From Cluster-Based Outlier Detection to Time Series Discord Discovery

Nguyen Huy Kha and Duong Tuan Anh(✉)

Faculty of Computer Science and Engineering,
Ho Chi Minh City University of Technology,
Ho Chi Minh City, Vietnam
dtanh@cse.hcmut.edu.vn

Abstract. Anomalous patterns or discords are just the kind of outliers in time series. In this paper, we present a new approach for time series discord discovery which is based on cluster-based outlier detection. In this approach, first, subsequence candidates are extracted from the time series using a segmentation method, then these candidates are transformed into the same length and are input for an appropriate clustering algorithm, and finally, we identify discords by using a measure suggested in the cluster-based outlier detection method given by He et al. 2003. The experimental results show that our approach is much more efficient than the HOTSAX algorithm in detecting time series discords while the anomalous patterns discovered by the two methods perfectly match with each other.

Keywords: Time series discord · Discord discovery · Segmentation · Important extreme points · Outlier detection · Cluster-based outlier detection

1 Introduction

Time series data are ubiquitous, touching almost every aspect of human life, in domains such as business, finance, industry, medicine, science or government. The problem of detecting unusual (abnormal, novel, deviant, anomalous, *discord*) time series has recently attracted much attention. Time series discord discovery brings out the "most unusual subsequence" in a time series. Areas that explore such time series discords are, for example, fault diagnostics, intrusion detection and data cleansing.

Some popular algorithms for discord discovery include window-based methods such as HOT SAX by Keogh et al. [14] and WAT by Bu et al. [1]; hidden Markov model-based such as TARZAN by Keogh et al. [13]; a method based on neural-network by Oliveira et al. [18] and a method based on one-class support vector machine by Ma et al. [17]. However, these methods still have two major limitations. First, they require the user to supply the length of the discord subsequence which is always unknown. Second, these algorithms involve high computational cost in the case of working with very large time series datasets.

Outlier detection is the process of detecting the data objects which are grossly different from or inconsistent with the remaining set of data. Searching for outliers is an important area of research in the world of data mining with numerous applications.

X.-L. Li et al. (Eds.): PAKDD 2015, LNCS 9441, pp. 16–28, 2015.
DOI: 10.1007/978-3-319-25660-3_2

Approaches for outlier detection can be classified into five major classes: distribution-based, distance-based, depth-based, density-based and cluster-based. Among all these classes of outlier detection, cluster-based is the latest and the most promising approach. The cluster-based approach considers outlier detection as by-product of clustering algorithm. This approach regards small clusters resulting from clustering as outliers and defines a measure for identifying the degree of each object being an outlier.

In this work, we address the second limitation of previous time series discord discovery algorithms by introducing a novel algorithm inspired by recent advances in the problem of outlier detection in the field of data mining. Anomalous patterns or discords are just the kind of outliers in time series. This work is the first of our attempt to apply outlier detection techniques in discord discovery for time series data. The main idea of our proposed method is to combine segmentation and cluster-based outlier detection. In this approach, first, pattern candidates are extracted from the time series using a segmentation method which is based on important extreme points, then these candidates are transformed into the same length and are input for an appropriate clustering algorithm, and finally, we identify discords by using a measure suggested in the cluster-based outlier detection method given by He et al. [9]. We experimented our proposed method for time series discord discovery on several real world datasets. The experimental results show that our approach is much more efficient than the HOTSAX algorithm in detecting time series discords while the anomalous patterns discovered by the two methods perfectly match with each other.

The rest of the paper is organized as follows. In Sect. 2 we give some essential definitions and explain briefly some basic ideas of cluster-based outlier detection algorithm. Section 3 introduces our proposed method for time series discord discovery. Section 4 reports the experiments on the proposed algorithm in comparison to HOTSAX. Section 5 gives some conclusions and remarks for future work.

2 Background

2.1 Time Series Discord

Intuitively a time series discord is a subsequence that is very different from its closest matching subsequence. However, in general, the best matches of a given subsequence (apart from itself) tend to be very close to the subsequence under consideration. For example, given a certain subsequence at position p, its closest match will be the subsequence at the position q where q is far from p just a few points. Such matches are called *trivial matches* and are not interesting [14].

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

Definition 2: (*Time Series Discord*) Given a time series T, the subsequence C of length n beginning at position p is said to be a discord of T if C has the largest distance to its nearest non-self match.

We may be interested in examining the top K discords, which is defined as:

Definition 3: (K-th Time Series Discord) Given a time series T, the subsequence D of length n beginning at position p is the K-th discord of T if D has the K-th largest distance to its nearest non-self match, with no overlapping region to the i-th discord beginning at position p_i, for all $1 \leq i \leq K$.

2.2 Related Works on Outlier Detection

Intuitively, outlier can be defined as given by Hawkins [10]:

> "*An outlier in a dataset is an observation that deviates so much from other observations as to arouse suspicion that it is generated from a different mechanism*".

Many data mining algorithms in literature find outliers as side-product of clustering algorithm, such as DBSCAN by Ester et al. [5], BIRCH by Zhang et al. [21], CURE by Guha et al. [6]. But these techniques define outliers as points which do not lie in clusters. Thus these techniques implicitly define outliers as the background noise in which the clusters are embedded.

Jiang et al. [11] regard *small* clusters as outliers, but a measure for identifying the degree of each object being outlier and how to distinguish small clusters from the rest are not presented in their work. He et al. [9], for the first time, not only regard small clusters as outliers but also define a measure for identifying the degree of each object being an outlier which is called CBLOF (Cluster-based local outlier factor) and present an efficient algorithm for mining outliers. Duan et al. [4] proposed another cluster-based outlier detection method which applies a new defined cluster-based outlier factor and use LDBSCAN algorithm (Duan et al. [3]) for clustering. One improvement in the method of Duan et al. 2009 is that it can evaluate the outlier factor for each cluster as a whole.

2.3 Cluster-Based Outlier Detection

The main ideas in the cluster-based outlier detection method proposed by He et al. [9] can be reviewed as follows.

It is reasonable to define the outliers from the point of view of clusters and identify those objects that do not lie in any large clusters as outliers. The algorithm for detecting outliers has two main parts: (1) clustering the dataset and (2) computing the value of *cluster-based local outlier factor* (CBLOF) for each object.

The clustering algorithm used is the Squeezer algorithm (He et al. [8]), which can produce good clustering results and at the same time preserves good scalability. The process of mining outliers is tightly coupled with the clustering algorithm.

The outlier factor CBLOF of each object which identifies the physical significance of an outlier is measured by both the size of the cluster the object belongs to and the distance between the object and its closest cluster (if the object lies in a small cluster) or the distance between the object and the cluster it belongs to (if the object lies in a large cluster).

3 From Cluster-Based Outlier Detection to Time Series Discord Discovery

This work is the first of our attempt to employ cluster-based outlier detection approach in time series discord discovery. The main idea of our proposed method is to combine segmentation and cluster-based outlier detection in the problem of time series discord discovery. Our proposed method for discovering discord is called FindCBD (Finding Cluster-based Discord). The FindCBD algorithm consists of the following steps:

Step 1. Divide the time series into segments (pattern candidates) using important extreme points method.
Step 2. Transform the pattern candidates to the same length by using homothetic transformation and discretize them using Symbolic Aggregate Approximation (SAX) discretization.
Step 3: Cluster the discretized pattern candidates using the Squeezer algorithm and calculate the anomaly scores of all the candidates using CBLOF factors.
Step 4: Identify the pattern candidate with the largest CBLOF score as the top discord of the time series. And the pattern with the K-th largest CBLOF score will be the K-th discord of the time series.

The details of all four steps are explained in the following subsections.

3.1 Time Series Segmentation Using Significant Extreme Points

In this work, we assume that the anomaly within a time series begins and ends at characteristic points such as maxima or minima. In Step 1, to extract a temporally ordered sequence of pattern candidates, significant extreme points of a time series have to be found. The definition of significant extreme points is given as follows.

Definition 4: (*Significant Extreme Points*) A univariate time series $T = t_1,...,t_N$ has a *significant minimum* at position m with $1 < m < N$, if a subsequence $(t_i, \ldots, t_j)$ with $1 \leq i < j \leq N$ in T exists, such that t_m is the minimum of all points of this subsequence and $t_i \geq R \times t_m$, $t_j \geq R \times t_m$ with user-defined $R \geq 1$.

Similarly, a *significant maximum* is existent at position m with $1 < m < N$, if a subsequence $(t_i, \ldots, t_j)$ with $1 \leq i < j \leq N$ in T exists, such that t_m is the maximum of all points of this subsequence and $t_i \leq t_m/R$, $t_j \leq t_m/R$ with user-defined $R \geq 1$.

Notice that in the above definition, the parameter R is called *compression rate* which is greater than one and an increase of R leads to selection of fewer significant extreme points. Figure 1 illustrates the definition of significant minima (a) and maxima (b).

Given a time series T, starting at the beginning of the time series, all significant minima and maxima of the time series are computed by using the procedure given in Pratt and Fink [19]. This procedure has linear time and constant memory. It can process new points as they arrive, without storing the original time series.

The details of the segmentation step for extracting pattern candidates are given as follows:

(i) We extract all significant extreme points of the time series T. The result of this step is a sequence of extreme points $EP = (ep_1,..., ep_l)$

(ii) We compute all the pattern candidates iteratively. A pattern candidate $PC_i(T)$, $i = 1,..., l - 2$ is the subsequence of T that is bounded by the two extreme points ep_i and ep_{i+2}. Pattern candidates are the subsequences that may have different lengths.

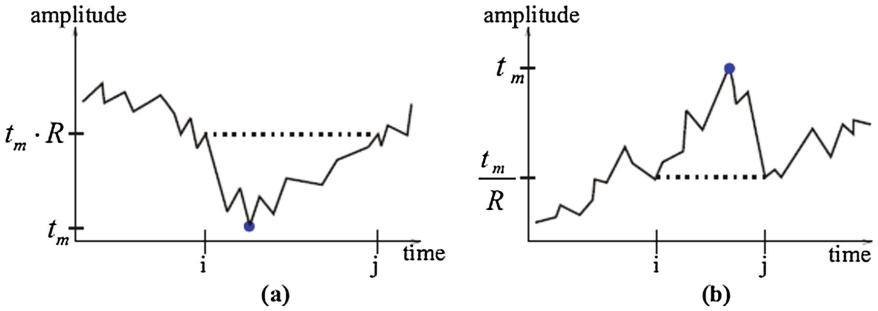

Fig. 1. Illustration of significant extreme points: (a) Minimum, (b) Maximum ([19])

3.2 Homothetic Transformation

To ensure the effectiveness of our proposed method, in Step 2 of our FindCBD algorithm, we have to apply some interpolation technique for transforming the pattern candidates of different lengths to those of the same length. Spline interpolation can be selected for this task as suggested in by Gruber et al. [7]. However, spline interpolation is not only complicated in computation, but also can modify undesirably the shapes of the motif candidates. In this work, we apply homothetic transformation, a simpler and more effective technique which also can transform the subsequences with different lengths to those of the same length. Due to the limit of space, we can not explain the use of homothetic transformation here, interested readers can refer to our previous work [20] for more details.

3.3 SAX Discretization

In our discord discovery method, we use Squeezer algorithm to cluster the pattern candidates (i.e. subsequences). But Squeezer algorithm operates on categorical dataset. Therefore, we need some discretization method to convert the subsequences to symbolic strings. Here we select to use Symbolic Aggregate Approximation (SAX) method ([16]) to transform the subsequences to symbolic strings. The SAX representation that is used in Step 2 of our FindCBD algorithm can be explained briefly as follows.

A time series $C = c_1 \ldots c_N$ of length N can be represented in a reduced w-dimensional-space as another time series $D = d_1 \ldots d_w$ by segmenting C into w equally-sized segments and

then replacing each segment by its mean value d_i. This dimensionality reduction technique is called Piecewise Aggregate Approximation (PAA) [12]. After this step, the time series D is transformed into a symbolic sequence $A = a_1 \ldots a_w$ in which each real value d_i is mapped to a symbol a_i through a table look-up. The lookup table contains the *breakpoints* that divide a Gaussian distribution in an arbitrary number (e.g. from 3 to 10) of equi-probable regions. This discretization is based on the assumption that the reduced time series have a Gaussian distribution.

Notice that since pattern candidates extracted in Step 1 are always short time series, after being transformed by PAA and SAX, the dimensionality of the pattern candidates becomes sufficiently small and Squeezer can cluster efficiently these objects with low dimensionality.

3.4 How to Compute CBLOF of Each Object

Definition 5: Let $A_1, \ldots, A_m$ be a set of categorical attributes with domains $D_1, \ldots, D_m$ respectively. Let the dataset D be the set of tuples where each tuple t: $t \in D_1 \times \ldots \times D_m$. The results of a clustering algorithm executed on D is denoted as: $C = \{C_1, C_2, \ldots, C_k\}$ where $C_i \cap C_j = \varnothing$ and $C_1 \cup C_2 \ldots \cup C_k = D$. The number of clusters is k.

Definition 6: (*Large and Small Cluster*) Suppose $C = \{C_1, C_2, \ldots, C_k\}$ is the set of clusters in the sequence that $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$. Given two numeric parameters α and β, we define b the boundary of large and small if one of the following formulas holds:

$$(|C_1| + |C_2| + \ldots + |C_b|) \geq |D| * \alpha \tag{1}$$

$$|C_b| / |C_{b+1}| \geq \beta \tag{2}$$

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

Definition 6 gives a measure to distinguish large and small clusters. Formula (1) considers the fact that most data points in the dataset are not outliers. Therefore, clusters that contain a large portion of data points should be taken as large clusters. For example, if α is set to 90 %, we intend to regard clusters contain 90 % of data points as large clusters. Formula (2) considers the fact that large and small clusters should have significant differences in size. For instance, if we set β to 5, the size of any cluster in LC is at least five times of the size of the cluster in SC.

Definition 7: (*Cluster-Based Local Outlier Factor*) Suppose $C = \{C_1, C_2, \ldots, C_k\}$ is the set of clusters in the sequence that $|C_1| \geq |C_2| \geq \ldots \geq |C_k|$ and the meanings of α, β, b, LC and SC are the same as they are formalized in Definition 4. For any records t, the cluster-based local outlier factor of t is defined as:

$$CBLOF(t) = |C_i| * \min(distance(t, C_j) \quad \text{if } t \in C_i, C_i \in SC \text{ and } C_j \in LC(j: 1 \rightarrow b)$$
$$= |C_i| * (distance(t, C_i)) \quad \text{if } t \in C_i, C_i \in LC$$

From Definition 7, the CBLOF of an object is determined by the size of its cluster, and the distance between the object and its closest large cluster (if this object lies in small cluster) or the distance between the object and the cluster it belongs to (if this object belongs to a large cluster).

3.5 The Squeezer Algorithm

Let $A_1, \ldots, A_m$ be a set of categorical attributes with domains $D_1, \ldots, D_m$. respectively. Let the dataset D be the set of tuples where each tuple t: $t \in D_1 \times \ldots \times D_m$. Let TID be the set of unique identifiers of every tuple. For each $tid \in TID$, the attribute value for A_i of corresponding tuple is represented as $tid.\ A_i$.

Definition 8: (*Cluster*) $Cluster \subseteq TID$ is a subset of TID.

Definition 9: Given a cluster C, the set of different attribute values on A_i with respect to C is defined as: $VAL_i(C) = \{tid\ .\ A_i \mid tid \in C\}$ where $1 \leq i \leq m$.

Definition 10: Given a cluster C, let $a_i \in D_i$, the support of a_i in C with respect to A_i is defined as: $Sup(a_i) = |\{tid \mid tid\ .\ A_i = a_i,\ tid \in C\}|$.

Definition 11: (*Similarity*) Given a cluster C and a tuple t with $tid \in TID$, the similarity between C and tid is defined as:

$$Sim(C, tid) = \sum_{i=1}^{m} \left(\frac{Sup(a_i)}{\sum_{a_j \in VAL_i(C)} Sup(a_j)} \right)$$

where $a_i = tid.\ A_i$

To compute CBLOF for object t, He et al. need a background clustering algorithm and the clustering algorithm used is the Squeezer algorithm (He et al. [8]). The Squeezer algorithm has some advantageous features. First, it achieves both high quality of clustering results and scalability. Second, it does not require the number of desired clusters as an input parameter. Third, it can handle high dimensional datasets effectively. However, Squeezer is for clustering categorical data.

From Definition 11, it is clear that the similarity used here is statistics based. That means if the similarity between a tuple and an existed cluster is large enough, it means that the probability of the tuple belongs to this cluster is larger.

The Squeezer algorithm has n tuples as input, similarity threshold s as a parameter and produces clusters as final results. Initially, the first tuple in the database is read in and the first cluster is constructed to contain the first tuple, i.e. $C = \{1\}$. Then, the consequence tuples are read iteratively. The outline of the Squeezer algorithm is described as follows.

```
Algorithm Squeezer (D, s)
begin
    while (D has unread tuple) {
      tuple = getCurrentTuple (D)
      if (tuple.tid = = 1) {
                addNewClusterStructure (tuple.tid)}
      else {
                for each existed cluster C
                          simComputation(C, tuple)
                get the max value of similarity: sim_max
                get the corresponding Cluster Index: index
                if sim_max >= s
                          addTupleToCluster (tuple, index)
                else
                          addNewClusterStructure (tuple.tid)}
                         }
}
    outputClusteringResult ()
end
```

Note: In the formula to compute the CBLOF of a tuple, the distance from a tuple to a given cluster can be computed using the similarity between that tuple to the cluster. That means

$$distance(t, C) = 1/(1 + Sim(C, t))$$

3.6 The Parameters of the FindCBD Algorithm

Due to the combination of segmentation, homothetic transform and cluster-based outlier detection, our FindCBD algorithm requires eight parameters which are defined as follows.

R: compression ratio for extracting important extreme points from a time series. If *R* is larger, fewer extreme points are selected, otherwise, more extreme points selected.

l_max: the maximum length of pattern candidates.

l_min: the minimum length of the pattern candidates. If a pattern candidate with length less than *l_min*, it will be ignored.

a: alphabet size, *w*: window size for SAX discretization.

s: similarity threshold. This threshold is used by the Squeezer algorithm to determine which cluster a pattern candidate belongs to.

Parameter α indicates the percentage of pattern candidates in the candidate set that belongs to large clusters. Parameter β determines the difference in size between a large cluster and a small cluster. The parameters α and β are used in computing the outlier factors of each pattern candidates.

Note that FindCBD does not require user to determine the discord length, but the user has to estimate the maximum and the minimum values for the discord length through the two parameters *l_max* and *l_min*.

4 Experimental Evaluation

We implemented the comparative methods with Microsoft Visual C# and conducted the experiments on an Intel(R) Core(TM) 2 Duo CPU T6400 @ 2.00 GHz, 2 GB RAM PC.

In this experiment, we compare the proposed algorithm to the HOT SAX algorithm. The HOT SAX is selected for comparison due to its popularity. It is the most cited algorithm for detecting time series discords up to date and was applied in many applications. The comparison is in terms of time efficiency and discord detection accuracy.

Table 1. Parameter settings of the two discord discovery algorithms on the five datasets

Dataset	FindCBD	HOT SAX
ECG	$R = 1.2$, $l\_max = 250$, $l\_min = 40$ $w = 20$, $a = 8$, $s = 12$, $\alpha = 0.9$, $\beta = 5$	$w = 20$, $a = 8$, $m = 100$
AEM	$R = 1.4$, $l\_max = 250$, $l\_min = 40$ $w = 20$, $a = 8$, $s = 12$, $\alpha = 0.9$, $\beta = 5$	$w = 20$, $a = 8$, $m = 200$
ERP	$R = 1.35$, $l\_max = 350$, $l\_min = 30$ $w = 20$, $a = 20$, $s = 12$, $\alpha = 0.9$, $\beta = 5$	$w = 20$, $a = 20$, $m = 1200$
STOCK	$R = 1.03$, $l\_max = 500$, $l\_min = 50$ $w = 20$, $a = 20$, $s = 15$, $\alpha = 0.9$, $\beta = 5$	$w = 20$, $a = 20$, $m = 600$
POWER	$R = 1.15$, $l\_max = 200$, $l\_min = 50$ $w = 20$, $a = 20$, $s = 12$, $\alpha = 0.9$, $\beta = 5$	$w = 20$, $a = 20$, $m = 800$

We conducted the experiment on five datasets: ECG (20000 data points), AEM (20000 data points), ERP (25000 data points), STOCK (20000 data points) and POWER (25000 data points). All the datasets are obtained from the UCR Time Series Data Mining Archive (Keogh et al. [15]).

The parameter settings for two algorithms: FindCBD and HOT SAX over each dataset are given in Table 1. For the HOT SAX algorithm, we have the three following parameters: SAX alphabet size a, the length of a SAX word w and the length of the discord m. For the FindCBD, we have eight parameters: the compression ratio R, the maximum length of pattern candidates, *l_max*, the minimum length of pattern candidates *l_min*, SAX alphabet size a, the length of a SAX word w, the similarity threshold s, the parameter α, and the parameter β (for distinguishing small clusters and large clusters).

Notice that with all five datasets, we choose the window size $w = 20$. Due to that after PAA and SAX transformation, the dimensionality of pattern candidates is 20. With this dimensionality, the Squeezer algorithm can work effectively. For all the

Table 2. Experimental results on the run times of the two discord discovery algorithms.

Dataset	Length	Method	Run time (s)	Speedup
ECG	20000	FindCBD	0.041	2390
		HOT SAX	98	
AEM	20000	FindCBD	0.016	16250
		HOT SAX	260	
ERP	25000	FindCBD	0.027	21629
		HOT SAX	584	
STOCK	20000	FindCBD	0.024	6250
		HOT SAX	150	
POWER	20000	FindCBD	0.052	3192
		HOT SAX	166	
Average				9942.2

experiments, the two parameters α, β needed when computing anomaly scores are set to 90 % and 5 separately.

The experimental results on the run times of the two algorithms on the five datasets are shown in Table 2.

From the experimental results, we can see that for all the datasets our FindCBD is remarkably faster than HOT SAX. In average, FindCBD is faster than HOT SAX about 10,000 times. Efficiency is achieved because of two reasons. First, sliding over the set of segments extracted from a time series in FindCBD is much faster than sliding the window over the time series one data point at a time in HOTSAX. Segment-based sliding window, which does not incur any information loss, has been supported by several previous works in time series anomaly detection as well as motif discovery

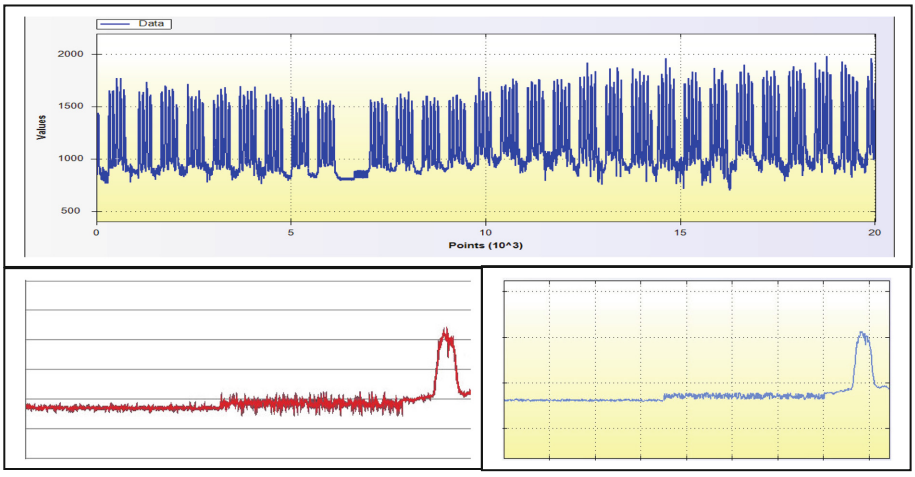

Fig. 2. POWER dataset (top). The top discord discovered in the dataset by HOT SAX (bottom-left). The top discord discovered in the dataset by FindCBD (bottom-right).

([2, 7, 20]). Second, both the segmentation basing on significant extreme points and the Squeezer algorithm for clustering can work very fast.

Now we turn our discussion to the accuracy of the proposed discord discovery algorithm. Following the tradition established in previous works, such as [1, 2, 13, 14], the accuracy of a given discord discovery algorithms is basically based on human analysis of the discords discovered by that algorithm. That means through human inspection we can check if the discords identified by a proposed algorithm on a given time series dataset are almost the same as those identified by the baseline discord discovery algorithm (here we select HOT SAX as the baseline algorithm). If the check result is positive in most of the test datasets, we can conclude that the proposed discord discovery algorithm brings out the same accuracy as the baseline algorithm.

Due to space limit, Fig. 2 shows only one example of the top discord discovered in the POWER dataset by the FindCBD and HOT SAX algorithm. From Fig. 2, we can see that the top discord discovered by our proposed algorithm is exactly the same as that discovered by HOT SAX algorithm. In fact, over all five datasets, the discords discovered by FindCBD are almost the same as those discovered by HOT SAX.

5 Conclusions

This work is the first of our attempt to apply outlier detection techniques in discord discovery for time series data. The main idea of our proposed method is to combine segmentation and cluster-based outlier detection in one framework. The segmentation is based on significant extreme points ([19]) and the cluster-based outlier detection is based on the method proposed by He et al. [8].

We experimented our proposed method for time series discord discovery on five real world datasets. The experimental results show that our approach is much faster than the HOTSAX algorithm in detecting time series discords while the anomalous patterns discovered by the two methods perfectly match with each other. One major conclusion we draw from this work is that it is promising to apply the techniques in the field of outlier detection to time series discord discovery. One limitation of our proposed method is that it requires several parameters for segmentation, homothetic transform, SAX discretization and clustering by Squeezer algorithm.

As future work, we intend to experiment our proposed method with more real world datasets and modify the Squeezer algorithm to adapt it to continuous data so that we do not have to use SAX discretization.

Acknowledgement. We are grateful to Prof. Eamonn Keogh for his kindly providing all the test datasets used in this work.

References

1. Bu, Y., Leung, T.W., Fu, A., 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 (SDM 2007), Minneapolis, MN, USA (26–28 April 2007)

2. Chuah, M.C., Fu, F.: ECG anomaly detection via time series analysis. In: Thulasiraman, P., He, X., Xu, T.L., Denko, M.K., Thulasiram, R.K., Yang, L.T. (eds.) ISPA Workshops 2007. LNCS, vol. 4743, pp. 123–135. Springer, Heidelberg (2007)
3. Duan, L.D., Xu, L., Guo, F., Lee, J., Yan, B.: A local density based spatial clustering with noise. Inf. Syst. **32**(7), 978–986 (2007)
4. Duan, L.D., Xu, L., Liu, Y., Lee, J.: Cluster-based outlier detection. Ann. Oper. Res. **168**, 151–168 (2009)
5. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noises. In: Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining, pp. 226–231. AAAI Press, Portland (1996)
6. Guha, S., Rastogi, R., Shim, K.: CURE: an efficient clustering algorithm for large databases. In: Tiwary, A., Franklin, M. (eds.) Proceedings of 1998 ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, pp. 73–84 (01–04 June 1998)
7. Gruber, C., Coduro, M., Sick, B.: Signature verification with dynamic RBF network and time series motifs. In: Proceedings of 10th International Workshop on Frontiers in Handwriting Recognition (2006)
8. He, Z., Xu, X., Deng, S.: Squeezer: an efficient algorithm for clustering categorical data. J. Comput. Sci. Technol. **17**(5), 611–624 (2002)
9. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recogn. Lett. **24** (9–10), 1641–1650 (2003)
10. Hawkins, D.: Identification of Outliers. Chapman and Hall, London (1980)
11. Jiang, M.F., Tseng, S.S., Su, C.M.: Two phase clustering process for outlier detection. Pattern Recogn. Lett. **22**(6–7), 691–700 (2001)
12. Keogh, E., Chakrabarti, K., Pazzani, M., Mehrotra, S.: Dimensionality deduction for fast similarity search in large time series database. J. Knowl. Inf. Syst. **3**(3), 263–286 (2001)
13. Keogh, E., Lonardi, S., Chiu, B.: Finding surprising patterns in a time series database in linear time and space. In: KDD 2002: Proceedings of 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, pp. 550–556 (2002)
14. Keogh, E., Lin, J. and Fu, A.: HOT SAX: efficiently finding the most unusual time series subsequence. In: Proceedings of 5th IEEE International Conference on Data Mining (ICDM), pp. 226–233 (2005)
15. Keogh E., Xi X., Wei L., Ratanamahatana C.A.: The UCR time series classification/clustering homepage (2013). www.cs.ucr.edu/∼eamonn/time_series_data
16. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discover (DMKD 2003), pp. 2–11 (13 June 2003)
17. Ma, J., Perkins, S.: Online novelty detection on temporal sequences. In: Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, NY, USA, pp. 614–618. ACM Press (2003)
18. Oliveira, A.L.I., Neto, F.B.L., Meira, S.R.L.: A method based on RBF-DAA neural network for improving novelty detection in time series. In: Proceedings of 17th International FLAIRS Conference, AAAI Press, Miami Beach, Florida, USA (2004)
19. Pratt, K.B., Fink, E.: Search for pattern in compressed time series. Int. J. Image Graph. **2**(1), 89–106 (2002)

20. Truong, C.D., Anh, D.T.: An efficient method for discovering motifs in large time series. In: Selamat, A., Nguyen, N.T., Haron, H. (eds.) ACIIDS 2013, Part I. LNCS, vol. 7802, pp. 135–145. Springer, Heidelberg (2013)
21. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, pp. 103–114 (04–06 June 1996)

http://www.springer.com/978-3-319-25659-7

Trends and Applications in Knowledge Discovery and Data Mining
PAKDD 2015 Workshops: BigPMA, VLSP, QIMIE, DAEBH, Ho Chi Minh City, Vietnam, May 19-21, 2015. Revised Selected Papers
Li, X.; Cao, T.; Lim, E.-P.; Zhou, Z.-H.; Ho, T.-B.; Cheung, D. (Eds.)
2015, XV, 289 p. 91 illus., Softcover
ISBN: 978-3-319-25659-7