TIME SERIES DISCORD DISCOVERY BASED ON R*-TREE

NGUYEN THANH SON*

ABSTRACT

In this paper, we propose a new algorithm for time series discord discovery based on R-tree. Our method is time and space efficient because it only saves Minimum Bounding Rectangles (MBR) of data in memory and needs a single scan over the entire time series database and a few times to read the original disk data in order to validate the results. The experimental results showed that our proposed algorithm outperforms the popular method, Hot SAX, in term of runtime and efficiency.*

Keywords: time series, multi-dimensional index, discord discovery, R*-tree.

TÓM TẮT

Khám phá bất thường trên chuỗi thời gian dựa vào R*-tree

Trong bài báo này, chúng tôi đề xuất một thuật toán mới cho bài toán khám phá bất thường trên chuỗi thời gian dựa vào cây R. Phương pháp này đạt hiệu quả về mặt thời gian lẫn không gian lưu trữ vì nó chỉ lưu vùng bao chữ nhật nhỏ nhất của chuỗi thời gian trong bộ nhớ và chỉ cần một lần quét qua toàn bộ dữ liệu chuỗi thời gian cùng một vài lần đọc dữ liệu trên đĩa để thẩm định lại kết quả. Kết quả thực nghiệm đã cho thấy phương pháp của chúng tôi thực hiện nhanh và hiệu quả hơn phương pháp thông dụng Hot SAX.*

Từ khóa: chuỗi thời gian, chỉ mục đa chiều, khám phá bất thường, cây R*.

1. Introduction

A time series is a sequence of real numbers where each number represents a value at a given point in time. Time series data arise in so many applications of various areas ranging from science, engineering, business, finance, economy, medicine to government.

An important research area in time series data mining which has received an increasing amount of attention lately is the problem of discovering discord in time series.

Time series discord is a subsequence of a longer time series which is the most different from all the rest of the time series subsequences. Time series discord discovering has been used to solve problems in several application areas such as fault diagnostics, intrusion detection, data cleansing and so on.

A formal definition of time series discord has been first introduced by Keogh et al. in 2005 [8]. In order to decrease the time complexity of the Brute Force Discord Discovery (BFDD) algorithm, in [8] authors have suggested a generic framework

* Ph.D., HCM City University of Technology and Education; *Email: sonnt@fit.hcmute.edu.vn*

called Heuristic Discord Discovery (HDD) algorithm instead of sequential searching. This algorithm uses two heuristics which determine the order in which the outer loop and inner loop visit the subsequences, respectively.

To improve the efficiency of the HDD algorithm, from this generic framework, Keogh et al. proposed a new discord discovery algorithm called Hot SAX [8]. In this algorithm, first the input time series is discretized into symbolic strings by Symbolic Aggregate Approximation (SAX). Then the two aforementioned heuristics can be applied as in HDD algorithm. The authors show that Hot SAX can run 3 to 4 orders of magnitude faster than BFDD. However, Hot SAX still based on the discretization process, the SAX approximation, without working on numerical time series data. So it requires some parameters such as the length of the discord, the alphabet size and the word size for the compression of subsequences.

In our work, we propose a new method for discovering time series discord, that is R*-tree-based method. This approach employs R*-tree index structure to speed up the search for the nearest neighbor of a sequence. Our proposed method is time and space efficient because it only requires a single sequential disk scan to read the time series database and a few times to read the original disk data to verify the result and only saves MBRs of data in memory. Besides, this method can directly work on numerical time series data transformed by some dimensionality reduction method but without applying some discretization process.

We experimented the proposed algorithm on real time series datasets of various areas. The experimental results show that this algorithm outperforms the popular method, Hot SAX algorithm, in terms of runtime and efficiency.

The rest of the paper is organized as follows. In Section 2 we examine background and related words. Section 3 describes our approach for discord discovering in time series. Section 4 presents our experimental evaluation on real datasets. In section 5 we include some conclusions.

2. Background and related works

2.1. Background

- **Definition 1.** ***Euclidean Distance*****:**

Euclidean distance is the simplest method to measure the similarity of time series. Given two time series $Q = \{q_1, \ldots, q_n\}$ and $C = \{c_1, \ldots, c_n\}$, the Euclidean distance between Q and C is defined as

$$D(Q,C) \equiv \sqrt{\sum_{i=1}^{n} (q_i - c_i)^2} \quad (1)$$

The Euclidean distance metric has been widely used for pattern matching [10].

- **Definition 2. *Time series*:** A *time series* is a real value sequence of length *n* over time, i.e. if T is a time series then $T = (t_1, \ldots, t_n)$ where t_i is a real number.
- **Definition 3. *Subsequence*:** Given a time series $T = (t_1, \ldots, t_n)$, a *subsequence* of length m ($m<n$) of T is a sequence $S = (t_i, \ldots, t_{i+m-1})$ with $1 \le i \le n - m + 1$.

Since all subsequences may potentially be discords, we have to compare any subsequence to all remaining subsequences. However, the best matches of a subsequence tend to be located some points to the left or to the right of the subsequence in question. Such matches are called trivial matches and they have to be excluded from the result of discovering discords.

Note that, all subsequences extracted from a time series T can form a *subsequence database* in which each subsequence can be regarded as a time series.

- **Definition 4.** *Non-trivial match*: Given a time series T, containing a subsequence C_p of length m beginning at position p and a matching subsequence C_q beginning at q, we say that C_q is a non-trivial match to C_p if $|p - q| \ge m$.
- **Definition 5.** *Time series discord*: Given a time series database T, the sequence $C \in T$ is the most significant discord in T if the distance to its nearest neighbor Q (or its nearest non-trivial match in case of subsequence databases) is largest. It means that for an arbitrary time series $M \in T$, $\min(D(C, Q)) \ge \min(D(M, P))$, where $Q, P \in T$ (and Q, P are non-trivial matches of C and M in case of subsequence database).
- **Indexing structure.**

The popular multidimensional index structures are R-tree and its variants ([5], [1]). An R-tree is a high balanced tree similar to a B-Tree. In a multidimensional index structure (e.g., R-tree or R*-tree), each node is associated with a minimum bounding rectangle (MBR). A MBR at a node is the minimum bounding box of the MBRs of its child nodes. A potential weakness in the method using MBR is that MBRs in index nodes can overlap. Overlapping rectangles could have negative effect on the search performance.

2.2. Related works

The problem of discovering unusual time series has attracted much attention, and various kinds of time series discord discovery methods have been introduced. The following are some typical methods for discord discovery in time series.

In [8] Keogh et al. proposed a fast heuristic technique (called Hot SAX) for pruning quickly the data space and focusing only on the potential discords. However, Hot SAX still based on the discretization process, the Symbolic Aggregate Approximation (SAX), without working on numerical time series data. So it requires some parameters such as the length of the discord, the alphabet size and the word size for the compression of subsequences.

In 2006 Fu et al. proposed a new algorithm based on Haar Wavelet transform to determine dynamically the word size for the compression of subsequences [4].

In [2] Bu et al. proposed a new method called WAT (Wavelet and augmented trie) which is based on Haar Wavelet transform and augmented trie to mine the top-*k* discords from time series data [2]. This algorithm requires fewer input parameters than that of Hot SAX due to exploiting the multi-resolution features of wavelet transform to determine dynamically a suitable word size for a particular dataset.

In [3] Chuah et al. proposed an anomaly detection method. It is based on time series analysis in order to determine whether a stream of real-time sensor data contains any abnormal heartbeats. If anomaly exists, that time series segment will be transmitted via the network to a physician so that experts can further diagnose the problem and take appropriate actions.

In [12], Lin et al. introduced a new approach for the anomaly detection problem. First, this method uses subseries join to obtain the similarity relationships among subseries of the time series data. Then it converts the anomaly problem to graph-theoretic problem which can be solve by existing graph-theoretic algorithm.

In 2011, Buu et al. proposed a new time series discord discovery algorithm, called HOTiSAX [6]. This algorithm incorporates iSAX (**i**ndexable **S**ymbolic **A**ggregate appro**X**imation) representation in Hot SAX instead of SAX representation. ISAX is a new symbolic representation proposed by Shieh and Keogh in 2008 which is an extension of SAX [17].

In 2012, Khanh et al. proposed a new method for discord discovery in time series, called WATiSAX [14]. This algorithm employs iSAX representation in WAT algorithm.

In 2013, Luo et al. [13] proposed a new method which exploits a recurrence structure of time series and uses a reference function that makes the search algorithm more efficient and robust.

In [7], Jones et al. introduced a new algorithm for discovering anomalies in real valued multidimensional time series. First this method uses an exemplar-based model for detecting anomalies in single dimensional time series, then uses a function that predicts one dimension from a related one.

In [16] Pavel Senin et al. proposed a new algorithm which use grammar induction to aid anomaly detection without any prior knowledge. First, this algorithm discretizes continuous time series values into symbolic form, then it infers a context free grammar. Finally, the algorithm uses its hierarchical structure to effectively and efficiently discover anomalies

3. Our proposed approach

Comparing to the case of database T which contained $|T|$ separate time series of length n, the fundamental algorithm for time series discord discovery in the case of

subsequences from a long time series remains unchanged with some additional miner bookkeeping to discount trivial matches. So, the discussion is limited to the case where database T contains $|T|$ separate time series of length n.

The basic intuition behind our algorithm is that a multidimensional index such as R*-tree [1] can be used for retrieving the nearest neighbor of a sequence.

To insert time series into R*-tree, we create a Minimum Bounding Rectangle (MBR) in the m dimensional space ($m<<n$) for each sequence of length n in the time series database. Then each sequence is inserted into R*-tree based on its MBR. To find a nearest match of a sequence s by searching in R*-tree, we need a distance function $D_{region}(s, R)$ of the sequence s from the MBR R associated with a node in the index structure such that $D_{region}(s, R) \leq D(s, C)$, $\forall C$, any sequence which is contained in the MBR R.

Before introducing a formal definition of $D_{region}(s, R)$, we need to describe how to calculate it in such a way that the correctness requirement is satisfied i.e. $D_{region}(s, R) \leq D(s, C)$, $\forall C$ in the MBR R.

Notice that each time series of length n can be considered as a point in n dimensional space. Supposed that we have built an index of time series by inserting the points $C = \{c_1, ., c_n\}$ into a MBR-based multi-dimensional index structure. Assume that we approximate a time series of length n by m equal-length constant value segments ($m<<n$). Let U be a leaf node of that index and $R = \{R_1, R_2, \ldots, R_m\}$ be the MBR associated with U where $R_j = \{L_j, H_j\} = \{(x_{jmin}, y_{jmin}), (x_{jmax}, y_{jmax})\}$ is a pair of endpoints which are the lower and higher endpoints of the major diagonal of R_j. R_j is defined as the 2-dimensional rectangular region in the *value-time space* that fully contains the j^{th} segment of all of time series stored under node U. *Value* and *time* are represented by y-dimension and x-dimension respectively. By definition, R is the smallest rectangle spatially contains each time series C which is inserted into U. The MBR associated with a non-leaf node would be a smallest rectangle that spatially contains the MBRs of its immediate children [5]. We can view the MBR as two sequences which are the lower bound $L = \{L_1, \ldots, L_m\}$ and upper bound $H = \{H_1, \ldots, H_m\}$ of time series stored under node U. Figure 1 illustrates a time series which is approximated by m (m=4) equal-length segments and its MBR $R = \{R_1, R_2, R_3, R_4\}$

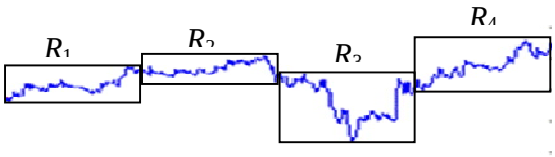

Fig.1. *An example of a time series approximated by m (m = 4) equal-length segments and its MBR*

In order to calculate the distance between a time series *s* and the bounding region *R*, $D_{region}(s, R)$, we accumulate distances from every value in the sequence *s* to *R* by calculating the distance, $d(s_{ji}, R_j)$, from each value s_{ji} in each segment *j* ($1 \le j \le m$) of time series *s* to the corresponding j^{th} bounding region R_j of MBR *R* which depends on the fact that it is above or in or under R_j.

Figure 2 shows an example of how to calculate $D_{region}(s, R)$. In this example $s = \{s_1, \ldots, s_9\}$, $R = \{R_1, R_2, R_3\}$ and we have $D_{region}(s, R) = (s_{11} - y_{1\max})^2 + (s_{21} - y_{2\min})^2 + (s_{32} - y_{3\min})^2$. Other remaining values are equal to zero since they are inside *R*.

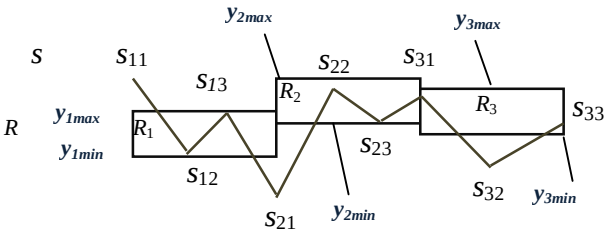

Fig. 2. *An example of how to calculate $D_{region}(s, R)$*

Definition 5. Given a time series *s* of length *n*, a set of time series *C* and a corresponding MBR *R* of *C* in the *m* dimensional space ($m << n$), i.e., $R = \{R_1, R_2, \ldots, R_m\}$, where $R_j = \{(x_{jmin}, y_{jmin}), (x_{jmax}, y_{jmax})\}$ is a pair of endpoints which are the lower and higher endpoints of the major diagonal of R_j. The distance function $D_{region}(s, R)$ of the sequence *s* from the MBR *R* is defined as follows [15].

$$D_{region}(s, R) = \sqrt{\sum_{j=1}^{m} D_{region_j}(s_j, R_j)} \quad (2)$$

where $D_{region_j}(s_j, R_j) = \sum_{i=1}^{N} d(s_{ji}, R_j)$

$$d(s_{j_i}, R_j) = \begin{cases} (y_{j\min} - s_{j_i})^2 & \text{if } s_{ji} < y_{jmin} \\ (s_{j_i} - y_{j\max})^2 & \text{if } s_{ji} > y_{jmax} \\ 0 & \text{otherwise} \end{cases}$$

N is the length of segment *j*.

To ensure the correctness of using $D_{\text{region}}(s, R)$ in searching nearest-neighbors of a query based on a multidimensional index, this group distance must satisfy the *group-lower-bound property* as follows.

Lemma 1. $D_{\text{region}}(s, R) \le D(s, C)$, $\forall C$ in the MBR *R*.

where

$$D(s,C) = \sqrt{\sum_{i=1}^{n}(s_i - c_i)^2} = \sqrt{\sum_{j=1}^{m}\sum_{i=1}^{N}(s_{ji} - c_{ji})^2}$$

Proof: According to the definition of the MBR associated with a node U in the index structure and the definition of the distance function $D_{\text{region}}(s, R)$, for any sequence C placed in a node U and the MBR R associated with U, we have

$y_{jmin} \le c_{ji} \le y_{jmax}, \forall i = 1, .., N, \forall j = 1, .., m$

$$D_{region_j}(s_j, R_j) \le D(s_j, C_j)$$

That implies

$$\forall j = 1, .., m$$

Where

$$D(s_j, C_j) = \sum_{i=1}^{N} (s_{ji} - c_{ji})^2$$

Hence

$D_{region}(s, R) \le D(s, C), \forall C$ in the MBR R.

Table 1. *Finding time series discord with the support of R*-tree*

Function [discord_id, dist] =
Finding_Discord(T, n)
in: T: a time series database
n: the length of sequence
out: discord_id: id of discord sequence
dist: the distance to its nearest neighbor
1: **for** j = 1 to $|T|$
2: Add(j, MBR$_j$, R*-tree)
3: **end for**
4: for j = 1 to $|T|$
5: x = Nearest_Neighbor(j, R*-tree)
6: **if** (Distance(T_j, T_x, n) > best_so_far_dist)
7: best_so_far_dist = Distance(T_j, T_x, n)
8: discord_id = i
9: **end if**
10: **end for**
11: **Return [**discord_id, best_so_far_dist]

Our algorithm for time series discord discovery with the support of R*-tree is shown in table 2. Table 3 illustrates a function for finding the nearest neighbor of a sequence using R*-tree. Table 4 illustrates a procedure for insert a sequence into R*-tree based on its minimum bounding rectangle (MBR).

Table 2. *Finding the nearest neighbor of T_j using R*-tree*

Function [*x*] = Nearest_Neighbor(*j*, R*-tree)
in: *R*: R*-tree
j: ID of sequence s_j
out: *x*: ID of the nearest neighbor of s_j
1: Traverse R*-tree downward from root node to find a leaf node *m* which MBR is nearest to the sequence s_j
2: For *i* = 1 to number of entries in the node *m*
3: Finding an entry *x* which is nearest to the sequence s_j
4: Return *x*

Table 3. *Insert the sequence j into R*-tree based on its minimum bounding rectangle (MBR)*

Procedure Add(*j*, MBR_j, R*-tree)
1: Choosing a sub-tree so that its MBRs require least overlap enlargement to include the sequence *j*.
2: Add the sequence *j* to the leaf node that is suitable for it.
3: If the leaf node is full
4: Split node with the criterion of minimizing the total area of the two covering rectangles after a split.
5: The node splitting process is repeated for father nodes if the father node is full due to splitting its child node.

4. Experimental evaluation

In this experiment, The methods are implemented with Microsoft Visual C# and conducted the experiments on a Core i3, Ram 2GB.

We compare the proposed approach to Hot SAX method. 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 runtime and efficiency. Here, we evaluate efficiency (*eff*) of the proposed algorithm by simply considering the ratio of how many times the Euclidean distance function must be evaluated by the proposed algorithm over the number of times it must be evaluated by the brute force algorithm described in [11].

$$eff = N_1/N_2 \quad (3)$$

where N_1 is the number of times proposed method calls Euclidean distance and N_2 is the number of times brute-force calls Euclidean distance.

The efficiency value is always less than 1. The method with lower efficiency value is better.

We tested on four different datasets which come from various sources publicly available through the Internet: Stock, ECG, Waveform and Burst. We conduct the experiments on the datasets with cardinalities ranging from 1000 to 10000 for each dataset. We consider the discord length ranging from 128 to 1024. In the methods using R*-tree, MBRs of time series are built with compression ratio 32:1. In Hot SAX, we use the same compression ratio and set the alphabet size of SAX to 5. For brevity, we report only some typical experimental results.

Figure 3 reports runtime and efficiency of two algorithms on different datasets with discord of length 128 and a fixed dataset size 4000. From the experimental result, we can see that for all the datasets used in the experiment our proposed method is faster and more efficient than HOT SAX.

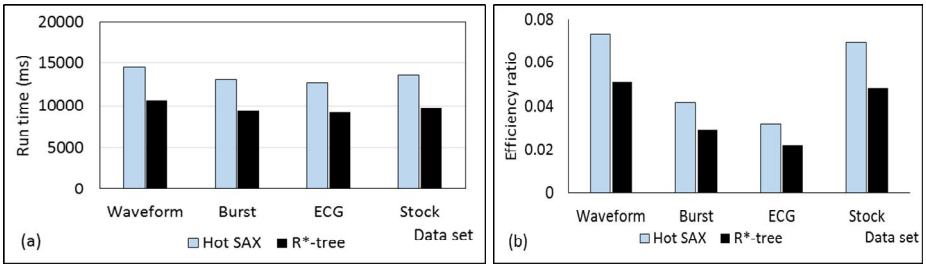

Fig.3. *The experimental results of runtime (a) and efficiency (b) of two algorithms on different datasets with a fixed discord length 128*

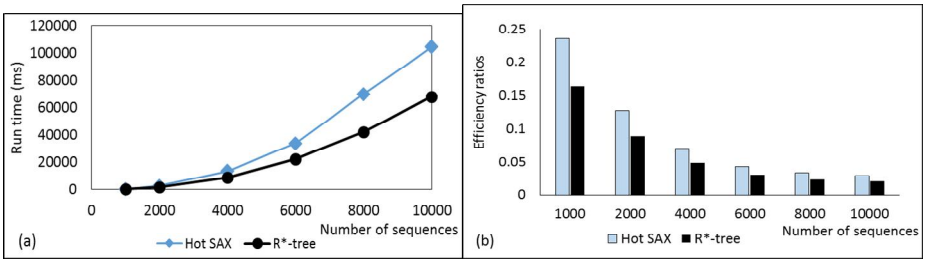

Fig.4. *The experimental results of runtime (a) and efficiency (b) of two algorithms on Stock dataset with different sizes and a fixed discord length 128*

Figure 4 shows runtime and efficiency of two algorithms on Stock dataset with cardinalities ranging from 1000 to 10000 and a fixed discord length 128.

From this experimental result, we can see that runtime of our proposed method is smaller than or equal to that of HotSAX. The efficiency in this case is also better than that of HotSAX. Figure 4(a) showed that the bigger a dataset size is, the longer the runtime is. However, figure 4(b) showed that the bigger a dataset size is, the smaller efficiency is. It is because the bigger a dataset size is, the greater the number of times brute-force calls Euclidean distance and therefore, the smaller efficiency is.

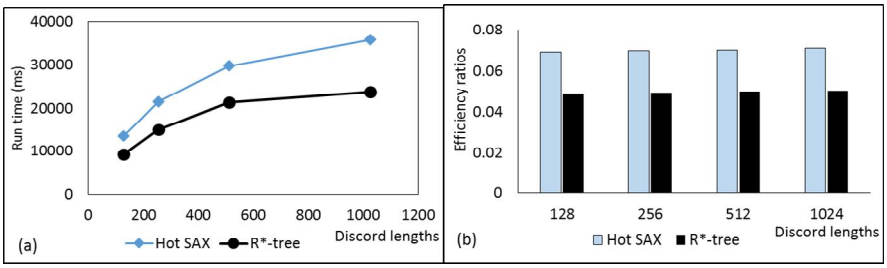

Fig.5. *The experimental results of runtime (a) and efficiency (b) of two algorithms on Stock dataset with different discord lengths and a fixed dataset size 4000*

Figure 5 reports runtime and efficiency of two algorithms on Stock dataset with discord lengths ranging from 128 to 1024 and a fixed dataset size 4000. From the experimental result, we can see that for different discord lengths our proposed method is also faster and more efficiency than HOT SAX. Figure 5(a) showed that the longer the discord length is, the longer the runtime is. It is true because the time needed to calculate Euclidean distance is longer. Figure 5(b) showed that efficiency of these experiment are approximate. It is because the number of times brute-force calling Euclidean distance is unaffected by the length of discord.

The accuracy of the proposed discord discovery algorithm is basically based on human analysis of the discords discovered by that algorithm ([2], [3], [9], [8]). That means if the discords identified by a proposed algorithm on most of the test dataset are almost the same as those identified by the baseline discord discovery algorithm (here HOT SAX is chosen as the baseline algorithm), we can say that the proposed discord discovery algorithm brings out the same accuracy as the baseline algorithm.

For brevity, figure 6 shows only one example of discord discovered in Stock dataset by two algorithms.

From figure 6, we can see that the discord discovered in Stock dataset by our proposed method is similar to that discovered by Hot SAX algorithm. Experimental results on the remaining datasets are similar.

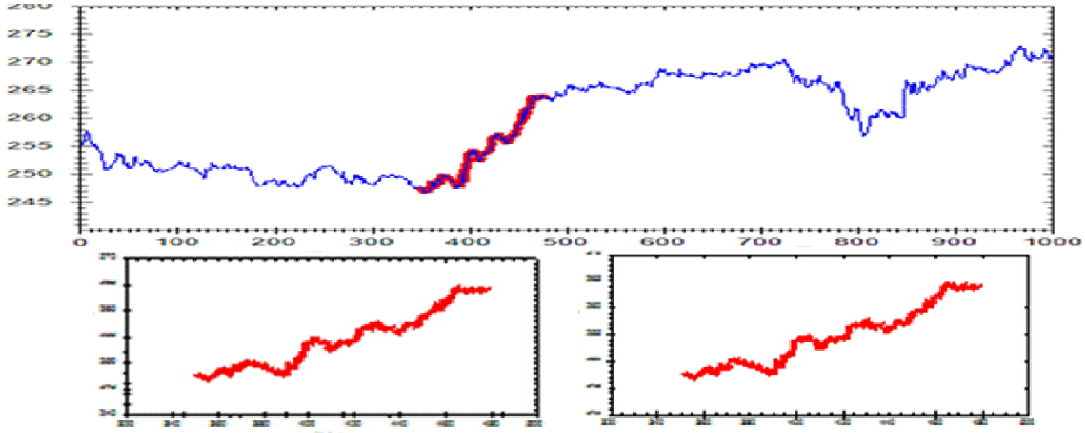

Fig. 6. *Stock dataset (top). The discord discovered in the Stock dataset by HOT SAX (bottom left). The discord discovered in the Stock dataset by our proposed algorithm (bottom right)*

5. Conclusions.

In this paper, we propose a new algorithm for time series discord discovery. This algorithm uses a multidimensional index structure, R*-tree, which help speeding up the search for the nearest neighbor of a sequence.

Our experiments on four real world datasets show that our approach is faster than HOTSAX algorithm in detecting time series discords while the anomalous patterns discovered by the two methods are similar.

In future, we plan to experiment our proposed method with more real world datasets and improve our algorithm to achieve more time efficiency.

REFERENCES

1. Beckman, N., Kriegel, H.P., Schneider, R. & Seeger, B. (1990). "The R*-tree: An efficient and robust access method for points and rectangles", *Proc. of 1990 ACM-SIGMOD Conf.*, Atlantic City, NJ, May 1990, pp. 322-331.
2. Bu, Y., Leung, T-W., Fu, A., Keogh, E., Pei, J. & Meshkin, S. (2007). "WAT: Finding Top-K Discords in Time Series Database". In *Proceeding of the 2007 SIAM International Conference on Data Mining* (SDM'07), Minneapolis, MN, USA.
3. Chuah, Mooi Choo & Fen Fu (2007). "ECG anomaly detection via time series analysis". *Frontiers of High Performance Computing and Networking ISPA 2007 Workshops*. Springer Berlin Heidelberg, 2007, pp 123-135.
4. Fu, A., Leung, O., Keogh, E. & Lin, J. (2006). "Finding Time Series Discords Based on Haar Transform". In *Lecture Notes in Computer Science, Advanced Data Mining and Applications*, Springer Berlin / Heidelberg, 31-41.
5. Guttman, A., (1984). "R-trees: a Dynamic Index Structure for Spatial Searching". *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, June 18-21, 47-57.
6. H. T. Q. Buu & D. T. Anh (2011). **"**Time Series Discord Discovery Based on iSAX Symbolic Representation". In *Proceeding of the Third International Conference on Knowledge and System Engineering (KSE 2011), Hanoi, Vietnam, October 14-17*. IEEE, pp 11-18.
7. Jones, M., Nikovski, D., Imamura, M. & Hirata, T., (2014) "Anomaly Detection in Real-Valued Multidimensional Time Series". In *Proc. of 2014 ASE BIGDATA/ SOCIALCOM/ CYBERSECURITY Conference*, Stanford University, May 27-31, 2014.
8. Keogh, E., Lin, J. & Fu, A. (2005). "HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence". In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, 226-233.
9. Keogh, E., Lonardi, S. & Chiu, B., (2002). "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.

10. Keogh, E. & S. Kasetty, S., (2002). "On the Need for Time series Data Mining Benchmarks: A Survey and Empirical Demonstration". In *the 8th ACM SIGKDD*, pp. 102-111.
11. Lin, J., Keogh, E., Lonardi, S. & Patel, P. (2002). "Finding motifs in time series". In: *Proc. 2nd Workshop on Temporal Data Mining*. Edmonton, Alberta, Canada.
12. Lin, Yi, Michael D. McCool & Ali A. Ghorbani. (2010). "Motif and anomaly discovery of time series based on subseries join". *IAENG International Conference on Data Mining and Applications, ICDMA*.
13. Luo W., Gallagher M. & Wiles J. (2013). "Parameter-free search of time-series discord". *Journal Of Computer Science And Technology* 28(2): 300-310 Mar. 2013. DOI 10.1007/s11390-013-1330-8.
14. N. D. K. Khanh & D. T. Anh (2012). "Time series discord discovery using WAT algorithm and iSAX representation". *In the Proceedings of the Third Symposium on Information and Communication Technology (SoICT'12)*, ACM New York, NY, USA, pp. 207-213.
15. N. T. Son & D. T. Anh (2016). "Discovery of time series k-motifs based on multidimensional index". *International Journal of Knowledge and Information Systems*, Springer, 46(1), pp. 59-86, 5 Jan 2016.
16. Pavel Senin, Jessica Lin, Xing Wang, Tim Oates & Sunil Gandhi. (2015). "Time series anomaly discovery with grammar-based compression". In *Proc. 18th International Conference on Extending Database Technology (EDBT)*, March 23-27, 2015, Brussels, Belgium
17. Shieh, J. & Keogh, E. (2008). "iSAX: Indexing and mining terabyte sized time series". In *Proceedings of SIGKDD*.

(*Received: 28/10/2016; Revised: 28/11/2016; Accepted: 16/12/2016)*