J-distance Discord: An Improved Time Series Discord Definition and Discovery Method

Tian Huang, Shanghai Jiao Tong University, ian_malcolm@sjtu.edu.cn
Yongxin Zhu, Shanghai Jiao Tong University, zhuyongxin@sjtu.edu.cn
Yafei Wu, Shanghai Jiao Tong University, wuyf0406@sjtu.edu.cn
Weiwei Shi, Shanghai Jiao Tong University, iamweiweishi@sjtu.edu.cn

***Abstract*—A time series discord is a subsequence that is maximally different to all the rest subsequences of a longer time series. Classic discord discovery has been used for detecting anomalous or interesting pattern, which usually represents the most unusual subsequences within a time series. However, an anomalous or interesting pattern may happen twice or more times so that any instance of this pattern is not distinct enough to be a top discord. To mitigate the issue, we propose an improved definition named *J-distance discord* (JDD), which incorporates the methodologies of KNN (k nearest neighbor) algorithm. JDD measures the similarity between a subsequence and its J^{th} most similar subsequence and ranks discords according to the similarity. We also propose a JDD discovery method to reduce the extra computational requirements brought by JDD definition. Experiments on synthetic and real world datasets show that JDD captures more de-facto anomalous and interesting patterns compared to the results of the original definition of discord. Besides, the JDD discovery method is as fast as the classic discord discovery method in terms of computational efficiency.**

I. Introduction

Time series discord is the most unusual subsequence of a time series [1], [2]. Discord can be discovered by calculating the distance of each subsequence to its non-overlapping nearest neighbor. The subsequence that has the greatest such value is the discord. Recently, finding time series discord has attracted much attention. The simple definition of discord captures an important class of anomalies, the relevance of which has been shown in several data mining applications [1], [3], [4] . For example, Fig.1 shows an electrocardiography (ECG) record, which is used in the detection of premature ventricular contraction arrhythmia [5]. The presence of arrhythmia is reflected by the time series discord shown in red. Discord discovery is also valuable in data mining tasks, such as anomaly detection, improving quality of clustering and data cleaning.

Discord discovery has the utility for detecting anomalous or interesting pattern, which however may appear twice or more times to form a *"Twin Freak"* problem [3] that current discord discovery methods fail to handle. In issues like Twin Freak problem, the instance of unusual subsequences is not distinct enough to be a discord. Consequently this pattern may not be among the best of the discord ranking list, or even never be captured as a discord.

In this paper we propose an improved definition named *J-distance discord* (JDD), which incorporates the methodologies of KNN algorithm. Our method measures similarity between a subsequence and its J^{th} most similar subsequence and ranks

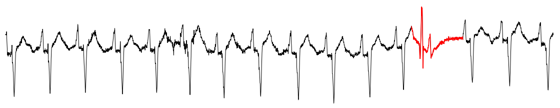

Fig. 1. ECG segment for a patient with premature ventricular contraction arrhythmia. Courtesy of Goldberger et al. [5].

discords according to the similarity. Besides, JDD imposes extra computational requirements, which is about finding J non-overlapping nearest neighbor of a subsequence. The prohibitive computational requirements would degrade the utility of discord discovery. Our challenge is to develop a JDD discovery method that is computationally efficient. We make contributions in the paper as follows.

- We define *J-distance discord* (JDD) so that it captures the multiple occurrence of anomalous or interesting patterns while still inherits the nice features of the original definition of discord, such as anomaly and change detection.
- We develop a JDD discovery method, which applies enhanced early abandon technique and reuses intermediate results to reduce the extra requirements brought by the definition of JDD.

We empirically evaluate JDD and discovery method with one synthetic dataset and three real world datasets. Experiments show that JDD captures more de-facto anomalous and interesting patterns compared to the results of the original definition of discord. Besides, the JDD discovery method is as fast as classic discord discovery method in terms of computational efficiency.

The rest of the paper is organized as follows. Section 2 presents a review on related works and their issues. Section 3 introduces the formal definition of JDD. Section 4 describes the details of the JDD discovery method. Section 5 empirically evaluates our method with synthetic and real world datasets. Section 5 also analyses the computational efficiency and the sensitivity of our method to its parameter J. Section 6 draws the conclusion.

II. Related Works and Issues

The discord of a time series can be found by computing the pair-wise distances among all subsequences. That means $O(m^2)$ comparisons for a time series of length m. Various algorithms have been proposed to reduce the computational complexity.

Keogh and his co-authors proposed *HOTSAX* [1] for electrocardiogram application. They first proposed a naive method that compares every pair of subsequences with a two-layer nested for-loops. The outer loop considers each possible candidate subsequence, and the inner loop is a linear scan to identify the non-overlapping nearest neighbor of the candidates. To reduce the computational complexity, the authors applied heuristic sorting techniques in outer and inner loops according to the occurrences of each word. A conditional branch of Early abandoning in inner loop help the algorithm skip the calculations of the normal subsequences. HOTSAX achieves three order of magnitude of speed-up compares to brute force method. The most of the following improvements of discord discovery are based on the method of [1].

Feature extraction is a key technique to reduce the dimensionality of time series data and the computational complexity of discover method. Fu and co-authors [6] propose WAT (wavelet and augmented trie), which employs Haar wavelet transform and symbol word mapping techniques to approximate the perfect search order of all candidate subsequences. Li et al. [7] introduced a PAA bit representation that improves the computational efficiency of the discord discovery.

The computational efficiency of discord discovery depends on proper settings of parameters. Basha and co-authors [8] use statistical decision making and time series data mining methods to find the size of the best sub-sequence sliding window for which similarities and discords could be found efficiently. Fu and co-authors [6] proposed a word size free algorithm by first converting subsequences into Haar wavelets, then used a breadth first search to approximate the perfect search order for outer loop and inner loop. Luo and co-authors [9] proposed DDS (Direct discord search) to find the top-K discords in a periodic or quasi-periodic time series without prebuilding an index or tuning parameters. Jones et al. [10] proposed a method that automatically determine a threshold ϵ to avoid detecting meaningless patterns from time series. [11] extended [9] by introducing GDS (general direct search) that eliminates the quasi-periodicity assumption in direct search.

Index allows discord discovery methods to quickly retrieve similar sequences from database. In [1], prefix tree is adopted to lexicographically index the SAX representations of subsequences. Li et al. [7] proposed improved K-Medoids clustering algorithm, called as BitCluster for indexing purpose. BitCluster excludes some unqualified subsequences and improve the speed of discord discovery. Clustering can be an effective method of primary filtering for subsequences in time series.

Most discord discovery algorithms assume that data resides in the memory. To solve disk aware discord discovery in huge date sets, Yankov et al. [12] proposed a two-phase discord detection method. The first phase selects the candidates and the second phase identifies discords. The algorithm requires two linear scans over the disk. However it is worthy noting that the computational complexity of the method in [12] is still quadratically increases with the size of dataset.

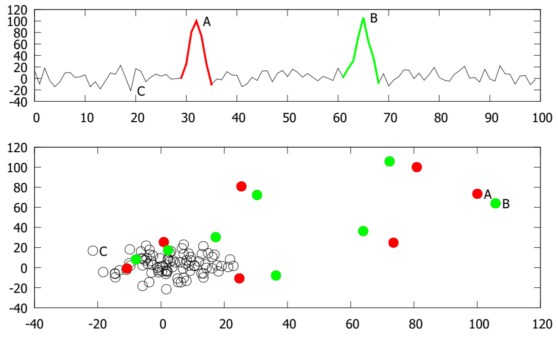

Fig. 2. A synthetic example of twin freak problem

A. Issues of Existing Methods

Existing works focus on faster discord discovery, but the quality of the results of these methods have rarely been discussed. Intuitively, an anomalous or interesting subsequence is maximally different to all the rest of subsequences of a longer time series. However, an anomalous or interesting pattern may happen two or more times such that any instance of this pattern is not different enough from the rest subsequences within the time series. The original definition of discord does not capture multiple occurrence of the actual interesting subsequences as the top discords rank list. In this case, normal or ordinary patterns have lower frequency of occurrences and therefore they are more likely to have higher rank in discords rank list. This problem is known as Twin Freak problem [3] in image related applications. We present a synthetic example to illustrate the problem of the original definition.

Fig.2 shows an synthetic example of time series discord discovery. The upper figure displays the complete time series, in which two coloured giant peaks are injected into a univariable Gaussian process. All subsequences of length two of the time series are plotted in the lower 2D figure. The subsequences overlapped with the two peaks are coloured correspondingly in the upper and lower figure.

Intuitively, subsequence A is a discord since it is quite different from most of the rest subsequences. However, according to the original definition of discord, A is not a discord because it has a similar neighbor B. Subsequence B is neither a discord because of the same reason. Moreover, subsequence C is intuitively normal but it has higher discord rank than A and B does, because C has larger distance to its nearest neighbor than A and B do.

As far as we know, none of the existing method for time series application takes the Twin Freak problem into consideration. We believe the main difficulty lies in the prohibitive computational complexity brought by searching J^{th} nearest neighbor of a subsequence. The complexity would be even higher in the situation where all sequences are extracted from a long run progress because we will inevitably engage with the overlapping problem, which will degrade the utility of time series related methods [13].

In this paper, we present a new definition of discord to mitigate the Twin Freak problem, as well as develop a new discovery method to reduce the computational requirements of discord discovery. We would like to start with the definition in Section III.

III. J-DISTANCE DISCORD

In this section we describe the concept, formal definitions and property of J-distance discord. Inspired by the idea of Local Outlier Factor (LOF) [14], we propose J-distance discord, which uses the distance of a subsequence to its J^{th} nearest neighbor as the criteria for discord discovery. We call the criteria *J-distance*. A normal or ordinary subsequence based on J-distance criteria has at least J neighbors close to it. A top discord based on J-distance criteria is the subsequence that has maximum J-distance among all subsequences of a time series.

A. Formal Definition

Before presenting the formal definition of J-distance Discord, we describe a set of preliminary notations. We use T to denote a *time series* $t_1, ..., t_m$, where m is the length of the time series, $t_i \in \mathbb{R}$. We denote a *subsequence* of a time series T as $C_{p,n} = t_p, ..., t_{p+n-1}$, where p is starting position, n is the length of the subsequence. Because subsequences of the same length are compared in discord discovery, we use C_p as an abbreviation of $C_{p,n}$ in the rest of the paper. Since every subsequence could be a discord, we will use a *sliding window*, whose size is equal to the length of subsequences, to extract all possible subsequences from T.

The original definition of the time series discord uses the distance of a subsequence to its nearest neighbor as an indicator of the degree of anomaly of a subsequence. The nearest neighbor of a subsequence C_p, denoted as $nn(C_p)$, is any subsequence C_o that has the smallest distance to C_p, where o is the starting position of C_o. The nearest neighbor of a subsequence C_p usually located one or two points to the left or the right of C_p. Such neighbors usually degrade the utility of time series related algorithms [13]. To avoid the issue, we exclude such neighors so that the J nearest neighbors conform to the following rules: (1) Any nearest neighbor C_o is not overlapped with C_p, that is $|o - p| > n$; (2) For any integer $i, j \in [1, J]$,$i \neq j$ the i^{th} nearest neighbor C_{o_i} has no overlapping region to the j^{th} nearest neighbor C_{o_j}. That is, $|o_i - o_j| > n$.

Having the preliminary notations, we begin to present the formal definitions. Discovering J-distance Discord requires finding the J^{th} nearest neighbor of all possible subsequences. We define the J^{th} nearest neighbor as:

Definition 1. *J^{th} nearest neighbor: For any integer $J > 1$, $1 \leq i < J$, suppose C_{o_i} is the i^{th} nearest neighbor of C_p. The J^{th} nearest neighbor of a subsequence C_p, denoted as $Jnn(C_p)$, is any subsequence C_o that has the smallest distance to C_p, with no overlapping region to C_p and i^{th} non-overlapping nearest neighbor, for all for $1 \leq i < J$. That is $|o - p| > n$, and $|o - o_i| > n$*

We are interested in the distance of a subsequence to its J^{th} nearest neighbor. We define the distance as:

Definition 2. *J-distance: For any positive integer J, the J-distance of subsequence C_p, denoted as J-$dist(C_p)$, is defined as the distance between C_p and $Jnn(C_p)$.*

Based on the definition of J-distance, we develop the definition of J-distance discord as follow:

Definition 3. *J-distance Discord (JDD): Given a time series T, the subsequence C_d of length n beginning at position d is said to be the discord of T, if C_d has the largest J-distance among all subsequences. That is, for $\forall$ subsequence C_o of T, $|d - o| > n$, $Jdist(C_d) \geq Jdist(C_o)$.*

We may be interested in examining the top K discords, which we define as:

Definition 4. *K^{th} J-distance Discord: Given a time series T, the subsequence C_d of length n beginning at position d is the K^{th} J-distance discord of T if C_d has the K^{th} largest J-distance, with no overlapping region to the i^{th} discord beginning at position d_i, for all $1 \leq i \leq K$. That is, $|d - d_i| \geq n$.*

The distance function of J-distance discord can be determined according to the application specific requirements. We will apply the widely used Euclidean distance measurement in the rest of this paper [13].

$$d(C_p, C_q) \equiv \sqrt{\sum_{i=0}^{n-1} (t_{p+i} - t_{q+i})^2}. \tag{1}$$

It is well known that it is meaningless to compare subsequences of different offset and amplitude [15]. We normalize each time series subsequence to have mean of zero and a standard deviation of one before calling the distance function.

B. Property of JDD

J-distance criteria allows our definition to be aware of multiple occurrence of similar discords, but introduces extra computational complexity. During the discovery of discord, our definition has some differences from the original definition:

1) J-distance criteria imposes $J - 1$ more calls to the distance function during the search of J nearest neighbors for each subsequence. Excluding the overlapping cases from the nearest neighbors list imposes additional calls to the distance function.
2) The NN distance of a subsequence C_p is greater or equal to the NN distance of C_p's nearest neighbor. Once a subsequence C_p is identified as a normal pattern, its nearest neighbor must also be a normal pattern. For a J-distance discord discovery, we cannot make the same assertion on C_p's J nearest neighbors based on the property of C_p.

These properties suggest that existing discord discovery methods are of little utility for finding good quality discords.

```
1   Function [$Jdist$, $pos$] = JDD discovery($T$,$n$,$J$)
2    $bsf\_Jdist = 0$      // best so far J-distance
3    $bsf\_pos = NaN$      // best so far beginning position
4   For Each $p$ in $T$ ordered by Outer
5       $Jdist^*(C_p) = infinity$
6       For Each $q$ in $T$ ordered by Inner
7           If $|p - q| < n$
8               continue
9           End
10          Update $Jdist^*(C_p)$ with $d(C_p, C_q)$
11          If $Jdist^*(C_p) < bsf\_Jdist$
12              Break
13          End
14      End
15      If $Jdist^*(C_p) > bsf\_Jdist$
16          $bsf\_Jdist = Jdist^*(C_p)$
17          $bsf\_pos = p$
18      End
19  End
20  Return[$bsf\_Jdist$, $bsf\_pos$]
```

TABLE I
J-DISTANCE DISCORD DISCOVERY

This motivates the proposal of an extended algorithm in the next section.

IV. Finding J-distance discords

Firstly, we develop a naive version of our JDD discovery method by refining the method in [1]. Table I shows the base method for JDD discovery.

T is the complete time series. n is the length of window. J is the parameter we introduce to calculate J-distance. $Jdist$ and pos is the J-distance and beginning position of the discord. $Jdist^*(C_p)$ is the distance of C_p to its neighbor that is J^{th} nearest so far has been searched in the inner loop. To calculate $Jdist^*(C_p)$ (Line 10), we have to complete: (1) Keep a list of the position and distance information of the nearest neighbors for the current candidate; (2) When new position and distance information are added into the list, find the top J non-overlapping nearest neighbors from the list; (3) Discard the neighbor C_o that has the distance $d(C_p, C_o)$ that is greater than $Jdist^*(C_p)$.

In the inner loop, we do not actually need to find the true J nearest neighbors of the current candidate. As soon as we find any J subsequences that are closer to the current candidate than the bsf_Jdist, we can abandon the current candidate of the outer loop (lines 11-13), safe in the knowledge that the current candidate could not be the J-distance discord. The utility of the above optimization depends on the order which the outer loop considers the candidates for the discord, and the order which the inner loop visits the other subsequences in its attempt to find J subsequences that will allow an early abandon of the inner loop. We adopt the heuristic ordering [1], which is denoted as *Outer* and *Inner* in Line 4 and 6.

A. Enhanced Early Abandon Technique

Since the J-distance criteria introduces extra computational complexity, we develop enhanced early abandon technique to mitigate the stress of extra computations. We observed that

```
1   Function [$Jdist$, $pos$] = JDD discovery($T$,$n$,$J$)
2    $bsf\_Jdist = 0$      // best so far J-distance
3    $bsf\_pos = NaN$      // best so far beginning position
4    For $\forall C_p$, $Jdist^*(C_p) = infinity$
5   For Each $p$ in $T$ ordered by Outer
6       If $C_p$ marked as normal
7           continue
8       End
9       For Each $q$ in $T$ ordered by Inner
10          If $|p - q| < n$
11              continue
12          End
13          Update $Jdist^*(C_p)$ with $d(C_p, C_q)$
14          If $Jdist^*(C_p) \times 2 < bsf\_Jdist$
15              Mark $N_J(C_p)$ as normal
16          End
17          Update $Jdist^*(C_q)$ with $d(C_p, C_q)$
18          If $Jdist^*(C_q) \times 2 < bsf\_Jdist$
19              Mark $N_J(C_q)$ as normal
20          End
21          If $Jdist^*(C_p) < bsf\_Jdist$
22              Break
23          End
24      End
25      If $Jdist^*(C_p) > bsf\_Jdist$
26          $bsf\_Jdist = Jdist^*(C_p)$
27          $bsf\_pos = p$
28      End
29  End
30  Return[$bsf\_Jdist$, $bsf\_pos$]
```

TABLE II
ENHANCED JDD DISCOVERY METHOD

A subsequence that is similar to a normal subsequence is likely to be a normal subsequence. This ideas can be translated into the follow rule: Given a subsequence C_p and its non-overlapping J nearest neighbor C_q, If $J = 1$, we have $Jdist(C_q) \leq Jdist(C_p)$. However, this feature does not hold when $J > 1$. We extend the rule so that it can be applied to our JDD discovery:

Theorem 1. *Given a subsequence C_p and its non-overlapping J^{th} nearest neighbor C_q, for any positive integer J, it holds $Jdist(C_q) \leq 2 \times Jdist(C_p)$.*

Due to the page limitation, we skip the proof of the theorem. Having developed the boundary of $Jdist(C_p)$, we improve the naive version of the JDD discovery method in Table I with Theorem 1.

The improved method is shown in Table II. During examining subsequences in the inner loop, we are looking for a candidate subsequence C_p that has $Jdist^*(C_q)$ smaller than $0.5 \times bsf\_Jdist$ (Line 14). Once we find it, we know according to Theorem 1 that the current J nearest neighbors of the candidate could not be the J-distance discord. Therefore it is safe to mark J nearest neighbors as normal (Line 15). Here we use $N_J(C_p)$ to represent the J nearest neighbors of C_p so far have being searching in the inner loop. In the outer loop, Lines 6-8 check the current candidate and skip it if the it has been marked as normal by Line 15.

B. Reusing Intermediate Results

We observe that storing the intermediate results of the embedded loop can also saves the computations in the following ways: (1) The distance between a candidate C_p and an instance C_q is not only helpful to calculate $Jdist(C_p)$, but can also be reused to calculate $Jdist(C_q)$; (2) The $Jdist^*(C_p)$ of C_p during the discovery of the K^{th} discord could also be useful to reduce the computational requirement for the discovery of the $(K+1)^{th}$ discord. Therefore we apply memoization technique [16] to further reduce the extra computational complexity brought by J nearest neighbor search.

We maintain a data structure to store intermediate results for finding $Jdist^*(C_p)$ of all subsequences. After calculating the distance between C_p and C_q, we update both $Jdist^*(C_p)$ and $Jdist^*(C_q)$ with $d(C_p, C_q)$, as shown in Line 13 and 17 of Table II. The update process will discard the distance results that larger than $Jdist^*(C_p)$ to save the memory requirement. Then we apply the enhanced early abandon technique described in previous section on C_p as well as C_q to discard normal subsequence in early time. (Lines 14-16 and Lines 18-20)

C. Finding Top K J-distance Discords

In order to find the top-K discords, we repeat similar steps to find the first discord, 2^{nd} discord, ..., K^{th} discord. During the discovery of k^{th} discord, for $1 \leq k \leq K-1$, we record and update the $Jdist^*(C_p)$ of each candidate and reuse these information in the discovery of $(k+1)^{th}$ discord. $Jdist^*(C_p)$ of each candidate are kept being updated during the discord discovery, they will continuously approach the true J-distance and therefore help the discovery method skip candidate in outer loop and early abandon the inner loop for discovery of each discord.

V. Empirical Evaluation

In this section, we empirically evaluate JDD and the discovery method with one synthetic and three real world datasets. We compare the JDD with HOTSAX in two perspectives: quality of discord rank list and the computational efficiency. We also evaluate the sensitivity of JDD to the size of sliding window and the parameter J. Unless specified, we use parameter $J = 3$ by default in our experiments. We choose HOTSAX as the baseline method For better comparability.

A. Anomaly Detection in Synthetic dataset

We begin the evaluation with a synthetic dataset. The upper part of the Fig.3 shows the synthetic dataset. It is a time series consists of non-stationary sine wave, whose period is 60. We manually inject three anomalies of length 20 at position 70, 133 and 761 of the time series and plot them with red thick lines. These anomalies cut-off three peaks of the sine wave and look similar in shape.

We perform the evaluation by applying JDD and HOTSAX method on the synthetic dataset. Generally, window length that equal to or similar to the period of the sine wave would a good choice for discord discovery. However in some applications we do not know the proper window length. We use window

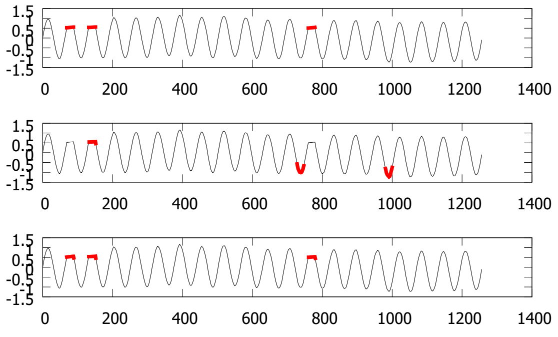

Fig. 3. A Synthetic dataset and the results of HOTSAX and JDD

	JDD	HOTSAX
Window length 20	3 (133,761,70)	1 (133,981,728)
Window length 40	3 (121,60,750)	2 (813,148,59)
Window length 60	3 (111,740,50)	3 (81,771,142)
Window length 80	3 (126,714,21)	3 (78,690,769)
Window length 100	3 (61,688,905)	3 (61,689,281)

TABLE III
THE RESULT ON SYNTHETIC DATASET WITH DIFFERENT WINDOW

length ranging from 20 to 100 with interval of 20 during the discovery. We find top three discords by using JDD and HOTSAX method respectively.

The middle and the lower part of Fig.3 respectively shows the top three discords discovered by HOTSAX and our JDD method. The discovered discords are plotted with red thick lines. For simplicity, we only plot the results with respect to the window length of 20 in the figures. As we can see from the middle part of Fig.3, HOTSAX only captures one of the three discords. The lower part of Fig.3 shows that JDD identifies all the three discords. Therefore JDD outperforms the original definition of time series discord on the synthetic dataset when window length is set to 20.

Table III lists the discovery results of different window length. It shows the number of discords being captured and the beginning positions of the top three discords. JDD captures all discords with under different settings of window length. HOTSAX fails to capture all discords when the window length is smaller than the period of the sine wave. In case the window length is 40, HOTSAX mistakenly identifies a normal subsequence starting from 813 as the top 1st discord, leading to a further degradation of the quality of the discovered results. This table indicates that JDD is less sensitive to the window length than the original definition of discord.

B. Arrhythmia Detection

Electrocardiogram (ECG) is a time series describing the electrical activity of beating heart. Keogh et al. [1] have already demonstrated the utility of discords in ECG. The ECG datasets being evaluated in [1] include limited number of anomalies. Meanwhile these anomalies are quite different in shapes, allowing the original method to identify easily.

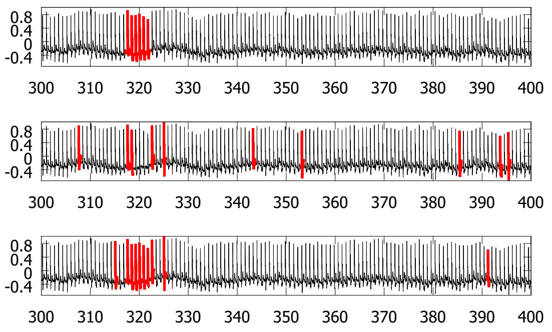

Fig. 4. An ECG time series and the results of HOTSAX and JDD

However, it is remarkable that similar anomalous heart beat may happen multiple times on one patient.

For example, the upper part of Fig.4 shows a ECG signal extracted from *sel102* dataset in MIT-BIH Arrhythmia Database [5]. The x axis represents the time in seconds while the y axis represents the voltage in mV. The ploted signal contains continuously occurrences of heart beat anomalies. The six anomalous heart beats plotted with red thick lines are manually annotated by cardiologists. We set the window length to 300 because, on average, approximately 300 points are sampled for each heart beat in this ECG graph. We use HOTSAX and our JDD method respectively to find top ten discords.

The middle and the lower part of Fig.4 shows the results of HOTSAX and our JDD method respectively. As shown in the middle part of the figure, HOTSAX identifies two anomalies on this dataset. The rank of two identified anomalies is number seven and number nine, which means the top six discords of HOTSAX does not capture any actual discord. As the anomalies happen continuously, we also consider the quality of the discords from the point view of change detection, which is another utility of discord discovery proposed by [1]. HOTSAX captures the entrance from normal status into anomalous status, but fails to capture the exit from the anomalous status back into normal status.

The lower part of Fig.4 shows the result of our JDD method. Our JDD method identifies all anomalies at the top six positions of the ranking list. Three of the rest irrelevant discord surround the anomalies, indicating the beginning and ending of the anomalies.

C. Heartbleed Detection

We also demonstrate the utility of JDD in applications other than healthcare. The Heartbleed vulnerability, which drew public attention in 2014, is a bug in the popular OpenSSL cryptographic software library, The vulnerability potentially allowed attackers to access confidential information from remote HTTPS server without being authorized [17]. One way to increase the possibility of getting confidential information is to repeatedly send malformed heartbeat requests to the remote server.

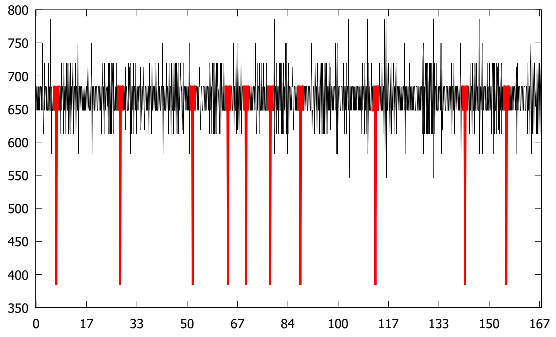

Fig. 5. Heartbleed dataset

	JDD	HOTSAX
Window length 10	1,2,4,9	1,2
Window length 12	1,2,3,5,8,10	1,2,3,7
Window length 14	1,2,3,5,7,8	1,2,3
Window length 16	1,2,3,4,5,8	1,2,3,4,8

TABLE IV
HEARTBLEED DETECTION WITH DIFFERENT WINDOW LENGTH

In this experiment we construct the situation of repeated heartbleed attacks and detect the attacks with JDD. We set up a web sever and established 20 TLS links from guest machine to the web server. We issue 100 malformed heartbeat requests, ten at a time. Each request only requires the server to send back 30 extra characters each time. We count the difference between the packet size of heartbeat requests and replies every five minutes. We apply JDD and HOTSAX method on the record of difference to detection the exploitation.

Fig.5 shows the time series of the difference counts of packet size between total heartbeat requests and replies. Each unit of x axis represents a five-minute period. Y axis represents the difference count. Each spike plotted with red thick lines represents a ten-times heartbleed attack. We used JDD and HOTSAX respectively to discover top ten discords. The discovery results are shown in Table IV.

Table IV shows the rank of attacks being identified by the two methods. We use different window length to discover attacks. As we can see from the table, neither of the methods identifies all attacks. But given any of these window length, our JDD method always captures more attacks and assigns the attacks with higher rank than original method does.

D. Human Activity Change Detection

Human activity recognition is an important technology in pervasive computing [18], where sensors are mounted on human bodies and generate various types of information for activity recognition. In this experiment, we perform change detection on a human activity dataset [19]. The data were collected from a wearable accelerometer mounted on the chest of the number one participant. The accelerometer recorded the accelerations in x,y and z directions at sampling frequency of 52 Hz. We calculate $g = \sqrt{x^2 + y^2 + z^2}$ as the time series for change detection. The time series of g is plotted in Fig.6.

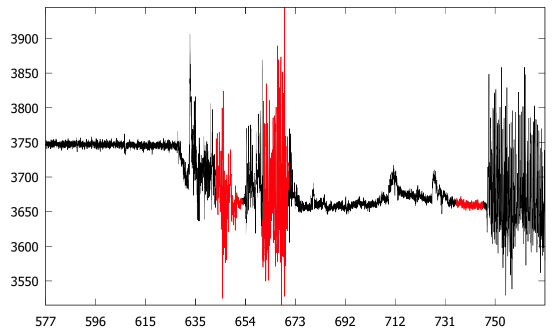

Fig. 6. Human activity dataset. Courtesy of Casale et al. [19].

	JDD	HOTSAX
Window length 5s	1,7	1,8
Window length 7.5s	1,5	1,6
Window length 10s	1,4,10	1,5

TABLE V
HUMAN ACTIVITY CHANGE DETECTION WITH DIFFERENT WINDOW

Fig.6 shows the time series of g acceleration measured by the accelerometer mounted on the chest of participant number one. X axis stands for the time in second. Y axis stands for uncalibrated g acceleration. There are four human activities in the figure. They are, in order, working at computer, going up/down stairs, standing and walking. Activity changes are ploted in red. We use both methods to discover top ten discords respectively. The discovered results are shown in Table V.

Table V shows the ranks of activity changes being identified by the two methods. We use different window length, which is represented in second, to detect activity change. As we can see from the table, given $window\ length = 10s$, JDD captures all activity changes. Besides, given any of these window length, JDD always assigns the captured activity changes with higher rank than HOTSAX does.

E. Computational Efficiency

We evaluate the computational efficiency of our JDD discovery method. We perform the evaluation by measuring the total number of calls to the distance function in Line 13 in Table II because it is a reproducible indicator for the computational efficiency of our method.

We apply JDD and HOTSAX method onto four representative real world datasets and discover top three discords for each dataset respectively. The four datasets are all distributed by [1]. The size of sliding window is set to $n = 100$.

Fig.7 shows the results of the experiments. X axis stands for the size of datasets, Y axis stands for the total number of calls to the distance function. Through the figure we observe that the curves of the JDD discovery method are very close to those of HOTSAX, indicating that our JDD discovery method shows computational efficiency similar to that of HOTSAX.

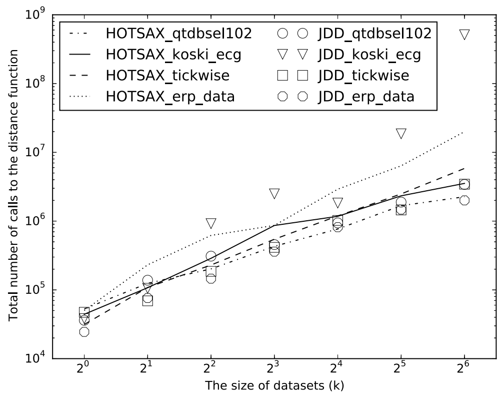

Fig. 7. The computational efficiency of HOTSAX and JDD discovery method on four representative datasets [1].

F. Sensitivity of JDD to Parameter J

We have already demonstrated the utility of JDD with $J = 3$ in previous experiments. We will show that $J = 3$ is not a fine-tuned result of our experiments. We evaluate the sensitivity of our JDD discovery method to the parameter J with respect to the computational efficiency. In this experiment, we use our JDD method to discover top ten discords from 8k subsets of the four real world datasets under the parameter J ranging from 2 to 8. The size of the sliding window is set to $n = 100$.

The bar chart in Fig.8 shows the total number of calls to the distance function of our JDD method with different settings of the parameter J. The bars for qtdbsel102, koski_ecg and erp_data slightly lengthen with the increase of J, as we expected. However the bar for tickwise shortens with the increase of J. After a deep investigation into the dataset tickwise, we confirm that a small cluster of subsequences are similar to each other, but are unusual to most of the subsequences in the dataset. JDD with larger J identifies the subsequences in the cluster as discords and therefore skips other candidates more quickly. In conclusion, the computational efficiency of our JDD method does not dramatically drop with the increase of the parameter J.

We also empirically evaluate the sensitivity of JDD with respect to the parameter J in terms of the consistency of the results. To present the evaluation, we first define the following measurement. We define $\mathbb{U}$ as an integer set of all settings of the parameter J, and define $\mathbb{V}$ as an integer set of all ranking number of discords. In the case of this experiment, $\mathbb{U} = [2, 8]$, $\mathbb{V} = [1, 10]$. The total numbers of elements in $\mathbb{U}$ and $\mathbb{V}$ are $|\mathbb{U}| = 7$ and $|\mathbb{V}| = 10$ respectively. For any integer $j \in \mathbb{U}$, $k \in \mathbb{V}$, we use D^j to represent the ten discords discoverd by our JDD method with the parameter $J = j$, and use d_k^j to represent the k^{th} discord in D^j. For any integer $i \in \mathbb{U}$, if d_k^j overlaps with any discord in D^i, denoted as $Ovlp(d_k^j, D^i) = 1$, we say that d_k^j is captured by JDD with the parameter $J = i$. Otherwise $Ovlp(d_k^j, D^i) = 0$. Then the percentage of a discord d_k^j being captured by JDD with different parameter

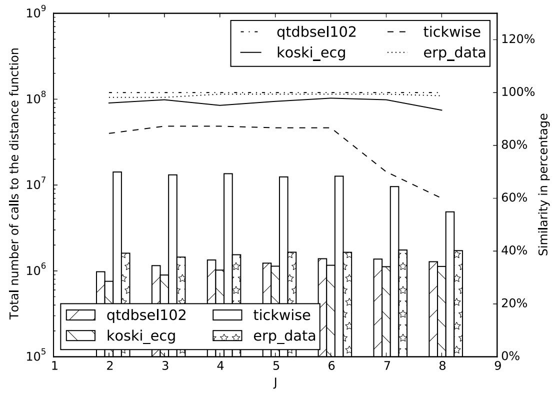

Fig. 8. Bars: number of calls to the distance function versus parameter J. Lines: consistency of the results under different parameter J.

J, denoted as $P(d_k^j)$, can be represented as:

$$P(d_k^j) = \frac{\sum_{h \in \mathbb{U}, i \neq j} Ovlp(d_k^j, D^i)}{|\mathbb{U}| - 1}. \quad (2)$$

To measure the similarity of D^j to all other results $D^i, i \in \mathbb{U}, i \neq j$, denoted as $S(D^j)$, we average the percentages of the discords in D^j. We take the ranking of discords into consideration and assign weight to each percentage, that is:

$$S(D^j) = \frac{\sum_{k \in \mathbb{V}} (|\mathbb{V}| - k + 1) * P(d_k^j)}{\sum_{k \in \mathbb{V}} k}. \quad (3)$$

According to the definitions, $S(D^j) \in [0\%, 100\%]$. It indicates the similarity of D^j to the results with different settings of the parameter J. That is, if D^j includes a list of discords that are mostly being captured by JDD with all other J, $S(D^j)$ would be close to 100%. This measurement roughly indicates the consistency of the results with different J.

In Fig.8, we plot lines using Eq.(3) over $j \in [2, 8]$ for the four real world datasets. The lines for qtdbsel102, koski_ecg and erp_data are close to 100% and do not change dramatically with different parameter J. The line for tickwise is over 80% when J is smaller than 6 and drops to 60% when J increases to 8. After a deep investigation into the dataset tickwise, we confirm that JDD with larger J identifies the small cluster of subsequences as discords, which is not fully captured by JDD with smaller J. This result indicates that JDD has the ability to capture small clusters of discords, which cannot be effectively captured by the original definition of discord.

VI. Conclusion

To tackle *Twin Freak* like issues in time series, we presented an improved definition, named J-distance discords (JDD), for time series data mining. We also proposed a discovery method for JDD to reduce the additional computation requirements raised by the definition of JDD. Experiments show that JDD captures more de-facto anomalous and interesting patterns compared to the original definition of discord. Besides, Our JDD discovery method is as fast as the classic discord discovery method, i.e. HOTSAX.

Acknowledgment

This paper is sponsored by the National Natural Science Foundation of China (No.61373032), the National Research Foundation Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) program and the National High Technology and Research Development Program of China (863 Program, 2015AA050204).

References

[1] E. Keogh, J. Lin, and A. Fu, “Hot sax: Efficiently finding the most unusual time series subsequence,” in *fifth IEEE international conference on Data Mining (ICDM)*. IEEE, 2005, pp. 8–pp.

[2] T.-c. Fu, “A review on time series data mining,” *Engineering Applications of Artificial Intelligence*, vol. 24, no. 1, pp. 164–181, 2011.

[3] L. Wei, E. J. Keogh, and X. Xi, “Saxually explicit images: Finding unusual shapes.” in *ICDM*, vol. 6, 2006, pp. 711–720.

[4] C. Miller, Z. Nagy, and A. Schlueter, “Automated daily pattern filtering of measured building performance data,” *Automation in Construction*, vol. 49, pp. 1–17, 2015.

[5] A. L. Goldberger, L. A. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley, “Physiobank, physiotoolkit, and physionet components of a new research resource for complex physiologic signals,” *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000.

[6] A. W.-C. Fu, O. T.-W. Leung, E. Keogh, and J. Lin, “Finding time series discords based on haar transform,” in *Advanced Data Mining and Applications*. Springer, 2006, pp. 31–41.

[7] G. Li, O. Bräysy, L. Jiang, Z. Wu, and Y. Wang, “Finding time series discord based on bit representation clustering,” *Knowledge-Based Systems*, vol. 54, pp. 243–254, 2013.

[8] R. Basha and J. Ameen, “Unusual sub-sequence identifications in time series with periodicity,” *International Journal of Innovative Computing, Information and Control*, vol. 3, no. 2, pp. 471–480, 2007.

[9] W. Luo and M. Gallagher, “Faster and parameter-free discord search in quasi-periodic time series,” in *Advances in Knowledge Discovery and Data Mining*. Springer, 2011, pp. 135–148.

[10] M. Jones, D. Nikovski, M. Imamura, and T. Hirata, “Anomaly detection in real-valued multidimensional time series,” 2014.

[11] W. Luo, M. Gallagher, and J. Wiles, “Parameter-free search of time-series discord,” *Journal of computer science and technology*, vol. 28, no. 2, pp. 300–310, 2013.

[12] D. Y. E. Keogh and U. Rebbapragada, “Disk aware discord discovery: Finding unusual time series in terabyte sized datasets,” in *in Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2007, pp. 381–390.

[13] B. Chiu, E. Keogh, and S. Lonardi, “Probabilistic discovery of time series motifs,” in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 493–498.

[14] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, “Lof: identifying density-based local outliers,” in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 93–104.

[15] E. Keogh and S. Kasetty, “On the need for time series data mining benchmarks: a survey and empirical demonstration,” *Data Mining and knowledge discovery*, vol. 7, no. 4, pp. 349–371, 2003.

[16] D. Michie, “Memo functions and machine learning,” *Nature*, vol. 218, no. 5136, pp. 19–22, 1968.

[17] Z. Durumeric, J. Kasten, D. Adrian, J. A. Halderman, M. Bailey, F. Li, N. Weaver, J. Amann, J. Beekman, M. Payer *et al.*, “The matter of heartbleed,” in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 475–488.

[18] Y.-S. Tak, J. Kim, and E. Hwang, “Hierarchical querying scheme of human motions for smart home environment,” *Engineering Applications of Artificial Intelligence*, vol. 25, no. 7, pp. 1301–1312, 2012.

[19] P. Casale, O. Pujol, and P. Radeva, “Personalization and user verification in wearable systems using biometric walking patterns,” *Personal and Ubiquitous Computing*, vol. 16, no. 5, pp. 563–580, 2012.