Segmentation-Based Methods for Top-*k* Discords Detection in Static and Streaming Time Series Under Euclidean Distance

Huynh Thi Thu Thuy[1,2(✉)], Duong Tuan Anh[1,3], and Vo Thi Ngoc Chau[1]

[1] Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology - VNU-HCM, Ho Chi Minh City, Vietnam
{8141217, chauvtn}@hcmut.edu.vn, anh.dt@huflit.edu.vn
[2] Center for Applied Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam
[3] Department of IT, HCM City University of Foreign Languages and Information Technology, Ho Chi Minh City, Vietnam

Abstract. Detecting top-*k* discords in time series is more useful than detecting the most unusual subsequence since the result is a more informative and complete set, rather than a single subsequence. The first challenge of this task is to determine the length of discords. Besides, detecting top-*k* discords in streaming time series poses another challenge that is fast response when new data points arrive at high speed. To handle these challenges, we propose two novel methods, TopK-EP-ALeader and TopK-EP-ALeader-S, which combine segmentation and clustering for detecting top-*k* discords in static and streaming time series, respectively. Moreover, a circular buffer is built to store the local segment of a streaming time series and calculate anomaly scores efficiently. Along with this circular buffer, a delayed update policy is defined for achieving instant responses to overcome the second challenge. The experiments on nine datasets in different application domains confirm the effectiveness and efficiency of our methods for top-*k* discord discovery in static and streaming time series.

Keywords: Anomaly detection · Top-*k* discords · Segmentation · Clustering · Streaming time series

1 Introduction

A streaming time series is an unbounded sequence of data points. In this sequence, new data points are continuously appended as time progresses. Recently, anomaly detection in streaming time series has emerged as an attractive topic because more applications need to be processed in real time rather than in batches. Such applications that include the anomaly detection task on streaming time series are listed as follows: detecting unusual patterns in flowing electrocardiograms (ECG) data from patients [11] and detecting anomalous patterns in streaming sensor data [1].

P. Cong Vinh and A. Rakib (Eds.): ICCASA 2021, LNICST 409, pp. 147–163, 2021.
https://doi.org/10.1007/978-3-030-93179-7_12

Compared to static time series, streaming time series have their own characteristics: (1) Data elements are frequently appended in streaming time series. (2) The size of a streaming time series is potentially unlimited. (3) Data synopsis and one-pass algorithms are often required to achieve a real time response as it is difficult to store all the data in memory or on disk with their frequent fast updates. Because of these three characteristics, several previous methods that were developed for static time series may not work in the streaming time series scenario.

On the other hand, discord detection in streaming time series is more difficult than similarity search in streaming time series. The major challenges for the existing anomaly detection methods on streaming time series are how to determine the length of anomalous subsequences and how to incrementally update the top-*k* discords whenever a new data point arrives.

Furthermore, finding top-*k* discords in time series is more important than only finding the most unusual subsequence since the set of top-*k* discords contains not only the most unusual subsequence but also some other important unusual subsequences. It makes the result more informative and complete.

In this paper, we devise two effective and efficient methods, named TopK-EP-ALeader and TopK-EP-ALeader-S, for detecting top-*k* discords in static and streaming time series, respectively, dealing with the aforementioned challenges. The main contributions of our paper are highlighted as follows.

The first contribution is TopK-EP-ALeader, our novel top-*k* discords detection algorithm in static time series under Euclidean distance. In TopK-EP-ALeader, a segmentation method based on the major extrema method proposed by Fink and Gandhi [4] is used to divide a time series into subsequences. After that, an incremental clustering method is applied to group similar subsequences into the same cluster and at the same time, dissimilar subsequences into different clusters. These subsequences in the resulting clusters are processed to return top-*k* discords. Through an empirical evaluation, TopK-EP-ALeader outperforms TopK-EP-ILeader, which is a variant of EP-ILeader [16] used for top-*k* discords detection in static time series.

The second contribution of our work is TopK-EP-ALeader-S, our extended version of TopK-EP-ALeader for detecting top-*k* discords in a more challenging scenario: streaming time series. TopK-EP-ALeader-S is a new algorithm that can find top-*k* discords immediately after one new pattern appears in the streaming time series. This remarkable feature of TopK-EP-ALeader-S is defined from the online property of the segmentation algorithm and the incrementality of the clustering algorithm. Indeed, the experimental results on various streaming time series show that TopK-EP-ALeader-S can bring out accurate top-*k* discords on the fly.

Furthermore, at the center of the two aforementioned contributions is the third contribution of our work. For this contribution, we propose A-Leader, an efficient subsequence clustering method used in TopK-EP-ALeader and TopK-EP-ALeader-S. A-Leader is defined as an improved version of I-Leader in our previous work [17] by speeding up Euclidean distance computation so that the subsequence clustering process can be accelerated.

The remainder of this paper is structured as follows. Section 2 introduces some background and related works. Section 3 describes supporting techniques to our proposed algorithms. Section 4 introduces our two proposed algorithms: TopK-EP-

ALeader and TopK-EP-ALeader-S for detecting top-k discords in static time series and streaming time series. The experimental results of our proposed algorithms TopK-EP-ALeader and TopK-EP-ALeader-S are reported in Sect. 5. Finally, Sect. 6 gives some conclusions and remarks on future works.

2 Background and Related Works

2.1 Background

Time series anomaly detection means finding the unusual (anomalous, novel, deviant, *discord*) subsequences in a time series. A time series discord is a subsequence that is maximally different from its closest matching subsequence. However, generally, the best matches of a given subsequence (apart from itself) tend to be very close to the subsequence under consideration. Such matches are called *trivial matches*. When detecting anomaly subsequences, we should discard trivial matches which can hinder our anomaly detection process. So, we define a non-trivial match (non-self match) as follows. Using this definition, discord-related concepts are presented.

Definition 1. (*Non-self match*) Given a time series T, containing a subsequence C of length n starting at position p and a matching subsequence M starting at position q, we say that M is a non-trivial match to C if $|p - q| \geq n$.

Definition 2. (*1-discord*) Given a time series T, the subsequence C in T is considered as the most important discord (also called 1-discord or top-anomaly) in T if C has the largest distance to its nearest non-trivial match.

In fact, detecting top-k discords in time series is more important than just only finding the most unusual subsequence. We are therefore interested in finding top-k discords in time series.

Definition 3. (k^{th} - *discord*) Given a time series T and the subsequence D of length n beginning at position p is the k^{th}-discord in T if it has the k^{th} largest distance to its nearest non-trivial match, with no overlapping region to the i^{th} discord beginning at position p_i, for all $1 \leq i \leq k$. That is, $|p - p_i| \geq n$.

These above definitions which are commonly-used in time series discord detection, are from the work by Keogh et al. [8]. The problem of 1-discord detection can be solved by the brute-force algorithm which is given in [8]. In the brute-force algorithm, using a sliding window, we extract all possible candidate subsequences in the outer loop and then find the distance to the nearest non-self match for each candidate subsequence in the inner loop. The candidate subsequence with the largest distance to its nearest non-self match is the 1-discord. Since the brute-force algorithm is a window-based method for detecting discords in a time series with a nested loop, it incurs high complexity. Its complexity is $\boldsymbol{O}(m^2)$, where m is the length of the time series.

Notice that we can modify the brute-force algorithm to obtain the algorithm which can detect top-k discords in time series.

2.2 Related Works on Anomaly Detection in Static Time Series

Bu et al. [2] proposed WAT algorithm for finding top-*k* discords in static time series. This algorithm needs to predetermine the discord length. It is based on a sliding-window approach for time series discord discovery. The discord length parameter setting is not easy to be done and the sliding window approach gives WAT algorithm a high computational cost. So, it might be very complicated to adapt WAT for detecting top-*k* discords in streaming time series.

Linardi et al. [9] proposed a variable-length motif and discord discovery framework. In this framework, the subsequence pairs with the largest Euclidean distances of each length in the user-defined range are returned as discords. This framework can find top k-discords with variable length in a time series.

Compared to our work, [9] is an interesting work for finding top k-discords with variable length in a time series. However, this algorithm requires complex parameter settings. Moreover, it is hard to determine which discord and which subsequence length are appropriate among many returned variable-length discords in user's applications. Above all, this algorithm aims to detect discords in static time series, while ours are dedicated to top-*k* discords detection from both static and streaming time series.

2.3 Related Works on Anomaly Detection in Streaming Time Series

For anomaly detection on streaming time series, a few research works have been proposed as briefly reviewed below.

Liu et al. [10] proposed a framework for anomaly detection in streaming time series called Detection of Continuous Discords (DCD). DCD can detect continuous 1-discords from local segments of a streaming time series which reside in a buffer with a predefined size. One major technique in DCD framework is that it limits the search space to further enhance the efficiency of the discord detection process. Since DCD framework is a window-based method in anomaly detection, DCD still has a high computational cost.

Sanchez and Bustos [14] presented a method for 1-discord detection in static time series which employed bounding rectangles and R-tree to establish two ordering heuristics for the two loops in the 1-discord detection process. Sanchez and Bustos extended this idea for discord detection in streaming time series. However, the detected anomalous subsequence is not really the top-discord of the whole streaming time series in the buffer. In other words, the algorithm can detect any approximate anomalous subsequence from a streaming time series rather than the top discord from the streaming time series in the buffer.

SKDIS method proposed by Giao and Anh [5] is a window-based method for detecting top-*k* discords in streaming time series. SKDIS is the incremental version of the brute-force algorithm for detecting top-*k* discords in static time series. SKDIS has to maintain and update the historic information of the interim top-*k* discords detected from the current buffer whenever a new data point arrives. To improve the efficienc of anomaly detection process, SKDIS applies some techniques from the suite of techniques UCR-ED [13] to speed up Euclidean distance computation.

All the aforementioned methods for discord detection in streaming time series belong to the window-based category which is often costly in terms of processing time. So, a new effective and efficient method is thus needed for detecting top-*k* discords in streaming time series.

3 Supporting Techniques

Before presenting the two proposed methods for finding top-*k* discords in static and streaming time series, we describe some supporting techniques for these two methods.

3.1 Accelerating Euclidean Distance Calculation in Clustering Algorithm

Euclidean distance (ED) calculates the distance between the two subsequences $C_1 = x_1, x_2,\ldots, x_m\}$ and $C_2 = \{y_1, y_2,\ldots, y_m\}$ with the same length m by taking the square root of the sum of squares of differences of the two corresponding points in two subsequences as shown in the following formula.

$$ED(C_1, C_2) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Since computing the distance between two time-series subsequences is time-consuming, in this work, we apply two techniques to accelerate ED calculation in I-Leader clustering algorithm. These two techniques are inspired from the UCR-ED suite introduced by Rakthanmanon et al. [13]. The two techniques are described as follows:

- *Using the Squared Distance.* ED uses a square root calculation. However, if we omit this step, the relative ranking of comparative subsequences is not changed, since the ED function is monotonic and concave. Furthermore, the absence of the square root function makes the ED computation faster.
- *Early Abandoning of ED.* During the ED computation, if the current sum of the squared differences between each pair of corresponding data points $(x_i - y_i)$ $(i = 1..k, k < m)$ exceeds a given threshold in the Leader clustering algorithm, then the computation is stopped. In other words, early abandoning technique can be used to prune the dissimilar subsequences from the set of subsequences so as to retain the candidates for further measuring their similarity with direct ED. Figure 1 illustrates the idea of early abandoning.

Fig. 1. An illustration of early abandoning in calculating Euclidean distance between C_1 and C_2.

When we apply these two techniques to improve I-Leader clustering algorithm [17], we obtain the new variant of I-Leader clustering and name it A-Leader algorithm (abbreviated for Accelerated variant of Leader).

3.2 A-Leader Clustering Algorithm

A-Leader is an improved version of I-Leader. I-Leader, proposed by [17], is also a variant of Leader algorithm, a well-known incremental clustering algorithm proposed by Hartigan [6]. I-Leader uses "*centroid*" as cluster representative rather than "*leader*" proposed in the original Leader algorithm.

In I-Leader, the distance between G_m and C_j is the distance between C_j and the centroid of cluster G_m. I-Leader computes the centroids of clusters in an incremental manner. Whenever a new subsequence t is put into a cluster, a technique is used to calculate a new centroid incrementally.

I-Leader clustering algorithm is described as follows.

- **Input**: A list C of subsequences and a user-specified distance threshold ε.
- **Output**: A list G of clusters.
- **Process**:

  Step 1: Assign the first subsequence, C_1, to the cluster G_1; set $i = 1$ and $j = 1$.
  Step 2: Set $j = j + 1$; consider clusters G_1 to G_i in ascending order of the index and assign C_j to cluster G_m ($1 \leq m \leq i$) if the distance between G_m and C_j is the smallest distance and this distance is less than a distance threshold ε. Otherwise, set $i = i + 1$ and assign C_j to a new cluster G_i.
  Step 3: Refine the clustering result of *Step 2* by merging the subsequences in *under-filled*" clusters to their nearest "*good*" clusters.
  Step 4: Repeat *Step 2* and *Step 3* until all of the subsequences in the list C are assigned to clusters.

Definitions of "*under-filled*" and "*good*" clusters are detailed in [17]. Here, we explain briefly about them. Clusters with few instances are called "*under-filled*" clusters and clusters with more instances are called "*good*" clusters. The number of instances in the cluster considered as "under-filled" or "good" is based on the distribution ratio of the instances in the clusters. So, Step 3 in I-Leader aims to improve the quality of the resultant clusters in terms of the distribution of the instances in the clusters. In Step 3, I-Leader merges the subsequences in "under-filled" clusters to their nearest "good" clusters. More details about I-Leader algorithm, interested readers can refer to [17].

In A-Leader, we apply some techniques to accelerate the computation of ED (as mentioned in the first part of Sect. 3) when we have to calculate the distance between the subsequence and the centroid of a particular cluster to decide whether the subsequence is pushed into the cluster or not. The threshold value for speeding up ED calculation here is the threshold ε of I-Leader algorithm. Thus, A-Leader differs from I-Leader in the way of calculating ED while all the other steps of A-Leader are the same as those of I-Leader.

3.3 EP-ALeader

EP-ALeader algorithm is an improved version of EP-ILeader by using A-Leader clustering instead of I-Leader clustering. EP-ILeader (abbreviated for Extreme Points and Improved Leader) is an anomaly detection algorithm in static time series proposed in our previous work [16].

For our approach, we do not use any sliding window to extract subsequences. Instead we use a segmentation method for extracting subsequences in time series and a clustering method to cluster the extracted subsequences. Then, based on the result of the clustering step, we can calculate the anomaly score of each subsequence. In order to calculate an anomaly score of each subsequence, we use the two following definitions given by [7].

Definition 4. (*Large and Small Clusters*). Given a set of patterns D that has been grouped into a set of clusters $C = \{C_1, C_2,\ldots, C_k\}$ such that $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$. Given two parameters α and β, if $|C_1| + |C_2| + \ldots + |C_b|)$ is greater than $|D| * \alpha$ and $|C_b| / |C_{b+1}| \geq \beta$ then clusters $C_1, C_2,\ldots, C_b$ are large clusters and clusters $C_{b+1}, C_{b+2},\ldots, C_k$ are small ones, in which b is an integer from 1 to k.

The set of large clusters is defined as $LC = \{C_i \mid i \leq b\}$ and the set of small clusters is defined as: $SC = \{C_j \mid j > b\}$.

Definition 5. (*Anomaly score*). Given a time series T that has been segmented into the set of subsequences and this set of subsequences has been grouped into a set of clusters $C = \{C_1, C_2,\ldots, C_k\}$ such that $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$. The meanings of LC and SC are the same as they are defined in Definition 4. For each subsequence t of time series T, an anomaly score of t is defined as follows:

$$Score(t) = \begin{cases} |C_i| * \min\left(dist\left(t, C_j\right)\right), if\ t \in C_i, C_i \in SC\ and\ C_j \in LC(j = 1..b) \\ |C_i| * dist(t, C_i), if\ t \in C_i, C_i \in LC \end{cases}$$

where $dist(t, C_i)$ is the distance from subsequence t to cluster C_i. In the context of EP-ALeader and EP-ILeader, the distance from subsequence t to cluster C_i is the distance from subsequence t to the centroid of cluster C_i.

EP-ALeader differs from EP-ILeader only in Step 3. In Step 3, EP-ILeader uses ordinary Euclidean distance computation while EP-ALeader uses Euclidean distance with the two speeding up techniques previously presented.

The pseudo code of EP-ALeader is describes as follows.

- **Input**: A time series T with length m; compression rate R.
- **Output**: The pattern and position of the top anomaly.
- **Process**:

  Step 1: Divide the time series T into subsequences (pattern candidates) using the major extreme point method proposed by Fink and Gandhi [4].

  Step 2: Transform the extracted subsequences to those with the same length by using homothetic transformation [18].

  Step 3: Cluster the pattern candidates using the A-Leader algorithm and calculate the anomaly scores of all the candidates.

  Step 4: Identify the pattern candidate with the largest anomaly score as the top anomaly pattern of the time series.

4 Proposed Methods

In this section, we describe two proposed methods: TopK-EP-ALeader and TopK-EP-ALeader-S for detecting the top *k*-discords in static time series and streaming time series, respectively.

4.1 TopK-EP-ALeader for Finding Top *k*-discords in Static Time Series

For top-*k* discords detection in static time series, we propose TopK-EP-ALeader, which is an extension of EP-ALeader previously introduced. In our method, the definition of time series discords given by Keogh et al. [8] is used.

TopK-EP-ALeader is different from EP-ALeader only in Step 4. In Step 4, TopK-EP-ALeader returns the top-*k* discords which are the subsequences with the *k*-th largest anomaly scores while EP-ALeader returns only the 1-discord which is the subsequence with the largest anomaly score. Thanks to the existence of anomaly scores, top-*k* discords detection in a time series is straightforward and thus incurs no overhead.

4.2 TopK-EP-ALeader-S for Finding Top *k*-discords in Streaming Time Series

For detecting top-*k* discords in streaming time series, we propose TopK-EP-ALeader-S as an extended version of our TopK-EP-ALeader algorithm. The extended features of TopK-EP-ALeader-S are determined to overcome the challenges in finding top-*k* discords over streaming time series.

Compared to TopK-EP-ALeader, TopK-EP-ALeader-S is defined with a *moving window* to store time series in the streaming time series context. In this window, a local segment of streaming time series is stored as time goes by. Only the most recent data points are included in the moving window.

In addition, TopK-EP-ALeader-S works with a delayed update policy instead of an instant update policy for more efficiency.

4.2.1 The Circular Buffer

In our work, the moving window is implemented as a *circular buffer*. Indeed, this circular buffer stores a segment of a streaming time series that contains the most recent data points of the streaming time series. In addition, this buffer operates its contents in a circular fashion.

The working of the circular buffer is illustrated in Fig. 2. When handling streaming time series, after the buffer is full, a newly incoming data point will *overwrite* the oldest point in the current circular buffer.

The length of the buffer might have an impact on the efficiency of the top-*k* discords detection in streaming time series. In this work, the buffer length is estimated based on the *period* of the time series under consideration. The buffer length should be a multiple of the period of the time series in order that for periodic time series, the newly incoming data point and the obsolete data point will have some similar variation. There have been a few methods for periodicity detection in a time series.

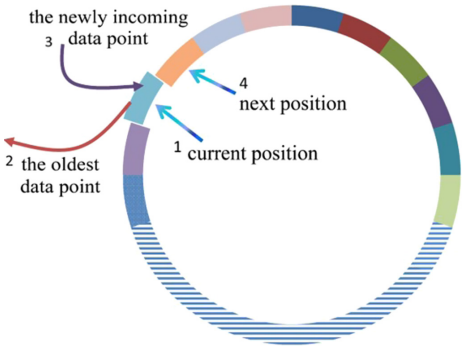

Fig. 2. Circular buffer.

To detect the period of a time series, we can apply a time series motif discovery method. Here we apply a method for estimating the length of the 1-motif in a time series, proposed by Phien [12]. This method is based on a variable-length motif discovery approach which uses grammar induction algorithm at its core.

4.2.2 The Delayed Update Policy

In the streaming time series context, as data points arrive continuously, it takes time to execute every process for detecting top-*k* discords with each new data point. For example, with our segment-based method, the Major Extrema method, A-Leader, and TopK-EP-ALeader need to be invoked to find top-*k* discords whenever a new data point arrives. Therefore, for more efficiency, we define a delayed update policy to determine when we should segment the time series again and then activate the anomaly detection process. In this delayed update policy, instead of the arrival of each new data point, we use the arrival of each new major extreme point to trigger the process of TopK-EP-ALeader-S. That is whenever a new major extreme point is found from newly incoming data points, a process of detecting new top-*k* discords in the current buffer is activated. At that time, we obtain a new subsequence that contains new data points. This subsequence is assigned to the nearest cluster. Furthermore, if the oldest data point in the buffer has not been the right end point of the oldest subsequence, the oldest subsequence is still kept in its certain cluster. The oldest subsequence is removed from its certain cluster if the oldest data point in the buffer is the last data point of the oldest subsequence.

4.2.3 TopK-EP-ALeader-S Algorithm

Using the circular buffer and the delayed update policy, TopK-EP-ALeader-S is described as follows. At first, when the circular buffer is full with the data points from a streaming time series, TopK-EP-ALeader-S invokes TopK-EP-ALeader to detect the top-*k* discords in the first buffer. After that starting history point, TopK-EP-ALeader-S begins to apply the delayed update policy for detecting more new top-*k* discords continuously from the remaining part of the streaming time series. From now on, TopK-EP-ALeader-S detects more top-*k* discords in the current buffer whenever the

latest data point in the buffer is identified as a major extreme point. The pseudo code of TopK-EP-ALeader-S is presented in Algorithm 1.

Algorithm 1 ***TopK-EP-ALeader-S algorithm***
Input: - *y*: a streaming time series
- *k*: the number of the desired top-*k* discords
- *R*: Compression rate
-*Threshold*: Threshold for cluster assignment
Output:- The top-*k* discords and positions of the top-*k* discords
1: Read the time series data points into buffer *B* /* Initialize the buffer */
2: Call TopK-EP-ALeader(*B,R,Threshold,Sub-List,Cluster,Positions-of-top-k-discords*)
/* Detect the top-*k* discords in the current buffer;
Sub-List is the list of the extracted subsequences. */
3: Print out the top-*k* discords found in the current buffer and their positions
4: **Repeat**
5: **if** Type(Last Extreme Point) = Max **then**
6: Find the next major minimum point in the newly incoming data points
7: **else** /* Last extreme point is minimum*/
8: Find the next major maximum point in the newly incoming data points
9: **endif**
10: ExtractNewSubsequence(*B*, *Sub*) /* *Sub* is the new extracted subsequence. */
11: Homothety(Sub, *NewSub*) /**NewSub* is the new transformed subsequence*/
12: Insert(Newsub, *NewSub-List*, *Cluster*)
/* Insert the Newsub into *Cluster*;
NewSub-List is the *Sub-List* after the Newsub is inserted. */
13: **if** all the data points in the first subsequence *FirstSub* in *NewList* are out of buffer *B* **then**
14: Delete(FirstSub, *NewSub-List*, *Cluster*)
/* Remove the FirstSub out of *Cluster* and the *NewSub-List*} */
15: **endif**
16: Calculate anomaly scores for all the subsequences in *Cluster*
17: Print out the top-*k* discords corresponding to *k* subsequences with the greatest anomaly scores and their positions
18: **Until** Stop

In Algorithm 1, on lines 1–3, TopK-EP-ALeader-S invokes TopK-EP-ALeader to detect the first top-*k* discords in the full buffer at the first time. From that moment, on lines 4–18, its process is executed repeatedly whenever a new major extreme point is obtained. In particular, on lines 5–9, TopK-EP-ALeader-S checks the arrival of a new major extreme point incrementally. On lines 10–12, a new subsequence is extracted, transformed with homothety, and then inserted into its nearest cluster. After that, on lines 13–15, the content of the circular buffer is checked for up-to-date data and the clusters are updated accordingly. On line 16, an anomaly score of each subsequence in

the clusters is calculated as described previously. Finally, the top-*k* discords are returned from *k* subsequences with the greatest anomaly scores and their corresponding positions. It is also noted for the termination condition on line 18 of the loop **repeat.. until**. Normally this stop condition is true whenever the entire dataset has been processed. In reality, this iterative process keeps working if new data points still come.

5 Empirical Evaluation

To evaluate our proposed methods, we conduct an empirical evaluation study with two experiments as follows:

- Experiment 1: Evaluate TopK-EP-ALeader for detecting top-*k* discords in static time series.
- Experiment 2: Evaluate TopK-EP-ALeader-S for detecting top-*k* discords in streaming time series.

In our experiments, we use 9 datasets including *ECG5000*, *ECG*, *AEM*, *ERP*, *POWER*, *STOCK*, *Power_demand_Italy*, *AHA_0001_ECG*, and *AF_learning-set-n01*. The datasets *AF_learning-set-n01*, and *AHA_0001_ECG* are from The Research Resource for Complex Physiologic Signals [15]. The rest datasets are downloaded from the UCR Time series Classification/Clustering [3]. These datasets are from different domains: finance, medicine and industry. Table 1 describes all the datasets used in our experiments.

Table 1. Descriptions of all datasets in experiments.

No	Dataset	Source	Domain	Length	Period	Experiment
1	ECG5000	[3]	Medicine	5000	33	2
2	ECG	[3]	Medicine	20000	60	1, 2
3	AEM	[3]	Industry	17000	80	1, 2
4	ERP	[3]	Industry	10000	50	1,2
5	POWER	[3]	Industry	20000	226	1,2
6	STOCK	[3]	Finance	20000	100	1,2
7	Power_demand_Italy	[3]	Industry	1000	80	2
8	AHA_0001_ECG_2500	[15]	Medicine	2500	106	2
9	AF_learning-set-n01	[15]	Medicine	1280	433	2

We implemented the comparative methods in Visual C#. The experiments were conducted on an HP Intel(R) Core i7–3630 QM CPU with 2.40 GHz processor, 8 GB RAM.

Before conducting Experiment 1 and Experiment 2, we empirically evaluate A-Leader in comparison with I-Leader for the clustering process in our proposed methods. The experimental results on five datasets show that the clustering results of

A-Leader are similar to those of I-Leader, but on average, the clustering process of A-Leader is about 1.37 times faster than that of I-Leader.

Besides, we empirically evaluate EP-ALeader in comparison with EP-ILeader for the anomaly detection process. The experimental results on five datasets show that the 1-discord results of EP-ALeader match with those detected by Brute-Force and EP-ILeader, but on average, the execution of EP-ALeader is about 19.1 times faster than that of EP-ILeader.

We do not need to compare the efficiency of EP-ALeader with that of HOT SAX [8] since we already compared the efficiency of EP-ILeader to that of HOT SAX in our previous work [16] and knew that EP-ILeader could perform much faster than HOT SAX.

5.1 Experiment 1: Evaluation of TopK-EP-ALeader

This subsection reports the experimental results on TopK-EP-ALeader, a method for detecting top-*k* discords in static time series.

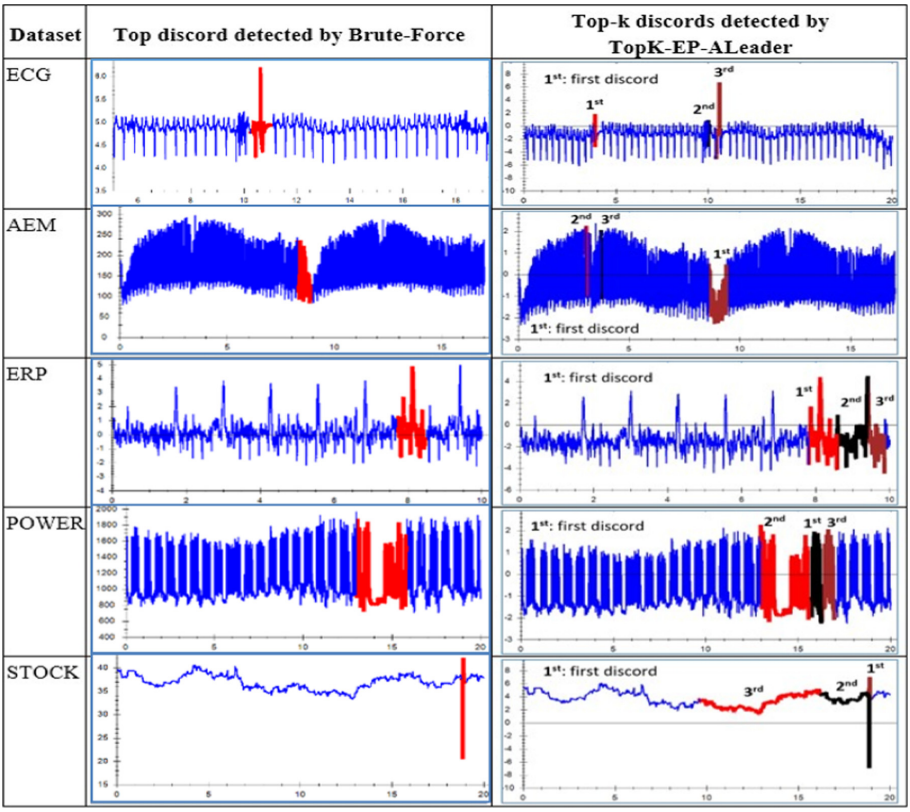

Fig. 3. Discords detected by Brute Force and TopK-EP-ALeader.

For checking the effectiveness of TopK-EP-ALeader, we check if top-*k* discords detected by TopK-EP-ALeader contain the top discord annotated by experts or not. Figure 3 shows that the top-*k* discords detected by TopK-EP-ALeader on each dataset include the top discord detected by Brute-Force. For example, Fig. 3 shows that with ECG dataset, Brute-Force only detects one top discord close to position 10000 and it ignores the 2nd discord and 3rd discord close to position 10000. Meanwhile, TopK-EP-ALeader exactly detects all three top discords in ECG dataset annotated by experts which are displayed in Fig. 5. This result shows that TopK-EP-ALeader is effective in finding top-*k* discords accurately from static time series.

For checking the efficiency of TopK-EP-ALeader, we compare the running time of TopK-EP-ALeader to that of TopK-EP-ILeader on five datasets. Since EP-ILeader is the algorithm that can detect the top discord in static time series [16], we have to modify EP-ILeader so that it can detect top-*k* discords in time series and name it TopK-EP-ILeader.

Table 2 depicts the running times (in seconds) of TopK-EP-ALeader and TopK-EP-ILeader over five test datasets. We can see that the runtime of TopK-EP-ALeader is less than that of TopK-EP-ILeader. In average, TopK-EP-ALeader can run faster than TopK-EP-ILeader about 1.9 times. This is obvious because TopK-EP-ALeader uses A-Leader clustering method and A-Leader is faster than I-Leader clustering method which is used in TopK-EP-ILeader.

Table 2. Running times (in seconds) of TopK-EP-ILeader and TopK-EP-ALeader.

Dataset	Length	Running time		TopK-EP- ILeader/ TopK-EP- ALeader
		TopK-EP- ILeader	TopK-EP- ALeader	
ECG	20,000	0.275	**0.234**	1.2
AEM	17,000	1.706	**0.508**	3.4
ERP	10,000	0.019	**0.011**	1.7
POWER	20,000	1.133	**0.661**	1.7
STOCK	20,000	0.016	**0.010**	1.6
			Average	**1.9**

5.2 Experiment 2: Evaluation of TopK-EP-ALeader-S

For TopK-EP-ALeader-S, we have to set one parameter: the buffer length which is based on the period of the time series. The period and buffer length used by TopK-EP-ALeader-S for each dataset are reported in Table 3. For example, as for POWER dataset, with its found period of 226, we set the buffer length to 452 (452 = 2*226).

Recall that TopK-EP-ALeader-S does not require the length of top-*k* discords as a parameter. The number *k* of top-*k* discords in our experiments is set to 3, the same value used in SKDIS method by Giao and Anh [5].

Table 3 describes nine datasets used in the evaluation of TopK-EP-ALeader-S for detecting top-*k* discords in streaming time series. Here, we set buffer length to a multiple of the period of each time series dataset.

Table 3. Buffer length for each dataset used in the experiments of detecting top-*k* discords in streaming time series.

Dataset	Length of dataset	Period	Buffer length
ECG	20000	60	12000
AEM	1000	80	480
ERP	10000	50	5000
POWER	1000	226	452
STOCK	20000	100	10000
Power_demand_Italy	1000	80	480
AHA_0001_ECG_2500	2500	106	1272
AF_learning-set-n01	1280	433	866
ECG5000	5000	33	2310

5.2.1 Effectiveness of TopK-EP-ALeader-S

For an illustration, we test the effectiveness of TopK-EP-ALeader-S in one dataset: ECG (electrocardiogram). Figure 4 shows the shapes of discords (in bold) in ECG dataset after TopK-EP-ALeader-S identifies some major extreme points in the streaming time series context. Figure 4 displays a view of the local segment of the time series after pushed into the buffer. The positions of discords in this case are about 4000, 10000, and 10500. It can be seen that our discord results over ECG dataset match those found by HOT SAX reported in [8] which are shown in Fig. 5. These three discords reported in [8] have also been detected as time progresses by TopK-EP-ALeader-S in our experiment. So, we can conclude that our discords detected by TopK-EP-ALeader-S match the top-3 discords in ECG dataset annotated by experts.

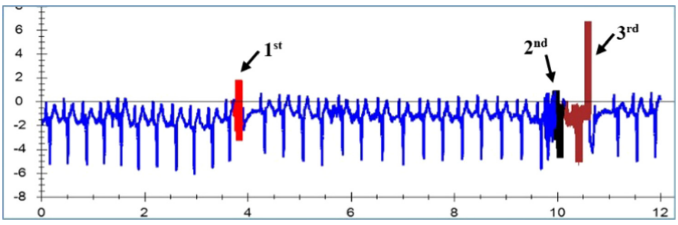

Fig. 4. Images of the top-3 discords detected by TopK-EP-ALeader-S.

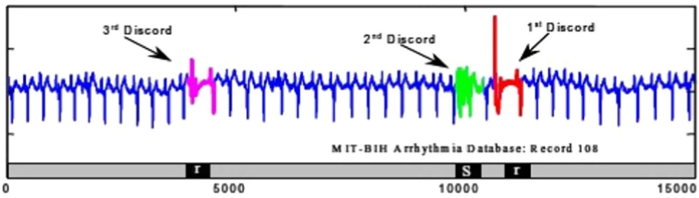

Fig. 5. Images of the top-3 discords in ECG dataset annotated by experts [8].

5.2.2 Efficiency of TopK-EP-ALeader-S

Table 4 shows the execution time (in seconds) of TopK-EP-ALeader-S after each new subsequence arrives over each of the nine datasets simulated as streaming time series. Moreover, we skip comparing the efficiency of TopK-EP-ALeader-S with those of the existing works like SKDIS [5] because of different approaches. SKDIS used the window-based approach while TopK-EP-ALeader-S uses the segmentation-based approach.

Table 4. Execution times (in seconds) of TopK-EP-ALeader-S after each new subsequence arrives.

No	Dataset	Execution time
1	ECG	0.006
2	AEM	0.005
3	ERP	0.004
4	POWER	0.007
5	STOCK	0.003
6	Power_demand_Italy	0.006
7	AHA_0001_ECG_2500	0.005
8	AF_learning-set-n01	0.007
9	ECG5000	0.005
Average		0.0053

To test the ability of immediate response of our proposed algorithm for streaming time series, we measure the execution time from the time point when a new major extreme point appears to the time point when the corresponding top-*k* discords are obtained. The response times for nine streaming time series are reported in Table 4. Table 4 shows that on average, TopK-EP-ALeader-S algorithm can respond immediately in about 5.3 ms whenever a new major extreme point appears.

Now, we answer the question how our online anomaly detection method meets the requirements of real world applications. Consider one experimented dataset: ECG (electrocardiogram data). As for ECG dataset, the full period of a heartbeat is about one second. Each extreme point will mark a half of the heartbeat (i.e. 0.5 s). Our TopK-EP-ALeader-S can detect top-*k* discords in about 6 ms (see Table 4). So, the speed of TopK-EP-ALeader-S is about 83 times faster than the transfer rate of ECG data. So, the

above analysis shows that TopK-EP-ALeader-S, our proposed algorithm for discovering top-k discords in streaming time series, can meet the practical requirement for real time anomaly detection on streaming power consumption data and ECG data.

6 Conclusions and Future Works

In this paper, we have proposed two effective and efficient methods for detecting top-k discords in static and streaming time series, TopK-EP-ALeader and TopK-EP-ALeader-S, respectively. These two methods hinge on major extrema-based segmentation and incremental subsequence clustering. In terms of effectiveness, the discords found by our two methods on several benchmark datasets match those annotated by the experts. In terms of efficiency, our two methods give fast response to discover top-k discords in both static and streaming time series via the experimental results on several time series in various application domains. Such better performance of our two methods is from a significant contribution of A-Leader clustering algorithm for incremental subsequence clustering and EP-ALeader algorithm for anomaly detection. As a result, our two methods can meet the practical requirements of time-critical real-world applications in both static and streaming time series contexts.

In the future, we plan to utilize TopK-EP-ALeader-S in some various real-world applications. Parallelizing the process of TopK-EP-ALeader on GPU is also of our interest for more efficiency.

References

1. Ahmad, S., Lavin, A., Purdy, S., Agha, Z.: Unsupervised real-time anomaly detection for streaming data. Neurocomputing **262**, 134–147 (2017)
2. Bu, Y., Leung, T.W., Fu, A.W.C., Keogh, E., Pei, J., Meshkin, S.: WAT: Finding top-k discords in time series database. In: Proceedings of the 2007 SIAM International Conference on Data Mining, pp. 449–454 (2007)
3. Chen, Y., et al.: The UCR Time series Classification/Clustering. https://www.cs.ucr.edu/~eamonn/time_series_data/. Accessed 2017
4. Fink, E., Gandhi, H.S.: Important extrema of time series. In: Proceedings of IEEE International Conference on System, Man and Cybernetics, pp. 366–372. Montreal, Canada (2007)
5. Giao, B.C., Anh, D.T.: Efficient search for top-k discords in streaming time series. Int. J. Bus. Intell. Data Min. **16**(4), 397–417 (2020)
6. Hartigan, J.A.: Clustering Algorithms. Wiley, New York (1975)
7. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recogn. Lett. **24** (9–10), 1641–1650 (2003)
8. Keogh, E., Lin, J., Fu, A.: HOT SAX: efficiently finding the most unusual time series subsequence. In: Proceedings of the Fifth IEEE International Conference on Data mining, pp. 226–233. Houston, Texas (2005)
9. Linardi, M., Zhu, Y., Palpanas, T., Keogh, E.: Matrix profile goes MAD: variable-length motif and discord discovery in data series. Data Min. Knowl. Disc. **34**(4), 1022–1071 (2020). https://doi.org/10.1007/s10618-020-00685-w

10. Liu, Y., Chen, X., Wang, F., Yin, J.: Efficient detection of discords for time series stream. In: Li, Q., Feng, L., Pei, J., Wang, S.X., Zhou, X., Zhu, Q.-M. (eds.) APWeb/WAIM -2009. LNCS, vol. 5446, pp. 629–634. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00672-2_62
11. Ngo, D.H., Veeravalli, B.: Design of a real-time morphology-based anomaly detection methods from ECG streams. In: Proceedings of IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 829–836 (2015)
12. Phien, N.N.: An efficient method for estimating time series motif length using sequitur algorithm. In: Meng, L., Zhang, Y. (eds.) MLICOM 2018. LNICSSITE, vol. 251, pp. 531–538. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00557-3_52
13. Rakthanmanon, T., et al.: Searching and mining trillions of time series subsequences under dynamic time warping. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 262–270, Beijing, China (2012)
14. Sanchez, H., Bustos, B.: Anomaly detection in streaming time series based on bounding boxes. In: Traina, A.J.M., Traina, C., Cordeiro, R.L.F. (eds.) SISAP 2014. LNCS, vol. 8821, pp. 201–213. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11988-5_19
15. The Research Resource for Complex Physiologic Signals. https://www.physionet.org. Accessed 22 Oct 2020
16. Thuy, H.T.T., Anh, D.T., Chau, V.T.N.: A novel method for time series anomaly detection based on segmentation and clustering. In: 2018 10th International Conference on Knowledge and Systems Engineering (KSE), pp. 276–281. IEEE (2018)
17. Thuy, H.T.T., Anh, D.T., Chau, V.T.N.: Incremental Clustering for time series data based on an improved leader algorithm. In: 2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF), pp. 1–6. IEEE (2019)
18. Truong, C.D., Anh, D.T.: An efficient method for motif and anomaly detection in time series based on clustering. Int. J. Bus. Intell. Data Min. **10**(4), 356–377 (2015)