Data Mining and Knowledge Discovery (2020) 34:1022–1071
https://doi.org/10.1007/s10618-020-00685-w

Matrix profile goes MAD: variable-length motif and discord discovery in data series

Michele Linardi[1] · Yan Zhu[3] · Themis Palpanas[2] · Eamonn Keogh[3]

Received: 13 January 2019 / Accepted: 10 April 2020 / Published online: 7 May 2020

Abstract

In the last 15 years, data series *motif* and *discord* discovery have emerged as two useful and well-used primitives for data series mining, with applications to many domains, including robotics, entomology, seismology, medicine, and climatology. Nevertheless, the state-of-the-art motif and discord discovery tools still require the user to provide the relative length. Yet, in several cases, the choice of length is critical and unforgiving. Unfortunately, the obvious brute-force solution, which tests all lengths within a given range, is computationally untenable. In this work, we introduce a new framework, which provides an exact and scalable motif and discord discovery algorithm that efficiently finds all motifs and discords in a given range of lengths. We evaluate our approach with five diverse real datasets, and demonstrate that it is up to 20 times faster than the state-of-the-art. Our results also show that removing the unrealistic assumption that the user knows the correct length, can often produce more intuitive and actionable results, which could have otherwise been missed.

Keywords Data series · Time series · Motif discovery · Variable length · Data mining

Responsible editor: Panagiotis Papapetrou

✉ Michele Linardi
michele.linardi@parisdescartes.fr

Yan Zhu
yzhu015@ucr.edu

Themis Palpanas
themis@mi.parisdescartes.fr

Eamonn Keogh
eamonn@cs.ucr.edu

[1] University of Paris, Paris, France

[2] French University Institute (IUF), University of Paris, Paris, France

[3] University of California at Riverside, Riverside, USA

1 Introduction

Data series[1] have gathered the attention of the data management community for more than 2 decades (Agrawal et al. 1993; Jagadish et al. 1995; Rafiei and Mendelzon 1998; Chakrabarti et al. 2002; Papadimitriou and Yu 2006; Camerra et al. 2010; Kashyap and Karras 2011; Wang et al. 2013b; Camerra et al. 2014; Dallachiesa et al. 2014; Zoumpatianos et al. 2016; Yagoubi et al. 2017; Jensen et al. 2017; Palpanas 2017; Kondylakis et al. 2018; Peng et al. 2018; Zoumpatianos and Palpanas 2018; Zoumpatianos et al. 2018; Gogolou et al. 2019; Echihabi et al. 2018, 2019; Yagoubi et al. 2020; Kondylakis et al. 2019; Boniol et al. 2020; Peng et al. 2020a; Boniol and Palpanas 2020; Peng et al. 2020b; Palpanas 2020; Gogolou et al. 2020). They are now one of the most common types of data, present in virtually every scientific and social domain (Palpanas 2015; Raza et al. 2015; Mirylenka et al. 2016; Keogh 2011; Palpanas and Beckmann 2019; Bagnall et al. 2019).

Over the last decade, data series motif discovery has emerged as perhaps the most used primitive for data series data mining, and it has many applications to a wide variety of domains (Whitney et al. 1998; Yankov et al. 2007a), including classification, clustering, and rule discovery. More recently, there has been substantial progress on the scalability of motif discovery, and now massive datasets can be routinely searched on conventional hardware (Whitney et al. 1998).

Another critical improvement in motif discovery, is the reduction in the number of parameters that require specification. The first motif discovery algorithm, PROJECTION (Chiu et al. 2003), requires that the users set seven parameters, and it still only produces answers that are approximately correct. Researchers have *"chipped"* away at this over the years (Mueen et al. 2009; Saria et al. 2011), and the current state-of-the-art algorithms only require the user to set a single parameter, which is the desired length of the motifs. Paradoxically, the ease with which we can now perform motif discovery has revealed that even this single burden on the user's experience or intuition may be too great.

For example, AspenTech, a company that makes software for optimizing the manufacturing process for the oil and gas industry, has begun to use motif discovery in their products both as a stand-alone service and also as part of a precursor search tool. They recently noted that, *"our lighthouse (early adopter) customers love motif discovery, and they feel it adds great value [...] but they are frustrated by the finicky setting of the motif length."*(Noskov Michael—Director, Data Science at Aspen Technology—Personal communication, February 3rd 2015). The issue, of being restricted to specifying length as an input parameter, has also been noted in other domains that use motif discovery, such as cardiology (Syed et al. 2010) and speech therapy (Wang et al. 2013a), as well as in related problems, such as data series indexing (Linardi and Palpanas 2018a, b).

The obvious solution to this issue is to make the algorithms search over all lengths in a given range and rank the various length motifs discovered. Nevertheless, this

[1] If the dimension that imposes the ordering of the series is time, then we talk about *time series*. However, a series can also be defined through other measures (e.g., angle in radial profiles in astronomy, mass in mass spectroscopy, position in genome sequences, etc.). We use the terms *time series*, *data series*, and *sequence* interchangeably.

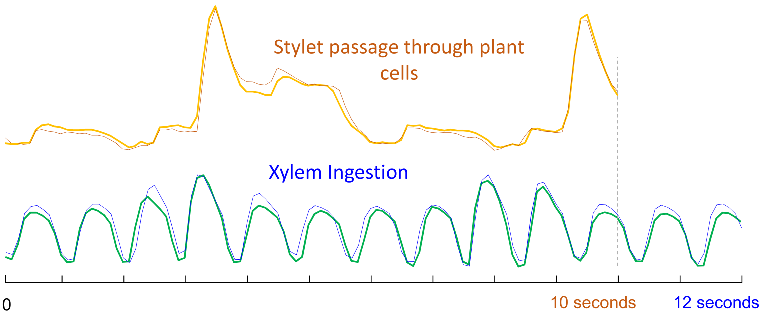


Fig. 1 An existence proof of semantically different motifs, of slightly different lengths, extracted from a single dataset

strategy poses two challenges. First, how we can rank motifs of different lengths? Second, and most important, how we can search over this much larger solution space in an efficient way, in order to identify the motifs?

In this work, we describe the first algorithms in the literature that address both problems. The proposed solution requires new techniques that significantly extend the state-of-the-art algorithms, including the introduction of a novel lower bounding method, which makes efficiently searching a large number of potential solutions possible.

Note that even if the user has good knowledge of the data domain, in many circumstances, searching with one single motif length is not enough, because the data can contain motifs of various lengths. We show an example in Fig. 1, where we report the 10-s and 12-s motifs discovered in the Electrical Penetration Graph (EPG) of an insect called Asian citrus psyllid. The first motif denotes the insect's highly technical probing skill as it searches for a rich leaf vein (stylet passage), whereas the second motif is just a simple repetitive "sucking" behavior (xylem ingestion). This example shows the utility of variable length motif discovery. An entomologist using classic motif search, say at the length of 12 s, might have plausibly believed that this insect only engaged in xylem ingestion during this time period, and not realized the insect had found it necessary to reposition itself at least twice.

The two motif pairs are radically different, reflecting two different types of insect activities. In order to capture all useful activity information within the data, a fast search of motifs over all lengths is necessary.

Another popular and well-studied data series primitive, the discord (Yankov et al. 2008; Keogh et al. 2005; Yeh et al. 2016; Senin et al. 2015; Luo et al. 2013), is proposed to discover subsequences that represent outliers. Surprisingly, the solutions to this problem that have been proposed in the literature are not as effective and scalable as practice requires. The reasons are twofold. First, they only support fixed-length discord discovery, and as we explained earlier, this rigidity with the subsequence length restricts the search space, and consequently, also the produced solutions and the effectiveness of the algorithm. Second, the existing techniques provide poor support for enumerating multiple discords, namely, for the identification of multiple anomalous subsequences. These works have considered only cases with up to *3* anomalous subsequences.

Therefore, we extend our motif discovery framework, and propose the first approach in the literature that deals with the variable-length discord discovery problem. Our approach leads to a scalable solution, enabling the identification of a large number of anomalous patterns, which can be of different lengths.

In this work,[2] we make the following contributions:

- We define the problems of variable-length motif and discord discovery, which significantly extend the usability of their operations, respectively.
- We propose a new data series motif and discord framework. The Variable Length Motif Discovery algorithm (VALMOD) takes as input a data series T, and finds the subsequence pairs with the smallest Euclidean distance of each length in the (user-defined) range $[\ell_{min}, \ell_{max}]$. VALMOD is based on a novel lower bounding technique, which is specifically designed for the motif discovery problem.
- Furthermore, we extend VALMOD to the discord discovery problem. We propose a new exact variable-length discord discovery, which aims at finding the subsequence pairs with the largest Euclidean distances of each length in the (user-defined) range $[\ell_{min}, \ell_{max}]$.
- We evaluate our techniques using five diverse real datasets, and demonstrate the scalability of our approach. The results show that VALMOD is up to 20 × faster than the state-of-the-art techniques. Furthermore, we present real case studies with datasets from entomology, seismology, and traffic data analysis, which demonstrate the usefulness of our approach.

2 Problem definition

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

Definition 1 (*Data series*) A data series $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$ $[t_1, t_2, \ldots, t_n]$, where n is the length of T.

We are typically not interested in the global properties of a data series, but in the local regions known as subsequences:

Definition 2 (*Subsequence*) A subsequence $T_{i,\ell} \in \mathbb{R}^\ell$ of a data series T is a continuous subset of the values from T of length ℓ starting from position i. Formally, $T_{i,\ell} = [t_i, t_{i+1}, \ldots, t_{i+\ell-1}]$.

2.1 Motif discovery

In this work, a particular local property we are interested in is data series motifs. A data series motif pair is the pair of the most similar subsequences of a given length, ℓ, of a data series:

[2] A preliminary version of this work has appeared elsewhere (Linardi et al. 2018a, b).

Definition 3 (*Data series motif pair*) $T_{a,\ell}$ and $T_{b,\ell}$ is a motif pair iff $dist(T_{a,\ell}, T_{b,\ell}) \leq dist(T_{i,\ell}, T_{j,\ell})$ $\forall i, j \in [1, 2, \ldots, n - \ell + 1]$, where $a \neq b$ and $i \neq j$, and dist is a function that computes the z-normalized Euclidean distance between the input subsequences (Chiu et al. 2003; Mueen et al. 2009; Wang et al. 2013a; Whitney et al. 1998; Yankov et al. 2007a).

Note, that we can consider more motifs, beyond the top motif pair. To that extent, we can simply build a rank of subsequence pairs in T (of length ℓ), according to their distances in ascending order. We call the subsequences pairs of this ranking *motif pairs of length* ℓ.

We store the distance between a subsequence of a data series with all the other subsequences from the same data series in an ordered array called a distance profile.

Definition 4 (*Distance profile*) A distance profile $D \in \mathbb{R}^{(n-\ell+1)}$ of a data series T regarding subsequence $T_{i,\ell}$ is a vector that stores $dist(T_{i,\ell}, T_{j,\ell})$, $\forall j \in [1, 2, \ldots, n - \ell + 1]$, where $i \neq j$.

One of the most efficient ways to locate the exact data series motif is to compute the matrix profile (Yeh et al. 2016; Zhu et al. 2016), which can be obtained by evaluating the minimum value of every distance profile in the time series.

Definition 5 (*Matrix profile*) A matrix profile $MP \in \mathbb{R}^{(n-\ell+1)}$ of a data series T is a meta data series that stores the z-normalized Euclidean distance between each subsequence and its nearest neighbor, where n is the length of T and ℓ is the given subsequence length. The data series motif can be found by locating the two lowest values in MP.

To avoid trivial matches, in which a pattern is matched to itself or a pattern that largely overlaps with itself, the matrix profile incorporates an "exclusion-zone" concept, which is a region before and after the location of a given query that should be ignored. The exclusion zone is heuristically set to $\ell/2$. The recently introduced STOMP algorithm (Zhu et al. 2016) offers a solution to compute the matrix profile MP in $\mathcal{O}(n^2)$ time. This may seem untenable for data series mining, but several factors mitigate this concern. First, note that the time complexity is independent of ℓ, the length of the subsequences. Secondly, the matrix profile can be computed with an anytime algorithm, and in most domains, in just $\mathcal{O}(nc)$ steps the algorithm converges to what would be the final solution (Yeh et al. 2016) (c is a small constant). Finally, the matrix profile can be computed with GPUs, cloud computing, and other HPC environments that make scaling to at least tens of millions of data points trivial (Zhu et al. 2016).

We can now formally define the problems we solve.

Problem 1 (*Variable-Length Motif Pair Discovery*) Given a data series T and a subsequence length-range $[\ell_{min}, \ldots, \ell_{max}]$, we want to find the data series motif pairs of all lengths in $[\ell_{min}, \ldots, \ell_{max}]$, occurring in T.

One naive solution to this problem is to repeatedly run the state-of-the art motif discovery algorithms for every length in the range. However, note that the size of this range can be as large as $\mathcal{O}(n)$, which makes the naive solution infeasible for even middle-size data series. We aim at reducing this $\mathcal{O}(n)$ factor to a small value.

Note that the motif pair discovery problem has been extensively studied in the last decade (Yeh et al. 2016; Zhu et al. 2016; Mueen and Chavoshi 2015; Li et al. 2015; Mueen et al. 2009; Mohammad and Nishida 2014, 2012). The reason is that if we want to find a collection of recurrent subsequences in T, the most computationally expensive operation consists of identifying the motif pairs (Zhu et al. 2016), namely, solving Problem 1. Extending motif pairs to sets incurs a negligible additional cost (as we also show in our study).

Given a motif pair $\{T_{\alpha,\ell}, T_{\beta,\ell}\}$, the data series motif set S_r^ℓ, with radius $r \in \mathbb{R}$, is the set of subsequences of length ℓ, which are in distance at most r from either $T_{\alpha,\ell}$, or $T_{\beta,\ell}$. More formally:

Definition 6 (*Data series motif set*) Let $\{T_{\alpha,\ell}, T_{\beta,\ell}\}$ be a motif pair of length ℓ of data series T. The motif set S_r^ℓ is defined as: $S_r^\ell = \{T_{i,\ell} | dist(T_{i,\ell}, T_{\alpha,\ell}) < r \vee dist(T_{i,\ell}, T_{\beta,\ell}) < r\}$.

The cardinality of S_r^ℓ, $|S_r^\ell|$, is called the frequency of the motif set.

Intuitively, we can build a motif set starting from a motif pair. Then, we iteratively add into the motif set all subsequences within radius r. We use the above definition to solve the following problem (optionally including a constraint on the minimum frequency for motif sets in the final answer).

Problem 2 (*Variable-Length Motif Sets Discovery*) Given a data series T and a length range $[\ell_{min}, \ldots, \ell_{max}]$, we want to find the set $S^* = \{S_r^\ell | S_r^\ell$ is a motif set, $\ell_{min} \leq \ell \leq \ell_{max}\}$. In addition, we require that if $S_r^\ell, S_{r'}^{\prime\ell'} \in S^* \Rightarrow S_r^\ell \cap S_{r'}^{\prime\ell'} = \emptyset$.

By abuse of notation, we consider an intersection non-empty in the case where subsequences have different lengths, but the same starting position offset (e.g., $S_r^{200} = \{T_{4,200}\}$, $S_{r'}^{500} = \{T_{4,500}\} \Rightarrow S_r^{200} \cap S_{r'}^{500} \neq \emptyset$).

Thus, the variable-length motif sets discovery problem results in a set, S^*, of motif sets. The constraint at the end of the problem definition restricts each subsequence to be included in at most one motif set. Note that in practice we may not be interested in all the motif sets, but only in those with the k smallest distances, leading to a *top-k* version of the problem. In our work, we provide a solution for the *top-k* problem (though, setting k to a very large value will produce all results).

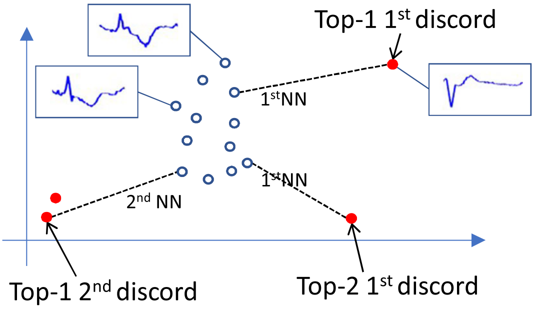
2.2 Discord discovery

In order to introduce the problem of discord discovery, we first define the notion of *best match*, or *nearest neighbor*.

Definition 7 (*mth best match*) Given a subsequence $T_{i,\ell}$, we say that its mth best match, or Nearest Neighbor (mth NN) is $T_{j,\ell}$, if $T_{j,\ell}$ has the mth shortest distance to $T_{i,\ell}$, among all the subsequences of length ℓ in T, excluding trivial matches.

In the distance profile of $T_{i,\ell}$, the mth smallest distance, is the distance of the mth best match of $T_{i,\ell}$. We are now in the position to formally define the discord primitives, we use in our work.

Fig. 2 A dataset with 12 subsequences (of the same length ℓ) depicted as points in 2-dimensional space. We report the *Top-k* mthdiscords. They belong to groups of subsequences, whose cardinality depends on m. The index k ranks the subsequences according their mth best match distances, in descending order



Definition 8 (*mth discord* (Keogh et al. 2005)) The subsequence $T_{i,\ell}$ is called the mth discord of length ℓ, if its mth best match is the largest among the best match distances of all subsequences of length ℓ in T.

Intuitively, discovering the $mthdiscord$ enables us to find an isolated group of m subsequences, which are far from the rest of the data. Furthermore, we can rank the $mthdiscords$, according to their mth best matches. This allows us to define the *Top-k* mthdiscords.

Definition 9 (*Top-k* mth *discord*) A subsequence $T_{i,\ell}$ is a Top-k mth-discord if it has the kth largest distance to its mth NN, among all subsequences of length ℓ of T.

In Fig. 2, we plot a group of *12* subsequences (represented in a 2-dimensional space), and we depict three *Top-k* mthdiscords (groups of red/dark circles). Remember that m represents the number of anomalous subsequences in a discord group. On the other hand, k ranks the discords and implicitly the groups, according to their mth best match distances, in descending order (e.g., Top-1 1st discord and Top-1 2nd).

Given these definitions, we can formally introduce the following problem:

Problem 3 (*Variable-Length Top-k mth Discord Discovery*) Given a data series T, a subsequence length-range $[\ell_{min}, \ldots, \ell_{max}]$ and the parameters $a, b \in \mathbb{N}^+$ we want to enumerate the *Top-k* mthdiscords for each $k \in \{1, .., a\}$ and each $m \in \{1, .., b\}$, and for all lengths in $[\ell_{min}, \ldots, \ell_{max}]$, occurring in T.

Observe that solving the Variable-Length *Top-k* mthDiscord Discovery problem is relevant to solving the Variable-Length Motif Set Discovery problem: in the former case we are interested in the subsequences with the most distant neighbors, while in the latter case we seek the subsequences with the most close neighbors. Therefore, the Matrix Profile, which contains all this information, can serve as the basis to solve both problems.

3 Comparing motifs of different lengths

Before introducing our solutions to the problems outlined above, we first discuss the issue of comparing motifs of different lengths. This becomes relevant when we want to rank motifs of different lengths (within the given range), which is useful in order to identify the most prominent motifs, irrespective of their length. In this section, we propose a length-normalized distance measure that the VALMOD algorithm uses in order to produce such rankings.

The increased expressiveness of VALMOD offers a challenge. Since we can *discover* motifs of different lengths, we also need to be able to *rank* motifs of different lengths. A similar problem occurs in string processing, and a common solution is to replace the edit-distance by the length-normalized edit-distance, which is the classic distance measure divided by the length of the strings in question (Marzal and Vidal 1993). This correction would find the pair {conca*t*e**n**ation, conca*m*e**r**ation} *more* similar than {cat, cot}, matching our intuition, since only 15% of the characters are different in the former pair, as opposed to 33% in the latter.

Researchers have suggested this length-normalized correction for time series, but as we will show, the correction factor is incorrect. To illustrate this, consider the following thought experiment. Imagine that some process in the system we are monitoring occasionally "injects" a pattern into the time series. As a concrete example, washing machines typically have a prototypic signature (as exhibited in the TRACE dataset (Roverso 2000)), but the signatures express themselves more slowly on a cold day, when it takes longer to heat the cooler water supplied from the city (Gisler et al. 2013). We would like all equal length instances of the signature to have approximately the same distance. As a consequence, we factorize the Euclidean distance by the following quantity: $sqrt(1/\ell)$, where ℓ is the length of the sequences. This aims to favor longer and similar sequences in the ranking process of matches that have different lengths.

In Fig. 3(*left*) we show two examples from the TRACE dataset (Roverso 2000), which will act as proxies for a variable length signature. We produced the variable lengths by down sampling. In Fig. 3(*center*), we show the distances between the patterns as their length changes. With no correction, the Euclidean distance is obviously biased to the shortest length. The length-normalized Euclidean distance looks "flatter" and suggests itself as the proper correction. However, its variation over the sequence length change is not visible due to the small scale. In Fig. 3(*right*), we show all of the measures after dividing them by their largest value. Now we can see that the length-normalized Euclidean distance has a strong bias toward the longest pattern. In contrast to the other two approaches, the $sqrt(1/length)$ correction factor provides a near perfect invariant distance over the entire range of values.

4 Proposed approach for motif discovery

Our algorithm, Variable Length Motif Discovery (VALMOD), starts by computing the *matrix profile* on the smallest subsequence length, namely ℓ_{min}, within a specified range $[\ell_{min}, \ell_{max}]$. The key idea of our approach is to minimize the work that needs to be done for subsequent subsequence lengths ($\ell_{min}+1, \ell_{min}+2, \ldots, \ell_{max}$). In Fig. 4, it can be observed that the motif of length 8 ($T_{33,8} - T_{97,8}$) has the same offsets as the motif of length 9 ($T_{33,9} - T_{97,9}$). Can we exploit this property to accelerate our computation?

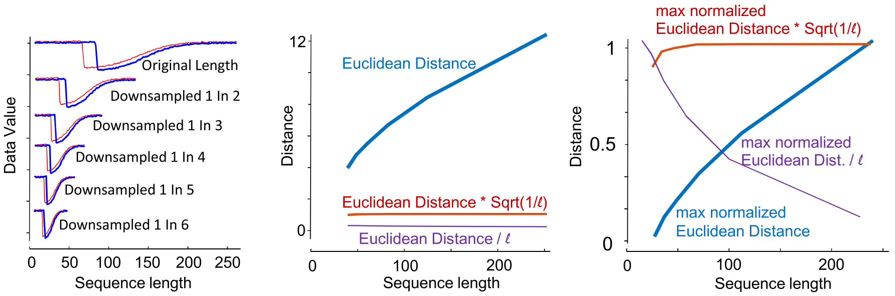


Fig. 3 (*left*) Two series from the TRACE dataset, as proxies for time series signatures at various speeds. (*center*) The classic Euclidean distance is clearly not length invariant. (*right*) After divide-by-max normalizing, it is clear that the length-normalized Euclidean distance has a strong bias toward the longest pattern

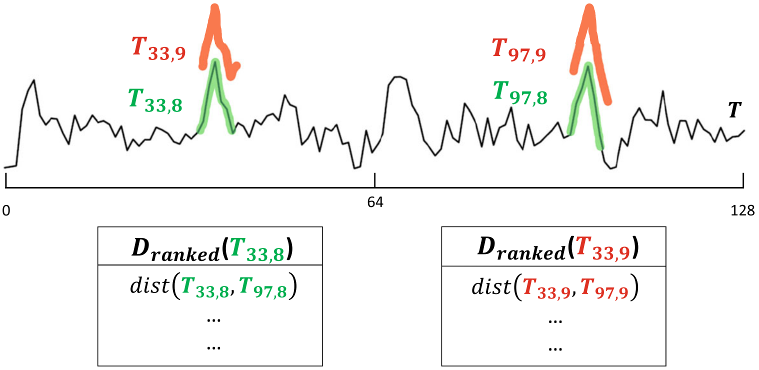


Fig. 4 (*top*) The top motifs of length 9 in an example data series have the same offsets as the top motifs of length 8. (*bottom*) The sorted distance profiles of $T_{33,8}$ and $T_{33,9}$

It seems that if the nearest neighbor of $T_{i,\ell_{min}}$ is $T_{j,\ell_{min}}$, then probably the nearest neighbor of $T_{i,\ell_{min}+1}$ is $T_{j,\ell_{min}+1}$. For example, as shown in Fig. 4(bottom), if we sort the distance profiles of $T_{33,8}$ and $T_{33,9}$ in ascending order, we can find that the nearest neighbor of $T_{33,8}$ is $T_{97,8}$, and the nearest neighbor of $T_{33,9}$ is $T_{97,9}$.

One can imagine that if the location of the nearest neighbor of $T_{i,\ell}$ ($i = 1, 2, \ldots, n-m+1$) remains the same as we increase ℓ, then we could obtain the matrix profile of length $\ell+k$ in $\mathcal{O}(n)$ time ($k = 1, 2, \ldots$). However, this is not always true. The location of the nearest neighbor of $T_{i,\ell}$ may not change as we slightly increase ℓ, if there is a substantial margin between the first and second entries of $D_{ranked}(T_{i,\ell})$. But, as ℓ gets larger, the nearest neighbor of $T_{i,\ell}$ is likely to change. For example, as shown in Fig. 5, when the subsequence length grows to 19, the nearest neighbor of $T_{33,19}$ is no longer $T_{97,19}$, but $T_{1,19}$. We observe that the ranking of the distance profile values may change, even when the data is relatively smooth. When the data is noisy and skewed, this ranking can change even more often. Is there any other rank-preserving measure that we can exploit to accelerate the computation?

The answer is yes. Instead of sorting the entries of the distance profile, we create and sort a new vector, called the *lower bound distance profile*. Figure 5(bottom) previews

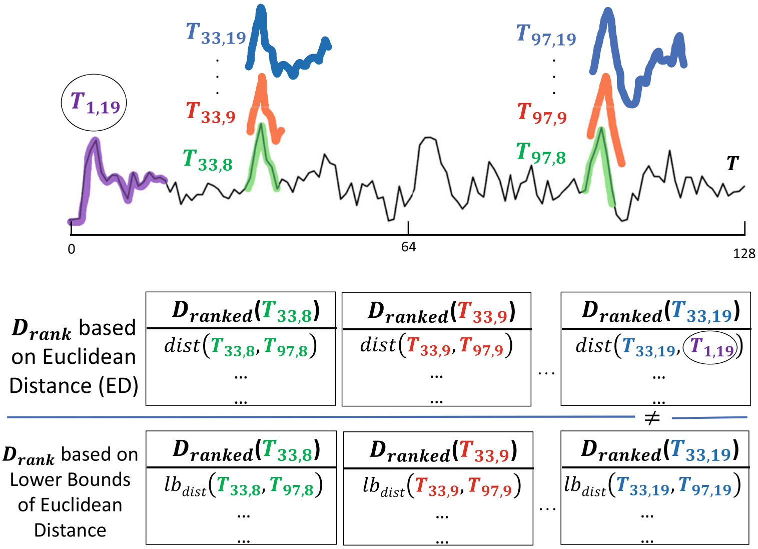

$T_{33,19}$
$T_{97,19}$
$T_{1,19}$
$T_{33,9}$
$T_{97,9}$
$T_{33,8}$
$T_{97,8}$
T
0
64
128
D_{rank} based on Euclidean Distance (ED)
$D_{ranked}(T_{33,8})$
$dist(T_{33,8}, T_{97,8})$
$D_{ranked}(T_{33,9})$
$dist(T_{33,9}, T_{97,9})$
$D_{ranked}(T_{33,19})$
$dist(T_{33,19}, T_{1,19})$
$\neq$
D_{rank} based on Lower Bounds of Euclidean Distance
$D_{ranked}(T_{33,8})$
$lb_{dist}(T_{33,8}, T_{97,8})$
$D_{ranked}(T_{33,9})$
$lb_{dist}(T_{33,9}, T_{97,9})$
$D_{ranked}(T_{33,19})$
$lb_{dist}(T_{33,19}, T_{97,19})$

Fig. 5 (*Top distance profiles*) Ranking by true distances leads to changes in the order of the pairs. (*Bottom distance profiles*) Ranking by lower bound distances maintains the same order of pairs over increasing lengths

the rank-preserving property of the lower bound distance profile. As we will describe later, once we know the distance between $T_{i,\ell}$ and $T_{j,\ell}$, we can evaluate a lower bound distance between $T_{i,\ell+k}$ and $T_{j,\ell+k}$, $\forall k \in [1,2,3,\ldots]$. The rank-preserving property of the lower bound distance profile can help us prune a large number of unnecessary computations as we increase the subsequence length.

4.1 The lower bound distance profile

Before introducing the lower bound distance profile, let us first investigate its basic element: the lower bound Euclidean distance.

Assume that we already know the z-normalized Euclidean distance $d_{i,j}^{\ell}$ between two subsequences of length ℓ: $T_{i,\ell}$ and $T_{j,\ell}$, and we are now estimating the distance between two longer subsequences of length $\ell+k$: $T_{i,\ell+k}$ and $T_{j,\ell+k}$. Our problem can be stated as follows: given $T_{i,\ell}$, $T_{j,\ell}$ and $T_{j,\ell+k}$ (but not the last k values of $T_{i,\ell+k}$), is it possible to provide a lower bound function $LB(d_{i,j}^{\ell+k})$, such that $LB(d_{i,j}^{\ell+k}) \leq d_{i,j}^{\ell+k}$? This problem is visualized in Fig. 6 .

One may assume that we can simply set $LB(d_{i,j}^{\ell+k}) = d_{i,j}^{\ell}$ by assuming that the last k values of $T_{i,\ell+k}$ are the same as the last k values of $T_{j,\ell+k}$. However, this is not an answer to our problem, as we need to evaluate z-normalized Euclidean distances, which are not simple Euclidean distances. The mean and standard deviation of a subsequence can change as we increase its length, so we need to re-normalize both $T_{i,\ell+k}$ and $T_{j,\ell+k}$. Assume that the mean and standard deviation of $T_{x,y}$ are $\mu_{x,y}$ and $\sigma_{x,y}$, respectively (i.e. $T_{j,\ell+k}$ corresponds to $\mu_{j,\ell+k}$ and $\sigma_{j,\ell+k}$). Since we do not

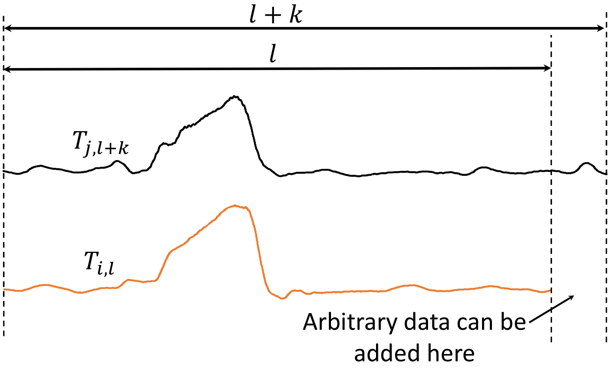


Fig. 6 Increasing the subsequence length from ℓ to $\ell + k$

know the last k values of $T_{i,\ell+k}$, both $\mu_{i,\ell+k}$ and $\sigma_{i,\ell+k}$ are unknown and can thus be regarded as variables. We recall that t_i denotes the ith point of a generic sequence T (or a subsequence $T_{a,b}$), we thus we have the following:

$$d_{i,j}^{\ell+k} \geq \min_{\mu_{i,\ell+k},\sigma_{i,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu_{i,\ell+k}}{\sigma_{i,\ell+k}} - \frac{t_{j+p-1} - \mu_{j,\ell+k}}{\sigma_{j,\ell+k}} \right)^2}$$

$$= \min_{\mu_{i,\ell+k},\sigma_{i,\ell+k}} \frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu_{i,\ell+k}}{\frac{\sigma_{i,\ell+k}\sigma_{j,\ell}}{\sigma_{j,\ell+k}}} - \frac{t_{j+p-1} - \mu_{j,\ell+k}}{\sigma_{j,\ell}} \right)^2}$$

Here, we substitute the variables $\mu_{i,\ell+k}$ and $\sigma_{i,\ell+k}$, respectively with μ' and σ'. Hence, we obtain:

$$= \min_{\mu',\sigma'} \frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} \sqrt{\sum_{p=1}^{\ell} \left(\frac{t_{i+p-1} - \mu'}{\sigma'} - \frac{t_{j+p-1} - \mu_{j,\ell}}{\sigma_{j,\ell}} \right)^2} \qquad (1)$$

Clearly, the minimum value shown in Eq. (1) can be set as $LB(d_{i,j}^{\ell+k})$. We can obtain $LB(d_{i,j}^{\ell+k})$ by solving $\frac{\partial LB(d_{i,j}^{\ell+k})}{\partial \mu'} = 0$ and $\frac{\partial LB(d_{i,j}^{\ell+k})}{\partial \sigma'} = 0$:

$$LB(d_{i,j}^{\ell+k}) = \begin{cases} \sqrt{\ell}\frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} & \text{if } q_{i,j} \leq 0 \\ \sqrt{\ell(1-q_{i,j}^2)}\frac{\sigma_{j,\ell}}{\sigma_{j,\ell+k}} & \text{otherwise} \end{cases} \qquad (2)$$

where $q_{i,j} = \frac{\sum_{p=1}^{\ell} \frac{(t_{j+p-1}t_{i+p-1})}{\ell} - \mu_{i,\ell}\mu_{j,\ell}}{\sigma_{i,\ell}\sigma_{j,\ell}}$.

$LB(d_{i,j}^{\ell+k})$ yields the minimum possible z-normalized Euclidean distance between $T_{i,\ell+k}$ and $T_{j,\ell+k}$, given $T_{i,\ell}$, $T_{j,\ell}$ and $T_{j,\ell+k}$ (but not the last k values of $T_{i,\ell+k}$). Now that we have obtained the lower bound Euclidean distance between two subsequences, we are able to introduce the lower bound distance profile.

Using Eq. (2), we can evaluate the lower bound Euclidean distance between $T_{j,\ell+k}$ and every subsequence of length $\ell+k$ in T. By putting the results in a vector, we obtain the lower bound distance profile $LB(D_j^{\ell+k})$ corresponding to subsequence $T_{j,\ell+k}$: $LB(D_j^{\ell+k}) = LB(d_{1,j}^{\ell+k})$, $LB(d_{2,j}^{\ell+k})$, ...,$LB(d_{n-\ell-k+1,j}^{\ell+k})$. If we sort the components of $LB(D_j^{\ell+k})$ in an ascending order, we can obtain the ranked lower bound distance profile: $LB_{ranked}(D_j^{\ell+k}) = LB(d_{r_1,j}^{\ell+k}), LB(d_{r_2,j}^{\ell+k}), \ldots, LB(d_{r_{n-\ell-k+1},j}^{\ell+k})$, where $LB(d_{r_1,j}^{\ell+k}) \leq LB(d_{r_2,j}^{\ell+k}) \leq \cdots \leq LB(d_{r_{n-\ell-k+1},j}^{\ell+k})$.

We would like to use this ranked lower bound distance profile to accelerate our computation. Assume that we have a best-so-far pair of motifs with a distance $dist_{BSF}$. If we examine the pth element in the ranked lower bound distance profile and find that $LB(d_{r_p,j}^{\ell+k}) > dist_{BSF}$, then we do not need to calculate the exact distance for $d_{r_p,j}^{\ell+k}, d_{r_{p+1},j}^{\ell+k}, \ldots, d_{r_{n-\ell-k+1},j}^{\ell+k}$ anymore, as they cannot be smaller than $dist_{BSF}$. Based on this observation, our strategy is as follows. We set a small, fixed value for p. Then, for every j, we evaluate whether $LB(d_{r_p,j}^{\ell+k}) > dist_{BSF}$ is true: if it is, we only calculate $d_{r_1,j}^{\ell+k}, d_{r_2,j}^{\ell+k}, \ldots, d_{r_{p-1},j}^{\ell+k}$. If it is not, we compute all the elements of $D_j^{\ell+k}$. We update $dist_{BSF}$ whenever a smaller distance value is observed. In the best case, we just need to calculate $\mathcal{O}(np)$ exact distance values to obtain the motif of length $l + k$. Note that the order of the ranked lower bound distance profile is preserved for every k. That is to say, if $LB(d_{a,j}^{\ell+k}) \leq LB(d_{b,j}^{\ell+k})$, then $LB(d_{a,j}^{\ell+k+1}) \leq LB(d_{b,j}^{\ell+k+1})$. This is because the only component in Eq. (2) related to k is $\sigma_{j,\ell+k}$. When we increase k by 1, we are just performing a linear transformation for the lower bound distance: $LB(d_{i,j}^{\ell+k+1}) = LB(d_{i,j}^{\ell+k})\sigma_{j,\ell+k}/\sigma_{j,\ell+k+1}$. Therefore, we have $LB(d_{r_p,j}^{\ell+k+1}) = LB(d_{r_p,j}^{\ell+k})\sigma_{j,\ell+k}/\sigma_{j,\ell+k+1}$, and the ranking is preserved for every k.

4.2 The VALMOD algorithm

We are now able to formally describe the VALMOD algorithm. The pseudocode for VALMOD is shown in Algorithm 1. With the call of *ComputeMatrixProfile()* in line 5, we build the matrix profile corresponding to ℓ_{min}, and in the meantime store the smallest p values of each distance profile in the memory. Note that the matrix profile is stored in the vector MP, which is coupled with the matrix profile index, IP, which is a structure containing the offsets of the nearest neighbor subsequences. We can easily find the motif corresponding to ℓ_{min} as the minimum value of MP. Then, in lines 7–16, we iteratively look for the motif of every length within ℓ_{min}+1 and ℓ_{max}. The $ComputeSubMP$ function in line 9 attempts to find the motif of length i only by evaluating a subset of the matrix profile corresponding to subsequence length i. Note that this strategy, which is based on the lower bounding technique introduced in Sect. 4.1, might not be able to capture the global minimum value within the matrix

```
Algorithm 1: $VALMOD$
   Input: DataSeries $T$, int $\ell_{min}$ int $\ell_{max}$, int $p$
   Output: $VALMP$
1  int $nDP \leftarrow |T| - \ell_{min} + 1$ ;
2  $VALMP \leftarrow$ new $VALMP(nDP)$;
3  $VALMP.MP = \{\perp, \ldots, \perp\}$;
4  MaxHeap[] $listDP$ , double [] $MP$, int [] $IP$ ;
5  $listDP$, $MP$ , $IP \leftarrow ComputeMatrixProfile(T, \ell_{min}, p)$; // listDP contains p entries of
       each distance profile
6  $VALMP \leftarrow updateVALMP(VALMP,MP,IP,nDP)$ ;
7  for $i \leftarrow \ell_{min} + 1$ to $\ell_{max}$ do
8      nDP $\leftarrow |T| - i + 1$ ;
       // compute SubMP and update listDP for the length i
9      bool bBestM, double [] $SubMP$, $IP \leftarrow ComputeSubMP(T,nDP,listDP,i,p)$;
10     if $bBestM$ then
           // SubMP surely contains the motif, update VALMP with it
11         $updateVALMP(VALMP,SubMP,IP,nDP)$;
12     else
13         $listDP,MP,IP \leftarrow ComputeMatrixProfile(T,i,p)$;
           // SubMP might not contain the motif, update VALMP computing MP
14         $updateVALMP(VALMP,MP,IP,nDP)$;
15     end
16 end
```

```
Algorithm 2: $updateVALMP$
Input: $VALMP$, double [] $MPnew$, int [] $IP$, $nDP$, $\ell$
Output: $VALMP$
for $i \leftarrow 1$ to $nDP$ do
    // length normalize the Euclidean distance
    double $lNormDist \leftarrow MPnew[i] * \sqrt{1/\ell}$;
    // if the distance at offset i of VALMP, surely computed with previous
       lengths, is larger than the actual, update it
    if $(VALMP.distances[i] > lNormDist$ or $VALMP.MP[i] == \perp)$ then
        $VALMP.distances[i] \leftarrow MPnew[i]$;
        $VALMP.normDistances[i] \leftarrow lNormDist$;
        $VALMP.lengths[i] \leftarrow \ell$;
        $VALMP.indices[i] \leftarrow IP[i]$;
    end
end
```

profile. In case that happens (which is rare), the Boolean flag $bBestM$ is set to false, and we compute the whole matrix profile with the $computeMatrixProfile$ procedure in line 13.

The final output of $VALMOD$ is a vector, which is called $VALMP$ (*variable length matrix profile*) in the pseudo-code. If we were interested in only one fixed subsequence length, *VALMP* would be the matrix profile normalized by the square root of the subsequence length. If we are processing various subsequence lengths, then as we increase the subsequence length, we update *VALMP* when a smaller length-normalized Euclidean distance is observed.

Algorithm 2 shows the routine to update the $VALMP$ structure. The final $VALMP$ consists of four parts. The ith entry of the $normDistances$ vector stores the smallest length-normalized Euclidean distance values between the ith subsequence and its nearest neighbor, while the ith place of vector $distances$ stores their straight Euclidean

Algorithm 3: *ComputeMatrixProfile*

```
   Input: DataSeries $T$, int $\ell$, int $p$
   Output: $MP$, $listDP$
1  int $nDP \leftarrow |T|$-$\ell$+1;
2  double [] $MP \leftarrow$ double [$nDP$];
3  int [] $IP \leftarrow$ int [$nDP$];
4  MaxHeap[] $listDP$= new MaxHeap(p)[$nDP$];
   // compute the dot product vector QT for the first distance profile
5  double [] $QT \leftarrow SlidingDotProduct(T_{1,\ell}, T)$;
   // compute sum and squared sum of the first subsequence of length $\ell$
6  $s \leftarrow sum(T_{1,\ell})$; $ss \leftarrow squaredSum(T_{1,\ell})$;
   // compute the first distance profile with distance formula (Eq.(3)) and
      store the minimum distance in MP and the offset of the nearest neighbor
      in IP
7  D($T_{i,\ell}$) $\leftarrow CalcDistProfile(QT, T_{i,\ell}, T, s, ss)$;
8  $MP[1], IP[1] \leftarrow$ min(D($T_{i,\ell}$));
   // iterate over the subsequences of T
9  for $i \leftarrow 2$ to $nDP$ do
       // update the dot product vector QT for the $i$th subsequence
10     for $j \leftarrow nDP$ down to $2$ do
11         $QT[j] \leftarrow QT[j-1] - T[j-1] \times T[i-1] + T[j+\ell-1] \times T[i+\ell-1]$ ;
12     end
       // update sum and squared sum of the $i$th subsequence
13     $s \leftarrow s - T[i-1] + T[\ell+i-2]$;
14     $ss \leftarrow ss - T[i-1]^2 + T[\ell+i-2]^2$;
15     D($T_{i,\ell}$) $\leftarrow CalcDistProfile(QT, T_{i,\ell}, T, s, ss)$;
16     $MP[i], IP[i] \leftarrow$ min($D(T_{i,\ell})$);
       // Store in listDP[i] the p entries e with smallest lower bounding
          distance
17     int c $\leftarrow 0$;
18     for each entry $e$ in $D(T_{i,\ell})$ do
           // Compute the lower bound for the length $\ell+1$
19         $e.LB \leftarrow compLB(\ell, \ell+1, QT[c], e.s1, e.s2, e.ss1, e.ss2)$;
           // save the entry only if is smaller than the max lb so far or if
              listDP[i] contains fewer than p elements
20         if $e.LB < max(listDP[i])$ or $|listDP[i]| < p$ then
21             $insert(listDP[i], e)$;
22         end
23         c$\leftarrow c+1$;
24     end
25 end
```

distance. The location of each subsequence's nearest neighbor is stored in the vector *indices*. The structure *lengths* contains the length of the ith subsequences pair.

In the next two subsections, we detail the two sub-routines, *computeMatrixProfile* and the *ComputeSubMP*.

4.3 Computing the matrix profile

The routine *ComputeMatrixProfile* (Algorithm 3) computes a matrix profile for a given subsequence length, ℓ. It essentially follows the STOMP algorithm (Zhu et al. 2016), except that we also calculate the lower bound distance profiles in line 18. In line 5, the dot product between the sequence $T_{1,\ell}$ and the others in T is computed in *frequency domain* in $\mathcal{O}(nlogn)$ time, where $n = |T|$. The dot product is computed in constant time in line 11 by using the result of the previous overlapping subsequences.

In line 7 we measure each z-normalized Euclidean distance, between $T_{i,\ell}$ and the other subsequence of length ℓ in T, avoiding trivial matches. The distance measure formula used is the following (Mueen et al. 2014; Yeh et al. 2016; Zhu et al. 2016):

$$dist(T_{i,\ell}, T_{j,\ell}) = \sqrt{2\ell(1 - \frac{QT_{i,j} - \ell\mu_i\mu_j}{\ell\sigma_i\sigma_j})} \tag{3}$$

In Eq. (3) $QT_{i,j}$ represents the dot product of the two sub-series with offset i and j respectively. It is important to note that, we may compute μ and σ in constant time by using the *running* plain and squared sum, namely s and ss (initialized in line 6). It follows that $\mu = s/\ell$ and $\sigma = \sqrt{(ss/\ell) - \mu^2}$.

In lines 8 and 16, we update both the matrix profile and the matrix profile index, which holds the offset of the closest match for each $T_{i,l}$.

Algorithm 3 ends with the loop in line 18, which evaluates the lower bound distance profile and stores the p smallest lower bound distance values in $listDP$. In line 19, the procedure $compLB$ evaluates the lower bound distance profile introduced in Sect. 4.1 using Eq. (2). The structure $listDP$ is a Max Heap with a maximum capacity of p. Each entry e of the distance profile in line 18 is a tuple containing the Euclidean distance between a subsequence $T_{j,\ell}$ and its nearest neighbor, the location of that nearest neighbor, the lower bound Euclidean distance of the pair, the dot product of them, and the plain and squared sum of $T_{j,\ell}$. In Fig. 7b, we show an example of the distance profile in line 18. The distance profile is sorted according to the lower bound Euclidean distance values (shown as LB in the figure). The entries corresponding to the p smallest LB values are stored in memory to be reused for longer motif lengths.

We note that this routine is called at least once, for the first subsequence length of the range, namely $\ell = \ell_{min}$. In the worst case, it is executed for each length in the range.

Complexity analysis In line 15 of Algorithm 3, the time cost to compute a single distance profile is $\mathcal{O}(n)$, where n is the number of subsequences of length ℓ. Therefore computing the n distance profiles takes $\mathcal{O}(n^2)$ time. In line 18, computing the lower bounds of the smallest p entries of each distance profile takes $\mathcal{O}(n \log(p))$ additional time. The overall time complexity of the $ComputeMatrixProfile$ routine is thus $\mathcal{O}(n^2 \log(p))$.

4.4 Matrix profile for subsequent lengths

We are now ready to describe our *ComputeSubMP* algorithm, which allows us to find the motifs for subsequence lengths greater than ℓ in linear time.

The input of *ComputeSubMP*, whose pseudo-code is shown in Algorithm 4, is the vector $listDp$ that we built in the previous step. In line 5, we start to iterate over the $p \times n$ elements of $listDp$ in order to find the motif pair of length $newL$, using a procedure that is faster than Algorithm 1, leading to a complexity that is now linear in the best case. Since $listDP$ potentially contains enough elements to compute the whole matrix profile, it can provide more information than just the motif pair.

Algorithm 4: *ComputeSubMP*

```
    Input: DataSeries $T$, int $nDp$, MaxHeap[] $listDP$, int $newL$, int $p$
    Output: bBestM, $SubMP$, $IP$
 1  double[] $SubMP \leftarrow$ double[$nDp$];
 2  int[] $IP \leftarrow$ int[$nDp$];
 3  double $minDistAbs \leftarrow$ inf, double $minLbAbs \leftarrow$ inf;
 4  List ⟨ int,double ⟩ $nonValidDP$;
    // iterate over the partial distance profiles in listDP
 5  for $i \leftarrow 1$ to $nDp$ do
 6      double $minDist \leftarrow$ inf;
 7      int $ind \leftarrow 0$;
 8      double $maxLB \leftarrow$ popMax($listDP[i]$);
        // update the partial distance profile for the length newL (true
           Euclidean and lower bounding distance )
 9      for each entry $e$ in $listDP[i]$ do
10          $e.dist$, $e.LB \leftarrow updateDistAndLB(e, newL)$;
11          $minDist \leftarrow$ min($minDist$,$e.dist$);
12          if $minDist == e.dist$ then
13              $ind = e.offset$;
14          end
15      end
        // check if the min (minDist) of this partial distance profile is the min
           of the complete distance profile
16      if $minDist < maxLB$ then
            // minDist is the real min; valid distance profile
17          $minDistABS \leftarrow$ min($minDistAbs$,$minDist$);
18          $SubMP[i] = minDist$;
19          $IP[i] = ind$;
20      else
            // minDist is not the real min; non-valid distance profile
21          $minLbAbs \leftarrow$ min($minLbAbs$, $maxLB$));
22          $SubMP[i] = \perp$;
23          $nonValidDP.add(\langle i, maxLB \rangle)$
24      end
25  end
26  bool $bBestM \leftarrow (minDistABS < minLbAbs)$ ;
    // if SubMP does not contain the motif distance (bBestM = false), compute
       the whole non-valid distance profiles, if it is faster then
       computeMatrixProfile (nDp / 2 = true)
27  if $!bBestM$ and $nonValidDP.size() < (\frac{n \log(p)}{\log(n)})$ then
28      for each pair $< ind, lbMax >$ in $nonValidDP$ do
29          if $lbMax < minDistABS$ then
30              $QT \leftarrow SlidingDotProduct(T_{ind,\ell}, T)$;
31              double $s \leftarrow sum(T_{ind,\ell})$; double $ss \leftarrow squaredSum(T_{ind,\ell})$;
32              $D(T_{ind,\ell}) \leftarrow CalcDistProf(QT, T_{ind,\ell}, T, s, ss)$;
33              $SubMP[ind]$, $IP[ind]$ = min($D(T_{ind,\ell})$);
34              $insert(listDP[ind], D(T_{ind,\ell}))$;
35          end
36      end
37      $bBestM \leftarrow 1$;
38  end
```

In the loop of line 9, we update all the entries of $listDP[i]$ by computing the Euclidean and lower bound distance for the length $newL$. This operation is valid, since the ranking of each $listDP[i]$ is maintained as the lower bound gets updated. Moreover, this latter computation is done in constant time (line 10), since the entries contain the statistics (i.e. sum, squared sum, dot product) for the length $newL - 1$. Also note that the routine $updateDistAndLB$ avoids the trivial matches, which may result from the length increment.

Subsequently, the algorithm checks in line 16 if $minDist$ is smaller than or equal to $maxLB$, the largest lower bound distance value in $listDP[i]$. If this is true, $minDist$ is the smallest value in the whole distance profile. In lines 17 and 18, we update the best-so-far distance value and the matrix profile. On the other hand, we update the smallest max lower bounding distance in line 21, recording also that we do not have the true min for the distance profile with offset i (line 23). Here, we may also note that even though the local true min is larger than the max lower bound (i.e., the condition of line 16 is not true), $minDist$ may still represent an approximation of the true matrix profile point.

When the iteration of the partial distance profiles ends (end of for loop in line 5), the algorithm has enough elements to know if the matrix profile computed contains the real motif pair. In line 26, we verify if the smallest Euclidean distance we computed ($minDistABS$) is less than $minLbAbs$, which is the minimum lower bound of the *non-valid* distance profiles. We call non-valid all the partial distance profiles, for which the maximum lower bound distance (i.e., the p-th largest lower bound of the distance profile) is smaller than the minimum true distance (line 20); otherwise, we call them valid (line 16).

As a result of the ranking preservation of the lower bounding function, if the above criterion holds, we know that each true Euclidean distance in the non-valid distance profiles must be greater than $minDistABS$. In line 27, the algorithm has its last opportunity to exploit the lower bound in the distance profiles, in order to avoid computing the whole matrix profile. If $bBestM$ is false (the motif has not been found), we start to iterate through the non-valid distances profiles. We perform this iteration, when their number is not larger than $\frac{n \log(p)}{\log(n)}$. This condition guarantees that Algorithm 4 is faster than Algorithm 3.

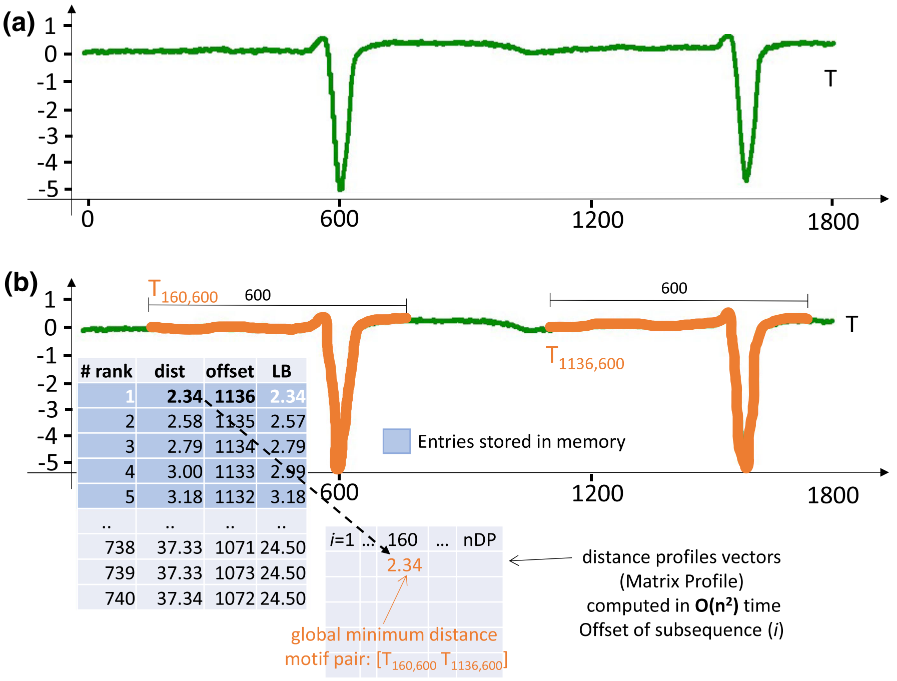
We present here two examples that explain the main procedures of $VALMOD$.

Example 1 In Fig. 7, we show a snapshot of a VALMOD run. In Fig. 7a, VALMOD receives as input a data series of length 1800. In Fig. 7b, the matrix profile for subsequence length $\ell = 600$ is computed (Algorithm 3). On the left, we depict the distance profile regarding $T_{160,600}$, and rank it according to the lower bound (LB) distance values. Although we are computing the entire distance profile, we store only the first $p = 5$ entries in memory.

Example 2 Figure 8 shows the execution of $ComputeSubMP$ (Algorithm 4), taking place after the step illustrated in Fig. 7b. In this picture, we show the distance profile of a subsequence belonging to the motif pair, for subsequence length $\ell = 601$. This time it is built by computing $p = 5$ distances (left side of the picture). We can now make the following observations:

(a) In the distance profile of the subsequence $T_{160,601}$ (left array): $minDist = 2.34 < maxLB = 3.18 \iff$ the value 2.34 is both a local and a global minimum (among all the distance profiles).

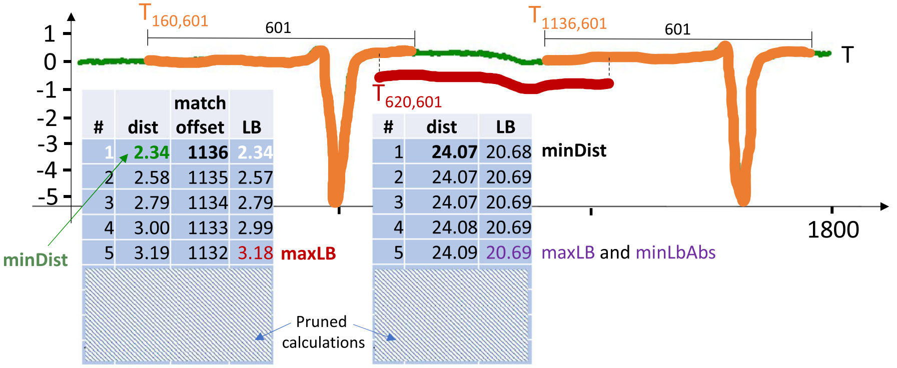
(b) Considering the partial distance profile of subsequence $T_{620,601}$ (right array), we do not know if its $minDist$ is its real global minimum, since 20.69 ($maxLB$) < 24.07 ($minDist$).



# rank	dist	offset	LB
1	2.34	1136	2.34
2	2.58	1135	2.57
3	2.79	1134	2.79
4	3.00	1133	2.99
5	3.18	1132	3.18
..	..	..	..
738	37.33	1071	24.50
739	37.33	1073	24.50
740	37.34	1072	24.50

i=1	…	160	…	nDP
		2.34		

Fig. 7 **a** Input time series, **b** Compute matrix profile snapshot: (on the left) distance profile of the subsequence $T_{160,600}$ which is part of the motif



#	dist	match offset	LB
1	2.34	1136	2.34
2	2.58	1135	2.57
3	2.79	1134	2.79
4	3.00	1133	2.99
5	3.19	1132	3.18

#	dist	LB
1	24.07	20.68
2	24.07	20.69
3	24.07	20.69
4	24.08	20.69
5	24.09	20.69

Fig. 8 Compute Sub Matrix profile: the partial distance profile of $T_{160,601}$ contains the motif's subsequences distance

(c) We know, that 20.69 ($maxLB$ of the distance profile of subsequence $T_{620,601}$) is the $minLbAbs$, or in other words, the smallest $maxLB$ distance among all the partial distance profiles in which $maxLB < minDist$ holds.

(d) We know that there are no true Euclidean distances (among those computed) smaller than 2.34. Since $minDist = 2.34 < minLbAbs = 20.69 \iff 2.34$ is the distance of the motif $\{T_{160,601}; T_{1136,601}\}$.

Complexity analysis In the best case, $ComputeSubMP$ can find the motif pair in $\mathcal{O}(np)$ time, where n is the total number of distance profiles. This means that no distance profile computation takes place, since the condition in line 26 of Algorithm 5 is satisfied. Otherwise, if we need to iterate over the non-valid distance profiles for finding the answer, the time complexity reaches its worst case, $\mathcal{O}(nC\log(n))$, with C denoting the number of non-valid distance profiles that are recomputed. When $C < \frac{n\log(p)}{\log(n)}$, the algorithm is asymptotically faster than re-executing *ComputeMatrixProfile*, which takes $\mathcal{O}(n^2\log(p))$ time.

Note that, each non-valid distance profile (starting in line 30) is computed by using the primitives introduced in the *ComputeMatrixProfile* algorithm, only if its maximum lower bound is less than the smallest true distance $minDistABS$. This indicates that the distance profile for length $newL$ may contain not yet computed distances smaller than $minDistABS$, which is our *best-so-far*. Therefore, the overall complexity of VALMOD is $\mathcal{O}(n^2\log(p) + (\ell_{max} - \ell_{min})np)$ in the best case, whereas the worst case time complexity is $\mathcal{O}((\ell_{max} - \ell_{min})n^2\log(p))$. Clearly, the $n^2\log(p)$ factor dominates, since $(\ell_{max} - \ell_{min})$ acts as a constant.

5 Finding motif sets

We finally extend our technique in order to find the variable-length motif sets. In that regard, we start to consider the top-k motif pairs, namely the pairs having the k smallest length-normalized distances. The idea is to extend each motif pair to a motif set considering the subsequence's proximity as a quality measure, thus favoring the motif sets, which contain the closest subsequence pairs. Moreover, for each top-K motif pair ($T_{a,\ell}$,$T_{b,\ell}$), we use a radius $r = D * dist(T_{a,\ell}, T_{b,\ell})$, when we extend it to a motif set. We call the real variable D *radius factor*. This choice permits us to tune the radius r by the user defined radius factor, considering also the characteristics of the data. Setting a unique and non data dependent radius for all motif sets, would penalize the results of exploratory analysis.

First, we introduce Algorithm 5, a slightly modified version of the $updateValmp$ routine (Algorithm 2). The new algorithm is called $updateVALMPForMotifSets$, and its main goal is to keep track of the best k subsequence pairs (motif pairs) according to the $VALMP$ ranking, and the corresponding partial distance profiles. The idea is to later exploit the lower bounding distances for pruning computations, while computing the motif sets.

In lines 4 to 7, we build a structure named $pair$, which carries the information of the subsequences pairs that appear in the $VALMP$ structure. During this iteration, we leave the fields $partDP1$ and $partDP2$ empty, since they will be later initialized with the partial distance profiles, if their $pair$ is in the top k of $VALMP$. In order to enumerate the best k pairs, we use the global maximum heap $heapBestKPairs$ in line 8. Then, we assign (or update) the corresponding partial distance profiles (line 15) to each pair.

We are now ready to present the variable length motif sets discovery algorithm (refer to Algorithm 6). Starting at line 1, the algorithm iterates over the best pairs. For each

Algorithm 5: *updateVALMPForMotifSets*

```
   Input: $VALMP$, double [] $MPnew$, int [] $IP$, $nDP$, $\ell$, MaxHeap[] $listDP$,
   Output: $VALMP$
1  for $i \leftarrow 1$ to $nDP$ do
       // length normalize the Euclidean distance
2      double $lNormDist \leftarrow MPnew[i] * \sqrt{1/\ell}$;
       // if the distance at offset i of VALMP, surely computed with previous
          lengths, is larger than the actual, update it
3      if ($VALMP.distances[i] > lNormDist$ or $VALMP.MP[i] == \perp$) then
4          entry $pair$;
5          $pair.off1 \leftarrow$ i, $pair.off2 \leftarrow IP[i]$ ;
6          $pair.distance \leftarrow MPnew[i]$, $pair.\ell \leftarrow \ell$;
7          $pair.partDP1 \leftarrow \perp$, $pair.partDP2 \leftarrow \perp$ ;
8          $insert(heapBestKPairs, pair)$;
9          $VALMP.distances[i] \leftarrow MPnew[i]$;
10         $VALMP.normDistances[i] \leftarrow lNormDist$;
11         $VALMP.lengths[i] \leftarrow \ell$;
12         $VALMP.indices[i] \leftarrow IP[i]$;
13     end
14 end
15 for each $pair$ in $heapBestKPairs$ do
16     if ($pair.partDP1 == \perp$) then
17         $pair.partDP1 \leftarrow listDP[pair.off1]$;
18         $pair.partDP2 \leftarrow listDP[pair.off2]$;
19     end
20 end
```

one of those, we need to check if the search range is smaller than the maximum lower bound distances of both partial distance profiles. If this is true, we are guaranteed to have already computed all the subsequences in the range. Therefore, in lines 7 and 14 we filter the subsequences in the range, sorting the partial distance profile according to the offsets. This operation will permit us to find the trivial matches in linear time.

On the other hand, if the search range is larger than the maximum lower bound distances of both partial distance profiles, we have to re-compute the entire distance profile (lines 11 and 18), to find all the subsequences in the range. Once we have the distance profile pairs, we need to merge them and remove the trivial matches (line 20). Each time we add a subsequence in a motif set, we remove it from the search space: this guarantees the empty intersection among the sets in S^*.

Complexity analysis The complexity of the *updateVALMPForMotifSets* algorithm is $\mathcal{O}(n \log(k))$, where n is the length of the $VALMP$ structure, which is linearly scanned and updated. $\mathcal{O}(\log(k))$ time is needed to retain the k best pairs of $VALMP$, using the heap structure in line 8. The final algorithm *computeVarLengthMotifSets* takes $\mathcal{O}(k \times p \times \log(p))$ time, in the best case. This occurs when, after iterating the k pairs in *heapBestKPairs*, each partial distance profile of length p, contains all the elements in the range r. In this case, we just need an extra $\mathcal{O}(p \log(p))$ time to sort its elements (line 7 and 14). On the other hand, the worst case time is bounded by $\mathcal{O}(k \times n \times \log(n))$, where n is the length of the input data series T. In this case, the algorithm needs to recompute k times the entire distance profile (line 11 and 18), at a unit cost of $\mathcal{O}(n \log(n))$ time.

Algorithm 6: $computeVarLengthMotifSets$

```
Input: DataSeries $T$, MaxHeap $heapBestKPairs$, double $D$
Output: Set $S^*$
for each $pair$ in $heapBestKPairs$ do
    double $r \leftarrow pair.distance * D$ ;
    double $maxLB1 \leftarrow$ popMax($pair.partDP1$);
    double $maxLB2 \leftarrow$ popMax($pair.partDP2$);
    $D(T_{pair.off1,pair.\ell}) \leftarrow \emptyset$, $D(T_{pair.off2,pair.\ell}) \leftarrow \emptyset$ ;
    if $maxLB1 > r$ then
        // sort according the offset, the partial distance profile contains
           all the elements in the range
        $D(T_{pair.off1,pair.\ell}) \leftarrow sortAndFilterRange(r, pair.partDP1$.toVector());
    else
        // re-compute the mat
        double $s \leftarrow sum(T_{ind,\ell})$;
        double $ss \leftarrow squaredSum(T_{ind,\ell})$;
        $D(T_{pair.off1,pair.\ell}) \leftarrow CalcDistProfInRange(r, QT, T_{pair.off1,pair.\ell}, T, s, ss)$;
    end
    if $maxLB2 > r$ then
        $D(T_{pair.off2,\ell}) \leftarrow sortAndFilterRange(r, pair.partDP2$.toVector());
    else
        double $s \leftarrow sum(T_{ind,\ell})$;
        double $ss \leftarrow squaredSum(T_{ind,\ell})$;
        $D(T_{pair.off2,pair.\ell}) \leftarrow CalcDistProfInRange(r, QT, T_{pair.off1,pair.\ell}, T, s, ss)$;
    end
    Set $S_r^{pair.\ell} \leftarrow mergeRemoveTM(D(T_{pair.off1,\ell}), D(T_{pair.off2,\ell}))$;
    $S^*.add(S_r^{pair.\ell})$;
end
```

6 Discord discovery

We now describe our approach to solving the Variable-Length *Top-k mth*Discord Discovery problem. First, we explain some useful notions, and we then present our discord discovery algorithm.

6.1 Comparing discords of different lengths

Before introducing the algorithm that identifies discords (from the Top-1 1st to the *Top-k mth*one), we define the data structure that allows us to accommodate them. We can represent this structure as a $k \times m$ matrix, which contains the best match distance and the offset of each discord.

More formally, given a data series T, and a subsequence length ℓ we define: $dkm_\ell = \begin{bmatrix} \langle d, o \rangle_{1,1} & \ldots & \langle d, o \rangle_{1,m} \\ \ldots & \ldots & \ldots \\ \langle d, o \rangle_{k,1} & \ldots & \langle d, o \rangle_{k,m} \end{bmatrix}$, where a generic pair $\langle d, o \rangle_{i,j}$ contains the offset o and the corresponding distance d of the Top-i jth discord of length ℓ ($1 \leq i \leq k$ and $1 \leq j \leq m$). In dkm_ℓ, rows rank the discords according to their positions (mth discords), and the columns according to their best match distance (Top-k). For each pair $\langle d, o \rangle_{a,b}, \langle d', o' \rangle_{a',b'} \in dkm_\ell$, we require that $T_{o,\ell}$ and $T_{o',\ell}$ are not trivial matches.

Since we want to compute dkm_ℓ for each length in the range $[\ell_{min}, \ell_{max}]$, we also need to rank discords of different lengths. In that regard, we want to obtain a unique matrix that we denote by $dkm_{\ell_{min},\ell_{max}}$. Therefore, we can represent a discord by the triple $\langle d^*, o^*, \ell^* \rangle_{i,j} \in dkm_{\ell_{min},\ell_{max}}$, where d^* is the ith greatest length normalized jth best match distance. More formally:

$$d^* = \max\left\{ \frac{d}{\sqrt{\ell min}} : d \in dkm_{\ell_{min}}[i][j]), \ldots, \frac{d}{\sqrt{\ell max}} : d \in dkm_{\ell_{max}}[i][j] \right\}$$

Each triple is also composed by the offset o^* and the length ℓ^* of the discord, where $\ell_{min} \leq \ell^* \leq \ell_{max}$.

As in the case of motifs discovery, we length-normalize the discord distances, while constructing the $dkm_{\ell_{min},\ell_{max}}$ ranking. Thus, we multiply each distance by the $1/\sqrt{\ell}$ factor. In this case, the length normalization aims to favor the selection of shorter discords. Therefore, if we compare two *Top-k mth*discord subsequences of different lengths, but equal best match distances, the shorter subsequence is the one with the highest point-to-point dissimilarity to its best match. This is guaranteed by dividing each distance by the discord length. Consequently, we promote the shorter subsequence as the more anomalous one.

6.2 Discord discovery algorithm

We now describe our algorithm for the *Top-k mth*discords discovery problem. We note that we can still use the lower bound distance measure, as in the motif discovery case. This allows us to efficiently build dkm_{ℓ}, for each ℓ in the $[\ell_{min}, \ell_{max}]$ range, incrementally reusing the distances computation performed. The final outcome of this procedure is the $dkm_{\ell_{min},\ell_{max}}$ matrix, which contains the variable length discord ranking. In this part, we introduce and explain the algorithms, which permit us to efficiently obtain dkm_{ℓ} for each length. We report the whole procedure in Algorithm 7.

Smallest length discords We start to find discords of length ℓ_{min}, namely the smallest subsequence length in the range. We can thus run Algorithm 3 in line 1, which computes the list of partial distance profiles of each subsequence of length ℓ_{min} ($listDP$), in the input data series T. Each partial distance profile contains the p smallest nearest neighbor distances of each subsequence. To that extent, we set $p \geq m$ in Algorithm 3 ($ComputeMatrixProfile$).

We then iterate the subsequences of T in line 6, using the index i. For each subsequence $T_{i,\ell min}$ that has no trivial matches in $dkm_{\ell_{min}}$, we invoke the routine $UpdateFixedLengthDiscords$ (line 8), which checks if $T_{i,\ell min}$ can be placed in $dkm_{\ell_{min}}$ as a discord. When $dkm_{\ell_{min}}$ is built, we update the variable length discords ranking ($dkm_{\ell_{min},\ell_{max}}$ matrix in line 11), using the procedure $UpdateVariableLengthDiscords$.

In the loop of line 12, we iterate the discord lengths greater than ℓ_{min}. Since we want to prune the search space, we consider the list of distance profiles in $listDP$, which also contains the lower bound distances of the p ($p > m$) nearest neighbors of each subsequence. In that regard, we invoke the routine $Topkm\_nextLength$ (line 15). Before we introduce the details, we describe the two routines we introduced, which allow to rank the discords.

Algorithm 7: *Topkm_DiscordDiscovery (Compute Top-k mthDiscords of variable lengths)*

```
Input: DataSeries $T$, int $\ell_{min}$, int $\ell_{max}$, int $k$, int $m$ , int $p$
Output: Matrix $dkm_{\ell_{min},\ell_{max}}$
MaxHeap[] $listDP$=$ComputeMatrixProfile(T, \ell_{min}, p)$;
int $nDp = (|T| - \ell_{min}) + 1$;
Matrix $dkm_{\ell_{min},\ell_{max}} = \{\{\langle -\infty, -\infty, -\infty\rangle, \ldots, \langle -\infty, -\infty, -\infty\rangle\}, \ldots, \{\ldots\}\}$;
Matrix $dkm_{\ell_{min}} = \{\{\langle -\infty, -\infty\rangle, \ldots, \langle -\infty, -\infty\rangle\}, \ldots, \{\ldots\}\}$;
if $p >= m$ then
    // iterate the partial distance profiles in listDP
    // and compute $dkm_{\ell_{min}}$
    for $i \leftarrow 1$ to $nDp$ do
        if $T_{i,\ell_{min}}$ has no Trivial matches in $dkm_{\ell_{min}}$ then
            $UpdateFixedLengthDiscords(dkm_{\ell_{min}}, listDP[i], i, k, m)$;

    end
    $UpdateVariableLengthDiscords(dkm_{\ell_{min}}, dkm_{\ell_{min},\ell_{max}}, k, m)$;
    // compute $dkm_{\ell_{nextL}}$ for each length, pruning distance
       computations
    for $nextL \leftarrow \ell_{min} + 1$ to $\ell_{max}$ do
        Matrix $dkm_{nextL} = \{\{\langle -\infty, -\infty\rangle, \ldots, \langle -\infty, -\infty\rangle\}, \ldots, \{\ldots\}\}$;
        $nDp = (|T| - nextL) + 1$;
        $dkm_{\ell_{nextL}}$=$Topkm\_nextLength(T, nDp, listDP, nextL, k, m)$;
        $UpdateVariableLengthDiscords(dkm_{\ell_{nextL}}, dkm_{\ell_{min},\ell_{max}}, k, m)$;
    end
end
```

Ranking fixed length discords In algorithm 8, we report the pseudo-code of the routine $UpdateFixedLengthDiscords$. This algorithm accepts as input the matrix dkm_{ℓ} to update, and a partial distance profile of the subsequence with offset off. It starts iterating the rows of $dkm_{\ell_{min}}$ in reverse order (line 1). This is equivalent to considering the discords from the mth one to the 1st. Hence, at each iteration we get the jth nearest neighbor of $T_{off,\ell_{min}}$ from its partial distance profile in line 2. Subsequently, the loop in line 3 checks if the $jthdist$ is among the k largest ones in the jth column of $dkm_{\ell_{min}}$. If it is true, the smallest elements in the column are shifted (line 6) and $T_{off,\ell_{min}}$ is inserted as the Top-i jth discord (line 7).

Ranking variable length discords Once we dispose of the matrix dkm_{ℓ}, we can invoke the procedure $UpdateVariableLengthDiscords$ for each length $\ell \in \{\ell_{min}, \ldots, \ell_{max}\}$ (Algorithm 9), in order to incrementally produce the final variable length discord ranking we store in $dkm_{\ell_{min},\ell_{max}}$. This algorithm accepts as input and iterates over the matrix $dkm_{\ell_{min},\ell_{max}}$. A position (discord) is updated if the length normalized best match distance of the discord in the same position of dkm_{ℓ} is larger (line 6).

Greater length discords In Algorithm 10, we show the pseudo-code of the routine $Topkm\_nextLength$. It starts performing the same loop of line 9 in Algorithm 1, iterating over the partial distance profiles (line 3), and updating the true Euclidean distances for the new length ($newL$) and the lower bounds (line 9) for the subsequent

Algorithm 8: *UpdateFixedLengthDiscords* (*Update dkm_ℓ*)

```
Input: Matrix $dkm_\ell$, MaxHeap $minMDist$, int $off$, int $k$, int $m$
for $j \leftarrow m$ down to 1 do
    double $jthdist \leftarrow minMDist.getMax(j)$;
    for $i \leftarrow 1$ to $k$ do
        $< d, o >_{i,j} = dkm_{newL}[i][j]$;
        if $jthdist > d$ then
            $shiftRankingTopK(dkm_{newL}[i][j])$;
            // update the ranking with the new Top-i jth discord $T_{off,\ell}$
            $dkm_\ell[i][j] \leftarrow \langle jthdist, off \rangle$;
            return;

    end
end
```

Algorithm 9: *UpdateVariableLengthDiscords* (*Update $dkm_{\ell_{min},\ell_{max}}$*)

```
Input: Matrix $dkm_{\ell_{min},\ell_{max}}$, Matrix $dkm_\ell$, int $k$, int $m$
for $i \leftarrow 1$ to $k$ do
    for $j \leftarrow 1$ to $m$ do
        $< d, o >_{i,j} = dkm_{newL}[i][j]$;
        $< d^*, o^*, l^* >_{i,j} = dkm_{\ell_{min},\ell_{max}}[i][j]$;
        // if length normalized distance is greater or equal for
           length $\ell$, update the rank.
        if $((d/\sqrt{\ell}) >= d^*)$ then
            $dkm_{\ell_{min},\ell_{max}}[i][j] = \langle (d/\sqrt{\ell}), o, \ell \rangle$

    end
end
```

length ($newL+1$). Since we need to know the distances from each subsequence to their m nearest neighbors, for each subsequence $T_{i,newL}$ that does not have trivial matches in dkm_{newL}, we check if the mth smallest distance is smaller than the maximum lower bound in the partial distance profile (line 14). If this is true, we have the guarantee that the partial distance profile $minMDist$ contains the exact m nearest neighbor Euclidean distances. Hence, in line 15, we can update the matrix dkm_{newL}. On the other hand, if the distances are not verified to be correct, we keep $minMDist$ in memory, which becomes a non-valid partial distance profile, along with the offset of the corresponding subsequence (line 17). Once we have considered all the partial distance profiles, we need to iterate the non-valid partial distance profiles (line 21).

We therefore recompute those that contain at least one true Euclidean distance greater than the distances in the last row of dkm_{newL}. The correctness of this choice is guaranteed by the fact that the distances of a non-valid partial distance profile can be only larger than the non-computed ones. Hence, if the condition of line 25 is not verified, no updates in dkm_{newL} can take place. Otherwise, we recompute the non-valid distance profile starting at line 26 from scratch. Note that when we re-compute a distance profile, we globally update the corresponding position of the partial distance profiles $listDP$ (line 29) and dkm_{newL} in the vector as well (line 30).

Algorithm 10: $Topkm\_nextLength$ (Compute *Top-k* mthDiscords of greater lengths)

```
Input: DataSeries $T$, int $nDp$, MaxHeap[] $listDP$, int $newL$, int $k$, int $m$, int $p$
Output: Matrix $dkm_{newL}$
Matrix $dkm_{newL} = \{\{\langle -\infty, -\infty\rangle, \ldots, \langle -\infty, -\infty\rangle\}, \ldots, \{\ldots\}\}$;
List ⟨MaxHeap,int⟩ $nonValidMindistList$;
// iterate over the partial distance profiles in listDP
for $i \leftarrow 1$ to $nDp$ do
    MaxHeap $minMDist \leftarrow$ new MaxHeap($p$);
    double $minDist \leftarrow$ inf;
    int $ind \leftarrow 0$;
    double $maxLB \leftarrow$ popMax($listDP[i]$);
    /* update the partial distance profile for the length newL (true
       Euclidean and lower bounding distance )                              */
    for each entry $e$ in $listDP[i]$ do
        $e.dist, e.LB \leftarrow updateDistAndLB(e, newL)$;
        // the m shortest neighbor distances are stored in minMDist
        $minMDist$.push($e.dist$);
    end
    // check if the $m$th shortest distance of this partial distance profile is
       the true $m$th shorthest.
    $mDist = minMDist.getMax(m)$;
    if $T_{i,newL}$ has no Trivial matches in $dkm_{newL}$ then
        if $mDist < maxLB$ then
            /* the discord ranking can be updated, without computing the whole
               distance profile                                             */
            $UpdateFixedLengthDiscords(dkm_{newL}, minMDist, i, k, m)$;
        else
            /* minMDist might not be exact, store the partial distance profile
               in memory.                                                   */
            $nonValidMindistList$.add($< minMDist, i >$);
        end

end
for each $< minMDist,\ i >$ in $nonValidMindistList$ do
    if $T_{i,\ell}$ has no Trivial matches in $dkm_\ell$ then
        for $j \leftarrow m$ down to 1 do
            $mDist = minMDist.getMax(j)$;
            if $mDist > dkm_{newL}[k][j].d$ then
                $QT \leftarrow SlidingDotProduct(T_{i,newL}, T)$;
                double $s \leftarrow sum(T_{ind,\ell})$; double $ss \leftarrow squaredSum(T_{i,newL})$;
                $D(T_{ind,\ell}) \leftarrow CalcDistProfAndLB(QT, T_{i,newL}, T, s, ss)$;
                $UpdatePartialDistanceProfile(listDP[i], D(T_{ind,\ell}))$ ;
                $UpdateFixedLengthDiscords(dkm_{newL}, listDP[i], i, k, m)$;
                break;

        end

end
```

Complexity analysis The time complexity of Algorithm 7 ($Topkm\_DiscordDiscovery$) mainly depends on the use of $ComputeMatrixProfile$ algorithm, which always takes $\mathcal{O}(n^2 \log(p))$ to compute the partial distance profiles for the n subsequences of length ℓ_{min} in T.

In order to compute the exact *Top-k* mthdiscord ranking in dkm_ℓ, the routine $UpdateFixedLengthDiscords$ takes $\mathcal{O}(km)$ time in the worst case. Recall that this latter algorithm is called only for subsequences that do not have trivial matches in dkm_ℓ. Checking if two subsequences are trivial matches takes constant time, if for each dkm_ℓ update, we store the ℓ trivial match positions. Given a series T, and the discord (subsequence) length ℓ, we can represent by $S = \frac{|T|}{l/2}$, the number of subsequences that are not trivial matches with one another. Therefore, updating the discord rank of each length has a worst case time complexity of $\mathcal{O}((\ell_{max} - \ell_{min}) \times S \times \ell \times k \times m \times \log(m))$, where the $\log(m)$ factor represents the time to get the mth largest distance in the partial distance profile (line 2 of Algorithm 8). Similarly, the construction of the variable length discord ranking in $dkm_{\ell_{min},\ell_{max}}$ takes: $\mathcal{O}((\ell_{max} - \ell_{min}) \times k \times m)$.

Observe also that the time performance of the $Topkm\_nextLength$ algorithm depends on the Euclidean distance computations pruning. If all the partial distance profiles contain the correct nearest neighbor's distances, computing the discords of each length greater than ℓ_{min} takes $\mathcal{O}(n \times p \times \log(m))$ time, with n equal to the number of subsequences in T. The worst case takes place when for each subsequence that can update dkm_{ℓ} (i.e., S), the complete distance profile is re-computed (Algorithm 10, line 26); in this case the algorithm takes $\mathcal{O}(n^2 \times \log(n) \times p \times \log(m))$.

7 Experimental evaluation

7.1 Setup

We implemented our algorithms in C (compiled with gcc 4.8.4), and we ran them in a machine with the following hardware: Intel Xeon E3-1241v3 (4 cores—8 MB cache—3.50 GHz—32 GB of memory).[3] All of the experiments in this paper are reproducible. In that regard, the interested reader can find the analyzed datasets and source code on the paper web page (Linardi 2017).

Datasets and benchmarking details To benchmark our algorithm, we use five datasets:

- GAP, which contains the recording of the global active electric power in France for the period 2006–2008. This dataset is provided by EDF (main electricity supplier in France) (Dua and Graff 2019);
- CAP, the Cyclic Alternating Pattern dataset, which contains the EEG activity occurring during NREM sleep phase (Terzano et al. 2001);
- ECG and EMG signals from stress recognition in automobile drivers (Healey and Picard 2016);
- ASTRO, which contains a data series representing celestial objects (Soldi et al. 2014).

Table 1 summarizes the characteristics of the datasets we used in our experimental evaluation. For each dataset, we report the minimum and maximum values, the overall mean and standard deviation, and the total number of points.

The (CAP), (ECG) and (EMG) datasets are available in Goldberger et al. (2000). We use several prefix snippets of these datasets, ranging from 0.1M to 1M of points.

In order to measure the scalability of our motif discovery approach, we test its performance along four dimensions, which are depicted in Table 2. Each experiment

[3] In order to validate the time performance results, we repeated our experiments on a second machine with different characteristics (Intel Xeon E5-2650 v4, 24 cores—30 MB cache—2.20 GHz, 250 GB of memory), where we observed the same trends.

Table 1 Characteristics of the datasets used in the experimental evaluation

	MIN	MAX	MEAN	STD-DEV	Number of points
ECG	– 2.182	1.543	0.006	0.24	2M
GAP	0.08	10.67	1.10	1.15	2M
ASTRO	– 0.00867	0.00447	0.00003	0.00031	2M
EMG	– 0.694	0.773	– 0.005	0.041	2M
EEG	– 966	920	3.34	41.36	0.5M

Table 2 Parameters of VALMOD benchmarking

Motif length (ℓ_{min})	Motif range ($\ell_{max} - \ell_{min}$)	Data series size (points)	p (elements of distance profiles stored)
256	**100**	0.1M	5
512	150	0.2M	10
1024	200	**0.5M**	15
2048	400	0.8M	20
4096	600	1M	**50**, 100, 150

Default values shown in bold

is conducted by varying the parameter of a single column, while for the others, the default value (in bold) is selected. In our benchmark, we have two types of algorithms to compare to VALMOD. The first are two state-of-the-art motif discovery algorithms, which receive a single subsequence length as input: QUICKMOTIF (Li et al. 2015) and STOMP (Yeh et al. 2016). In our experiments, they have been run iteratively to find all the motifs for a given subsequence length range. The other approach in the comparative analysis is MOEN (Mueen and Chavoshi 2015), which accepts a range of lengths as input, producing the best motif pair for each length.

For VALMOD, we report the total time, including the time to build the matrix profile (Algorithm 3). The runtime we recorded for all the considered approaches is the average of five runs. Prior to each run we cleared the system cache.

7.2 Motif discovery results

Scalability over motif length In Fig. 9, we depict the performance results of the *four* motif discovery approaches, when varying the motif length. We note that the performance of VALMOD remains stable over the five datasets. On the other hand, we observe that a pruning strategy based on a summarized version of the data is sensitive to subsequence length variation. This is the case for QUICK MOTIF, which operates on Piecewise Aggregate Approximation (PAA) discretized data. Figure 9 shows that the performance of QUICK MOTIF varies significantly as a function of the motif length range, growing rapidly as the range increases, and failing to finish within a reasonable amount of time in several cases.

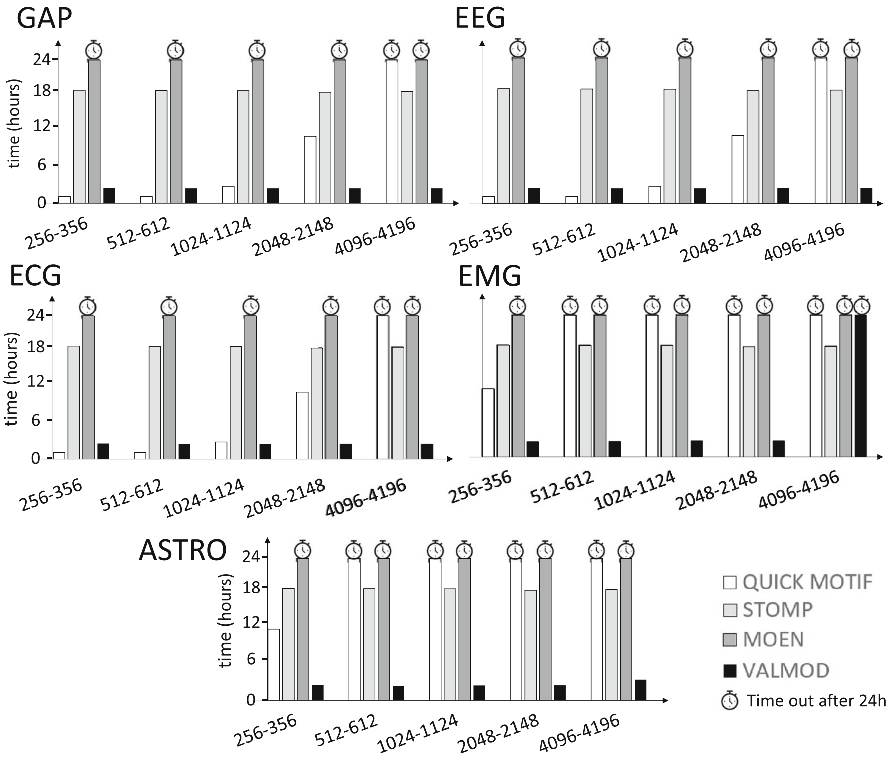
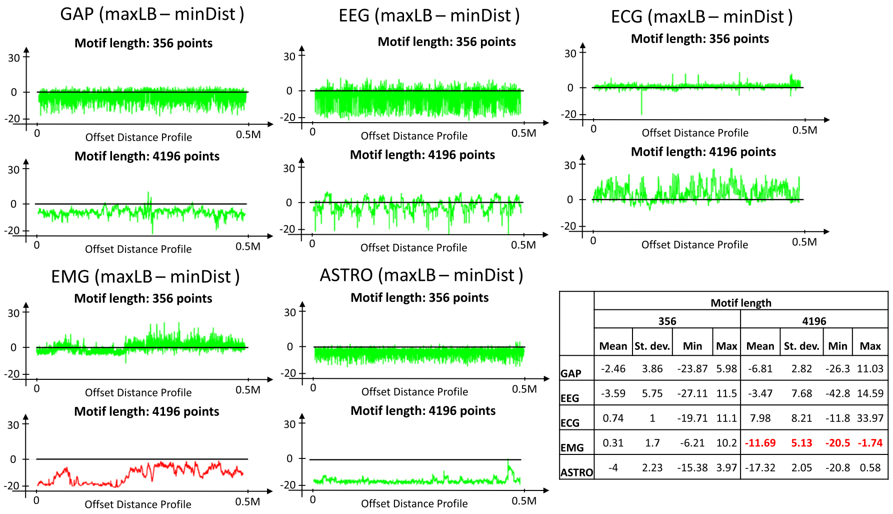


Fig. 9 Scalability for various motif length ranges

Moreover, we argue that our proposed lower bounding measure enables our method to improve upon MOEN, which clearly does not scale well in this experiment (see Fig. 9). The main reason for this behavior is that the effectiveness of the lower bound of MOEN decreases very quickly as we increase the subsequence length ℓ. When we increase the subsequence length by 1, MOEN multiplies the lower bound by a value smaller than 1 ((Mueen and Chavoshi 2015), Sect. IV.B), thus making it less tight. In contrast, the lower bound of VALMOD does not always decrease (refer to Eq. (2)): $\frac{\sigma_{j,l}}{\sigma_{j,l+k}}$ may be larger than 1. Consequently, the lower bound of VALMOD can remain effective (i.e., tight) even after several steps of increasing the subsequence length.

Concerning the VALMOD performance, we note a sole exception that appears for the noisy EMG data (Fig. 9), for a relatively high motif length range (4096–4196). The explanation for this behavior is that the lower bounding distance used by VALMOD is coarse, or in other words, it is not a good approximation of the true distance. Figure 10 shows the difference between the greater lower bounding distance ($maxLB$) and the smaller true Euclidean distance for each distance profile. We use the subsequence lengths *356* and *4196*, which are respectively the range's smallest and largest extremes in this experiment. In this last plot, each value greater than 0 corresponds to a valid condition in line 16 of the *ComputeSubMP* algorithm. This indicates that we found the smallest value of a distance profile, while pruning computations over the entire



	Motif length							
	356				4196			
	Mean	St. dev.	Min	Max	Mean	St. dev.	Min	Max
GAP	-2.46	3.86	-23.87	5.98	-6.81	2.82	-26.3	11.03
EEG	-3.59	5.75	-27.11	11.5	-3.47	7.68	-42.8	14.59
ECG	0.74	1	-19.71	11.1	7.98	8.21	-11.8	33.97
EMG	0.31	1.7	-6.21	10.2	**-11.69**	**5.13**	**-20.5**	**-1.74**
ASTRO	-4	2.23	-15.38	3.97	-17.32	2.05	-20.8	0.58

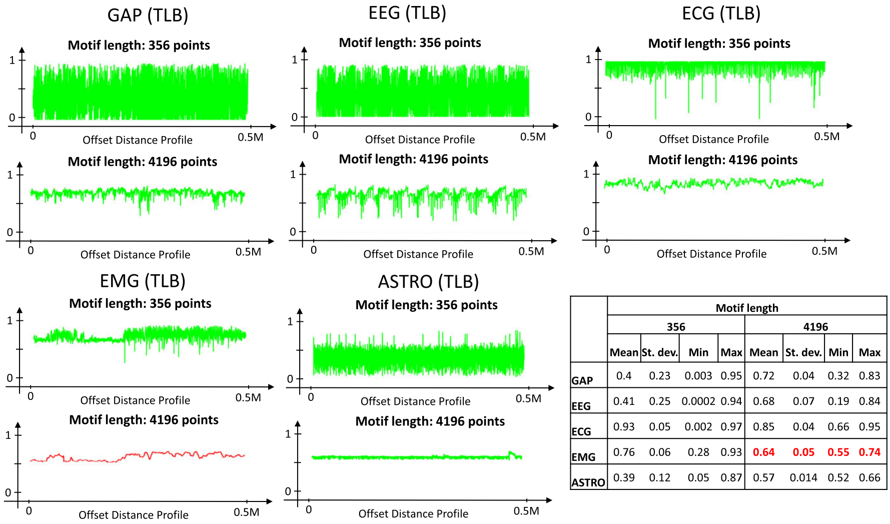
Fig. 10 The difference between the max lower bounding distance (maxLB) and the min Euclidean distance of partial distance profiles for all the datasets. Subsequence lengths: 356/4196. (We report the results for the EMG dataset in red, which corresponds to VALMOD's worst case for lengths 4096–4196, as shown in Fig. 9) (Color figure online)

subsequence length range. As the subsequence length increases, VALMOD's pruning becomes less effective for the EMG (observe that there are no, or very few values above zero in the distances profiles for subsequence length *4196*). On the other hand, we observe the presence of values above zero in the other datasets. This confirms that motifs in those cases are found, while pruning the search space.

In order to further evaluate the pruning capability of VALMOD, we report the measurements for the Tightness of the Lower Bound (TLB) (Shieh and Keogh 2008; Zoumpatianos et al. 2015) performed during the previous experiment (Fig. 9). The TLB is a measure of the lower bounding quality; given two data series t_1 and t_2, the TLB is computed as follows: $LB_{dist}(t_1, t_2)/EuclideanDistance(t_1, t_2)$. Note that TLB takes values between 0 and 1. A TLB value of 1 means that the lower bound distance is the same as the Euclidean distance; this corresponds to the optimal case.

In Fig. 11, we show the average TLB for each (partial) distance profile. In the EMG dataset, when using the larger subsequence length, we observe a sharp decrease of the lower bounding quality (small TLB values), explaining the behavior observed for the EMG dataset (refer to Fig. 10(bottom-left)). We also note similar results for the ASTRO dataset. As we have noted for this last case, the performance is not negatively affected, since we dispose of several partial distance profiles that provide the correct minimum distances, and thus permit us to find the motifs, without recomputing all the distance profiles. In contrast, in the other datasets, we note a smaller negative impact on TLB for the case of subsequence length *4196*.

In Fig. 12, we also show the distance distribution of the pairwise subsequences, using the same datasets and subsequences lengths. Here, we plot the distances without length normalization, since the algorithm uses it to rank the motifs in the trailing part.



	Motif length							
	356				4196			
	Mean	St. dev.	Min	Max	Mean	St. dev.	Min	Max
GAP	0.4	0.23	0.003	0.95	0.72	0.04	0.32	0.83
EEG	0.41	0.25	0.0002	0.94	0.68	0.07	0.19	0.84
ECG	0.93	0.05	0.002	0.97	0.85	0.04	0.66	0.95
EMG	0.76	0.06	0.28	0.93	**0.64**	**0.05**	**0.55**	**0.74**
ASTRO	0.39	0.12	0.05	0.87	0.57	0.014	0.52	0.66

Fig. 11 Average of the tightness of the lower bound (TLB) for every Distance profile for all datasets. Subsequence lengths: 356/4196. (We report the results for the EMG dataset in red, which corresponds to VALMOD's worst case for lengths 4096–4196, as shown in Fig. 9) (Color figure online)

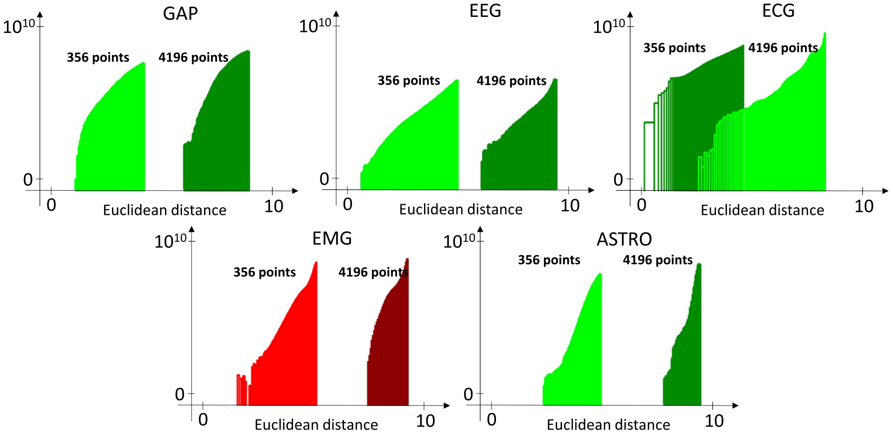


Fig. 12 Distribution of Euclidean distance of pairwise subsequences in all the datasets. Subsequence lengths: 356/4196. (We report the results for the EMG dataset in red, which corresponds to VALMOD's worst case for lengths 4096–4196, as shown in Fig. 9) (Color figure online)

For the EMG and ASTRO datasets, in the case of length *4196*, the distance distribution includes many small and large values, which does not suggest the presence of motifs, but affects VALMOD negatively. Observe that in the other datasets, the values are more uniformly distributed over all the subsequence lengths. This denotes the presence of subsequence pairs that are substantially closer than the rest, which typically identifies the occurrence of motifs. In this case, VALMOD is able to prune more distance profile computations, leading to better performance.

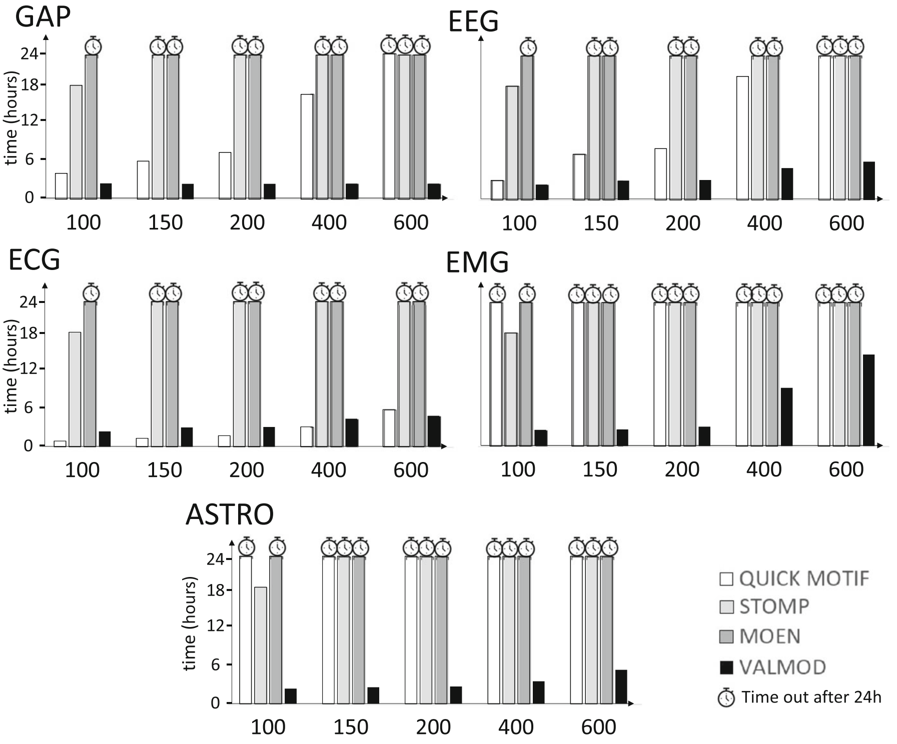


Fig. 13 Scalability with increasing motif range

Scalability over motif range In Fig. 13, we depict the performance results as the motif range increases. VALMOD gracefully scales on this dimension, whereas the other approaches can seldom complete the task. Not only does our technique address the intrinsic problem of STOMP and QUICK MOTIF, which independently process each subsequence length, but it also exhibits a substantial improvement over MOEN, the existing state-of-the-art approach for the discovery of variable length motifs.

Scalability over data series length In Fig. 14, we experiment with different data series sizes. For the EEG dataset we only report three measurements, since this collection contains no more than 0.5M points. We observe that QUICK MOTIF exhibits high sensitivity, not only to the various data sizes, but also to the different datasets (as in the previous case, where we varied the subsequence length). It is also interesting to note that QUICK MOTIF is slightly faster than VALMOD on the ECG dataset, which contains regular and similar heartbeat patterns, and is a relatively easy dataset for motif discovery. Nevertheless, QUICK MOTIF, as well STOMP and MOEN, fail to terminate within a reasonable amount of time for the majority of our experiments. On the other hand, VALMOD does not exhibit any abrupt changes in its performance, scaling gracefully with the size of the dataset, across all datasets and sizes.

Large datasets and length ranges Here we report two further experiments that we have conducted on larger snippets of the datasets—namely, 2 million points—and over

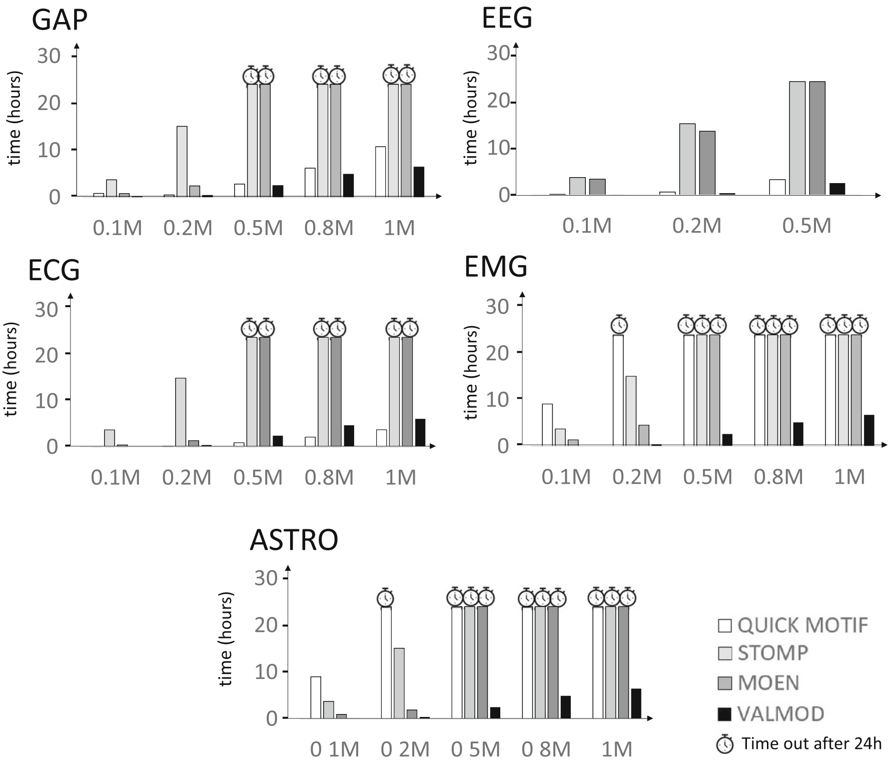


Fig. 14 Scalability with increasing data series size

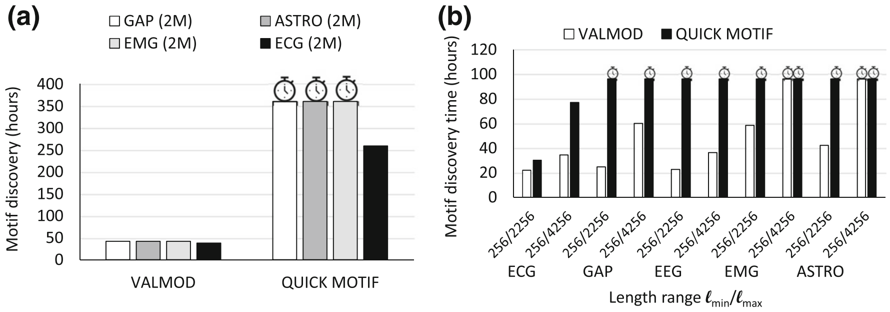


Fig. 15 Scalability of VALMOD and QUICKMOTIF using large datasets (2M of points) and large length ranges

a larger range of motif lengths. To that extent, we want to test the scalability of our approach, considering two *extreme* cases. We compare VALMOD to QUICKMOTIF, since the latter is the sole approach that can scale to data series lengths beyond half a million points, and to motif length ranges larger than *100*.

In Fig. 15a, we report the motif discovery time on four datasets that contain *2* million points. We pick the default length boundaries, namely $\ell_{min} = 1024$ and $\ell_{max} = 1124$, discovering motifs of each length in between them. The results show that VALMOD gracefully scales, and is always one order of magnitude faster than QUICKMOTIF, which does not reach the timeout only in the case of the ECG datasets.

The same observations hold for the results of the experiments that vary the motif length range. Figure 15b, shows the results for length ranges *2000* and *4000*, on all five datasets in our study (at their default sizes). Once again, QUICKMOTIF reaches the timeout state in all datasets, except for ECG, where for the larger length ranges is two times slower than VALMOD. On the other hand, VALMOD scales well and remains the method of choice (with the exception of the largest length ranges for the EMG and ASTRO datasets, where it reaches the timeout).

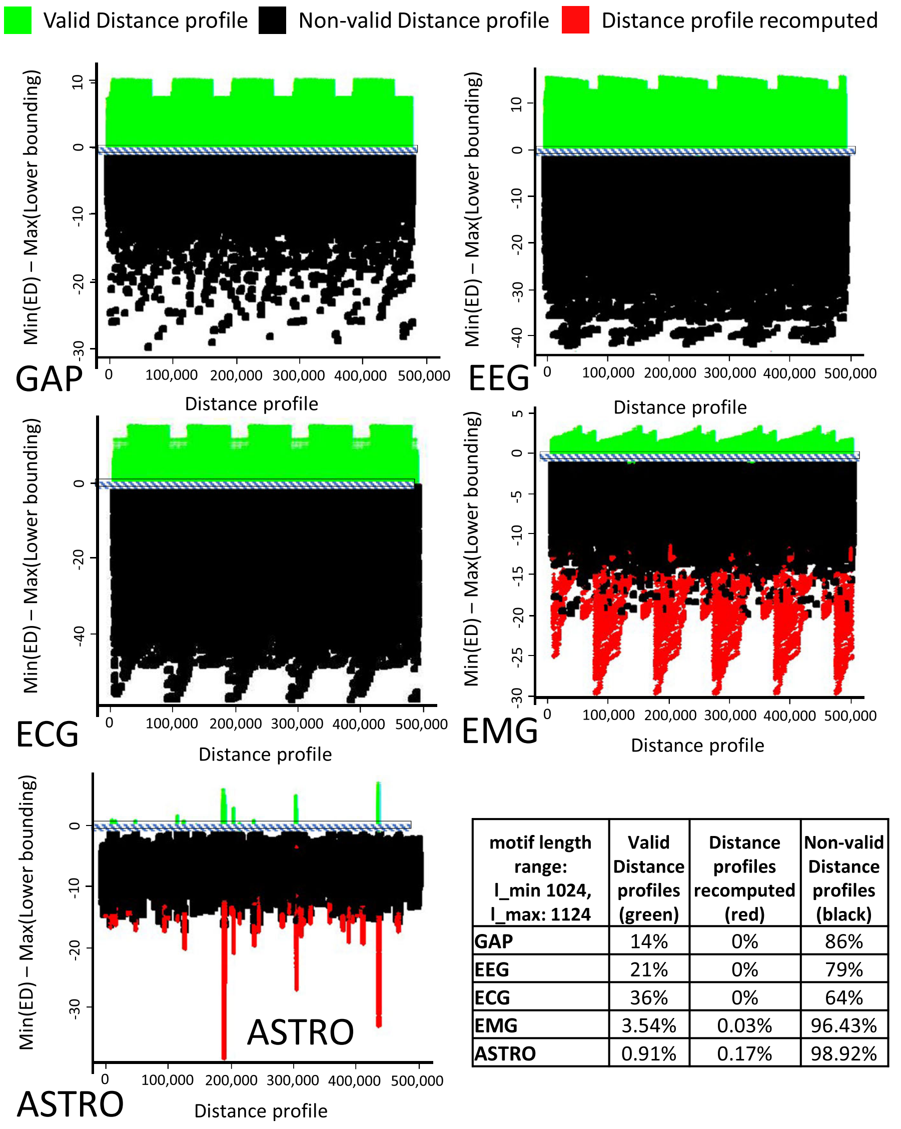
The above results demonstrate the superiority of VALMOD, but also show its limits, which open possibilities for future work.

Overall pruning power In order to show the global effect of VALMOD's pruning power, we conduct an experiment recording the number of distance profile computations performed by procedure *ComputeSubMP*, which extracts motifs of length greater than ℓ_{min}, pruning the unpromising calculations. We recall that this algorithm computes for each subsequence $T_{i,\ell}$ with $\ell > \ell_{min}$ a subset of distances (Euclidean and lower bounding), called partial distance profiles. If the smallest Euclidean distance computed is also smaller than the larger lower bounding distance, we know it is the true distance of the nearest neighbor of $T_{i,\ell}$. In this case, we call the partial distance profile *valid*. Otherwise, we do not know the true nearest neighbor distance, and we call the partial distance profile *non-valid*. In order to identify the correct motifs, the algorithm only needs to recompute the entire *non-valid* distance profiles that might contain distances shorter than those already found in the *valid* distance profiles.

In Fig. 16, we depict the difference between the minimum Euclidean distance and the maximum lower bounding distance of each distance profile computed in the subsequence length range (*1025/1124*). In the plots, the values above zero refer to the *valid* ones (green points), whereas values under zero are either *non-valid* (black points) or *recomputed* (red/triangular points). We observe that in the first three datasets, namely EEG, ECG and GAP, there are no distance profiles that are recomputed, meaning that the motifs are always found in the *valid* (partial) distance profiles in the shortest time possible (*best case*). Concerning the EMG and ASTRO datasets, several re-computations take place (red/triangle points). As we can see from the table in the bottom part of Fig. 16 though, the computed distance profiles are not more than the *0.20%* of the total. This means that the algorithm successfully prunes a high percentage of the computations, thanks also to the effectiveness of the proposed lower bounding measure.

At this point, we can further analyze the reasons behind the pruning capability of our approach. To that extent, in Fig. 17a we plot the number of distance profiles that VALMOD recomputes at each subsequence length for the EMG and ASTRO datasets. These two datasets both contain noisy data, which influence re-computations. However, they differ according to the length for which these re-computations take place.

Figure 17b shows the position of the Top-1 motif along the subsequence length. Note that the Top-1 motif is always placed around the same offset region in the ASTRO



motif length range: l_min 1024, l_max: 1124	Valid Distance profiles (green)	Distance profiles recomputed (red)	Non-valid Distance profiles (black)
GAP	14%	0%	86%
EEG	21%	0%	79%
ECG	36%	0%	64%
EMG	3.54%	0.03%	96.43%
ASTRO	0.91%	0.17%	98.92%

Fig. 16 Partial distance profile repartition (*valid*, *non-valid*, *recomputed*), in the motif discovery task on the five considered datasets. Default parameters are used in this experiment (Color figure online)

dataset, suggesting the presence of a few similar data segments, which is also verified by the high number of *non-valid* distance profiles we observe in Fig. 16 (ASTRO). On the other hand, in the EMG dataset, the motif location changes several times, denoting the presence of different segments, which contain motifs of different lengths. This is also confirmed by the more prevalent presence of *valid* distance profile in the EMG dataset. In this last case, the re-computation number drops to zero as soon as the motif positions start to change, i.e., at length *1058*, maintaining the same trend until the end.

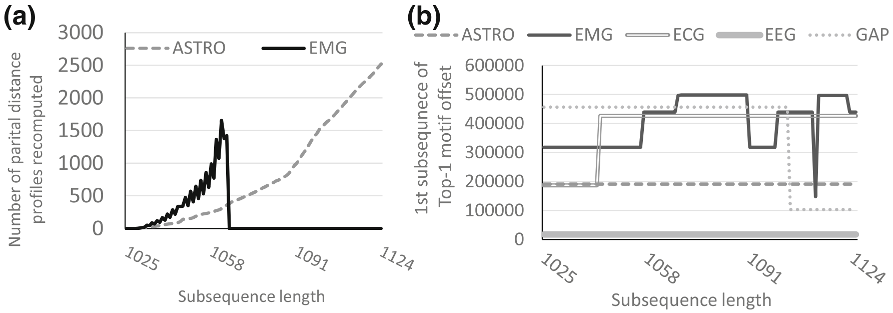


Fig. 17 **a** Distribution of *recomputed* distance profiles for each subsequence length considered in the EMG and ASTRO datasets. **b** Offset of the first subsequence in the discovered motif for all the length in the EMG and ASTRO datasets

Effect of changing parameter p. In Fig. 18, we study the effect of parameter p on VALMOD's performance. The p value determines how many distance profile entries we compute and keep in the memory. Increasing p leads to increased memory consumption, but could also translate to an overall speed-up, since having more distances may guarantee a larger margin between the greater lower bounding distance and the minimum true Euclidean distance in a distance profile. As we can see on the left side of the plot, increasing p does not provide any significant advantage in terms of time complexity. Moreover, the plots on the right-hand side of the figure demonstrate that the size of the Matrix profile subset ($subMP$), computed by the *computeSubMP* procedure, decreases in the same manner at each iteration (i.e., as we increase the length of the subsequences that the algorithm considers), regardless of the value of p.

It is important to note that irrespective of its size, $subMP$ *always* contains the smallest distances of the matrix profile, namely the distances of the motif pair. Having a larger $subMP$ does not represent an advantage w.r.t. motif discovery, but rather an opportunity to view and analyze the subsequence pairs, whose distances are close to the motif.

7.3 Motif sets

We now conduct an experiment to show the time performance of identifying the variable length motif sets. We use the default values of Table 2, varying K and the radius factor D for each dataset. In Fig. 19 we report the results; we also show the time to compute $VALMP$ (the output of VALMOD). We note that once we build the pairs ranking of $VALMP$ ($heapBestKPairs$ in Algorithm 5), we can run the procedure that computes the motif sets (Algorithm 6). The results show that this operation is 3–6 orders of magnitude faster than the computation of $VALMP$. The advantage in time performance is pronounced for the ECG and EEG datasets, thanks to the pruning we perform with the partial distance profiles.

The fast performance of the proposed approach also allows for a fast exploratory analysis over the radius factor, which would otherwise (i.e., with previous approaches) be extremely time-consuming to set for each dataset.

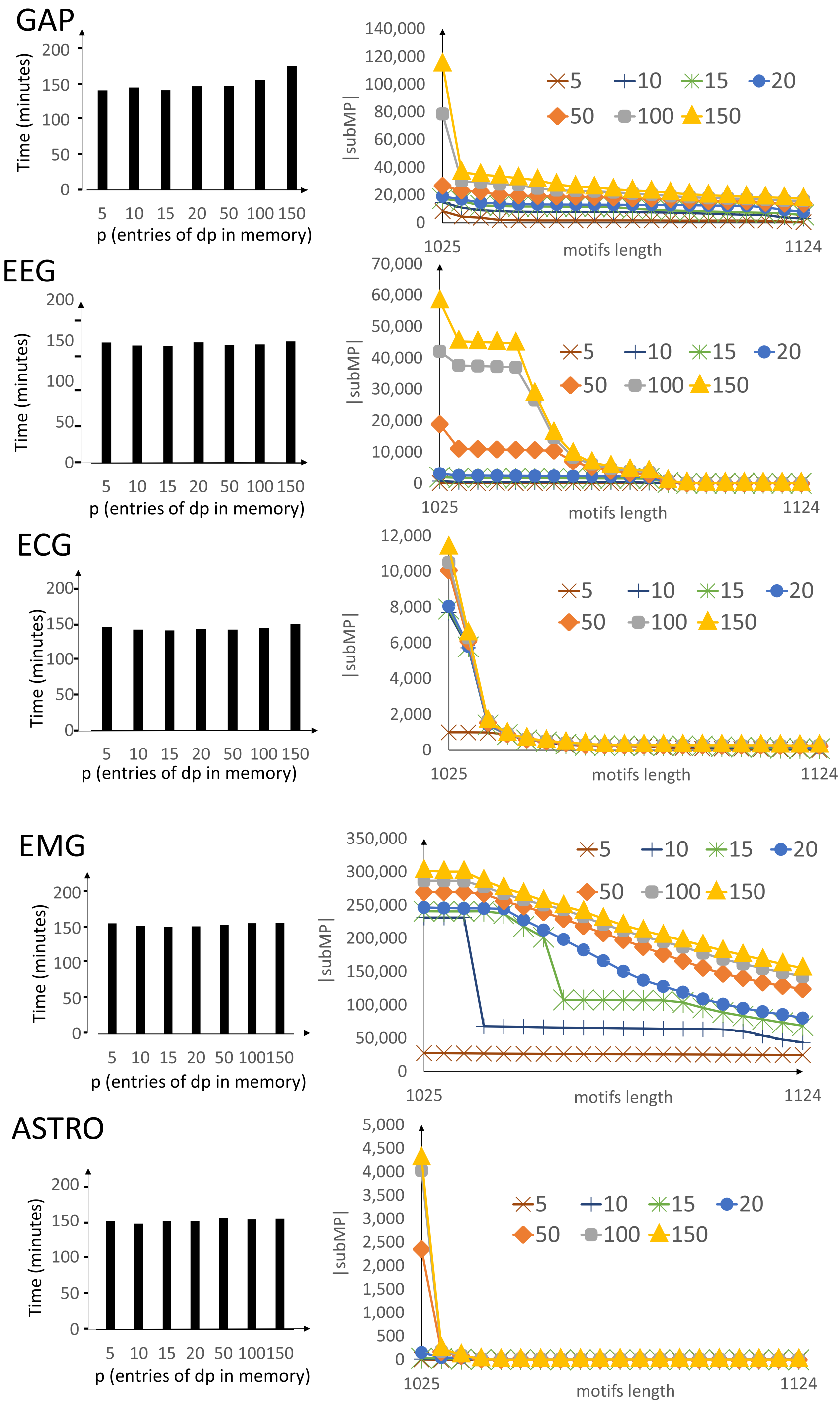
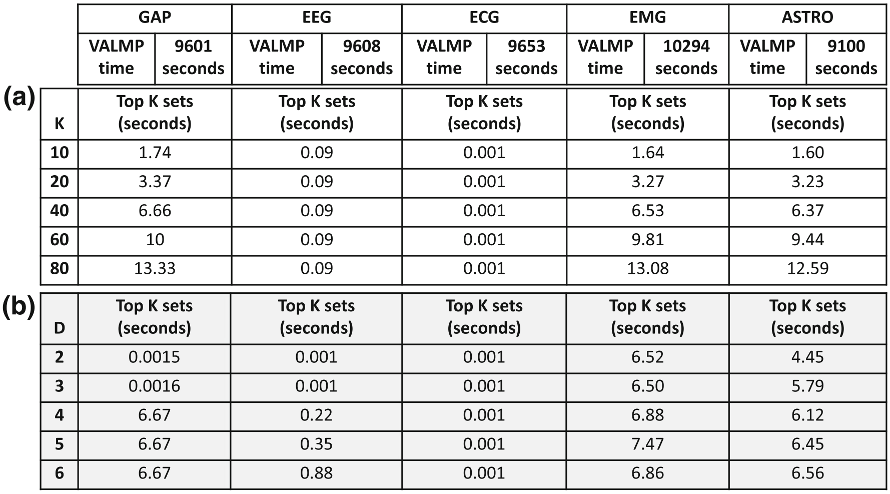


Fig. 18 Scalability with increasing parameter p



	GAP	EEG	ECG	EMG	ASTRO
	VALMP time 9601 seconds	VALMP time 9608 seconds	VALMP time 9653 seconds	VALMP time 10294 seconds	VALMP time 9100 seconds

(a)

K	Top K sets (seconds)	Top K sets (seconds)	Top K sets (seconds)	Top K sets (seconds)	Top K sets (seconds)
10	1.74	0.09	0.001	1.64	1.60
20	3.37	0.09	0.001	3.27	3.23
40	6.66	0.09	0.001	6.53	6.37
60	10	0.09	0.001	9.81	9.44
80	13.33	0.09	0.001	13.08	12.59

(b)

D	Top K sets (seconds)	Top K sets (seconds)	Top K sets (seconds)	Top K sets (seconds)	Top K sets (seconds)
2	0.0015	0.001	0.001	6.52	4.45
3	0.0016	0.001	0.001	6.50	5.79
4	6.67	0.22	0.001	6.88	6.12
5	6.67	0.35	0.001	7.47	6.45
6	6.67	0.88	0.001	6.86	6.56

Fig. 19 Time performance of variable length motif sets discovery. **a** Varying K (default D = 4). **b** Varying radius factor D (default K = 40)

7.4 Discord discovery

In this last part, we conduct the experimental evaluation concerning discord discovery. In the following experiments, we use the same datasets as before.

We identify two state-of-the-art competitors to compare to our approach, the Motif And Discord (MAD) framework. The first one, Disk Aware Discord Discovery (DAD) (Yankov et al. 2007b), implements an algorithm suitable for enumerating the *fixed-length* mth discords of a data series collection stored on a disk. We adapted this algorithm, as suggested by the authors, in order to extract discords from data series loaded in main memory. The second approach, GrammarViz (Senin et al. 2015), is the most recent technique, which discovers *Top-k 1st*discords. It operates by means of grammar rules compression, which further operate on a summarized data series representation, in order to find the rare segments of the data (discords) in a reduced search space. To the best of our knowledge, there exist no techniques capable of finding the *Top-k* mthranked variable-length discords as MAD, using a single execution of an algorithm.

Mth discord discovery In Fig. 20a, b, we present the performance comparison between MAD and DAD for finding the mth discords, when we vary m, for all datasets. (All other parameters are set to their default values, as listed in Table 2.)

Since DAD discovers fixed-length mth discords, we report its execution time *only* for the first length in the range, namely ℓ_{min}. We observe that MAD, which enumerates the mth discords of *100* lengths ($\ell_{min} = 1024$, $\ell_{max} = 1124$) is still one order of magnitude faster than these DAD performance numbers, for all datasets, when m is larger or equal to *5*. Moreover, the performance trend of MAD remains stable over all datasets, whereas DAD has different execution times. We observe that the computational time of DAD depends on the subsequence length, since it computes

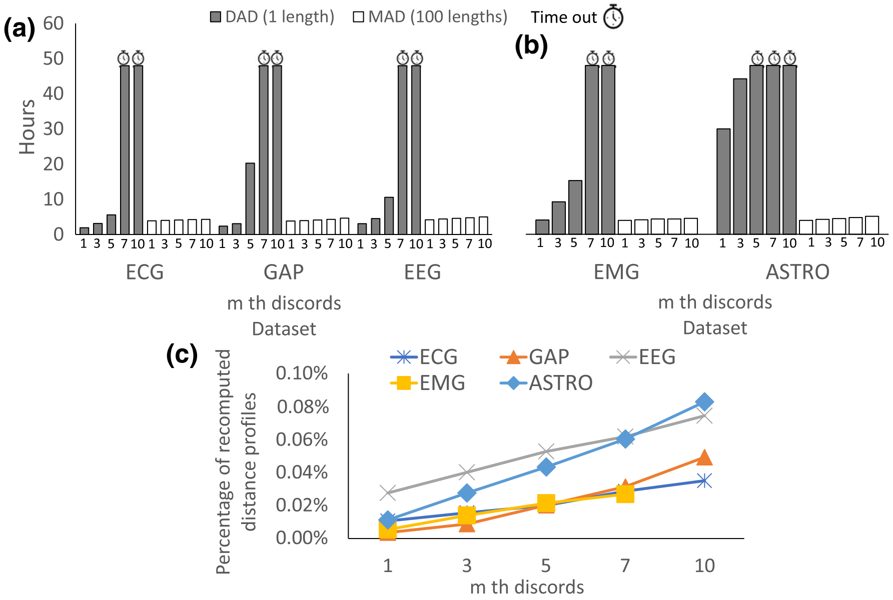


Fig. 20 **a**,**b** DAD (one length) and MAD (*100* lengths) *Top-k mth*discords discovery time. **c** Percentage (on the total distance profiles) of *non-valid* partial distance profiles recomputed by Algorithm 10

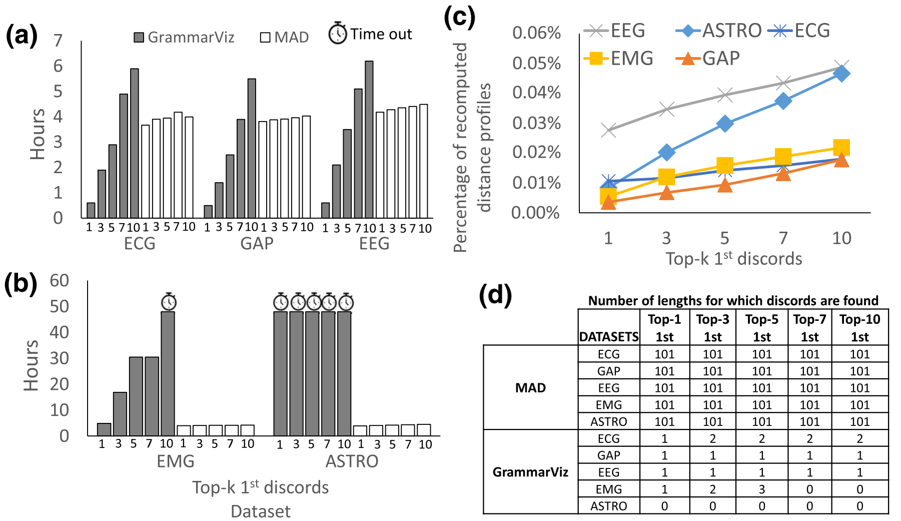
Euclidean distances in their entirety (only applying early abandoning based on the best so far distance). How effective this early abandoning mechanism is, depends on the characteristics of the data. On the other hand, our algorithm computes all distances for the first subsequence length in constant time, and then prunes entire distance computations for the larger lengths.

In Fig. 20c, we report the percentage of non-valid distance profiles that are recomputed, over the total number of distance profiles considered during the entire task of variable-length discord discovery. We note that the number of re-computations is limited to no more than *0.10%*, in the worst case. This demonstrates the high computation pruning rate achieved by our algorithm, justifying the considerable speed-up achieved.

***Top* − *k 1st* discord discovery** In Fig. 21, we depict the performance comparison between GrammarViz and MAD. We do not report results for DAD, since it always reaches the imposed time-out, even for the variable length *Top-k 1st*discord discovery task. Therefore, we consider *Top-k 1st*discords discovery, as previously introduced. (We maintain the same parameter settings in this experiment.)

First, we note that GrammarViz outperforms MAD in the first three datasets, for *k* smaller or equal to *5*, as depicted in Fig. 21a. Nevertheless, the experiment shows that MAD scales better over the number of discovered *Top-k 1st*discords, as its execution time increases only by a small constant factor. A different trend is observed for GrammarViz, whose performance significantly deteriorates as *k* increases from 1 to 6.

Moreover, this technique is highly sensitive to the dataset characteristics, as we observe in Fig. 21b, where the two noisy datasets, i.e., EMG and ASTRO, are con-



Number of lengths for which discords are found

	DATASETS	Top-1 1st	Top-3 1st	Top-5 1st	Top-7 1st	Top-10 1st
MAD	ECG	101	101	101	101	101
	GAP	101	101	101	101	101
	EEG	101	101	101	101	101
	EMG	101	101	101	101	101
	ASTRO	101	101	101	101	101
GrammarViz	ECG	1	2	2	2	2
	GAP	1	1	1	1	1
	EEG	1	1	1	1	1
	EMG	1	2	3	0	0
	ASTRO	0	0	0	0	0

Fig. 21 **a**, **b** GrammarViz and MAD (*100* lengths) *Top-k 1st*discords discovery time. **c** Percentage (on the total distance profiles) of *non-valid* partial distance profiles recomputed by Algorithm 10

sidered. This is a direct consequence of the data summarization sensitivity to the data characteristics, which then influences the ability to prune distance computations.

In Fig. 21c, we report the percentage of non-valid distance profiles that MAD needed to recompute. In this case, too, this percentage is very low.

To conclude, since GrammarViz is a variable length approach that selects the most promising discord lengths according to the distribution of the data summarization (by picking the lengths of the series, whose discrete versions represent a rare occurrence), we report in Fig. 21d the number of lengths, for which discords are found. We observe that our framework always enumerates and ranks discords of all lengths in the specified input range, based on the exact Euclidean distances of the subsequences. On the other hand, GrammarViz selects the most promising length based on the discrete version of the data, and only identifies the exact *Top-k 1st*discords for *3* (out of *100*) different lengths in the best case.

Top-k mth **Discord Discovery.** Figure 22 depicts the execution time for the *Top-k mth*discord discovery task, and the percentage of recomputed distance profiles for MAD, when varying k and m. We observe that the pruning power remains high: the percentage of distance profile re-computations averages around 0.05%.

Utility of variable-length discord discovery We applied MAD on a real use case, a data series containing the average number of taxi passengers for each half hour over 75 days at the end of 2014 in New York City (Rong and Bailis 2017), depicted in Fig. 23a. We know that this dataset contains an anomaly that occurred during the daylight savings time end, which took place the 2nd of November 2014 at 2 a.m. At that time, the clock was set back at 1 a.m. Since the recording was not adjusted, two samples (corresponding to a *1* h recording) are summed up with the two subsequent ones.

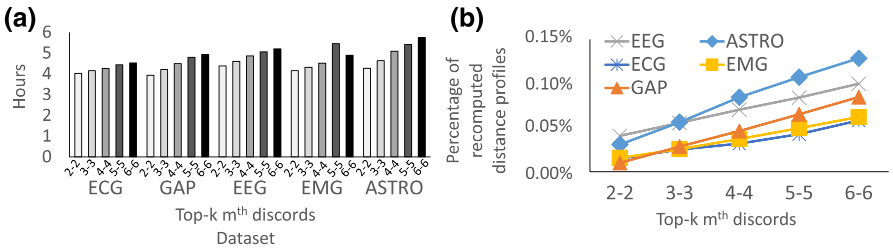


Fig. 22 **a** MAD (*100* lengths)*Top-k mth*discords discovery time on the five datasets. **b** Percentage (on the total distance profiles) of *non-valid* partial distance profiles recomputed by Algorithm 10

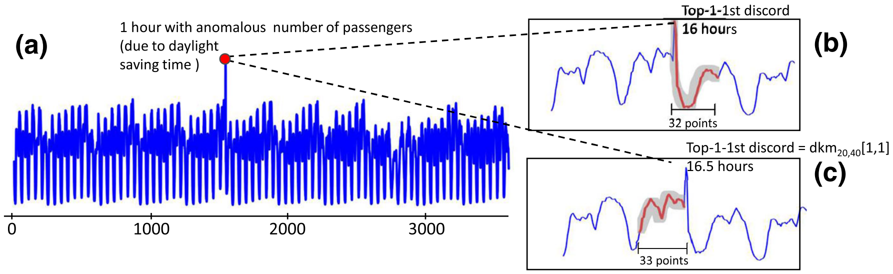


Fig. 23 **a** Data series reporting the number of taxi passengers over 75 days at the end of 2014 in New York City. **b** Top-1 1st discord of length *32*, which contains the abnormal peak generated by the double recording problem of daylight savings time. **c** Top-1 1st discord of length *33*, which represents an anomalous trend for the number of taxi passengers due to the daylight savings time

We ran the variable-length discord discovery task using the length range $\ell_{min} = 20$ and $\ell_{max} = 48$, in order to cover subsequences that correspond to recordings between 10- and 24-h. Our algorithm correctly identifies the anomaly for subsequence length *32*, shown in Fig. 23b. Changing the window size does not allow the detection of the anomaly. For example, enlarging the window by *just* 1 point, the *Top-k 1st*discord corresponds to a pattern *before* the abnormality (refer to Fig. 23c).

These results showcase the importance of efficient variable-length discord discovery. It permits us to discover rare, or abnormal events with different durations, which can be easily missed in the fixed length discord discovery setting, where the analyst is constrained to examine a single length (or time permitting, a few fixed lengths).

7.5 Exploratory analysis: motif and discord length selection

In this part, we present the results of an experiment we conducted to test the capability of MAD to suggest the most promising length/s for motifs and discords.

Given a data series, the user may have no clear idea about the motif/discord length. Therefore, we present use cases that examine the ability of MAD to perform a wide length-range search, providing the most promising results at the *correct* length.

We used MAD for finding motifs and discords in the length range: $\ell_{min} = 256$ and $\ell_{max} = 4096$. We conducted this experiment in the first $500K$ points of the datasets listed in Table 1. The considered motif/discord length range covers the user studies

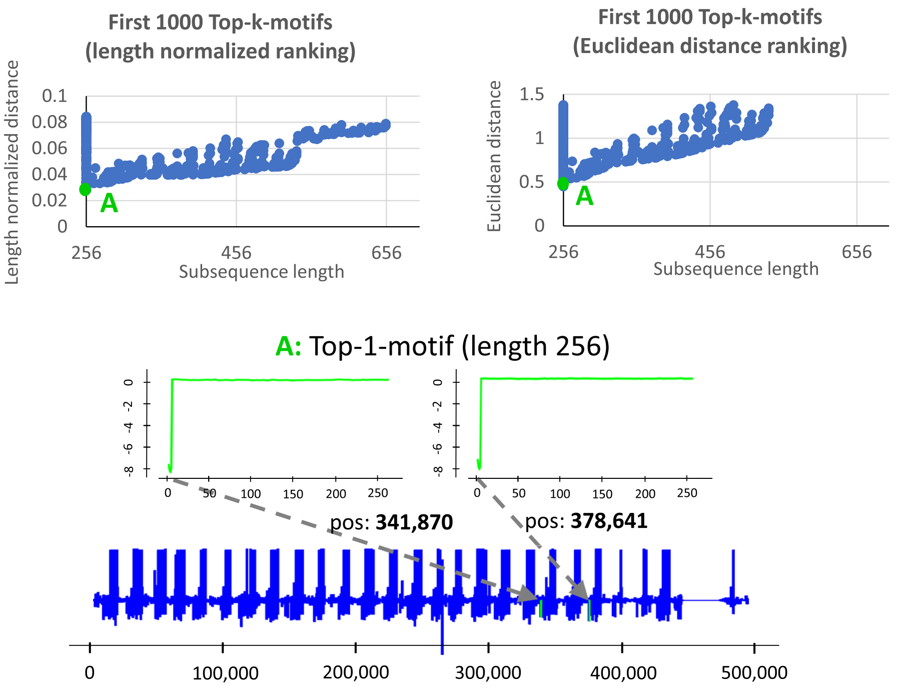


Fig. 24 Top-1 motif (of length 256) in the EEG data set. The subsequences pairs composing this motif have the smallest distance in both the Euclidean distance and length normalized ranking

that have been presented so far in the literature (where knowledge of the exact length was always assumed).

Scalability The MAD framework completed the motif/discord discovery task within *2 days* (on average), enumerating the motifs and the Top-1 discords of each length in the given range. Concerning the competitors, we estimated that STOMP, which is the state-of-the-art solution for fixed length motif/discord discovery would take 320 days for the same experiment (a little bit more than 2 h for each of the lengths we tested). QUICK MOTIF, which has data dependent time performance, takes up to more than *6* days (projection) for all datasets but ECG (which completes in *38* h). We note that the variable-length motif discovery competitor (MOEN) never terminates before *24* h when searching motifs of 600 different lengths, while in this experiment, the length range is composed of 3841 different lengths. Considering discord discovery, we observed that GrammarViz does not enumerate all the discords in the given length-range, since it selects the length according to the data summarizations. Thus, we are obliged to run this technique independently for each length, which would take at least 320 h in the best case (projection based on results of Fig. 21).

Select the most promising length in motif discovery Once the search is completed, the MAD framework enumerates the motifs and discords ranking them in a second step, according to the proposed distance normalization strategy. In Fig. 24, we show the results of motif discovery for the EEG dataset.

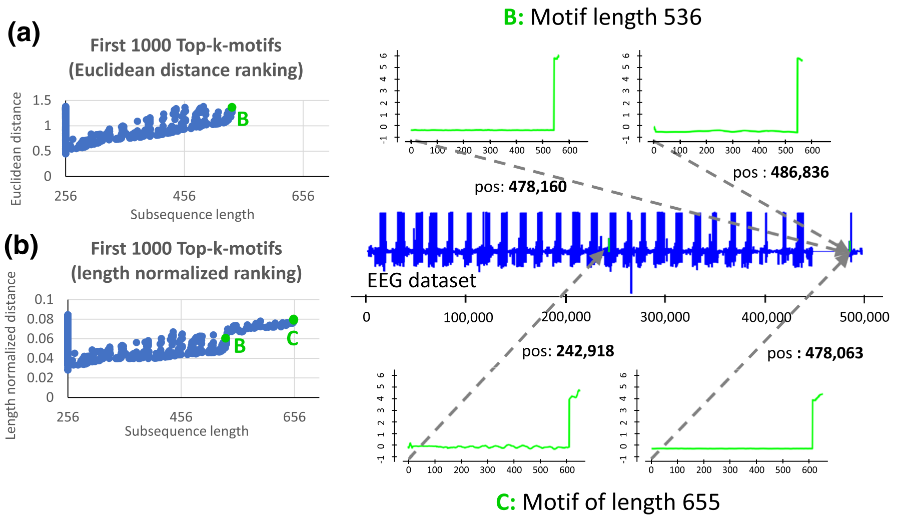


Fig. 25 **a** Top-1000 motifs according the length normalized distance (top), and the Euclidean Distance (bottom). **b** Motif pair of the largest length (656) in the length normalized ranking (top) and motif pair of the largest length (536) in the Euclidean distance ranking (red/bottom) (Color figure online)

The objective of this experiment is to evaluate the proposed length-normalized correction strategy. In this regard, we compare the motifs sorted by using length-normalization, and by Euclidean distances.

On the top part of Fig. 24, we report the distance/length values of the Top-1000 motifs ranked by the length-normalized measure (left), which comprise a subset of the results we store in the VALMP structure (Algorithm 1). In the right part of the figure, we report the Top-1000 motifs ordered by their Euclidean distances.

We observe that the Top-1 motif, i.e., the subsequence pair with the smallest distance (marked by the letter A) is the same in both rankings. We report this motif in the bottom part of Fig. 24, which is composed of two quasi-identical patterns in the EEG data series.

We now evaluate motifs of larger lengths in the same dataset, which may reveal other interesting and similar patterns at different resolutions (lengths). In Fig. 25a, we report again the distance/length values of the Top-1000 motifs ranked by their Euclidean distance, which reveal that the longest motif, marked as B, has length 536. We observe that this subsequence pair substantially differs from the Top-1 motif of Fig. 24.

Subsequently, in Fig. 25b, we report the longest motif (marked as C) of length 655 that we found in the Top-1000 motif ranking, based on length-normalized distances. We note that 6% of the length-normalized motifs are longer than those in the Top-1000 of the Euclidean ranking. The example of motif C, which is a longer version of B, shows that this pattern appears much earlier in the sequence than B. If we considered just the Top-1000 motifs ranked by their Euclidean distance, we would have missed this insight (motif C appears in the Euclidean distance ranking only in the Top-4000 motifs).

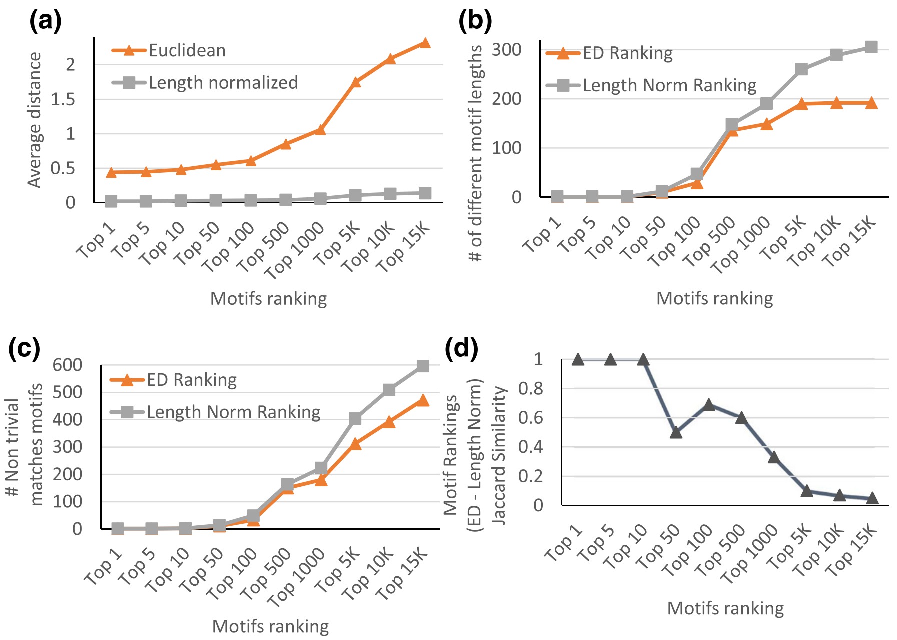


Fig. 26 Euclidean and length-normalized *Top-k* motifs properties. **a** Average distance. **b** Number of motif lengths. **c** Number of non- trivial match motifs. **d** Jaccard similarity of the ranking using the Euclidean and length-normalized Euclidean distances

Unfolding Top-k motifs When considering the *Top-k* motif ranking, we could manually inspect all the subsequence pairs. However, this is a cumbersome (and unnecessary) task for a user that would like to focus directly on the most interesting motifs. In the previous experiment on EEG data, we examined the number of motifs we need to consider. We now examine the value of k that allows us to find interesting patterns within the *Top-k* motif ranking.

In Fig. 26a we report the average distance for the *Top-k* motif rankings that we built considering Euclidean and length-normalized distances, varying k. We note that the average distance exhibits a steep increase in the Euclidean distance rankings, starting from $k = 500$. This is due to the presence of motifs of larger lengths, as depicted in Fig. 26b, since these pairs of longer subsequences have also a larger distance. In this specific case, the user may choose to discard motifs beyond the *Top*-500, thus, disregarding several motifs of different lengths. In contrast, we note that length-normalized distance is not heavily affected by longer motifs (Fig. 26a). This will urge users to continue the exploration beyond the *Top*-500, and consider motifs of several different lengths that (as discussed earlier) represent different kinds of insights.

Another important factor to account in *Top-k* motif analysis is the redundancy in the reported motifs. In that respect, we can eliminate the motifs composed by subsequences that are trivial matches of motif subsequences that appear earlier in the ranking. In Fig. 26c, we plot the motifs that we retain (i.e., the motifs that are not trivial matches) from the Euclidean and length-normalized *Top-k* rankings. We notice that

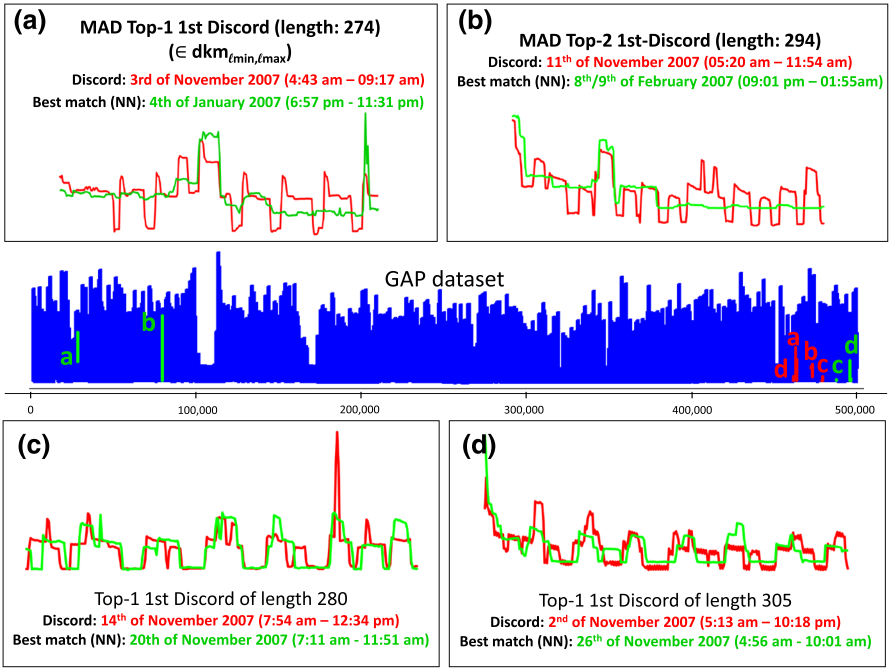


Fig. 27 Four discords of different length in the GAP dataset. Each discord (red subsequence) is coupled with its nearest neighbor (green subsequence). **a** The discord, with the highest length-normalized distance to its nearest neighbor has length 274. **b** Discord with the second highest length-normalized distance. **c**, **d** discords with a smaller length-normalized distance to their nearest neighbor (Color figure online)

as k increases these retained motifs represent only a small subset of the motifs in the original rankings (up to 4%), which renders their examination easier. Furthermore, we observe that the Length-normalized Top-k rankings contain up to 130 more non-trivial match motifs than the Euclidean rankings, which translates to more useful results.

To conclude, we depict in Fig. 26d the Jaccard similarity between the two ranking types (i.e., length-normalized and Euclidean) as we vary k. While computing the intersection and the union of the two rankings, we discard the motifs that are trivial matches. As k increases, and consequently the motif length increases as well (refer to Fig. 26b), we observe that set similarity decreases. This means that the new motifs of different lengths are not trivial matches of motifs found in higher ranking positions, but they represent new, useful results.

Select the most promising length in discord discovery In this part, we show the results of discord discovery performed in the GAP dataset. We recall that in this case, the discord ranking performed according to their length normalized distances aims to favor smaller discords, which have a high point to point distance.

In Fig. 27, we report some of the discords we found in the length range $\ell_{min} = 256$ and $\ell_{max} = 4096$. The discord with the highest length-normalized distance, *best* Top-1 1st discord, is the one depicted in the top-left part of the figure, and has length 274. We plot it in red (dark), whereas its nearest neighbor appears in green (light). We note that this discord drastically differs from its nearest neighbor: it represents a fluctuating cycle of global power activity, while its nearest neighbor exhibits the expected behavior of two major peaks, in the morning and around noon. In Fig. 27b we report the Top-2 1st discord in the length range 256–4096 identified by MAD, which corresponds to the subsequence in that length range with the second highest length-normalized distance to its nearest neighbor. Once again, we observe a high degree of dissimilarity between the pattern of this discord and its nearest neighbor. On the contrary, Fig. 27c, d report the Top-1 1st discords for two specific lengths (i.e., 280 and 305, respectively). These discords correspond to patterns that are not significantly different from their nearest neighbors. Therefore, they represent discords that are less interesting than the ones reported by MAD in Fig. 27a, b, which examines a large range of lengths.

This experiment demonstrates that MAD and the proposed discord ranking allows us to prioritize and select the correct discord length.

8 Related work

While research on data series similarity measures and data series query-by-content date back to the early 1990s (Palpanas 2016), *data series motifs* and *data series discords* were both introduced just 15 and 12 years ago, respectively (Chiu et al. 2003; Fu et al. 2006). Following their definition, there was an explosion of interest in their use for diverse applications. Analogies between *data series motifs* and sequence *motifs* exist (in DNA), and have been exploited. For example, discriminative motifs in bioinformatics (Sinha 2002) inspired discriminative data series motifs (i.e., data series shapelets) (Neupane et al. 2016). Likewise, the work of Grabocka et al. (2016) on generating idealized motifs is similar to the idea of consensus sequence (or canonical sequence) in molecular biology. The literature on the general data series motif search is vast; we refer the reader to recent studies (Zhu et al. 2016; Yeh et al. 2016) and their references.

The QUICK MOTIF (Li et al. 2015) and STOMP (Zhu et al. 2016) algorithms represent the state of the art for fixed-length motif pair discovery. QUICK MOTIF first builds a summarized representation of the data using Piecewise Aggregate Approximation (PAA), and arranges these summaries in Minimum Bounding Rectangles (MBRs) in a Hilbert R-Tree index. The algorithm then prunes the search space based on the MBRs. On the other hand, STOMP is based on the computation of the matrix profile, in order to discover the best matches for each subsequence. The smallest of these matches is the motif pair. We observe that both of the above approaches solve a restricted version of our problem: they discover motif sets of cardinality two (i.e., motif pairs) of a fixed, predefined length. On the contrary, VALMOD removes these limitations and proposes a general and efficient solution. Its main contributions are the novel algorithm for examining candidates of various lengths and corresponding lower bounding distance: these techniques help to reuse the computations performed so far, and lead to effective pruning of the vast search space.

We note that there are only three studies that deal with issues of variable length motifs, and attempt to address them (Minnen et al. 2007; Gao et al. 2016; Yingchareonthawornchai et al. 2013; Gao and Lin 2018). While these studies are pioneers in demonstrating the *utility* of variable length motifs, they cannot serve as practical solutions to the task at hand for two reasons: (i) they are all approximate, while we need to produce exact results; and (ii) they require setting many parameters (most of which are unintuitive). Approximate algorithms can be very useful in many contexts, if the amount of error can be bounded, or at least known. However, this is not the case for the algorithms in question. Certain cases, such as when analyzing seismological data, the threat of litigation, or even criminal proceedings (Cartlidge 2016), would make any analyst reluctant to use an approximate algorithm.

The other work that explicitly considers variable length motifs is MOEN (Mueen and Chavoshi 2015). Its operation is based on the distance computation of subsequences of increasing length, and a corresponding pruning strategy based on upper and lower bounds of the distance computed for the smaller length subsequences. Unlike the algorithms discussed above, MOEN is exact and requires few parameters. However, it has been tuned for producing only a single motif pair for each length in the range, and as our evaluation showed, it is not competitive in terms of time-performance with our approach. This is due to its relatively loose lower bound and sub-optimal search space pruning strategy, which force the algorithm to perform more work than necessary.

Exact discord discovery is a problem that has attracted lots of attention. The approaches that have been proposed in the literature can be divided in the following two different categories. First, the *index-based* solutions, i.e., Haar wavelets (Fu et al. 2006; Bu et al. 2007) and SAX (Keogh et al. 2005, 2007; Senin et al. 2015), where series are first discretized and then inserted in an index structure that supports fast similarity search. Second, the *sequential scan* solutions (Yeh et al. 2016; Liu et al. 2009; Fu et al. 2006; Luo and Gallagher 2011; Yankov et al. 2007b; Zhu et al. 2016), which consider the direct subsequence pairwise distance computations, and the corresponding search space optimization.

Indexing techniques are based on the discretization of the real valued data series, with several user defined parameters required for this operation. In general, selecting and tuning these parameters is not trivial, and the choices made may influence the behavior of the discord discovery algorithm, since it is strictly dependent on the quality of the data representation. In this regard, the most recent work in this category, GrammarViz (Senin et al. 2015), proposes a method of *Top-k 1st*discord search based on grammar compression of data series represented by discrete SAX coefficients. These representations are then inserted in a hierarchical structure, which permits us to prune unpromising candidate subsequences. The intuition is that rare patterns are assigned to representations that have high *Kolmogorov complexity*. This means that a rare SAX string is not compressible, due to the lack of repeated terms.

The state of the art for the sequential scan methods is represented by STOMP, since computing the matrix profile permits to discover, in the same fashion as motifs, the *Top-k 1st*discords. Surprisingly, just one work exists that addresses the problem of mth discord discovery (Yankov et al. 2007b). The authors of this work, proposed the Disk Aware discords Discovery algorithm (DAD), which is based on a smart sequential scan performed on disk resident data. This algorithm is divided into two parts. The first is discord candidate selection, where it identifies the sequences, whose nearest neighbor distance is less than a predefined range. The second part, which is called refinement, is applied in order to find the exact discords among the candidates. Despite the good performance that this algorithm exhibits in finding the first discord, when *m* is greater than one, it becomes hard to estimate an effective range. In turn, this leads to scalability problems, due to the explosion of the number of distances to compute.

In summary, while there exists a large and growing body of work on the motif and discord discovery problems, this work offers the first scalable, parameter-light, *exact* variable-length algorithm in the literature for solving both these problems.

9 Conclusions

Motif and discord discovery are important problems in data series processing across several domains, and key operations necessary for several analysis tasks. Even though much effort has been dedicated to these problems, no solution had been proposed for discovering motifs and discords of different lengths.

In this work, we propose the first framework for variable-length motif and discord discovery. We describe a new distance normalization method, as well as a novel distance lower bounding technique, both of which are necessary for the solution to our problem. We experimentally evaluated our algorithm by using five real datasets from diverse domains. The results demonstrate the efficiency and scalability of our approach (up to 20 × faster than the state of the art), as well as its usefulness.

In terms of future work, we would like to further improve the scalability of VALMOD. We also plan to extend VALMOD in order to efficiently compute a complete matrix profile for each length in the input range. This would enable us to support more diverse applications, such as discovery of *shapelets* (Ye and Keogh 2009).

References

Agrawal R, Faloutsos C, Swami AN (1993) Efficient similarity search in sequence databases. In: Foundations of data organization and algorithms, 4th international conference, FODO'93, pp 69–84

Bagnall A, Cole RL, Palpanas T, Zoumpatianos K (2019) Data series management (Dagstuhl seminar 19282). Dagstuhl Rep 9(7):24–39

Boniol P, Palpanas T (2020) Series2Graph: graph-based subsequence anomaly detection for time series. In: PVLDB

Boniol P, Linardi M, Roncallo F, Palpanas T (2020) Automated anomaly detection in large sequences. In: ICDE

Bu Y, Leung OT, Fu AW, Keogh EJ, Pei J, Meshkin S (2007) WAT: finding top-k discords in time series database. In: SIAM, pp 449–454

Camerra A, Palpanas T, Shieh J, Keogh E (2010) iSAX 2.0: indexing and mining one billion time series. In: IEEE ICDM, pp 58–67

Camerra A, Shieh J, Palpanas T, Rakthanmanon T, Keogh EJ (2014) Beyond one billion time series: indexing and mining very large time series collections with *i*SAX2+. KAIS 39(1):123–151

Cartlidge E (2016) Seven-year legal saga ends as Italian official is cleared of manslaughter in earthquake trial. Science. 3 Oct 2016

Chakrabarti K, Keogh EJ, Mehrotra S, Pazzani MJ (2002) Locally adaptive dimensionality reduction for indexing large time series databases. ACM Trans Database Syst 27(2):188–228

Chiu BY, Keogh EJ, Lonardi S (2003) Probabilistic discovery of time series motifs. In: ACM SIGKDD, pp 493–498

Dallachiesa M, Palpanas T, Ilyas IF (2014) Top-k nearest neighbor search in uncertain data series. In: PVLDB, vol 8, no 1, pp 13–24

Dua D, Graff C (2019) UCI machine learning repository. http://archive.ics.uci.edu/ml

Echihabi K, Zoumpatianos K, Palpanas T, Benbrahim H (2018) The Lernaean Hydra of data series similarity search: an experimental evaluation of the state of the art. In: PVLDB, vol 12, no 2, pp 112–127

Echihabi K, Zoumpatianos K, Palpanas T, Benbrahim H (2019) Return of the lernaean hydra: experimental evaluation of data series approximate similarity search. In: PVLDB, vol 13, no 3, pp 403–420

Fu AW, Leung OT, Keogh EJ, Lin J (2006) Finding time series discords based on Haar transform. In: ADMA, vol 4093, pp 31–41

Gao Y, Lin J (2018) Exploring variable-length time series motifs in one hundred million length scale. Data Min Knowl Discov 32(5):1200–1228

Gao Y, Lin J, Rangwala H (2016) Iterative grammar-based framework for discovering variable-length time series motifs. In: IEEE ICMLA, pp 7–12

Gisler C, Ridi A, Zufferey D, Khaled OA, Hennebert J (2013) Appliance consumption signature database and recognition test protocols. In: 2013 WoSSPA, pp 336–341

Gogolou A, Tsandilas T, Palpanas T, Bezerianos A (2019) Progressive similarity search on time series data. In: EDBT/ICDT CEUR workshop proceedings, vol 2322

Gogolou A, Tsandilas T, Echihabi K, Palpanas T, Bezerianos A (2020) Data series progressive similarity search with probabilistic quality guarantees. In: ACM SIGMOD

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE (2000) Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. Circulation 101(23):e215–e220

Grabocka J, Schilling N, Schmidt-Thieme L (2016) Latent time-series motifs. In: TKDD, vol 11, no 1, pp 6:1–6:20

Healey J, Picard R (2016) Detecting stress during real-world driving tasks using physiological sensors. IEEE Trans Intell Transp Syst 6(2):156–166

Jagadish HV, Mendelzon AO, Milo T (1995) Similarity-based queries. In: ACM SIGACT-SIGMOD-SIGART, pp 36–45

Jensen SK, Pedersen TB, Thomsen C (2017) Time series management systems: a survey. IEEE Trans Knowl Data Eng 29(11):2581–2600

Kashyap S, Karras P (2011) Scalable KNN search on vertically stored time series. In: ACM SIGKDD, pp 1334–1342

Keogh EJ (2011) Machine learning in time series databases (tutorial). In: AAAI

Keogh EJ, Lin J, Fu AW (2005) HOT SAX: efficiently finding the most unusual time series subsequence. In: IEEE ICDM, pp 226–233

Keogh EJ, Lonardi S, Ratanamahatana CA, Wei L, Lee S, Handley J (2007) Compression-based data mining of sequential data. Data Min Knowl Discov 14(1):99–129

Kondylakis H, Dayan N, Zoumpatianos K, Palpanas T (2018) Coconut: a scalable bottom-up approach for building data series indexes. In: PVLDB, vol 11, no 6, pp 677–690

Kondylakis H, Dayan N, Zoumpatianos K, Palpanas T (2019) Coconut palm: static and streaming data series exploration now in your palm. In: ACM SIGMOD, pp 1941–1944

Li Y, Hou L, Yiu ML, Gong Z (2015) Quick-motif: an efficient and scalable framework for exact motif discovery. In: ICDE, pp 579–590

Linardi M (2017) VALMOD support web page. http://www.mi.parisdescartes.fr/~mlinardi/VALMOD.html. Accessed Dec 2017

Linardi M, Palpanas T (2018a) Scalable, variable-length similarity search in data series: the ULISSE approach. In: PVLDB, vol 11, no 13, pp 2236–2248

Linardi M, Palpanas T (2018b) ULISSE: ultra compact index for variable-length similarity search in data series. In: ICDE, pp 1356–1359

Linardi M, Zhu Y, Palpanas T, Keogh EJ (2018a) Matrix profile X: VALMOD-scalable discovery of variable-length motifs in data series. In: ACM SIGMOD, pp 1053–1066

Linardi M, Zhu Y, Palpanas T, Keogh EJ (2018b) VALMOD: a suite for easy and exact detection of variable length motifs in data series. In: ACM SIGMOD, pp 1757–1760

Liu Y, Chen X, Wang F (2009) Efficient detection of discords for time series stream. Advances in data and web management. Springer, Berlin, pp 629–634

Luo W, Gallagher M (2011) Faster and parameter-free discord search in quasi-periodic time series. PAKDD 6635:135–148

Luo W, Gallagher M, Wiles J (2013) Parameter-free search of time-series discord. J Comput Sci Technol 28(2):300–310

Marzal A, Vidal E (1993) Computation of normalized edit distance and applications. IEEE Trans Pattern Anal Mach Intell 15(9):926–932

Minnen D, Jr CLI, Essa IA, Starner T (2007) Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In: AAAI conference on artificial intelligence, pp 615–620

Mirylenka K, Christophides V, Palpanas T, Pefkianakis I, May M (2016) Characterizing home device usage from wireless traffic time series. In: EDBT, pp 551–562

Mohammad Y, Nishida T (2012) Unsupervised discovery of basic human actions from activity recording datasets. In: 2012 IEEE/SICE international symposium on system integration (SII), pp 402–409

Mohammad YFO, Nishida T (2014) Exact discovery of length-range motifs. In: Intelligent information and database systems—6th Asian conference, vol 8398. ACIIDS, pp 23–32

Mueen A, Chavoshi N (2015) Enumeration of time series motifs of all lengths. Knowl Inf Syst 45(1):105–132

Mueen A, Keogh EJ, Zhu Q, Cash S, Westover MB (2009) Exact discovery of time series motifs. In: SIAM SDM, pp 473–484

Mueen A, Hamooni H, Estrada T (2014) Time series join on subsequence correlation. In: IEEE ICDM, pp 450–459

Neupane D, Moss CB, van Bruggen AH (2016) Estimating citrus production loss due to citrus huanglongbing in Florida. Annual meeting, Southern Agricultural Economics Association, San Antonio, TX

Palpanas T (2015) Data series management: the road to big sequence analytics. In: SIGMOD Record, vol 44, no 2, pp 47–52

Palpanas T (2016) Big sequence management: a glimpse of the past, the present, and the future. In: SOFSEM 2016, vol 9587, pp 63–80

Palpanas T (2017) The parallel and distributed future of data series mining. In: High performance computing & simulation (HPCS), pp 916–920

Palpanas T (2020) Evolution of a data series index. In: CCIS, vol 1197, pp 61–75

Palpanas T, Beckmann V (2019) Report on the first and second interdisciplinary time series analysis workshop (ITISA). In: SIGMOD record, vol 48, no 3, pp 916–920

Papadimitriou S, Yu PS (2006) Optimal multi-scale patterns in time series streams. In: ACM SIGMOD, pp 647–658

Peng B, Fatourou P, Palpanas T (2018) Paris: the next destination for fast data series indexing and query answering. In: IEEE big data, pp 791–800

Peng B, Fatourou P, Palpanas T (2020a) MESSI: in-memory data series indexing. In: ICDE

Peng B, Palpanas T, Fatourou P (2020b) ParIS+: data series indexing on multi-core architectures. In: TKDE

Rafiei D, Mendelzon AO (1998) Efficient retrieval of similar time sequences using DFT. In: Foundations of data organization and algorithms, 4th international conference, FODO'98, pp 249–257

Raza U, Camerra A, Murphy AL, Palpanas T, Picco GP (2015) Practical data prediction for real-world wireless sensor networks. IEEE Trans Knowl Data Eng 27(8):2231–2244

Rong K, Bailis P (2017) ASAP: prioritizing attention via time series smoothing. In: PVLDB, vol 10, no 11, pp 1358–1369

Roverso D (2000) Multivariate temporal classification by windowed wavelet decomposition and recurrent networks. In: ANS international topical meeting on nuclear plant instrumentation, control and human–machine interface

Saria S, Duchi A, Koller D (2011) Discovering deformable motifs in continuous time series data. In: IJCAI, pp 1465–1471

Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, Chen C, Frankenstein S (2015) Time series anomaly discovery with grammar-based compression. In: EDBT, pp 481–492

Shieh J, Keogh EJ (2008) iSAX: indexing and mining terabyte sized time series. In: ACM SIGKDD, pp 623–631

Sinha S (2002) Discriminative motifs. In: Proceedings of the sixth annual international conference on computational biology, RECOMB 2002, pp 291–298

Soldi S, Beckmann V, Baumgartner WH, Ponti G, Shrader CR, Lubinski P, Krimm HA, Mattana F, Tueller J (2014) Long-term variability of AGN at hard X-rays. Astron Astrophys 563:A57

Syed Z, Stultz CM, Kellis M, Indyk P, Guttag JV (2010) Motif discovery in physiological datasets: a methodology for inferring predictive elements. In: TKDD, vol 4, no 1, pp 2:1–2:23

Terzano MG, Parrino L, Sherieri A, Chervin R, Chokroverty S, Guilleminault C, Hirshkowitz M, Mahowald M, Moldofsky H, Rosa A, Thomas R, Walters A (2001) Atlas, rules, and recording techniques for the scoring of cyclic alternating pattern (cap) in human sleep. Sleep Med 2(6):537–553

Wang J, Balasubramanian A, Mojica de la Vega L, Green JR, Samal A, Prabhakaran B (2013a) Word recognition from continuous articulatory movement time-series data using symbolic representations. In: Proceedings of the fourth workshop on speech and language processing for assistive technologies (SLPAT), pp 119–127

Wang Y, Wang P, Pei J, Wang W, Huang S (2013b) A data-adaptive and dynamic segmentation index for whole matching on time series. In: PVLDB, vol 6, no 10, pp 793–804

Whitney CW, Gottlieb DJ, Redline SS, Norman RG, Dodge RR, Shahar E, Surovec SA, Nieto FJ (1998) Reliability of scoring respiratory disturbance indices and sleep staging. Sleep 21(7):749–57

Yagoubi DE, Akbarinia R, Masseglia F, Palpanas T (2017) DPiSAX: massively distributed partitioned iSAX. In: IEEE ICDM, pp 1135–1140

Yagoubi DE, Akbarinia R, Masseglia F, Palpanas T (2020) Massively distributed time series indexing and querying. IEEE Trans Knowl Data Eng 32(1):108–120

Yankov D, Keogh EJ, Medina J, Chiu BY, Zordan VB (2007a) Detecting time series motifs under uniform scaling. In: ACM SIGKDD, pp 844–853

Yankov D, Keogh EJ, Rebbapragada U (2007b) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. In: IEEE ICDM, pp 381–390

Yankov D, Keogh EJ, Rebbapragada U (2008) Disk aware discord discovery: finding unusual time series in terabyte sized datasets. Knowl Inf Syst 17(2):241–262

Ye L, Keogh EJ (2009) Time series shapelets: a new primitive for data mining. In: ACM SIGKDD, pp 947–956

Yeh CM, Zhu Y, Ulanova L, Begum N, Ding Y, Dau HA, Silva DF, Mueen A, Keogh EJ (2016) Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: IEEE ICDM, pp 1317–1322

Yingchareonthawornchai S, Sivaraks H, Rakthanmanon T, Ratanamahatana CA (2013) Efficient proper length time series motif discovery. In: IEEE ICDM, pp 1265–1270

Zhu Y, Zimmerman Z, Senobari NS, Yeh CM, Funning G, Mueen A, Brisk P, Keogh EJ (2016) Matrix profile II: exploiting a novel algorithm and GPUs to break the one hundred million barrier for time series motifs and joins. In: IEEE ICDM, pp 739–748

Zoumpatianos K, Palpanas T (2018) Data series management: fulfilling the need for big sequence analytics. In: ICDE, pp 1677–1678

Zoumpatianos K, Lou Y, Palpanas T, Gehrke J (2015) Query workloads for data series indexes. In: ACM SIGKDD, pp 1603–1612

Zoumpatianos K, Idreos S, Palpanas T (2016) ADS: the adaptive data series index. VLDB J 25(6):843–866

Zoumpatianos K, Lou Y, Ileana I, Palpanas T, Gehrke J (2018) Generating data series query workloads. VLDB J 27(6):823–846

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.