ICIC Express Letters
Volume **13**, Number **8**, August **2019**

ICIC International ©2019 ISSN 1881-803X
pp. 673–680

AN EFFICIENT BIT SERIALIZATION AND PROBABILITY-BASED ALGORITHM FOR TIME SERIES DISCORD DISCOVERY

Thuc-Doan Do[1,2], Phuong-Dai Ngo[2], Hong-Van T. Hoang[1,2] 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 }@tdtu.edu.vn; *Corresponding author: duongthithuyvan@tdtu.edu.vn

Received December 2018; accepted March 2019

Abstract. *Time series discord is a subsequence which has the largest difference compared to the other subsequences of a time series. It can be considered as a type of anomalies in time series and it has many uses in a wide range of areas. In this paper, we propose a novel approach based on bit serialization and probability for time series discord discovery. This work continues our first attempt in reducing the time and space cost of constructing heuristics with an aim of pruning away unnecessary searches in the brute-force algorithm. In our algorithm, using bit representation and formulas based on probability enables to discover potential discord candidates quickly. Experimental results on some time series datasets demonstrate the effectiveness and efficiency of our proposed algorithm compared to HOT SAX and our recent algorithm while the discords discovered by these methods are the same.*

Keywords: Discord discovery, Time series, Bit serialization, Probability

1. Introduction.

Time series discord is defined as a subsequence of a longer time series that is maximally different to other subsequences [1]. Time series discords have demonstrated their uses in many data mining tasks such as summarization, clustering, anomaly detection, and data cleaning. They are also used in a variety of areas including medicine, surveillance, and industry [2,3].

A naïve algorithm for time series discord discovery (brute-force) requires two nested loops, in which each subsequence is compared to the remaining subsequences of the time series to find out the discord. The time complexity of this algorithm is $O(m^2)$, with m being the length of the time series. It is clearly inefficient so there have been many works on suggesting heuristics to reorder the subsequences considered in two loops with the hope that it is possible to exit early from the loops. To the best of our knowledge, most algorithms reduce the time spent on the loops at the expense of much time and space for building heuristics [1-8]. In this paper, we propose a novel bit serialization and probability-based algorithm to discover time series discord which is especially efficient in time and space in building heuristics and in general. Moreover, our method still returns exact discord. The main contributions of our work are the following.

- Propose a new algorithm for time series discord discovery which is effective and efficient in time and space.
- Do the experiments on the various data sets to show the effectiveness and the efficiency of our algorithm.
- Compare our new algorithm with HOT SAX, the most well-known algorithm for time series discord discovery and with our recent algorithm [12] adequately in time

DOI: 10.24507/icicel.13.08.673

efficiency, memory cost, the number of times distance function called, the number of parameters, and then discuss obtained results.

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

2. **Background and Related Works.**

Definition 2.1. *Time Series Discord: Given a time series T, the subsequence D of length n is called the discord of T if D has the largest distance to its nearest non-self match. That is, for every subsequence C of T different from D, non-self match MC of C, and non-self match MD of D:* $\min(\mathit{Dist}(D, MD)) > \min(\mathit{Dist}(C, MC))$.

In it, *Dist*() is a distance function to measure distance from two subsequences of the same length. In our paper, we use the Euclidean distance function as a *Dist*() due to its simplicity and efficiency.

Definition 2.2. *Euclidean Distance: Given two time series Q and C of length n, the Euclidean distance between them is defined as*

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

In it, q_i and c_i are the i^{th} value of the time series Q and C respectively.

Definition 2.3. *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 $\mathit{Dist}(M, C)$ if $|p - q| \geq n$.*

From the aforementioned definitions, it can be seen that brute-force is the simplest algorithm to find discord. We just need two nested loops, in which the outer loop considers each candidate subsequence of the time series, and the inner loop scans other subsequences to identify the candidate's nearest non-self match. The candidate with the largest distance to its nearest non-self match is considered the discord. This algorithm can find the exact discord with only one parameter which is the length of the discord. However, it is impractical for large datasets due to its time complexity $O(m^2)$.

The authors in [1] suggested a generic framework based on two heuristics to improve the brute-force algorithm, one for the order in which the outer loop visits each candidate subsequence, and the other to determine the order in which the inner loop visits the remaining subsequences. It is expected that the outer heuristic will consider the subsequences with the highest ability to become the discord first while the inner heuristic will set higher priority to the subsequences with shorter distances to the current candidate subsequence of the outer loop in order to exit early from the loops of the brute-force. This framework is called the heuristic discord discovery and illustrated in Table 1.

The first and most well-known algorithm following the generic framework is HOT SAX [1]. The authors first reduced the dimensionality of subsequences by Piecewise Aggregate Approximation (PAA) representation before discretizing them into words by Symbolic Aggregate Approximation representation (SAX) and taking advantage of these words' frequencies to build two heuristics. The intuitive idea behind this is that the word with the lowest frequency is more likely to be the discord. Two data structures used to support this process include: an array of words with their corresponding frequencies, and an augmented trie to store the positions of words in the original time series. Similarly, in [4,5], subsequences were transformed and discretized into words, using the same data

Table 1. Heuristic discord discovery [1]

```
**Function** $[dist, loc] = Heuristics\_Search(T, n, Outer, Inner)$
  $best\_so\_far\_dist = 0$
  $best\_so\_far\_loc = NaN$
**For Each** $p$ **in** $T$ ordered by heuristic Outer          //Begin Outer Loop
    $nearest\_neighbor\_dist = infinity$
    **For Each** $q$ **in** $T$ ordered by heuristic Inner      //Begin Inner Loop
        **If** $|p - q| \geq n$                                 //non-self match?
            **If** $Dist(t_p, \ldots, t_{p+n-1}, t_q, \ldots, t_{q+n-1}) < best\_so\_far\_dist$
                **Break**                                       //Break out of Inner Loop
            **EndIf**
            **If** $Dist(t_p, \ldots, t_{p+n-1}, t_q, \ldots, t_{q+n-1}) < nearest\_neighbor\_dist$
                $nearest\_neighbor\_dist = Dist(t_p, \ldots, t_{p+n-1}, t_q, \ldots, t_{q+n-1})$
            **EndIf**
        **EndIf**
    **EndFor**                                                  //End Inner Loop
    **IF** $nearest\_neighbor\_dist > best\_so\_far\_dist$
        $best\_so\_far\_dist = nearest\_neighbor\_dist$
        $best\_so\_far\_loc = p$
    **EndIf**
**EndFor**                                                      //End Outer Loop
**Return** $[best\_so\_far\_dist, best\_so\_far\_loc]$
```

structures. Using Haar wavelet transform and constructing gradually the trie enable to exit earlier from the loops and determine the word size dynamically. The method proposed in [6] replaces SAX with aSAX, in which how to discretize subsequences into words depends on data instead of being fixed like SAX. This method takes more time and space for pre-processing stage while it allows the loops to exit earlier than HOT SAX. The authors in [7] suggested to transform subsequences into bit serialization, and then cluster those bit series. Clusters with the fewest elements are considered first in the outer loop while the inner loop sets higher priority for elements in the same cluster. In [8], the piecewise vector quantized approximation technique is used to discretize subsequences. Two data structures including a Vector Quantization (VQ) representation table and an augmented trie support building heuristics with the idea that the VQ representation with the lowest frequency has more chance to be the discord. In general, these solutions require remarkable memory cost for auxiliary data structures and time for constructing two heuristics.

Recently, in [9], the authors suggested a novel algorithm for the J-distance discord discovery, in which J-distance discord is an extended definition of discord to solve the problem of "Twin Freak" in industries. This algorithm can identify anomalies which happen several times and it is applied to detecting anomalies in aircraft engine gearbox data. The authors in [10,11] suggested to combine time series discord discovery and privacy preserving. Apart from detecting discord, they must conduct encryption and decryption processes to ensure data privacy. In general, in order to solve their own problems, the recent algorithms have to suffer high computational cost.

In terms of the original time series discord discovery problem, our first solution in [12] proposed an information theory-based algorithm (IDD) in an attempt to reduce memory cost and time due to two data structures including an array and a dictionary, and simple formulas for building heuristics. To be specific, subsequences were transformed and discretized into words like HOT SAX before the weighted density of every subsequence was calculated and used for heuristics. The intuitive idea behind this is that the word with the lowest weighted density is more likely to be the discord. In the next section, we present our new bit serialization and probability-based algorithm which improves the algorithm in [12] in time efficiency, memory cost. Furthermore, it uses fewer parameters and has fewer numbers of times distance function called.

3. **The Bit Serialization and Probability-Based Algorithm.** Our algorithm also follows the generic framework in which we build two heuristics based on bit serialization and probability to determine order of subsequences considered in two loops. Firstly, subsequences are extracted from the original time series T (length m) by a sliding window of length n. These subsequences are then reduced dimensionality using PAA representation to length $w << n$. This representation divides each subsequence into w equal segments and the value of each segment is the average value of all points in that segment. After that, in order to transform PAA segments into bit series, we start from the first segment and compare the values of two successive segments. If there is an upward trend, we have a bit 1 while a downward trend is represented by a bit 0. The result obtained after this process is a bit series with length $w - 1$. An example of bit series is shown in Figure 1. The details are presented in [7].

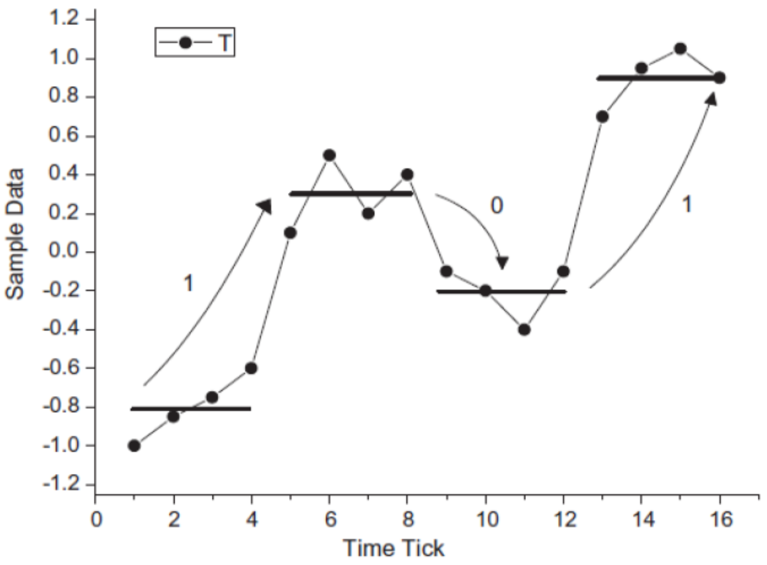

FIGURE 1. An example of bit series T with the length of $3 = \{1, 0, 1\}$ transformed from 4 segments [7]

Given that:
B^k is the kth bit of bit series for $1 \le k \le w - 1$;
v_x^k is the value of kth bit of the bit series x, $v_x^k \in \{0, 1\}$.
The probability that B^k equals 1:

$$P\left(B^k = 1\right) = \frac{\textit{the number of bit series with } B^k = 1}{m - n + 1} \tag{1}$$

in which $m - n + 1$ is the number of bit series or the number of subsequences.
The probability that B^k equals 0:

$$P\left(B^k = 0\right) = 1 - P\left(B^k = 1\right) \tag{2}$$

The probability of bit series x:

$$P(x) = \prod_{k=1}^{w-1} P\left(B^k = v_x^k\right) \tag{3}$$

with the assumption that values of all positions of the bit series are independent [13].

Based on the computation of the probabilities of all bit series, we construct two heuristics as follows.

Firstly, we reorder the candidate subsequences in the outer loop using following heuristic.

Outer Loop Heuristic: The subsequences, which have the lowest probability for their bit serialization, are considered first in the outer loop. After that, the remaining subsequences are visited in a random order.

The intuition behind this heuristic is that the lower the probability of bit series is, the more likely that bit series is a discord.

When the subsequence s_i is considered in the outer loop, we reorder the subsequences considered in the inner loop according to the following inner loop heuristic.

Inner Loop Heuristic: We find subsequences which have the same representation in bit serialization with s_i to consider them first in the inner loop. After that, the remaining subsequences are visited in a random order.

The reason behind this heuristic is that two subsequences with the same representation in bit serialization are likely to be similar. Therefore, the distance between them is expected to be minimal and the inner loop is likely to exit early.

We use two data structures to support the Outer Loop Heuristic and Inner Loop Heuristic: an array of bit series where the last column contains the probability of those bit series and a dictionary which helps to retrieve instantly the positions of subsequences in the original time series mapped to each bit series by using the key or that bit series. Figure 2 presents these data structures.

1	**1**	**0**	**0**	0.01
2	**1**	**1**	**1**	0.005
3	**1**	**0**	**0**	0.01
…	**…**	**…**	**…**	…
m-n+1	**1**	**1**	**0**	0.002

Key	**Values**		
100	1	3	
111	2		
…	…		
110	m-n+1		

FIGURE 2. Two data structures are used to support heuristics: (Left) an array of bit series with their probabilities; (Right) a dictionary to look up the positions of subsequences corresponding to bit series in the original time series.

In our method, the transformed subsequences or bit series are only used to construct two heuristics for the outer and inner loops. For the distance function in our proposed main algorithm, we use Euclidean distance function between the original subsequences to find the exact discord. This is because in Definition 2.1, discord is defined according to distance between original subsequences. Transformed subsequences are approximate subsequences of the original ones and distance between them only gives approximate results.

The proposed algorithm is called BPDD (**B**it serialization and **P**robability based **D**iscord **D**iscovery) and presented in Table 2.

4. Experimental Evaluation.

4.1. **Experiments.** For experimental evaluation, we implemented three algorithms: HOT SAX, IDD and BPDD. They are compared in terms of time efficiency, memory cost, the number of times distance function called and the number of parameters. All experiments were implemented by Matlab R2016a on a Core i7, 2 CPU 2.9GHz, 8GB RAM, Windows 7 64-bit laptop.

TABLE 2. Bit serialization and Probability based Discord Discovery (BPDD)

```
Input: A time series with length of $m$
Output: Discord length of $n$
---
Step 1: All subsequences are extracted from the original time series by sliding
window length $n$ and converted to bit series with length of $w-1$.
Step 2: Compute the probability of each transformed subsequence by using for-
mulas (1)-(3).
1       For $k = 1$ to $w-1$ do
2           Compute $P\left(B^k = 1\right)$ according to (1)
3           Compute $P\left(B^k = 0\right)$ according to (2)
4       For $x = 1$ to $m-n+1$ do
5           Compute $P(x)$ according to (3)
Step 3: Based on the probability in bit serialization of each subsequence that we
have computed, we construct two heuristics for the outer and inner loops and run
the heuristic discord discovery framework in Table 1.
```

TABLE 3. Characteristics of datasets and parameters chosen

Dataset	**Time series length**	n	w	A
ECG	20000	255	5	3
Respiration	4000	150	5	3
Power Demand	20000	750	5	3
Space Shuttle	5000	100	5	3
Video Surveillance	5000	200	5	3

TABLE 4. Runtime of HOT SAX, IDD and BPDD (in seconds)

Dataset	**HOT SAX**	**IDD**	**BPDD**
ECG	412.05	80.71	31.90
Respiration	18.84	5.13	4.41
Power Demand	623.35	254.91	81.63
Space Shuttle	39.20	18.13	5.52
Video Surveillance	29.98	15.52	5.57

We use 6 datasets from the UCR Time Series Data Mining archive for discord discovery [14]. They are from different areas such as medicine, manufacturing and science.

For each dataset, we set the same parameters for the three algorithms to evaluate them fairly. While HOT SAX and IDD have three parameters: length of discord n, word size w and alphabet size a, BPDD has two parameters: length of discord n, word size w. The lengths of discords are chosen according to experts [3]. The parameters used for the datasets are presented in Table 3.

The experimental results are obtained by executing each of the three algorithms ten times and the average of results is taken.

4.2. **Results and discussion.** To evaluate the time efficiency of the three algorithms, we measure their runtime on aforementioned datasets. The results are presented in Table 4.

It can be seen that BPDD algorithm is much faster than HOT SAX. It is even remarkably faster than IDD algorithm which is our first attempt in reducing computational cost of discord discovery. This improvement can be explained by the fact that instead of discretizing subsequences into characters by SAX representation like HOT SAX and IDD, BPDD uses bit representation which makes the computation more efficient. The formulas proposed in BPDD are simpler than those of IDD while the dictionary of BPDD helps to retrieve information faster than the augmented trie of HOT SAX.

In the heuristic discord discovery framework, the distance function between two subsequences in the original time series must be called to calculate their real distance, which enables to find discord. The number of times distance function called in three algorithms is presented in Table 5.

Table 5. The number of times distance function called in HOT SAX, IDD and BPDD

Dataset	**HOT SAX**	**IDD**	**BPDD**
ECG	2,506,345	2,724,931	2,675,592
Respiration	580,355	476,111	585,025
Power Demand	19,266,502	15,250,559	5,284,319
Space Shuttle	1,114,635	1,735,844	846,990
Video Surveillance	1,561,344	1,951,510	670,898

From the above results, it is clear that, the number of distance function calls in BPDD is remarkably lower than that of IDD, which indicates that the heuristics of BPDD is better in general. Compared to HOT SAX, heuristics in BPDD is better than those of HOT SAX in three datasets: Power Demand, Space Shuttle and Video Surveillance while the effectiveness of heuristics in BPDD is a bit lower in the two remaining datasets. In general, the effectiveness of heuristics in BPDD and that in HOT SAX are competitive.

In terms of memory cost, BPDD is more efficient than HOT SAX and IDD. To be specific, for HOT SAX, the array of SAX words requires $(m - n + 1) * (w + 1)$, with $(m-n+1)$ being the number of words or subsequences, w being the number of characters in each word (the length of word), and one for the last column. Similarly, IDD requires $(m - n + 1) * (w + 1)$ for its array of SAX words. Meanwhile, the memory required by BPDD for its array of bit series is $(m - n + 1) * w$ because the length of bit series is $w - 1$. In order to calculate heuristics, IDD needs three arrays to store the computed values while BPDD needs two arrays. However, the sizes of these arrays are $w << m$. On the other hand, the dictionary used in BPDD requires at least $(m - n + 1) + 1$ if all bit series are the same and at most $(m - n + 1) * 2$ if all bit series are different one from another. The memory cost for the dictionary of IDD is similar to that of BPDD while the augmented trie of HOT SAX needs at least $(m-n+1)+w$ and at most $(m-n+1)*w$.

5. Conclusion and Future Works.

In this work, we proposed a bit serialization and probability-based algorithm for time series discord discovery. Our algorithm uses bit representation and simple formulas based on probability to improve the performance of calculating heuristics before applying the generic framework. We conducted experiments on six datasets to evaluate our algorithm, HOT SAX and IDD. Experimental results have shown that our BPDD algorithm outperforms HOT SAX and IDD in terms of time efficiency and memory cost. The heuristics constructed by BPDD are better than IDD and competitive to those of HOT SAX. Furthermore, the number of parameters of BPDD is fewer than IDD and HOT SAX while their accuracies are the same.

In the future, we are working in some directions to extend our algorithm. Firstly, we will take advantage of computational power of bit serialization to construct better heuristics while maintaining the time efficiency and memory cost. We also continue reducing the number of parameters, towards parameter-free algorithms. Finally, we will extend our work on data streams, which is the most challenging problem recently.

REFERENCES

[1] 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.

[2] E. Keogh et al., Finding unusual medical time-series subsequences: Algorithms and applications, *IEEE Transactions on Information Technology in Biomedicine,* vol.10, no.3, pp.429-439, 2006.

[3] 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.

[4] A. Fu et al., Finding time series discords based on Haar transform, in *Advanced Data Mining and Applications*, X. Li, O. R. Zaïane and Z. Li (eds.), Springer, Berlin, Heidelberg, 2006.

[5] 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.

[6] N. D. Pham, Q. L. Le and T. K. Dang, HOT aSAX: A novel adaptive symbolic representation for time series discords discovery, in *Intelligent Information and Database Systems*, N. T. Nguyen, M. T. Le and J. Świątek (eds.), Springer, Berlin, Heidelberg, 2010.

[7] G. Li et al., Finding time series discord based on bit representation clustering, *Knowledge-Based Systems*, vol.54, pp.243-254, 2013.

[8] 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.

[9] Z. Wang, D. Pi and Y. Gao, A novel unsupervised time series discord detection algorithm in aircraft engine gearbox, *Proc. of the 14th International Conference on Advanced Data Mining and Applications*, pp.202-210, 2018.

[10] C. Zhang, H. Liu and Y. Li, Time series discord discovery under multi-party privacy preserving, *Proc. of the 2017 IEEE Second International Conference on Data Science in Cyberspace*, pp.467-474, 2017.

[11] C. Zhang et al., Fast time series discords detection with privacy preserving, *Proc. of the 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2018.

[12] T.-D. Do, H.-V. T. Hoang, C. Huynh and T.-V. T. Duong, An information theory-based approach for time series discord discovery, *ICIC Express Letters*, vol.12, no.10, pp.1071-1078, 2018.

[13] D. P. Bertsekas and J. N. Tsitsiklis, *Introduction to Probability*, 2nd Edition, Athena Scientific, 2008.

[14] E. Keogh, *www.cs.ucr.edu/∼eamonn/discords/*, accessed in December 2018.