Discord Detection in Streaming Time Series with the Support of R-Tree

Pham Minh Chau, Bui Minh Duc, Duong Tuan Anh
Faculty of Computer Science and Engineering
Ho Chi Minh City University of Technology
Ho Chi Minh City, Vietnam
{51300366, 51300904}@hcmut.edu.vn, dtanh@cse.hcmut.edu.vn

***Abstract*—Time series discord detection is an important problem in various applications. In this work, we propose a version of Heuristic Discord Discovery (HDD) algorithm, called HDD-MBR, for efficient discord detection in static time series. HDD-MBR employs R-tree as a supporting data structure to arrange two ordering heuristics in HDD. Furthermore, we introduce HDD-MBR-Stream, the application of HDD-MBR in the framework given by Liu et al. for detecting local discords in streaming time series. The experimental results reveal that HDD-MBR can detect the same quality discords as those detected by HOT SAX but with much shorter run time. Moreover, HDD-MBR-Stream demonstrates a fast response in handling time series streams with quality local discords detected.**

Keywords—streaming time series, discord discovery, R-Tree

I. INTRODUCTION

The problem of detecting unusual (abnormal, novel, deviant, anomalous, *discord*) subsequences in a time series has recently attracted much attention. Time series anomaly detection brings out the abnormal patterns present in a time series. Application areas that explore such time series anomalies are, for example, fault diagnosis, intrusion detection, fraud detection, auditing and data cleansing.

There has been an extensive study on time series anomaly detection in the literature. Various algorithms for time series discord discovery have been proposed, such as brute-force and HOT SAX by Keogh et al. [9] and WAT by Bu et al. [1]; a method based on neural-network by Oliveira et al. [19]; a method based on one-class support vector machine by Ma and Perkins [18]; a method based on segmentation and Finite State Automata by Salvador and Chan [21]; a method based on time series segmentation and anomaly scores by Leng et al. [14]; an extension of HOT SAX based on iSAX symbolic representation by Buu and Anh [2]; a method based on Piecewise Aggregate Approximation (PAA) bit representation and clustering by Li et al. [15]; and a method based on segmentation and cluster-based outlier detection by Kha and Anh [13]. These above-mentioned algorithms for anomaly detection are classified into three categories: window-based methods, segmentation-based methods and classification-based methods [3]. In the window-based category, HOT SAX has been considered as the most popular algorithm. HOT SAX is an unsupervised method of anomaly detection and has been applied in several real applications. However this algorithm still has some weaknesses as follows. Users of HOT SAX are required to choose suitable values for the parameters such as the discord length, word-length and/or alphabet-size, which are not intuitive. Besides, HOT SAX still suffers from a high computational cost.

Nowadays, discord detection in streaming time series has emerged as an attractive problem. The task of finding the most important discord in streaming time series arises in diverse applications. Examples of these applications are online stock analysis, sensor networks, health-care monitoring, earthquake monitoring. However, discord detection in streaming time series is a non-trivial task. A streaming time series is possible to be read only once and the length of a streaming time series can be very large. Therefore, it is challenging to adapt a discord detection algorithm in static time series to a discord detection algorithm for streaming time series.

As for streaming time series, there have been a few research works of anomaly detection which will be listed as follows. Zhu and Shasha, in 2003 [23] introduced an online algorithm to find abnormal aggregates or busts in data streams. This online algorithm could monitor streaming time series on elastic windows. However, the definition of bursts introduced by Zhu and Shasha is different from that of time series discords given by Keogh et al. [9] which is commonly-used. Liu et al. in 2009 presented a framework for discord detection in streaming time series called Detection of Continuous Discords (DCD) [17]. DCD can find continuous discords from local segments of a time series stream. One important technique in DCD is that it limits the search space to further enhance the discord detection efficiency. Toshniwal and Yadav [22] introduced an extension of HOT SAX to find outliers in local segments of a streaming time series. In this work, the criterion to determine time series outliers is the type of deviation from normal behavior, i.e. above or below normal. When there is a newly incoming subsequence, the data distribution of these distances is reevaluated. Due to this reason, this algorithm has high computational complexity. Sanchez and Bustos in 2014 [20] proposed a method for discord detection in streaming time series which employs bounding boxes and R-tree to arrange two ordering heuristics for two loops in discord detection process.

Motivated by the framework given by Liu et al. [17] and the

978-1-5386-9186-1/18/$31.00 ©2018 IEEE
DOI 10.1109/ACOMP.2018.00023

idea of using bounding boxes from the work by Sanchez and Bustos, in this work we aim to develop an efficient method for discord detection in streaming time series. This work makes two contributions. First, we propose a version of HDD, called HDD-MBR, for efficient discord discovery in static time series. In this algorithm, we use a supporting data structure, R-tree, to arrange two ordering heuristics in the discord discovery process. Second, we introduce HDD-MBR-Stream, the application of HDD-MBR in the framework given by Liu et al. [17] for detecting local discords in streaming time series. The experimental results reveal that HDD-MBR can detect the same quality discords as those detected by HOT SAX but with much shorter run time. Furthermore, HDD-MBR-Stream demonstrates a fast response in handling time series streams with quality local discords detected.

The rest of our paper is organized as follows. Some background related to finding discords in streaming time series is provided in Section 2. In section 3, we introduce the HDD-MBR algorithm. In section 4, we introduce our algorithm for discord detection in streaming time series. The extensive experiments of our two proposed algorithms are reported in Section 5. Finally, Section 6 presents some conclusions and remarks for future works.

II. BACKGROUND AND RELATED WORKS

A. Definitions

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. When finding discords, we should exclude trivial matches and keep only *non-self* matches defined as follows [9].

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 $|p - q| \geq n$, i.e. C does not overlap M.

Definition 2. (*Time series discord*): Given a time series T, the subsequence D in T is called the most significant discord in T if the distance to its nearest non-self match is largest.

A streaming time series T is a semi-infinite time series sequence of real numbers $t_1, t_2, \ldots, t_n, \ldots$ where t_n is the most recent data point. Given a streaming time series T, the problem is finding the most significant discord of length l as soon as a newly incoming subsequence C of length l exists. This implies that the discord detection might be repeatedly executed once for every newly incoming data point of T. In storing a streaming time series, to avoid memory overflow, we use large-size *buffer* B to contain the local segment of the streaming time series under consideration. In general, at time point t, suppose B contains n last values of the time series, $B = t_1, t_2, \ldots, t_n$ and t_{new} is the new coming data point. At time point $t + 1$, the time series in the buffer becomes $B = t_2, \ldots, t_n, t_{new}$. That means the data point t_1 is removed out of the buffer at time point $t + 1$and it is considered as the *obsolete* data point. Notice that the length of the buffer is specified in advance.

B. HDD Algorithm

The discord detection problem can be easily solved by a brute force search using a nested loop. The outer loop takes each subsequence as a possible candidate, and the inner loop is used to find the candidate's nearest non-self match. The candidate that has the greatest such value is the discord. The complexity of the brute-force algorithm is $O(N^2)$ where N is the number of subsequences.

To improve the brute-force algorithm, Keogh et al., 2005 [9] proposed a generic algorithm, called Heuristic Discord Discovery (HDD), for efficient discord detection. This algorithm requires two heuristics that generate two ordered list of subsequences: one for the outer loop and the other one for the inner loop. The heuristic *Outer* is useful for quickly finding the best candidate, and the heuristic *Inner* is useful for quickly finding the best nearest non-self match. We break out of the inner loop if the distance is less than the best-so-far discord distance.

C. A Lower Bounding Technique

Keogh et al. proposed a lower bounding function, called *LB_Keogh*, in 2004 [12] which is described as follows.

Given Q a sequence of length n, we can use a parameter r to define two new sequences U and L:

$$U_i = \max(Q_{i-r}: Q_{i+r})$$

$$L_i = \min(Q_{i-r}:Q_{i+r})$$

U and L stand for *Upper* and *Lower*, respectively; and the two sequences form a bounding envelope that encloses Q from above and below. An obvious property of U and L is the following: $\forall i, L_i \leq Q_i \leq U_i$

Having defined U and L, Keogh et al. use them to define a lower bounding measure between a sequence C with a query sequence Q.

$$\text{LB\_Keogh}(Q,C) = \sqrt{\sum_1^n \begin{cases} (C_i - U_i)^2 \; if \; C_i > U_i \\ (C_i - L_i)^2 \; if \; C_i < L_i \\ 0 \, , \; else \end{cases}} \qquad (1)$$

To reduce the dimensionality, a sequence C of length n can be represented by Piecewise Aggregate Approximating (PAA) transform [8] in a lower w dimension to a vector $\bar{C} = \overline{c_1}, \overline{c_2}, \ldots, \overline{c_w}$ $(w < n)$. The value $\overline{c_i}$ can be computed by the formula:

$$\overline{c_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_i \qquad (2)$$

Using PAA representation of a sequence, the two PAA transformed sequences of U and L, denoted as $\hat{U}$ and $\hat{L}$, can be computed by the following.

$$\hat{U}_i = \max(U_{\frac{n}{w}(i-1)+1}, \ldots, U_{\frac{n}{w}(i)})$$

$$\hat{L}_i = \min(L_{\frac{n}{w}(i-1)+1}, \ldots, L_{\frac{n}{w}(i)})$$

The two sequences $\hat{U}$ *and* $\hat{L}$ can work as the upper boundary and the lower boundary which form a bounding rectangle enclosing the sequence Q with the same roles as the curves U and L (see Fig. 1).

Given a candidate sequence C, transformed to $\bar{C}$ by PAA. Suppose our index structure contains a leaf node and let $R = (L, H)$ be the minimum bounding rectangle (MBR) associated with the leaf node, where $L = \{l_1, l_2, \ldots, l_N\}$ and $H = \{h_1, h_2, \ldots, h_N\}$ are the lower and higher endpoints of the major diagonal of R. Given the above, the MINDIST function that returns the lower bounding measure of the distance between a sequence C and a minimum bounding rectangle R is defined as.

$$\text{MINDIST}(\bar{C}, R) = \sqrt{\sum_1^w \frac{N}{w} \begin{cases} (\bar{C}_i - h_i)^2 \; if \; \bar{C}_i > h_i \\ (\bar{C}_i - l_i)^2 \; if \; \bar{C}_i < l_i \\ 0, \; else \end{cases}} \qquad (3)$$

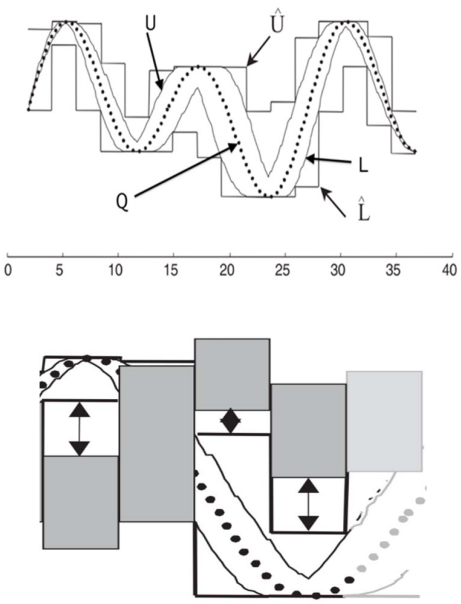

Fig. 1. (Top) A subsection of a sequence with its associated functions. (Bottom) An illustration of the MINDIST function. ([12]).

D. Using R-Tree in Discord Detection in Streaming Time Series

The discord detection algorithm that uses bounding boxes (i.e. minimum bounding rectangles) was proposed by Sanchez and Bustos [20] in 2014.

This algorithm uses an R-tree to search on the subsequences extracted from a time series. For each subsequence $C_p = \{c_1, \ldots, c_n\}$ from time series T, we generate a minimum bounding rectangle $(\hat{U}, \hat{L})$ of C_p, $\hat{U}$ and $\hat{L}$ are the PAA representations of the bounding curve U and L of C_p, then this rectangle will be inserted to a leaf node of the R-tree. After constructing the R-tree, the start position of each subsequence is stored into an integer array which is associated with a particular entry in a leaf node.

There are three important operations in R-Tree: searching, inserting and deleting. In inserting a subsequence into a leaf node, the subsequence is stored as an entry in a leaf node. If this node is overflow, it will be split and the splits may propagate up the tree. According to Guttman [6], there are three techniques for splitting: exhaustive, quadratic-cost and linear-cost.

Given a leaf node, if a subsequence in it is deleted and the node might have too few entries, we have to eliminate the node and relocate its entries. Node elimination may propagate upward as necessary.

Based on the R-tree, the outer order and the inner order can be established as follows.

- *Outer Loop Heuristic*: First, the algorithm visits all the subsequences in the MBR which contains the minimum number of subsequences. Then, the algorithm visits the rest of the subsequence in random number. The heuristic ensures that the subsequences that are most isolated will be visited at the beginning of the search as potential candidates.

- *Inner Loop Heuristic*: First, the algorithm visits all subsequences bounded in R_j, such that MINDIST $(\bar{C}_q, R_j)$ < MINDIST $(\bar{C}_q, R_{i,\forall i \neq j})$. Then, the algorithm visits the rest of the subsequence in random number. This heuristic enables us to first visit all the subsequences that are most similar to C_p, increasing the probability of early termination of the loop. The MINDIST function is calculated by Eq. 3.

E. A Framework for Discord Detection in Streaming Time Series

Liu et al. in 2009 proposed a framework, called DCD (Detection of Continuous Discords), for discord detection in streaming time series [17]. One important technique in DCD is that it limits the search space to further enhance the detection efficiency. The pseudo-code of the framework DCD is described as in Fig. 2.

Procedure DCD
Input: A given time series stream and the length of discord *n*
Output: The local discords (discord in the buffer)
1. Initialize ();
2. Read the time series data points into *B*; // *B* is the buffer
3. [*prev_loc*, *dist*] = FindDiscord(*B, n*);
4. Output the local discord LD ≡ [*prev_loc*, *t*]; // *t* is the current time;
5. Add *dist* into array *V*;
6. **while** (the time series stream is not stopped) **do**
7. Read the next data point into *B*;
8. [*loc*, *dist*] = FindDiscord(*B*, *n*);
9. **if** *loc* != *prev_loc* - 1 && *valid(dist,V)* **then**
10. Output the local discord LD ≡ [*loc, t*]; // *t* is the current time;
11. **end** // end of if
12. *prev_loc* = *loc*;
13. Add *dist* into array *V*;

14. **end** // end of While

Fig. 2. The framework DCD for discord detection in streaming time series

In the framework DCD, we use two data structures: the buffer B and array V. In explaining the use of array V we need the definition of *nearest non-self neighbor distance* of a subsequence as follows.

Definition 3 (*Nearest non-self neighbor distance*): Given a time series T, for any subsequence P, Q is the nearest nonself match of P, the distance from P to Q is the *nearest non-self neighbor distance* of P.

The new incoming data points will arrive to the buffer continuously. With the assumption that we start to find the discord when the buffer is full. At that moment the *FindDiscord* procedure is invoked to detect the discord in the time series segment in the buffer B. The nearest non-self neighbor distance of the current discord is stored in the array V. Next, we read the new incoming data point from the stream, we put it in the right end of the buffer B and remove the left most data point in the buffer out of the buffer. At this moment, we invoke again the *FindDiscord* procedure to find the new discord. Whenever the discord is found, we output the discord if its location is different from that of the previous discord. The loop continues until the stream terminates.

To find the interesting discords from the given time series stream, we need to use the *valid* function which is defined as follows.

$$valid(dist, V) \text{ is true if } dist > mean(V)*threshold$$

where $mean(V)$ is the mean value of all elements in the array V, *threshold* and the size of the array V are specified in advance by user.

A simple solution for *FindDiscord* procedure is to use the existing window-based algorithm such as HOT SAX to find the discord from the current buffer. In that case, we called the online discord detection algorithm as Brute-force HOT SAX (BFHS) algorithm. BFHS is not efficient since it has to search the whole buffer in each time of discord detection.

Liu et al. proposed a new *FindDiscord* procedure that can limit the search space to further enhance the discord detection efficiency. Before describing the *FindDiscord* procedure, we need one more related definition.

Definition 4 (*Small match*): Given a time series T, for any subsequence P of T, Q is the non-self match of P, if $Dist(P, Q) < dist$, where *dist* is the nearest non-self neighbor distance of the current local discord of the time series, then Q is a small match of P and P is a small match of Q.

The pseudo-code of Procedure *FindDiscord* is described as in Fig. 3.

Let *loc* be the position of local discord at time t, *dist* be the nearest non-self neighbor distance of local discord at time t and *currDist* be the distance between the local discord at the time point t and the new arriving subsequence. Procedure *FindDiscord* considers two possible cases:

- Case *a*: If *currDist* < *dist* or *loc*=1, we have to search all possible subsequences in the buffer to find the new local discord at time point $t+1$. For this case, the search space is the *Candidates* set which consists of all the subsequences in the buffer from location 1 to location $|B|-n+1$.
- Case *b*: Otherwise, there may be some subsequences whose nearest non-self neighbor distances become larger than that of the local discord at time point t. For this case, Liu et al. suggested that the search space can be reduced to the *Candidates* set that consists of: (i) the small match of the first subsequence in the buffer, (ii) the local discord at the time point t, and (iii) the new arriving subsequence.

Procedure FindDiscord

Input: B: the time series buffer at time $t+1$; n: the length of discord;

Output: The position and non-similar distance of local discord at time $t+1$.

1. read the next data point t_{new};
2. *currDist* = Dist(t_{loc},…,$t_{loc+n-1}$, t_{m-n+2},…, t_m, t_{new}); /* *loc*: the position of local discord at time t;
   /* *dist*: the nearest non-self neighbor distance of local discord at time t.
3. **if** *currDist* < *dist* // The case (a1)
4. *Candidate* = 1:|B|-n+1;
5. **else if** *loc*=1 // The case (a2)
6. *Candidates* = 1:|B|-n+1;
7. **else** // The case (b)
8. *Candidates* = {The small match of subsequence (1, n)(t)}∪ {The local discord at time t}∪ {The subsequence (m-n+1, n)(t+1) };
9. [*loc, dist*] = Search(*Candidates, n, B*);

Fig. 3. Procedure FindDiscord

Liu et al. [17] suggested that the *Search* function in *FindDiscord* procedure can be some discord detection function, such as HOT SAX algorithm. When HOT SAX is used as the *Search* function in *FindDiscord* procedure, we call the resultant online discord detection algorithm HOTSAX-Stream.

III. PROPOSED METHOD FOR DISCORD DETECTION: HDD-MBR

In this section, we propose an algorithm with the support of R-tree, called HDD-MBR, for discord discovery in static time series. The algorithm HDD-MBR consists of the five following steps.

Step 1: Extracting subsequences of length n by sliding a window of length n cross the time series.

Step 2: From each extracted subsequence, applying PAA to reduce it to a w-dimensional subsequence.

Step 3: Inserting all the reduced subsequences in the R-tree.

Step 4: Creating two ordering heuristics based on the R-tree as described in Section II.D

Step 5: Applying discord detection process as in HDD algorithm with the two nested loops using the two created ordering heuristics.

The pseudo-code of *HDD-MBR* is described in Fig. 4.

Algorithm HDD-MBR

Input: Time series T and discord length n

Output: Discord starting position *best_so_far_loc*, the largest distance from discord to nearest neighbor *best_so_far_dist*.

1. *best_so_far_dist* = 0
2. *best_so_far_loc* = NaN
3. **for** $i = 1$ **to** $|T| - n+1$
4. extract subsequence $t_i \ldots t_{i+n-1}$, use PAA to get a reduced subsequence of $t_i \ldots t_{i+n-1}$ then insert it to the R-tree
5. **endfor**
6. CreateOuterOrder;
7. CreateInnerOrder;
8. **for each** p **in** T ordered by heuristic *Outer* *// Begin Outer Loop*
9. *nearest_neigh_dist = infinity*
10. **for each** q **in** T ordered by heuristic *Inner* *// Begin Inner Loop*
11. **if** $|p - q| \geq n$ *// non-seft match?*
12. **if** $Dist(t_p \ldots t_{p+n-1}, t_q \ldots t_{q+n-1}) < best\_so\_far\_dist$
13. **break** *// Break out of Inner Loop*
14. **end**
15. **if** $MINDIST(t_p \ldots t_{p+n-1}, R[q]) < nearest\_neigh\_dist$
16. **if** $Dist(t_p \ldots t_{p+n-1}, t_q \ldots t_{q+n-1}) < nearest\_neigh\_dist$
17. $nearest\_neigh\_dist = Dist(t_p \ldots t_{p+n-1}, t_q \ldots t_{q+n-1})$
18. **end**
19. **end**
20. **end** *// End non-seft match test*
20. **endfor** *// End Inner Loop*
21. **if** $nearest\_neigh\_dist > best\_so\_far\_dist$
22. $best\_so\_far\_dist = nearest\_neigh\_dist$
23. $best\_so\_far\_loc = p$
24. **end**
25. **endfor** *// End Outer Loop*
26. **return** [*best_so_far_dist, best_so_far_loc*]

Fig. 4. HDD-MBR for discord discovery in time series

Our HDD-MBR has the same spirit as the algorithm given by Sanchez and Bustos for discord detection in static time series. However, there is one major improvement in our proposed algorithm, described as follows.

The MINDIST function is employed by Sanchez and Bustos [20] for arranging the two ordering heuristics: one for outer loop and one for inner loop. In HDD-MBR, besides using MINDIST function to create the two ordering heuristics, we also employ the MINDIST function as a lower bounding technique to prune off the entries in the R-tree during the search in the inner loop. In line 15 of HDD-MBR, the MINDIST function is used to prune off the subsequences which cannot be the nearest non-self match of the subsequence under consideration (i.e. $MINDIST(t_p \ldots t_{p+n-1}, R[q]) \geq nearest\_neigh\_dist$).

As for splitting operation in R-tree, in HDD-MBR we employ *linear cost splitting* technique [3] when inserting a new subsequence into a leaf node of the R-tree and this node is full. According to Guttman [6], linear splitting technique generates the fewest MBRs, incurs the least memory space and performs very fast.

Besides, to estimate the suitable length of PAA-frame for each dataset (i.e. the PAA transformation in Step 1), we apply PLA segmentation with bottom up algorithm [7] to segment the time series into several linear segments. The average of the lengths of all these segments will be used to estimate the length of the PAA-frame.

IV. HDD-MBR-STREAM FOR DISCORD DETECTION IN STREAMING TIME SERIES

Our proposed algorithm for online discord detection, called *HDD-MBR-Stream*, applies the framework DCD given by Liu et al. [17]. HDD-MBR-Stream still uses the *Candidates* set proposed in DCD to limit the search space in finding discord subsequence. The important point in our proposed algorithm is that while in DCD Liu et al. suggest to use the original HOT SAX in the role of the *Search* function in the *FindDiscord* procedure, our proposed algorithm uses HDD-MBR as the *Search* function in the *FindDiscord* procedure. Due to that, we name our proposed algorithm as *HDD-MBR-Stream*.

HDD-MBR-Stream has to update the supporting data structure, i.e. the R-tree, during the discord detection process in a streaming time series. At every new incoming data point, the subsequence which contains the new data point, after PAA reduction, will be assigned to the suitable leaf node in the R-tree (insert operation) and the subsequence which contains the out-of-date data point will be removed from its current leaf node (delete operation). Due to the dynamic property of the R-tree, this update work can be achieved efficiently.

Notice that the objective of discord detection in streaming time series in HDD-MBR-Stream is different from that of the algorithm proposed by Sanchez and Bustos. In HDD-MBR-Stream, the algorithm has to find and report the discord found in the buffer whenever a new data point arrives. But in the algorithm proposed by Sanchez and Bustos, it has to determine a detection starting point in the stream such that it can apply discord detection up to that point to obtain *a threshold distance*. And after that, the algorithm alerts whenever a new data point generates an anomalous subsequence whose distance to its nearest non-self match exceeds the threshold distance. In other words, the method by Sanchez and Bustos focuses on determine whether the new subsequence is a discord or not.

Due to this reason, we will not compare empirically our HDD-MBR-Stream to the algorithm proposed by Sanchez and Bustos.

A. How to determine the buffer size

The size of the buffer can affect on the efficiency of the discord detection in streaming time series. If the buffer size is small, the detected discord results will vary so frequently. Otherwise, if the buffer size is too large, the computational cost will increase remarkably. In this work, we estimate the buffer

size based on the *period* of the time series under consideration. The buffer size should be a multiple of the period of the time series in order that for cyclic time series, the new incoming data point and the obsolete data point will have some similarity.

There exist a few methods for periodicity detection in time series. In this work, we employ Autocorrelation Function to estimate the period of the time series [4]. The main idea of this method is that if a time series has a period, a significant autocorrelation coefficient will occur at the time lag equal to the period or multiples of the period. Therefore, in this work we calculate the autocorrelation coefficients between two subsequences $C(1, n - k)$ and $C(1+ k, n - k)$ in the same time series T at the same time lag k varying from 1 to $n/2$. The estimated period is the value of the lag k which brings out the largest autocorrelation coefficient.

The autocorrelation coefficient between two subsequences $C(1, n - k)$ and $C(1+ k, n - k)$ can be calculated by the formula:

$$\mathrm{AC} = \frac{Covariance(C(1,n-k),C(1+k,n-k))}{Std(C(1,n-k)).Std(C(1+k,n-k))}$$

$$= \frac{\frac{1}{n-k}\sum_{1}^{n-k}(t_i - \overline{t_1})(t_{i+k} - \overline{t_{1+k}})}{Std(C(1,n-k)).Std(C(1+k,n-k))}$$

where $\overline{t_1}$, $\overline{t_{1+k}}$ are the two mean values of the subsequences $C(1, n - k)$ and $C(1+ k, n - k)$, respectively and $Std(C(1, n - k))$, $Std(C(1+ k, n - k))$ are the standard deviations of the subsequences $C(1, n - k)$ and $C(1+ k, n - k)$, respectively.

V. Experimental Evaluation

In this section, we describe the results of the two main experiments: Experiment 1 for discord detection in static time series (called *offline discord detection*) and Experiment 2 for discord detection in streaming time series (called *online discord detection*).

A. Offline Discord Detection

For this experiment, we implemented two algorithms for discord detection in static time series: HOT SAX and HDD-MBR. The experiment aims to compare HDD-MBR with the original HOT SAX algorithm in terms of time efficiency and discord detection accuracy.

This experiment was conducted on the datasets from the UCR Time Series Data Mining archive ([10], [11]). There are 7 datasets used in this experiment. The datasets are from different areas such as medicine, industry and science. The names and lengths of the seven datasets are shown in TABBLE I. For each dataset in TABLE I, we also give the discord length *n_length* in the fourth column.

So the parameters for the two comparative algorithms are selected as follows:

- HOT SAX: $w = 3$, $a = 3$.
- HDD-MBR: $w = n\_length/4$, $r = 2$, *MaxEntryPerNode* = 25 and *MinEntryPerNode* = 12.

where w is the SAX word length (the reduced dimension of PAA), a is the size of the alphabet used in SAX transform and MaxEntryPerNode and MinEntryPerNode are the maximum number and the minimum number of entries in a node of R-tree, respectively.

TABLE I. Length and discord length for each dataset

Dataset Description	Dataset Name	Length of time series	Discord length
Space Shuttle Marotta Valve	(*TEK16*, *TEK17*)	4993, 5000	128
Electrocardiograms	(*ECG*)	21600	40
Power Data	(*Power_data*)	35040	200
Patient's respiration data	(*nprs43*, *nprs44*)	18020, 24125	160
Earth Rotation Parameters	*ERP*	198400	64

1) Effectiveness

Following the tradition established in previous works, such as [1], [9], [15], [20], the accuracy of a given discord discovery algorithm is basically based on human inspection of the discords detected by that algorithm. That means by eye, 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 which is the HOT SAX algorithm in this paper. If the checked 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. Notice that in most of the seven datasets in Experiment 1, the discords have been annotated by experts; therefore, we can spot the discords by eye with not much effort.

We compare HDD-MBR and HOT SAX in discord detection accuracy over the 7 datasets. For each dataset, we found out that the discord detected by HDD-MBR is exactly the same as the discord detected by HOT SAX.

2) Efficiency

Table II shows the runtimes (in seconds) of the two algorithms: HOT SAX, and HDD-MBR in discord detection over the seven datasets. From the experimental results in Table II, we can see that HDD-MBR performs much faster than HOT SAX in all the datasets. On average, HDD-MBR runs about 20.943 times faster than HOT SAX.

TABLE II Execution times of HOT SAX and HDD-MBR

Dataset	HOT SAX	HDD-MBR
TEK16	2.116	0.703
TEK17	2.554	0.521
ECG	14.979	0.612
Power_data	93.189	8.691
nprs43	16.599	2.122
nprs44	25.955	2.889
ERP	2056.595	23.724

We also compare the time efficiency of HDD-MBR to that of the algorithm proposed by Sanchez and Bustos called Bounding-Boxes. Table III shows the runtimes (in seconds) of the two algorithms: HDD-MBR and Bounding-Boxes in discord detection over the seven datasets. On average, HDD-MBR runs about 1.788 times faster than Bounding-Boxes.

TABLE III. EXECUTION TIMES OF HDD_MBR AND Bounding-Boxes

Dataset	HDD_MBR	Bounding-Boxes (Sanchez and Bustos)
TEK16	1.687	3.893
TEK17	1.446	3.459
ECG	2.278	4.137
power_data	45.604	72.057
nprs43	16.608	19.193
nprs44	17.530	22.732
ERP	73.409	144.690

B. Online Discord Detection

For this experiment, we implemented three algorithms for discord detection in streaming time series: HOTSAX-Stream, HDD-MBR-Stream and Brute Force HOT SAX (BFHS). BFHS is a concrete version of the framework DCD (Fig. 1) in which HOT SAX plays the role of the *FindDiscord* procedure for detecting discord in the time series segment currently in the buffer. It means that BFHS has to search the whole buffer in each time of discord detection.

We do not compare empirically HDD-MBR-Stream to the method proposed by Sanchez and Bustos [20] for discord detection in streaming time series due to the different purposes of the two methods.

We implemented all the algorithms with Visual C# 2013 (Window Form), and conducted the experiment on a HP, Intel(R) Core(TM) i5 CPU M430 @ 2.27GHz (4 CPUs), 4GB RAM, Windows 8.1 Pro 64-bit.

This experiment uses the datasets from the UCR Time Series Data Mining archive ([8], [9]). There are 5 datasets used in this experiment. The names and lengths of the five datasets are shown in TABLE IV.

TABLE IV. LENGTH AND DISCORD LENGTH FOR EACH DATASET

Dataset description	Dataset Name	(Time series) length	Length of streaming time series	Discord length
Space Shuttle Marotta Valve	*TEK16*	2992	2000	128
Electrocardiograms	*ECG*	16600	5000	40
Power Data	*Power_data*	30040	5000	200
Patient's respiration data	*nprs43*	13020	5000	160
Earth Rotation Parameters	*ERP*	193400	5000	64

For HDD-MBR-Stream we have to estimate six parameters: the period of the time series, the buffer length of the time series, the reduced dimension of PAA w, the parameter for LB_Keogh r, the two parameters for R-tree: MaxEntryPerNode (*maxE*) and MinEntryPerNode (*minE*). For BFHS and HOTSAX-Stream, we need two more parameters: the length of SAX word w, the alphabet size a. The values of the parameters in the three algorithms for each dataset are shown in TABLE V.

TABLE V. PARAMETERS FOR EACH DATASET

Dataset	Period length	Buffer length	*w*	*a*	*r*	*MaxE*	*MinE*
TEK16	1007	1007*2	5	3	2	25	12
ECG	371	371*10	5	3	2	25	12
Power_data	672	672*5	4	3	2	25	12
Nprs43	40	40*75	3	3	2	25	12
ERP	1280	1280*2	4	3	2	25	12

1) Effectiveness

We compare HDD-MBR-Stream and BFHS in discord detection accuracy over the 5 streaming time series datasets. For each dataset, we found out that the discord detected by HDD-MBR-Stream is almost the same as the discord detected by BFHS.

Fig. 5 shows the TEK16 dataset with 5 periods (period length is about 1000).

In Fig. 6 we evaluate our HDD-MBR-Stream algorithm in a real case (with dataset TEK16). The top graph shows the locations of the discords found by HDD-MBR-Stream, at each time point. The bottom graph shows the nearest non-self neighbor distance of each input subsequence at each time point.

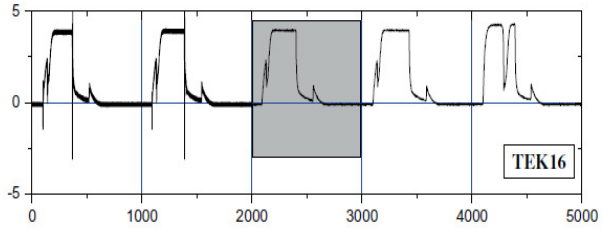

Fig. 5 TEK16 time series with five periods (period lenghth ≈1000)

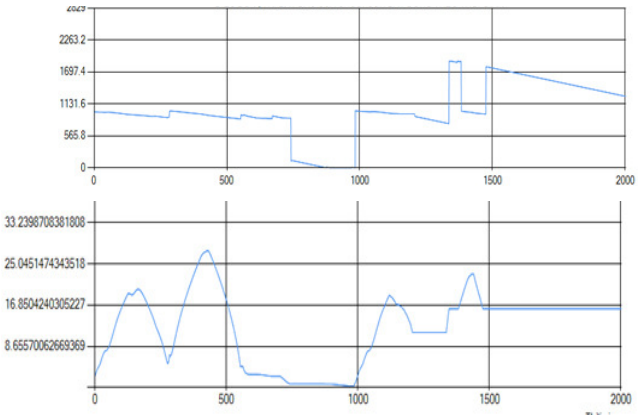

Fig. 6 Online discord detetion on TEK 16 dataset. (Top) Start locations of discords. (Bottom) Nearest non-self distances of discords (HDD-MBR-Stream)

2) Efficiency

TABLE VI shows the runtimes (in seconds) of the three algorithms: BFHS, HOTSAX-Stream and HDD-MBR-Stream in discord discovery over the 5 streaming datasets. From the experimental results in TABLE VI, we can see that HDD-MBR-Stream performs much faster than BFHS and HOTSAX-Stream in all the datasets. On average, HDD-MBR-Stream runs about 8.064 times faster than BFHS and about 4.332 times faster than HOTSAX-Stream.

TABLE VI. EXECUTION TIMES (IN SECONDS) OF BFHS, HOTSAX-STREAM AND HDD-MBR-STREAM

Dataset	BFHS	HOTSAX-Stream	HDD-MBR-Stream
TEK16	1226	620	444
ECG	8940	2838	454
Power_data	11400	2765	2016
nprs43	7740	4440	1283
ERP	3618	2538	394

VI. CONCLUSIONS

Both discord detection in static time series and streaming time series are challenging problems. In this work, we proposed an efficient algorithm which uses R-tree, called HDD-MBR, for efficient time series discord detection. R-Tree helps HDD-MBR in arranging two ordering heuristics for outer loop and inner loop. In addition, we extend HDD-MBR to a new algorithm, called HDD-MBR-Stream for discord detection in streaming time series. We experimentally showed that HDD-MBR is much faster than HOT SAX in offline discord detection and the online detection algorithm HDD-MBR-Stream is remarkably faster than BFHS, the brute force search approach which is based on the original HOT SAX.

HDD-MBR-Stream still belongs to the category of window-based methods for time series discord detection. In the future, we intend to develop a much more efficient algorithm for discord discovery in streaming time series which is based on segmentation and incremental clustering.

ACKNOWLEDGMENT

We are grateful to Prof. Eamonn J. Keogh for kindly providing necessary datasets for the research work.

REFERENCES

[1] Y. Bu, T.W. Leung, A.W.C. Fu, E. Keogh, J. Pei, and S. Meshkin, "WAT: Finding top-k discords in time series database", *Proceedings of the 2007 SIAM International Conference on Data Mining*, Minneapolis, Minnesota, USA, pp. 449-454, 2007

[2] H.T . Q. Buu and D. T. Anh, ""Time series discord discovery based on iSAX symbolic representation", *Proceedings of 2011 Third International Conference on Knowledge and Systems Engineering* (KSE), Hanoi, Vietnam, pp. 11-18, 2011.

[3] D. Cheboli, "Anomaly Detection of Time Series", Master Thesis, University of Minnesota, 2010.

[4] F. X. Diebold, Elements of Forecasting, Fourth Edition, Thomson South-Western, 2007.

[5] J. Han, M. Kamber, J. Pei, Data Mining: Concepts and Techniques, 3rd Edition, Morgan Kaufmann, 2011.

[6] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," *SIGMOD'84, Proceedings of Annual Meeting,* Boston, Massachusetts, June 18-21, 1984.

[7] E. Keogh, S. Chu, D. Hart, M. Pazzani, "An Online Algorithm for Segmenting Time Series," *Proceedings of IEEE International Conference on Data Mining*, pp.289-296, 2001.

[8] E. Keogh, K. Chakrabatti, M. Pazzani, "Dimensionality reduction for fast similarity search in large time series databases," *Knowledge and Information Systems*, vol. 3, no. 3, pp. 263-286, 2000.

[9] E. Keogh, J. Lin and A. Fu, "HOT SAX: efficiently finding the most unusual time series subsequence," *Proc. of 5th IEEE Int. Conf. on Data Mining* (ICDM), Houston, Texas, pp.226–233, 2005.

[10] E. Keogh, J. Lin, and A. Fu, [online] http://www.cs.ucr.edu/~eamonn/discords/

[11] E. Keogh and T. Folias, The UCR Time Series Data Mining Archive, [http://www.cs.ucr.edu/~eamonn/TSDMA/index.html]. Assessed in Jan. 2017.

[12] E. Keogh, C. Ratanamahatana, "Exact indexing of dynamic time warping," *Knowledge and Information Systems*, Volume 7, Issue 3, pp 358–386, March 2005

[13] N.H. Kha, D.T. Anh, "From cluster-based outlier detection to time series discord discovery", *Trends and Applications in Knowledge Discovery and Data Mining-PAKDD 2015 Workshops: BigPMA, VLSP, QIMIE, BAEBH*, Ho Chi Minh City, Vietnam, May 19-21, 2015, X..L. Li et al. (Eds.), LNAI 9441, Springer, 16-28.

[14] M. Leng, X. Chen and L. Li, "Variable length methods for detecting anomaly patterns in time series," *Proc. of Int. Symposium on. Computational Intelligence and Design* (ISCID'08), Vol. 2, 2008.

[15] G. Li, O. Braysy, L. Jiang, Z. Wu, Y. Wang, "Finding time series discord based on bit representation clustering," *Knowledge-Based Systems*, vol.52, pp. 243-254, 2013.

[16] J. Lin, E. Keogh, S. Lonardi, B. Chiu, "Symbolic representation of time series, with implications for streaming algorithms," *Proc. of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, San Diego, CA, June 13, 2003.

[17] Liu, Y., Chen, X., Wang, F., Yin, J., "Efficient Detection of Discords for Time Series Stream," Li Q., Feng L., Pei J., Wang S.X., Zhou X., Zhu QM. (eds), *Advances in Data and Web Management*, vol. 5446, pp. 629-634, 2009.

[18] J. Ma, S. Perkins, "Time series novelty detection using one-class support vector machines," *Proc. of Int. Joint Conf. on Neural Networks*, Vol. 3, pp.1741-1745, 2003.

[19] A. L. I. Oliveira, F.B.L. Neto, S.R.L. Meira, "A method based on RBF-DAA neural network for improving Novelty detection in time series," *Proc. of 17th International FLAIRS Conference*, AAAI Press, Miami Beach, Florida, USA., 2004.

[20] H. Sanchez, B. Bustos, "Anomaly Detection in Streaming Time Series Based on Bounding Boxes," *Traina A.J.M., Traina C., Cordeiro R.L.F. (eds) Similarity Search and Applications. SISAP 2014.* LNCS 8821. Springer, 2014.

[21] S. Salvador, P. Chan, "Learning states and rules for time series anomaly detection," *Applied Intelligence*, vol. 23, no.3, pp. 241 -255, 2005.

[22] D. Toshniwal, S. Yadav, "Adaptive outlier detection in streaming time series', *Proceedings of International Conference on Asia Agriculture and Animal. ICAAA 2011*, Hong Kong, China, pp. 186-191, 2011.

[23] Y. Zhu and D. Shasha, "Efficient elastic burst detection in data streams", *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, Washington, DC, USA, pp. 181–192, 2003.