorithm 3 kMotif(w, m, k)

```
 1: $\mathcal{L} \longleftarrow \emptyset$
 2: while New point $P_t$ arrives do
 3:   $\mathcal{L}$.PopFront()
 4:   $L_t \leftarrow \emptyset$
 5:   $\mathcal{L}$.PushBack($L_t$)
 6:   Let $V$ be an empty dequeue
 7:   for $i = t-1$ to $t-w+1$ do
 8:     while $P_l \neq$ nil do
 9:       $P_l \longleftarrow L_i$.next()
10:       if $d(P_l, P_i) < V[k-1]$ then
11:         $V$.insert($d(P_l, P_i)$)
12:       else
13:         $L_i$.delete($P_l$)
14:       end if
15:     end while
16:     $d(P_t, P_i) = $ distance($P_t, P_i, V[k-1]$)
17:     if $d(P_t, P_i) < V[k-1]$ then
18:       $V$.insert($d(P_t, P_i)$)
19:       $L_i$.insert($P_t$)
20:     end if
21:   end for
22: end while
```

We also perform the same check with every incoming pair (P_t, P_i) in lines 16-20. First we calculate the Euclidean distance from P_t to P_i (line 16). In the next step we check if (P_t, P_i) is promising and insert P_t into the sorted promising list L_i (lines 17-20). In doing so, the summary update phase costs about $\mathbf{O}(|V|(k + log|V|))$ in the worst case. Notice that in order to guarantee the correctness of the algorithm the insert and delete element functions are designed such that the list is always sorted increasingly according to the Euclidean distance.

Lemma 3. (Correctness of *kMotif*) *Using the promising data structure handled by algorithm kMotif, we can find the true top-k motifs. Besides, the promising summary is at most as large as the forward nearest neighbor adopted by nMotif algorithm.*

5.2 Complexity of *kMotif* The working space of algorithm *kMotif* consists of a portion of memory storing the window and another one containing the lists $\mathcal{L}$ and V. The former portion requires $\mathbf{O}(w)$ float numbers.The size of the latter portion depends on the number of promising pairs after every summary update. We consider this number as a random variable X. The following lemma shows how large X is on average :

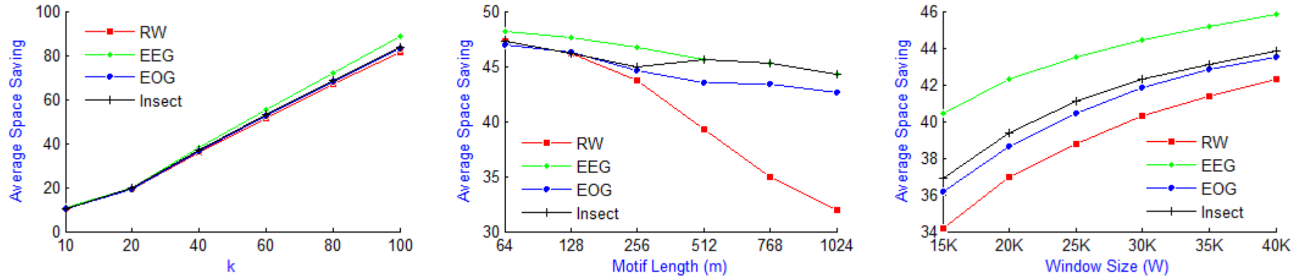


Figure 6: A comparison of *kMotif* and *nMotif* in terms of space saving. The space saving ratio is measured as the fraction between memory usage of algorithm *nMotif* and algorithm *kMotif*. Algorithm *kMotif* is 30-50 times more space-efficient than algorithm *nMotif* when $k = 50$.

Lemma 4. (Average Memory) *Given a window, assuming that any pair (P_i, P_j) has the same opportunity being in the top-k closest pairs of that window, the average number of promising pairs at any time point is $E(X) \simeq 2k\log(w)$*

Proof. Let $X_{ij}(j > i)$ be the indicator of the event that (P_i, P_j) is a promising pair. The probability of $X_{ij} = 1$ is equal to the probability of (P_i, P_j) is a *top-k* closest pairs among $\frac{(w-i)(w-i+1)}{2}$ pairs of points in this window if $\frac{(w-i)(w-i+1)}{2} > k$ and is equal to 1 otherwise. Since we assume that this probability is the same for any pair in the window we have:

$$Pr(X_{ij} = 1) = \begin{cases} \frac{2k}{(w-i)(w-i+1)} & if\ \frac{2k}{(w-i)(w-i+1)} < 1 \\ 1 & otherwise \end{cases}$$

On the other hand, as X stands for the number of promising pairs in list L we get:

$$X = \sum_{i=1}^{w} \sum_{j=i+1}^{w} X_{ij} \tag{5.1}$$

from which we further imply that:

$$E(X) = \sum_{i=1}^{w} \sum_{j=i+1}^{w} E(X_{ij}) \tag{5.2}$$

$$\simeq \sum_{i=1}^{w} \frac{2k}{(w-i+1)} \tag{5.3}$$

$$\simeq 2k \log w \tag{5.4}$$

The last equation proves the lemma. □

It is important to note that in this analysis we allow overlapping motifs, however, a similar result $E(X) \simeq 2k\log(w-m)$ with non-overlapping motifs can be proved similarly. Generally, the expected number of promising pairs is small compared to the size of the window w. We will see that for most of the datasets in our experiments the number of promising pairs does not deviate too far from this theoretical average value. Let us denote this theoretical average value as $E_p = 2k\log(w)$. We can theoretically bound the probability of the event that the number of promising pairs deviating from E_p by a constant factor as follows:

Lemma 5. (By Markov's inequality) *Given a constant a, the probability that the number of promising pairs deviating from its expected value by the constant factor a is bounded by $Pr(X > aE_p) \leq \frac{1}{a}$*

The aforementioned bound is naive but it is useful because there is no condition on the independence of X_{ij}. However, it is not tight since we can further tighten the bound based on the assumption that X_{ij}'s are independent from each other. In particular, we have the following result:

Lemma 6. (By the Chernoff's bound) *If we assume that X_{ij} are independent from each other we have: $Pr(X > eE_p) \leq \frac{1}{e^{E_p}}$*

Proof. According to the Chernoff's bound we have:

$$Pr(X > (1+\delta)\mu) < \left(\frac{e^{\delta}}{(1+\delta)^(1+\delta)}\right)^{\mu} \tag{5.5}$$

If we replace the value of δ by $e - 1$ and the value of μ by E_p then we have:

$$Pr(X > eE_p) < \left(\frac{e^{(e-1)}}{e^{e}}\right)^{E_p} \tag{5.6}$$

$$< \frac{1}{e^{E_p}} \tag{5.7}$$

The last inequality proves the lemma. □

Theoretically, Lemma 6 shows a tighter upper-bound on the probability of the event that the number

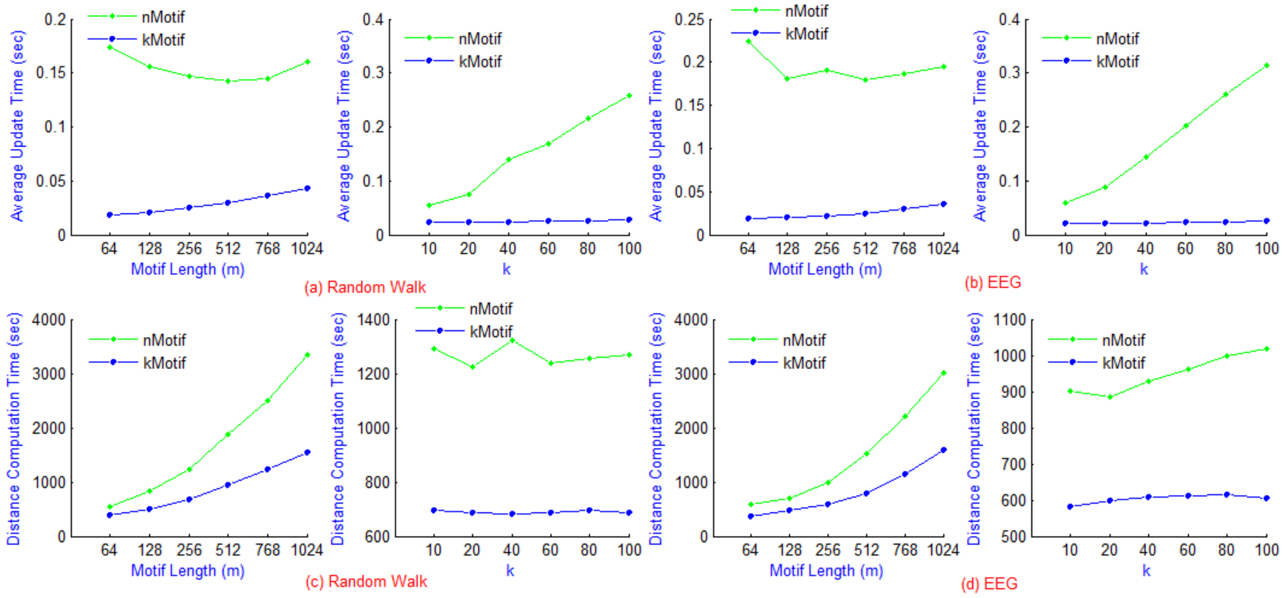


Figure 7: Update time and Euclidean distance calculation time for *kMotif* and *nMotif*.

of promising pair deviates far from E_p with a constant factor e. That probability is small, i.e. about 10^{-44} when k is in order of hundreds.

5.3 Space usage: *kMotif* vs. *nMotif* In summary, algorithm *kMotif* consumes $\Omega(w+2k\log(w))$ float numbers on average which is k times more space-efficient than algorithm *nMotif*. Another interesting property of algorithm *kMotif* is that although it seems to be theoretically less time-efficient than *nMotif* ($\mathbf{O}(wm+|V|(k+\log|V|)$ versus $\mathbf{O}(w(m+\log(k))))$, where $|V| = 2klogw$ on average. However, in practise it turns out to be faster in all experiments in both real-life and synthetic datasets. The reason of this effect will be discussed in the experimental section.

6 Experiments and Results

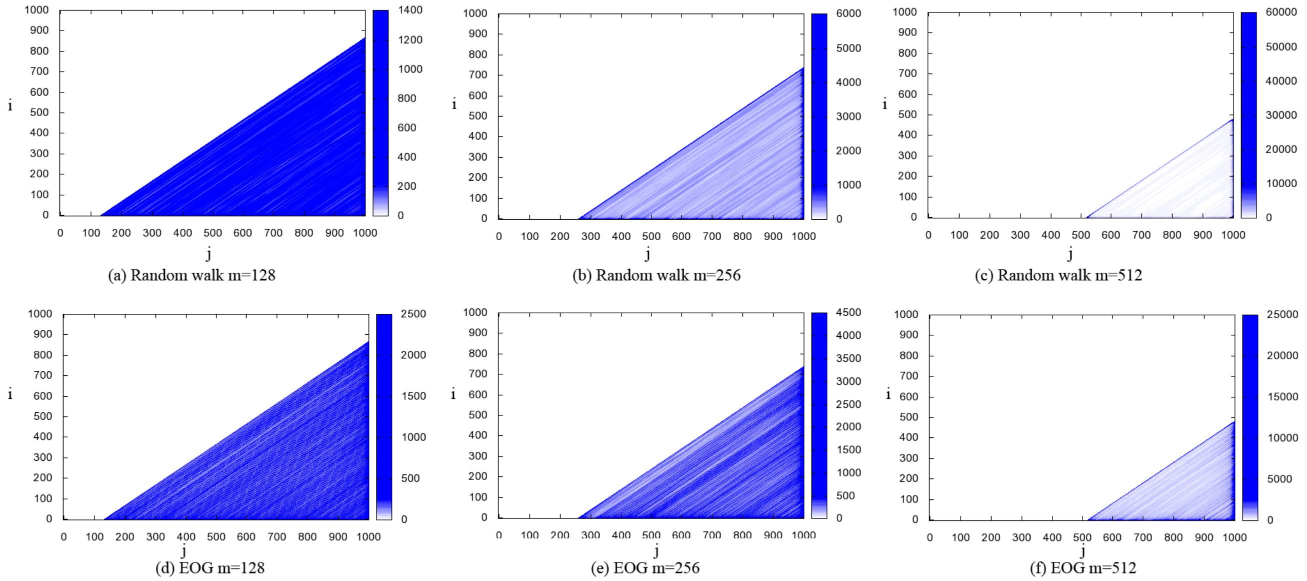
We implement algorithms *kMotif* and *nMotif* in C++ with STL library. We run all the programs in a 2.4 GHz core 2 dual Windows platform with 2GB of RAM. The datasets we use in the experiments are the same as in [2], including 3 real-life time series *EEG*, *EOG*, the *Insect* trace data and a synthetic dataset *random walk*. In all experiments Z-normalization is applied to all vectors and overlapping motifs are not allowed. The source code with demo and related resources of our work are available for download on our project website[1]. It is also worth noting that in this particular section we aim at discussing the significance of *kMotif* in comparison to *nMotif*. The significance of *nMotif* compared to *oMotif* is already discussed in section 4 where it was shown that *nMotif* outperforms *oMotif*.

In the first experiment, we fix $k = 50$ and $W = 50000$ while m is varied from 64 to 1024. For each value of m the ratio between the working space of *nMotif* and *kMotif* is calculated. The larger the ratio is, the more effective algorithm *kMotif* is. Figure 6 in the middle shows the result of this experiment on 4 datasets. On all datasets *kMotif* achieves 30 to 50 times space-saving as compared to *nMotif*. Another important observation is that the space-saving slightly degrades when m is large. The explanation of this effect will be discussed later.

Similarly, we carry out another experiment with fixing $m = 256$ and $W = 50000$ and varying the value of k. Figure 6 on the left shows the dependency of space-saving ratio on k. Four lines corresponding to the 4 datasets follow the following rule: the space-saving ratio increases linearly with the value of k. This result is consistent with our theoretical analysis in section 5.3 stating that when m and w are fixed the space-saving ratio is approximately equal to k. Finally, Figure 6 on the right illustrates the result when $m = 256$ and $k = 50$ are fixed while W is varied. We observe that the space-saving ratio is steadily increasing from 30 up to 50 along with the window length in this experiment.

6.1 Update time: *kMotif* versus *nMotif* We plot the update time and the Euclidean distance cal-

[1] http://www.win.tue.nl/∼lamthuy/projects/kmotif.html



(a) Random walk m=128

(b) Random walk m=256

(c) Random walk m=512

(d) EOG m=128

(e) EOG m=256

(f) EOG m=512

Figure 8: Density of the event (P_i, P_j) being a *top-k* motifs where i, j are the time-stamps at which the points start relative to the beginning of the window. When m is large parts close to the triangle edges are more dense.

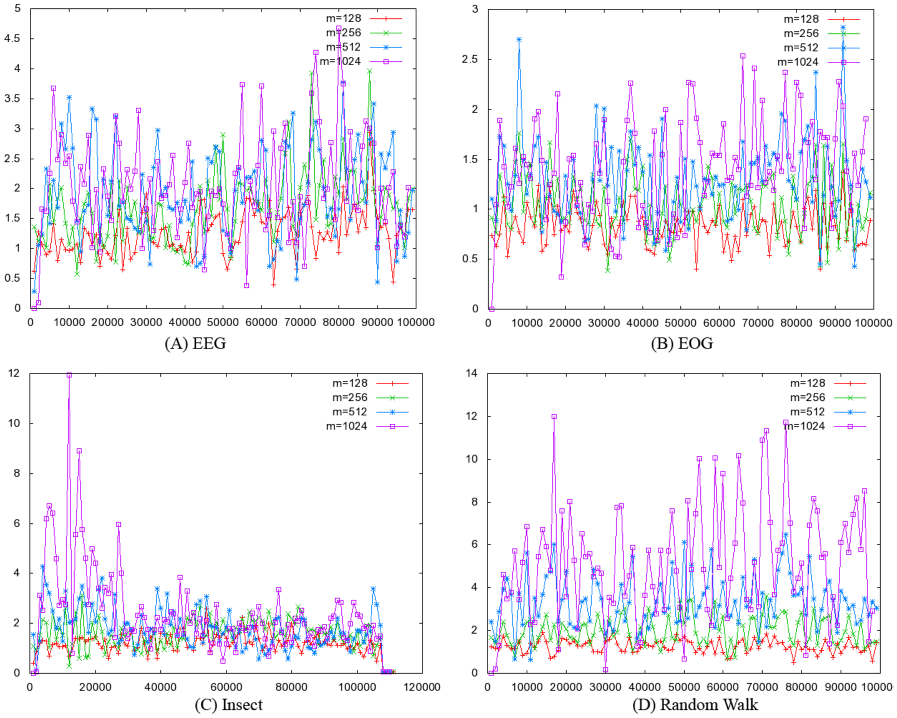
culation time of *kMotif* and *nMotif* in Figure 7. It is clear from Figure 7.a and 7.b that *kMotif* is steadily several times faster than *nMotif*. One of the reasons for the speed-up is explained in Figure 7.c and Figure 7.d in which the Euclidean distance calculation times are compared. *kMotif* adopts tighter upper-bounds in the distance calculation function leading to earlier termination of the calculation process. As a result, the summary update phase is more efficient. For brevity reason, the result is illustrated with only two datasets but similar behavior is observed with the other ones.

6.2 Distribution of promising pairs Recall that in order to derive the average number of promising pairs we had to assume a prior knowledge about the distribution of X_{ij}. In this section, we will discuss the validity of the assumption that we use in lemma 4. We first analyze how far the number of promising pairs deviates from the theoretical average value $E_p = 2k\log(w)$. In order to do so, the number of promising pairs in each window is divided by the theoretical expectation E_p to obtain the ratio which we call the *deviation ratio*. When we measure the deviation ratio for various values of k, m and w, we observe that it mostly depends on m. Therefore, we plot the deviation ratio when k and w are fixed to 100 and 1000 respectively and m varies from 128 to 512 in Figure 9.

Each portion of this figure corresponds to a dataset and the deviation ratios are plotted for 3 values of m corresponding to three lines $m = 128$, $m = 256$ and $m = 512$. The first behavior we can observe in all datasets is that the number of promising pairs is not deviated very far from the theoretical expectation E_p. In this particular setting a constant factor of 12 is an upper-bound for the deviation ratio.

Another important observation is that the variance of the deviation ratio in creases with increasing m. This explains the role of m in the degradation of the space-saving ratio as shown in subsection 5.3. In order to get an insight into the reason of this behavior we plot the distribution of the event (P_i, P_j) being a *top-k* motif in Figure 8 where i, j are the offsets of the time-stamps from the beginning of the window. In particular, we fix the value of $k = 100$ and $w = 1000$ and vary m as 128, 256 and 512. Then we count the number of times (P_i, P_j) is a *top-k* motif in the window of length w. The plot shows the result of only two datasets but similar behavior was observed in the other datasets.

Since we are only interested in non-overlapping motifs ($j - i \geq m$) only the right bottom triangle is dense. If the assumption in lemma 4 is true we must observe the same density at any point in this triangle. Actually, this is the case when m is small; cfr. Figures 8.a and 8.d. However, when m is large the *kMotif* algorithm favors the points close to the triangle's edges causing an interesting effect which can be observed in Figures 8.b.c.e.f. The areas closed to the triangle's edges are more dense. Currently we do not have a satisfactory explanation for this observation. Because of the observed anomaly, when m becomes



(A) EEG

(B) EOG

(C) Insect

(D) Random Walk

Figure 9: The ratio between the number of promising pairs and the theoretical average number of promising pairs E_p. Picture looks better in color.

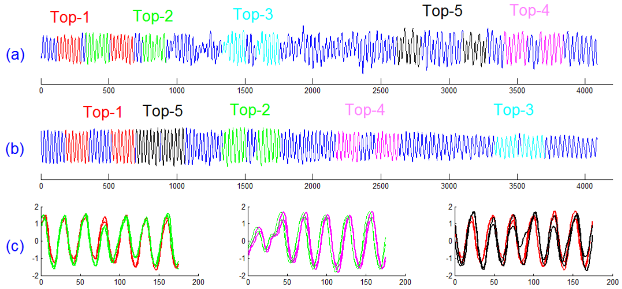


Figure 10: Motifs discovered in brain activity data. *Top-k* motifs discovery provide the biologist with more insight information about the similarity structure of the motifs. Picture looks better in color.

large the performance of *kMotif* can slightly degrade. Nevertheless, as we have seen in the experiments, when m is as large as 1024, *kMotif* still preserves very high space saving ratio (at least 30 times when $k = 50$). Besides, in many applications with multi-dimensional time series the typical dimensionality is only in order of hundreds which remains in the range where *kMotif* exposes its maximum performance.

7 Applications of the *Top-k* Motifs Discovery

Various applications of the online motif discovery problem have been discussed by Mueen and Keogh [2]. In that work, the authors demonstrate many interesting applications of the given problem in data compression, in acoustic data management and in the robotics domain. In the current paper, the *top-k* motifs discovery is introduced as a generalized version of the exact motif discovery problem. Therefore, the result of this work can be indirectly and efficiently be applied to the same applications. However, in other to emphasize on the necessity of *top-k* motif discovery we will focus on an example which actually shows that the *top-k* motifs can be used as the supplement to the exact motif discovery in order to find meaningful patterns of the time series.

Finding meaningful motifs In brain activity a motif may be in the form of the largest connected components (brain areas) preceding a seizure. Therefore, finding the significant motifs can be valuable for predicting the seizure periods. In other words, the *strong similarity in motif occurrences in seizure data not only can be used in forecasting the unobserved outcome but also can help to isolate groups of neurons that trigger identical activity during the epileptic attacks* [12].

In order to identify significant motifs we need powerful visualization tools which show the similarity structure between the discovered motifs. For instance, we carried out a simple experiment with the brain activity datasets from epileptic patients during epileptic attacks [7]. This dataset consists of 100 time series of length 4087 each. The motif length m is set to 174 (the number of recordings per sec.) and the sliding window length is set to $w = 4087$.

In Figure 10, an example of the *top-5* motifs found in the 4^{th} (Figure 10.a) and 8^{th} (Figure 10.b) time series are displayed. Figure 10.c highlights the strong similarity of the *top-5* motifs not only in the same time series but also over the different time series. Particularly, Figure 10.c on the left shows the similarity between the *top-1* and the *top-2* motif detected in Figure 10.a, and Figure 10.c in the middle shows the similarity between the *top-3* motif in Figure 10.a and the *top-2* motif in Figure 10.b. Finally, Figure 10.c on the right shows the similarity of the *top-1* motif and the *top-5* motif detected in Figure 10.b. These interpretable and visually meaningful motifs provide biologists with a chance to understand the similarity structure across the *top-k* motifs. Choosing the most significant one is easier with this interpretable visualization.

8 Conclusions and Future Work

We introduced and discussed the online *top-k* motifs discovery problem. A theoretical memory lower-bound for any exact algorithm was shown. This lower-bound is tight and easily achievable by simply storing the complete window. Based on the *oMotif* algorithm, a more efficient yet simpler algorithm *nMotif* was proposed. Algorithm *nMotif* was then further improved to get *kMotif*. From both theoretical and empirical analysis it followed that *nMotif* is much more space and time efficient than *oMotif* for the single motif discovery problem, and that *kMotif* outperforms *nMotif* for the general *top-k* discovery problem.

In addition to that, we also show interesting application of the *top-k* motifs discovery problem. In particular, we showed by demonstration that the *top-k* motifs with $k > 1$ provide more information about the inherent structure of the motifs. Thus, it provides the users with better chances to choose the meaningful motifs for their application.

For the future work we suggest to focus on the methods which exactly identify the meaningful motifs for a specific application domain. A remarkable observation in the experimental data shows that it is not easy to find the meaningful motifs based only on the Euclidean distance. Therefore, in order to identify more meaningful motifs we need a post-processing phase in which the help from the experts in the specific application domain is necessary. In doing so, the motif discovery process is no longer online. Our suggestion is to re-define the motif concepts matching the application needs instead of using the current definition based solely on the Euclidean distance. In doing so, we can have an automatic meaningful motifs identification process making it suitable for online decision.

9 Acknowledgements

We deeply thank A. Mueen and professor E. Keogh for their released datasets, source code and useful discussion in the early stage of the project. We also thank all the anonymous reviewers for their useful comments which help us improve our work significantly.

References

[1] Arno J. Knobbe, Hendrik Blockeel, Arne Koopman, Toon Calders, Bas Obladen, Carlos Bosma, Hessel Galenkamp, Eddy Koenders, Joost N. Kok: *InfraWatch: Data Management of Large Systems for Monitoring Infrastructural Performance.* IDA 2010: 91-102

[2] Abdullah Mueen, Eamonn J. Keogh: *Online discovery and maintenance of time series motifs.* KDD 2010: 1089-1098

[3] Chiu, B., Keogh, E. and Lonardi, S. Probabilistic Discovery of Time Series Motifs. ACM SIGKDD 2003, pp. 493-498.

[4] Patel, P., Keogh, E., Lin, J. and Lonardi, S. Mining Motifs in Massive Time Series Databases. ICDM 2002, pp. 370 - 377.

[5] Lin, J., Keogh, E., Lonardi, S., and Patel, P. Finding Motifs in Time Series, 2nd Workshop on Temporal Data Mining (KDD'02), pp. 53 - 68.

[6] Ferreira, P., Azevedo, P.J., Silva, C. and Brito, R. Mining Approximate Motifs in Time Series, Discovery Science, 2006.

[7] Yankov, D., Keogh, E., Medina, J., Chiu, B. and Zordan B. Detecting Motifs under Uniform Scaling. SIGKDD 2007, pp 844 - 853.

[8] Shieh, J. and Keogh, E. iSAX: Indexing and Mining Terabyte Sized Time Series, SIGKDD 2008, pp. 623-631.

[9] Mueen, A., Keogh, E., Zhu, Q., Cash, S. and Westover, B. Exact Discovery of Time Series Motif. SDM 2009, pp. 473-484.

[10] Castro, N. and Azevedo, P. Multiresolution Motif Discovery in Time Series. SDM 2010, pp. 665-676.

[11] Mueen, A., Keogh, E. and Shamlo, N.B. Finding Time Series Motifs in Disk-Resident Data. ICDM 2009, pp. 367-376.

[12] T. Sauer. Time series prediction by using delay coordinate embedding. Time Series Prediction. Forecasting the Future and Understanding the Past, 59(8):175-193, August 1994.