ICIC Express Letters
Volume **12**, Number **10**, October **2018**
 ISSN 1881-803X
pp. **1071–1078**

AN INFORMATION THEORY-BASED APPROACH FOR TIME SERIES DISCORD DISCOVERY

Thuc-Doan Do[1,2], Hong-Van T. Hoang[1,2], Cam Huynh[3]
and Thuy-Van T. Duong[1,2,*]

[1]Center for Applied Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam

[2]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam
{ dothucdoan; hoangthihongvan }@tdt.edu.vn; *Corresponding author: duongthithuyvan@tdt.edu.vn

[3]Faculty of Information Technology, Post and Telecommunication Institute of Technology
Ho Chi Minh City, Vietnam

Received March 2018; accepted May 2018

Abstract. *Time series discord is a subsequence which is the most different from the remaining subsequences of a time series. In this paper, we propose a novel approach based on information theory to discover time series discord. Many previous papers rely on prebuilding an index for subsequences to make heuristic for pruning away unnecessary searches. However, building such data structures has a significant effect on running performance and memory cost. In our approach, using information theory rather than data structure reduces the preprocessing time and space to make heuristic, and helps our method run faster with less space. Experimental results on some time series datasets demonstrate the effectiveness and efficiency of our proposed algorithm compared to HOT-SAX while the discords discovered by two methods completely match each other.*
Keywords: Discord discovery, Time series, Outlier detection, Information theory, Entropy

1. **Introduction.** Time series discord is defined as a subsequence of a longer time series that is maximally different to other subsequences. Time series discord discovery plays an important role and has proven to be useful for data mining such as data cleaning, data description, clustering. It captures the idea of anomalous subsequences in time series but takes only one intuitive parameter – the length of the subsequence while most anomaly detection algorithms require many parameters.

A simple way to find the discord is the brute force algorithm. It requires comparisons among $O(m^2)$ pair-wise distances, where m is the length of the time series. Therefore, it is not applicable in practice. To search for discords more efficiently, most efforts focus on building heuristics for two nested loops in the brute force. They are based on data structures to reorder the subsequences with the hope that it is possible to exit quickly from two nested loops. It takes time and space to build such data structures, some of which often have unintuitive parameters (e.g., word length and alphabet size) to tune. There is a trade-off between effectiveness and efficiency in this case. That is the reason why evaluating the performance of the algorithm based on only the number of calls to distance function is not exact. In this paper, we propose a novel approach to discovering time series discord, which is information theory-based to reduce the time and space compared to the above-mentioned methods. The main contributions of our work are the following.

- Propose a new solution that reduces the preprocessing time and space so that it can work efficiently in the whole process of discord discovery.
- Do the experiments on the various data sets to show the efficiency of our algorithm.

DOI: 10.24507/icicel.12.10.1071

- Compare the results to HOT SAX and discuss obtained results.

The rest of the paper is organized as follows. In Section 2, related works on outlier detection and time series discord discovery are presented. Then, we give some essential definitions, present generic framework to solve discord discovery and propose our solution in Section 3. Section 4 presents the experiments, results obtained and discussion. Finally, Section 5 draws the conclusion and future works.

2. Related Works.

2.1. Outlier detection.

Outlier is defined in [1] as an observation that deviates so much from other observations as to arouse suspicion that it is generated from a different mechanism. In [2], approaches for outlier detection consist of seven groups: classification-based, clustering-based, nearest neighbor-based, statistical, information theory-based, spectral decomposition-based, visualization-based.

Among these approaches, information theory-based approach has a solid foundation as it is based on mathematics and yields promising results. This method uses information theoretic measures, such as entropy and relative uncertainty to compute the disorder of a dataset after removing outliers as it assumes that outliers cause irregularities in the data and affect information consistency. In some papers, the problem of outlier detection is considered as an optimization problem and solved by a local search heuristic-based algorithm [3] and a greedy algorithm [4] to minimize the objective function. In [5], outlier detection is also considered as an optimization problem, in which the efficiency is improved by combining the correlation among attributes and the importance of each attribute. In [6], the authors proposed an algorithm based on entropy using a weighted density definition for categorical data and showed the superiority of that algorithm compared with the existing outlier detection methods.

2.2. Time series discord discovery.

When finding outliers within a time series, outliers can be points or subsequences. When it comes to the later one, the most famous problem is discord discovery, which was firstly defined in [7] and has confirmed the utility in many fields [8,9]. While few papers find discords approximately like [10] with the idea that subsequences in a smaller sized cluster have the higher probability to be a discord, most studies find exact discords by building heuristics for two nested loops in the brute force.

The authors in [7] suggested an algorithm called HOT SAX with a heuristic technique for pruning quickly the data space and focusing only on the potential discords. They use two data structures: an array of SAX word and a trie to support in approximating the magic outer and inner loops. In [11], WAT algorithm is based on Haar wavelet and augmented trie to mine the top-K discords from a time series database. It provides better pruning power and can dynamically determine the word size. Vector quantization, a time series discretization technique, is used to reduce dimensions and discretize time series data. In [12], the authors create two data structures: a table of VQ representations for subsequences of the input data and an augmented trie to lookup occurrences of VQ representations.

3. Problem and Solution.

3.1. Problem definition.

Definition 3.1. Time Series Discord*: Given a time series T, the subsequence D of length n beginning at position l is said to be the discord of T if D has the largest distance to its nearest non-self match, that is, $\forall$ subsequence C of T, non-self match MD of D, and non-self match MC of C,* $\min(Dist(D, MD)) > \min(Dist(C, MC))$.

Here, $Dist(Q, C)$ is a distance function to measure distance from subsequence Q to subsequence C of the same length. In our paper, we use the Euclidean distance function as a Dist due to its simplicity and effectiveness.

Definition 3.2. Euclidean Distance*: Given two time series Q and C of length n, the Euclidean distance between them is defined as:*

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

Definition 3.3. 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 q, we say that M is a non-self match to C at distance of $Dist(M, C)$ if $|p - q| \geq n$.*

Brute force algorithm is a simple and easy way to finding discord. We just run two nested loops, in which the outer loop considers each possible candidate subsequence, and the inner loop is a linear scan to identify the candidate's nearest non-self match. This algorithm can find the exact discord with only one parameter, the length of the discord. However, it is impractical for large datasets due to its time complexity $O(m^2)$.

Motivated by reducing the time complexity, Keogh et al. [7] suggested a generic framework based on two heuristics, one to determine the order in which the outer loop visits each subsequence, and one to determine the order in which the inner loop visits the other subsequences. This framework is called the heuristic discord discovery and illustrated in Table 1.

Now, discord discovery problem has been transformed to determine good heuristic orderings with as little time as possible. However, the time for finding out good heuristic orderings is a part of the time complexity of algorithm. Therefore, if we spend much time on this process, the algorithm is not an efficient solution for the discord discovery.

TABLE 1. Heuristic discord discovery [7]

```
1  Function [dist, loc]= Heuristics_Search (T, n, Outer, Inner)
2    best_so_far_dist = 0
3    best_so_far_loc = NaN
4  For Each p in T ordered by heuristic Outer          //Begin Outer Loop
5       nearest_neighbor_dist = infinity
6       For Each q in T ordered by heuristic Inner     //Begin Inner Loop
7       IF |p − q| ≥ n                                  //non-self match?
8            IF Dist(t_p, ..., t_{p+n−1}, t_q, ..., t_{q+n−1}) < best_so_far_dist
9                 Break                                //Break out of Inner Loop
10           End
11           IF Dist(t_p, ..., t_{p+n−1}, t_q, ..., t_{q+n−1}) < nearest_neighbor_dist
12                nearest_neighbor_dist = Dist(t_p, ..., t_{p+n−1}, t_q, ..., t_{q+n−1})
13           End
14      End                                             //End non-self match test
15   End                                                //End Inner Loop
16      IF nearest_neighbor_dist > best_so_far_dist
17           best_so_far_dist = nearest_neighbor_dist
18           best_so_far_loc = p
19      End
20   End                                                //End Outer Loop
21   Return[ best_so_far_dist, best_so_far_loc ]
```

3.2. **Solution.** Our solution follows the generic framework but we make an approximate magic ordering of heuristic discord discovery based on weighted density, an information measure defined in [6] instead of using some data structures like in the literature. This helps to reduce time for both preprocessing and two nested loops. The proposed algorithm is called IDD (Information theory based Discord Discovery) and presented in Table 2.

TABLE 2. Our proposed algorithm – information theory based discord discovery

Input: A time series with length of m
Output: 1st discord length of n

Step 1: All subsequences are extracted from the original time series by sliding window length n and converted to SAX words with length of w.
Step 2: Apply Formulas (1), (2), (3), (4) and (5) to computing the weighted density for each transformed subsequence.

```
For i = 1 to w do
    Compute U/IND(C^i) according to (1)
    Compute E(C^i) according to (2)
    Compute W(C^i) according to (3)
For i = 1 to m − n + 1 do
    For j = 1 to w do
        Compute Den_Cj(x_i) according to (4)
    Compute WDen(x_i) according to (5)
```

Step 3: Based on the weighted density of each subsequence that we have computed, we will make heuristics for Outer and Inner loops and run the heuristic discord discovery in **Table 1**.

As the method suggested in [6] is for categorical data, we have to convert the original time series to categorical data using Symbolic Aggregate approXimation (SAX) which is presented by Lin et al. [13].

Firstly, all subsequences are extracted from the original time series by a sliding window of length n (length of discord) across time series T (length m) and converted to SAX words. Each subsequence is considered as an object with set of categorical attribute values being its characters. As a result, the original time series is transferred to categorical data and we can map our problem to the method in [6] as follows.

U is the set of all subsequences, which are presented in SAX word with length w.

A is a set of attributes in which C^k is the kth character in each subsequence.

The binary relation on the character C^k between subsequences:

$$IND\left(C^k\right) = \left\{(x, y) \in U \times U | x \cdot C^k = y \cdot C^k\right\}$$

The equivalence class on the character C^k determined by subsequence x:

$$[x]_{C^k} = \left\{y \in U | (x, y) \in IND\left(C^k\right)\right\}$$

The partition of U on the character C^k:

$$U/IND\left(C^k\right) = \left\{[x]_{C^k} | x \in U\right\} \tag{1}$$

Assume that $U/IND(C^k) = \{X_1, X_2, \ldots, X_p\}$.

The complementary entropy of the character C^k:

$$E(C^k) = \sum_{i=1}^{p} \frac{|X_i|}{|U|}\left(1 - \frac{|X_i|}{|U|}\right) \tag{2}$$

in which $|X|$ represents the number of items in X.

The weight of the character C^k:

$$W\left(C^k\right) = \frac{1 - E\left(C^k\right)}{\sum_{i=1}^{w}(1 - E(C^i))} \tag{3}$$

The density of the subsequence x with respect to the character C^k, given by:

$$Den_{C^k}(x) = \frac{|[x]_{C^k}|}{|U|} \tag{4}$$

The weighted density for subsequence x:

$$WDen(x) = \sum_{i=1}^{w} Den_{C^i}(x) \cdot W(C^i) \tag{5}$$

In our method, we impose the weighted density with the assumption that the lower the frequency of each attribute value is, the more likely the object is a discord. Furthermore, an object whose each attribute value is extremely irregular is an absolutely discord.

Outer Loop Heuristic: After computing the weighted density for each subsequence, we can reorder the candidate subsequences using following heuristic. **Heuristic**: *The subsequences, which have the lowest weighted density, are considered first in the outer loop. After that, the rest are visited in random order.*

The simple intuition behind our heuristic is that the lower the weighted density value of the subsequence is, the higher possibility it is a discord.

Inner Loop Heuristic: When the subsequence s_i is considered in the outer loop, we reorder the inner loop search order by inner loop heuristic. **Heuristic**: *We find subsequences, which have the same weighted density value with s_i to consider them first in the inner loop. After that, the unsearched subsequences are visited in a random order.*

The reason behind our inner heuristic is as follows. As two subsequences that have the same weighted density value are very likely to be similar, the distance between them is minimal and they should be chosen first in the inner loop.

In our method, the transformed subsequences are only used for heuristic outer loop and heuristic inner loop. For the distance calculation, we will use Euclidean distance between the original subsequences to find the exact discord.

4. Experiments and Results.

For an experiment evaluation, we implemented our proposed algorithm – IDD and HOT SAX algorithm in [7] using Matlab R2016a programming language. All experiments were conducted on a notebook with an Intel (R) 2.53GHz Intel® Core ™ i3 Processor, 4G RAM, 64-bit Microsoft Windows 7 operating system, Home Premium PC. The results were recorded to compare the cost of running time and the number of Euclidean distance function calls of both algorithms.

4.1. Dataset.

We use some benchmark real-life datasets obtained from the UCR Time Series Data Mining archive for discord discovery [14]. The datasets are from many different areas with different lengths. In all experiments, the length of discord for each dataset was fixed and chosen according to experts as shown in Table 3.

Table 3. Characteristics of datasets

Data set	Time series length	Discord length
Video	5000	200
ECG	20000	255
Power	20000	750
Patient	4000	150
Space Shuttle	5000	100

4.2. **Experimental setup.** Both HOT SAX and IDD have three parameters: length of discord n, word size w and alphabet size a to set before running the algorithm. Among these parameters, word size w and alphabet size a do not affect the correctness of both algorithms, but they may affect the performance of both. In the literature, most of works just provide performance evaluation based on the number of calls to the distance function with the best parameter setting. This evaluation is not fair for the methods which have less much time of ordering heuristics and choosing the parameters. Therefore, in our experiment, we evaluate two metrics with various parameters: (1) the number of calls to the distance function and (2) CPU runtime to give the adequate performance evaluation.

To evaluate the direct impact of alphabet size a and word size w on the performance of both algorithms, we conduct experiments Group 1 and Group 2 as follows.

Group 1: We set $w = 5$ for Video, ECG, Power, Patient datasets and $w = 10$ for Space Shuttle. Keep the word size w unchanged, and adjust alphabet size a to evaluate the dependence of each algorithm performance on alphabet size a.

Group 2: Alphabet size a is kept unchanged (3 for HOT SAX and 21 for IDD based on results of Group 1) while the word size w is adjusted to evaluate the dependence of each algorithm performance on word size w.

4.3. **Results and discussion.** The experimental results for running time and the number of calls to distance function in Group 1 are shown in Figure 1 and Figure 2, respectively.

The experimental results in Group 1 show that with the same discord found for both algorithms, IDD has much less running time than HOT SAX despite the fact that the number of calls to distance function of IDD is a bit higher than that of HOT SAX in some cases. The reason is that the preprocessing time of IDD is not as much as HOT SAX when it takes the advantages of information theory-based method, whereas HOT SAX has to spend much time building an array and a trie to support the next steps. The results also show that the IDD algorithm is more stable than the HOT SAX over five datasets. In general, most of experiments show alphabet size of 3 is better for HOT SAX

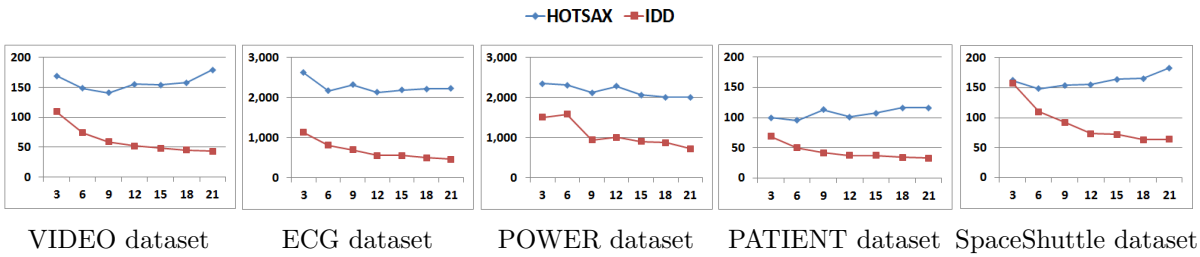

FIGURE 1. Running time (on vertical axis) of each algorithm on each dataset: Keep word size w unchanged and increase the alphabet size a (on horizontal axis)

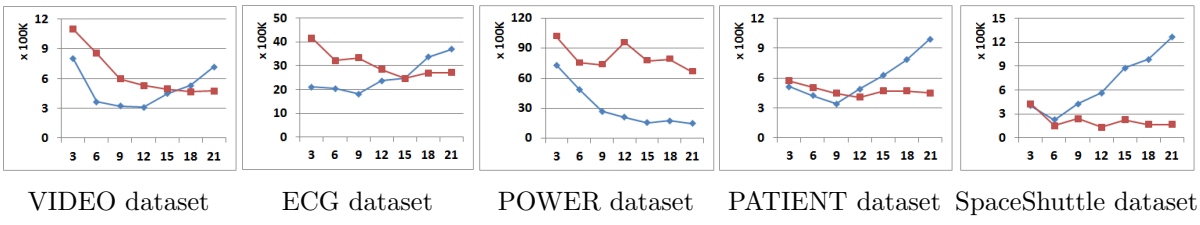

FIGURE 2. Number of calls to distance function (on vertical axis) of each algorithm on each dataset: Keep word size w unchanged and increase the alphabet size a (on horizontal axis)

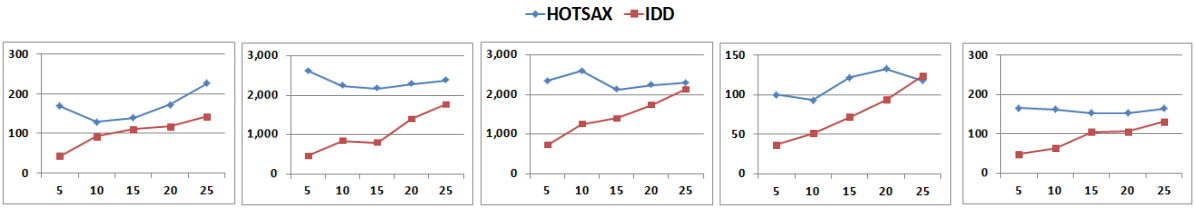

FIGURE 3. Running time (on vertical axis) of each algorithm on each dataset: Keep alphabet size a unchanged and increase the word size w (on horizontal axis)

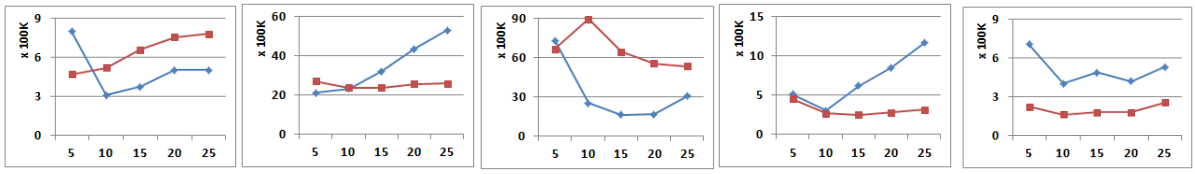

FIGURE 4. Number of calls to distance function (on vertical axis) of each algorithm on each dataset: Keep alphabet size a unchanged and increase the word size w (on horizontal axis)

while IDD prefers the higher value a of 21. In the next experiments in Group 2, we set 3 for HOT SAX and 21 for IDD.

The experimental results for running time and the number of calls to distance function in Group 2 are shown in Figure 3 and Figure 4, respectively.

The results shown above in Group 2 suggest that the proposed algorithm has much more efficient runtime than the HOT SAX algorithm, especially when w is small. As the w increases, the running time of our algorithm increases, too. According to lines 1 and 6 in Table 2, our algorithm's running cost depends on w, the smaller the value of w is, the more efficient our algorithm is. Unlike our method, the running time of the HOT SAX does not linearly increase when w rises.

In terms of memory cost, our algorithm is more efficient than HOT SAX. As presented in Table 2, our algorithm just uses four arrays to store the computed value for $E(C^i)$, $W(C^i)$, $Den_{Cj}(x_i)$ and $WDen(x_i)$ to make heuristic for the next step. Three of them have size of w and the other has size of $(m-n+1)$ which is the number of subsequences. When w is small, most of space we need for preprocessing is nearly an array of $(m-n+1)$, whereas HOT SAX is much more expensive due to using an array and a trie for considering subsequences in two nested loops. General formula to compute space for HOT SAX is $(m-n+1)*w$. Therefore, our algorithm reduces space to one wth times compared to HOT SAX.

5. **Conclusion and Future Works.** In this work, we propose an information theory-based approach to solving the problem of discord discovery in time series with detailed analysis of its efficiency for the metrics of running time, the number of calls to distance function and the memory cost compared to HOT SAX. Our method is evaluated more efficiently on both running cost and memory cost than HOT SAX while guaranteeing to produce identical results even though our number of calls to distance function is a bit higher than HOT SAX in some cases. Furthermore, the parameter selection in our algorithm is easier due to the simplicity and the data independence of the method.

In the future, we are working in the extension of our algorithm in many directions. Firstly, we will apply some other time series discretization methods in our algorithm that

is free-parameter in order to make it more efficiently. Secondly, we may modify the formula of weighted density to make better heuristic for two nested loops. Finally yet importantly, we will extend our work on data streams, which is the most challenging problem recently.

REFERENCES

[1] D. Hawkins, *Identification of Outliers*, Chapman and Hall, London, 1980.
[2] V. Chandola, A. Banerjee and V. Kumar, Outlier detection: A survey, *Technical Report 07-017*, University of Minnesota, 2007.
[3] Z. Y. He, S. C. Deng and X. F. Xu, An optimization model for outlier detection in categorical data, *Proc. of 2005 International Conference on Advances in Intelligent Computing*, Hefei, China, pp.400-409, 2005.
[4] Z. Y. He, S. C. Deng and X. F. Xu, A fast greedy algorithm for outlier mining, *Proc. of the 10th Pacific Asia Conference on Knowledge and Data Discovery*, pp.567-576, 2006.
[5] S. Wu and S. Wang, Information-theoretic outlier detection for large-scale categorical data, *IEEE Trans. Knowledge Engineering and Data Engineering*, vol.25, no.3, pp.589-602, 2013.
[6] X. W. Zhao, J. Y. Liang and F. Y. Cao, A simple and effective outlier detection algorithm for categorical data, *International Journal of Machine Learning Cybernet*, vol.5, no.3, pp.469-477, 2014.
[7] E. Keogh, J. Lin and A. Fu, HOT SAX: Efficiently finding the most unusual time series subsequence, *Proc. of the 5th IEEE International Conference on Data Mining (ICDM)*, pp.226-233, 2005.
[8] E. Keogh, J. Lin and H. V. Herle, Finding unusual medical time-series subsequences: Algorithms and applications, *IEEE Trans. Information Technology in Biomedicine*, vol.10, no.3, pp.429-439, 2006.
[9] E. Keogh et al., Finding the most unusual time series subsequence: Algorithms and applications, *Knowledge and Information Systems*, vol.11, no.1, pp.1-27, 2007.
[10] N. H. Kha and D. T. Anh, From cluster-based outlier detection to time series discord discovery, *Trends and Applications in Knowledge Discovery and Data Mining*, pp.16-28, 2015.
[11] Y. Bu et al., WAT: Finding top-K discords in time series database, *Proc. of the 2007 SIAM International Conference on Data Mining*, Minneapolis, USA, 2007.
[12] L. V. Q. Anh et al., Vector quantization: A discretization technique for fast time series discord discovery, *The NAFOSTED Conference on Information and Computer Science 2015 (NICS'15)*, 2015.
[13] J. Lin et al., A 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 Discover (DMKD 2003)*, pp.2-11, 2003.
[14] E. Keogh, *www.cs.ucr.edu/∼eamonn/discords/*, 2017.