A Dynamic Programming Approach for Time Series Discord Detection

Duong Tuan Anh[1(✉)] and Nguyen Van Hien[2]

[1] Department of Information Technology, HCMC University of Foreign Languages and Information Technology, Ho Chi Minh City, Vietnam
anh.dt@huflit.edu.vn
[2] VNG Company, Ho Chi Minh City, Vietnam

Abstract. There have been several methods to search the top anomaly subsequence (1-discord) in a time series. Most of these methods belong to the window-based category which uses a sliding window with a pre-specified length to extract subsequences. However, one of the main shortcomings of these window-based methods for discord detection is that their computational cost is still high in the cases of very large time series data. In this paper, we propose a new dynamic programming approach for discord detection in time series under Euclidean distance in order to improve further its time efficiency. We evaluate our proposed dynamic programming approach on several time series datasets and the results show that our method provides performance up to 25.2 times faster than HOT SAX algorithm.

Keywords: Time series · Discord · Discord detection · Dynamic programming

1 Introduction

Time series discord is the subsequence of a time series which is the most dissimilar to the rest of the subsequences. In [5], Keogh et al. introduced the term *discord* to refine the concept of an anomalous subsequence. The problem of discord (also called anomaly, novelty, deviant pattern) detection in time series has attracted much attention because of the wide diversity of its practical applications. Some typical applications of time series discord discovery in real world can be listed as follows. Nicoli et al. (2013) proposed a method to use anomaly detection in monitoring system's measurements of a hydroelectric dam [1]. Sivaraks and Ratanamahatana (2015) proposed a method for anomaly detection in electrocardiogram (ECG) time series [2]. Ortiz et al. (2019) proposed an anomaly detection approach in electroencephalogram (EEG) time series for dyslexia diagnosis [3]. Munir et al. (2017) proposed a method for anomaly detection on operational time series data of Internet of Things (IoT)- based household devices [4].

There has been several research works on time series discord search in the literature. Various algorithms proposed for this problem will be summarized as follows. Brute-Force, by Keogh, Lin, and Fu (2005) [5], is a simple algorithm for discord detection which comprises two nested loops. HOT SAX, by Keogh et al. (2005) [5], uses Symbolic Aggregate Approximation (SAX) (Lin et al., 2003 [6]) as a

P. Cong Vinh and A. Rakib (Eds.): ICCASA 2021, LNICST 409, pp. 255–266, 2021.
https://doi.org/10.1007/978-3-030-93179-7_20

symbolization transform and utilizes two ordering heuristics for the inner and outer loops to improve the discord search process. Bu et al. (2007) [7] proposed WAT algorithm which searches top k-discords by using Haar Wavelet Transform and augmented trie. Oliveira et al. (2004) [8] found discords in time series by using RBF neural networks. Sanchez and Bustos (2014) [9] proposed a method for time series discord search based on bounding boxes which does not require normalizing the subsequences. Huang et al. (2015) proposed a new discord definition named J-distance discord and a new discord search algorithm which can handle the "twin freak" problem in time series discord discovery [10]. Thuy et al. (2016) proposed some segmentation-based techniques to improve HOT SAX algorithm in time series discord search [11]. Vy and Anh (2016) found discords by using segmentation and anomaly pattern scoring [12]. And Buda et al. (2018) proposed a framework for time series anomaly detection which applies long short term memory (LSTM) network in a prediction-based approach [13].

Time series anomaly detection methods can be categorized into four kinds: window-based methods, segmentation-based methods, prediction-based and classification-based methods. Most of the above-mentioned methods belong to the window-based category which uses a sliding window with a pre-specified length to extract subsequences from a long time series before discord search. However, one of the main shortcomings of these window-based methods for discord detection is that their computational cost is still high in the cases of large scale time series.

In this paper, we propose a new algorithm, called DPDD, which is based on dynamic programming for discord search in time series under Euclidean distance. This dynamic programming algorithm has the same structure as the Brute-Force algorithm for discord discovery and hinges on the data reuse of the distance computations in the first iteration of Brute-Force algorithm for all the rest of iterations in this algorithm. The distance computations for the pairs of subsequences in the Brute-Force algorithm can be seen as overlapping sub-problems in the DPDD algorithm. The DPDD algorithm has time complexity just $O(m)$ where m is the length of the time series. To the best of our knowledge, this algorithm is the first one for time series discord detection which has linear time complexity. We evaluate our proposed dynamic programming approach on eight benchmark datasets and the results show that DPDD can run faster than HOT SAX about 25.2 times while brings out the same accuracy for discord detection.

The rest of the paper is structured as follows. Section 2 reviews some basic backgrounds about discord and Brute-Force algorithm for discord detection. Section 3 introduces the proposed dynamic programming algorithm, called DPDD, for discord detection in time series. Section 4 reports the experiments to evaluate the performance of DPDD in discord detection. Finally, Sect. 5 presents some conclusions and expectations for future work.

2 Background

2.1 Definitions

A time series $T = \{t_1, t_2, ..., t_m\}$ is an ordered list of m data points $t_i \in \Re$, measured at equal intervals.

Definition 1 (*Subsequence*): Given a time series T of length m, a subsequence C of T is a segment of length $n < m$ of consecutive positions from T, that is, $C = t_p, \ldots, t_{p+n-1}$ for $1 \leq p \leq m-n+1$.

We can denote the subsequence C as $T_{p:p+n-1}$.

Definition 2. (*Non-trivial match*): Given a time series T, including a subsequence C of length n beginning at position p and a matching subsequence D beginning at the position q. D is called as a non-trivial match to C if $|p-q| \geq n$.

Definition 3. (*Discord*): Given a time series T, the subsequence D in T is called the most significant discord (1-discord) in T if it has the largest distance to its nearest non-trivial match.

2.2 Brute-Force Algorithm

The problem of finding 1-discord can be solved by the Brute-Force algorithm (called BFDD) is given in [5]. In Brute-Force algorithm, using a sliding window, all possible candidate subsequences can be extracted in the outer loop, and then the algorithm finds the distance to the nearest non-trivial match for each candidate subsequence in the inner loop. The candidate subsequence with the largest distance to its nearest non-trivial match is the 1-discord.

Since the brute-force algorithm is a window-based method for searching discords in a time series with a nested loop, it incurs high complexity. Its complexity is $O(m^2)$, where m is the length of the time series.

3 A Dynamic Programming Approach for Time Series Discord Detection

3.1 The Main Idea

Euclidean distance is the most widely-used distance measure in time series data mining. Given two time series $X = x_1, x_2, .., x_n$ and $Y = y_1, y_2, .., y_n$, the Euclidean distance between X and Y is given as in the following formula:

$$Dist(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

In the context of time series discord detection, if we do not use square root in the above formula, this does not change the relative rankings of nearest matches, since Euclidean function is monotonic and concave [14].

The main idea of dynamic programming approach for time series discord search is that we have to compute directly the distances between the pairs of subsequences in the first iteration (in the BFDD) only and store them in a table and reuse these values to compute the distances between the pairs of subsequences for all the rest of iterations. In a time series T, the distance between the subsequence at position p and the subsequence

at position q of the same length n is denoted as $D(p, q)$. Here we index the elements in T as the integers 0,.., $|T| - 1$. So the distance between two subsequences which is denoted as $D(p, q)$ is computed as:

$$\left(T_p - T_q\right)^2 + \left(T_{p+1} - T_{q+1}\right)^2 + .. + \left(T_{p+n-1} - T_{q+n-1}\right)^2$$

And $D(p + 1, q + 1)$, the distance between the subsequence at position $p + 1$ and the subsequence at position $q + 1$ of the same length n, is computed as:

$$\left(T_{p+1} - T_{q+1}\right)^2 + \left(T_{p+2} - T_{q+2}\right)^2 + .. + \left(T_{p+n} - T_{q+n}\right)^2$$

We can see that the distance $D(p+1, q+1)$ is different from $D(p, q)$ at only the first and the last quantity in the computing formula. So we can view computing $D(p, q)$ and $D(p + 1, q + 1)$ as overlapping subproblems. Basing on this finding, we compute $D(p + 1, q + 1)$ by using the stored value of $D(p, q)$ as in the following formula.

$$D(p+1, q+1) = D(p, q) - \left(T_p - T_q\right)^2 + \left(T_{p+n} - T_{q+n}\right)^2 \quad (2)$$

Now consider a toy example. Given a time series T with length 11, we index T with the values 0, 1, 2,…, 10. Assume that the length of sliding window is 3. Based on the above mentioned dynamic programing idea, we can compute the distances between pairs of subsequences in the Brute-Force algorithm for this example by using a table which is illustrated in Fig. 1. Remember that $D(1, 4)$ means $Dist(T_{1:3}, T_{4:6})$.

In Fig. 1, we store the computed distances in a table which is implemented as a one-dimensional array. We can used $D(0, 3)$ (i.e. $Dist(T_{0:2}, T_{3:5})$) to compute $D(1, 4)$ (i.e. $Dist(T_{1:3}, T_{4:6})$) and all these data reuses are highlighted by the arrows between two adjacent cells in the table in Fig. 1. So we have to compute only the distances between the pairs of subsequences in the first iteration of the Brute-Force algorithm. In all the other iterations, we can derive the distances from the stored distances in the table rather than computing them from the scratch.

Besides, in the cases that p is greater than q, we can derive $D(p, q)$ from $D(q, p)$ due to the *symmetry* of the Euclidean distance $(Dist(X, , Y) = Dist(Y, X))$. Therefore, in Fig. 1 we do not have to compute $D(6, 1)$, $D(6, 2)$, $D(6, 3)$ since $D(1, 6)$, $D(2, 6)$, $D(3, 6)$ are already available in the table at that iteration. In Fig. 1 for all the distances in the iterations 6,.., 8 we can refer to the stored distance values in the table basing on the symmetry of Euclidean distance. All these distances are highlighted in bold in Fig. 1 and we do not have to store them in the table. As for the case of a time series of the length 11 and the sliding window of the length 3, we need a one-dimensional array of length (11–2*3 + 1)* (11–2*3 + 2)/2 = 21 to store necessary distance values.

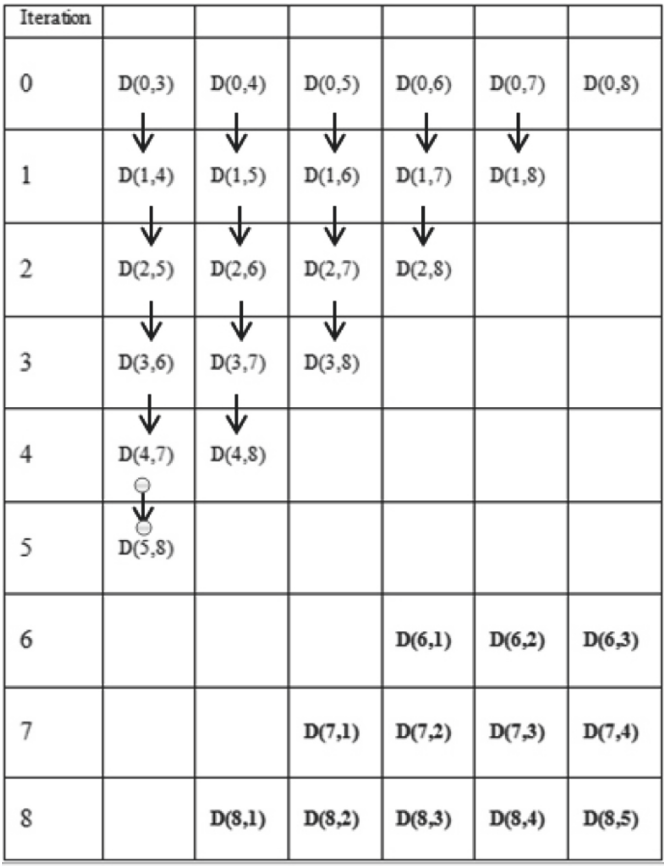

Fig. 1. Illustration of dynamic programming technique in computing Euclidean distances between pairs of subsequences for BFDD algorithm.

3.2 Dynamic Programming Algorithm for Discord Detection

Based on the main idea explained in the above subsection, we come with the dynamic programming algorithm for time series discord detection, which is called DPDD (abbreviated for Dynamic Programming for Discord Detection). The pseudo code for DPDD algorithm is described as follows:

Algorithm 1: (DPDD)
Input: *T* is a time series and *n* is discord length
Output: discord_dist and discord_loc: distance and location of the found 1-discord.

```
1. discord_dist = infinity;
2. discord_loc = NaN
3. k = 0;
4. for j = n to |T|- n
5.    dist = Dist(T0:n-1, Tj:j+n-1]
6.    a[k] = dist
7.    k = k+1
8.    if dist < discord_dist
9.       discord_dist = dist
10   endif
11 endfor
12 discord_loc = 0
13 for p = 1 to |T| - n
14    nearest_match_dist = infinity
15    for q = 0 to |T| - n
16       if abs(p – q) ≥ n
17          if q ≥ p
18             index = (|T| - n*2)*(p-1) – (p-1)*(1+ (p-1))/2 +q – n
19          dist = a[index] – (Tp-1- Tq-1)2 + (Tp-1+n – Tq-1+n)2
20          else
21             index = (|T| - n*2)*q – q*(1+q)/2 + p –n
22             dist = a[index]
23          endif
24          a[k] = dist
25          k = k+1
26          if dist < nearest_match_dist
27             nearest_match_dist = dist
28          endif
29       endif
30    endfor
31    if nearest_match_dist > discord_dist
32       discord_dist = nearest_match_dist
33       discord_loc = p
34    endif
35 endfor
36 return <discord_dist, discord_loc>
```

Even though the table in Fig. 1 looks like a matrix data structure, in DPDD, for saving memory space, we use a one-dimensional array *a* to implement this table for storing the computed distances. The length of array *a* for a time series *T* with a sliding window of length *n* is computes as follows:

$$l = (|T| - 2 * n + 1)(|T| - 2 * n + 2)/2 \tag{3}$$

In Algorithm 1, the first *for* loop (lines 4–11) performs the direct distance computations for the pairs of subsequences in the first iteration of the algorithm. All the distances computed in the first iteration are stored in the array a. The next *for* loop (lines 13–35) performs the distance computations in all the rest of iterations for finding the 1-discord in the time series T. Notice that $D(p, q)$ is stored in array a at the element with the index $(|T| - n * 2) * (p - 1) - (p - 1) * ((p - 1) + 1)/2 + q - n$.

If the basic operation in the DPDD is the distance computation for a pair of subsequences, it is obvious that the time complexity of DPDD is about O(m) where m is the length of the time series. But DPDD requires an array to store intermediate values and this space complexity is O(m^2) due to Formula (3). So, dynamic programming approach always exhibits the trade-off between time and space cost in a given algorithm. However, with large capacity of RAM in current computers, all the data involved in the computations in DPDD for a given time series can fit into the main memory.

4 Empirical Evaluation

For empirical evaluation, we investigate the performance of two algorithms, DPDD and HOT SAX. The comparative methods are implemented with C++ and the experiments are conducted on MacBook Pro 2015 - MJLQ2, 2.2 GHz Quad-Core Intel Core i7, RAM: 16 GB.

The experiments aim to compare DPDD, the proposed dynamic programming approach for discord search with HOT SAX in two perspectives: effectiveness and computational efficiency. The HOT SAX is selected as the baseline algorithm due to its popularity in time series discord detection.

4.1 Datasets and Parameter Setting

The test datasets used in our experiments are obtained from the UCR time series archive for discord detection [15]. These 8 test datasets are from different domains (finance, health care, industry). Table 1 shows the description and the length of each tested dataset. Figure 2 shows the plots of some time series out of eight tested time series.

Table 1. Description and length of eight datasets

Dataset	Description	Length
TEK16	Industry	4992
TEK17	Industry	5000
TEK14	Industry	5000
ECG	Medicine	21600
stock_20_0	Finance	5000
Power_demand_italy	Industry	29931
Power_data	Industry	35040
memory	Industry	6875

We have set some parameters for HOT SAX as follows: size of the alphabet is 3 and the SAX word length is 8.

For all the eight test datasets and for both HOT SAX and DPDD, we select the sliding window length (i.e. the length of 1-discord) as 128. These parameter selections are based on the experiments from several previous works in time series discord detection which used the same benchmark datasets.

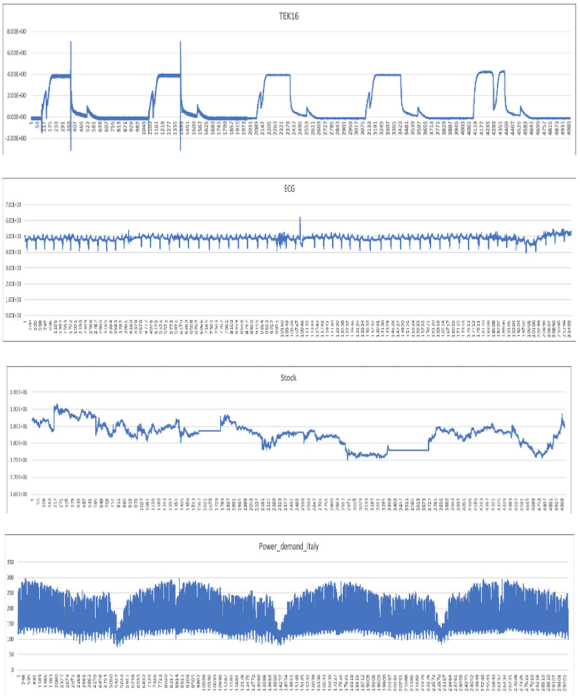

Fig. 2. Some time series out of eight test time series: (a) TEK16, (b) ECG, (c) Stock, and (d) Power_demand_italy

4.2 Experimental Results

From our experiments, through human inspection, we found out that the resultant 1-discord subsequence detected by DPDD for each test dataset matches with the one found by HOT SAX. As for TEK16, TEK17, TEK14 and ECG datasets, the resultant 1-discord subsequence detected by DPDD for each test dataset also matches with the one annotated by experts [5]. Besides, for each test dataset, the start location of the 1-discord detected by DPDD is exactly the same as that of the 1-discord found by HOT SAX. Table 2 reports the start locations of the 1-discords found by DPDD and HOT SAX on eight test datasets. Figure 3 shows the test dataset TEK16 and the 1-discord found in the dataset by DPDD which matches with the one found by HOT SAX.

Table 2. Locations of discords detected by DPDD and HOT SAX

Dataset	DPDD	HOT SAX
TEK16	4253	4253
TEK17	2101	2101
TEK14	1091	1091
ECG	10864	10864
stock_20_0	122	122
Power_demand_italy	26343	26343
power_data	4594	4594
memory	6260	6260

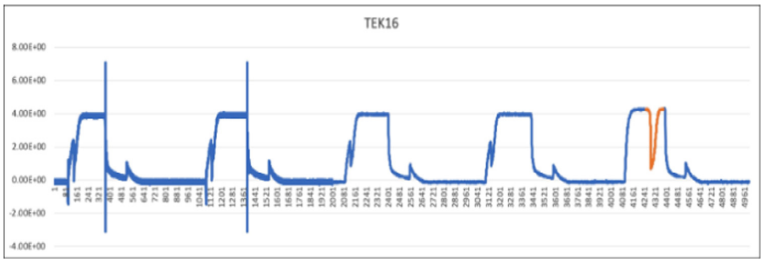

Fig. 3. Time series TEK16 and the 1-discord found by DPDD (in red) in this time series.

Table 3 reports the running times in milliseconds of DPDD and HOT SAX on eight datasets. The speed-up rates of DPDD over HOT SAX on each test dataset are reported in the 4th column of Table 3. This speed-up rate depends on the characteristics of each time series and the size of this time series. The experimental results in Table 3 show that in average, DPDD can execute faster than HOT SAX about 25.2 times.

Table 3. Execution times (in milliseconds) of DPDD and HOT SAX on eight test datasets

Dataset	DPDD	HOT SAX	Speed-up
TEK16	1392	7962	5.720
TEK17	1402	61402	43.796
TEK14	1393	58464	41.790
ECG	32626	124782	3.824
stock_20_0	1408	13968	9.920
Power_demand_italy	67392	3980000	59.057
power_data	82953	2374770	28.627
memory	2753	24250	8.809

4.3 Discussion

In comparison to HOT SAX, DPDD algorithm brings out remarkably higher time efficiency since the time complexity of DPDD is linear. But one more important fact is that a higher efficiency can still be obtained if this dynamic programming algorithm is properly adapted to the GPU programming environment [16, 17]. There have been some research works which proposed some efficient parallelization strategies for dynamic programming algorithms on GPUs, for example, the work [18] by Wani and Quadri in 2013 and the work [19] by Berger et al. in 2013. These research works suggest some guidelines which can be used in designing a parallel implementation for DPDD algorithm on GPU environment.

5 Conclusion and Future Work

This paper has introduced a new dynamic programming algorithm, called DPDD, for discord detection in time series under Euclidean distance. This algorithm has the same structure as the Brute-Force algorithm for discord discovery and hinges on the data reuse of the distance computations in the first iteration for all the rest of iterations. The DPDD algorithm has time complexity just O(m) where m is the length of the time series and memory complexity O(m^2). We evaluate our proposed dynamic programming approach on eight benchmark datasets and the results show that DPDD can run faster than HOT SAX about 25.2 times while brings out the same accuracy for discord detection.

This work is just the first part of our ongoing research project. As for future work, we intend to improve DPDD algorithm to address the "twin freak" problem in time series discord detection [10, 20]. Besides, we plan to accelerate DPDD algorithm further by implementing this algorithm as a parallel version on GPU computing [16, 17].

References

1. Sivaraks, H., Ratanamahatana, C.A.: Robust and accurate anomaly detection in ECG artifacts using time series motif discovery. In: Computational and Mathematical Methods in Medicine, vol. 2015, 20 p. (2015)
2. Nicoli, S., Jargini, J.A., Magrini, L.G., Miranda, C.D.M.: Detection of nonconformities in monitoring system's measurements. In: IEEE PES Conference on Innovative Smart Grid Technologies - Latin America (ISGT LA), Sao Paulo, 15–17 April (2013)
3. Ortiz, A., López, P.J., Luque, J.L., Martínez-Murcia, F.J., Aquino-Britez, D.A., Ortega, J.: An anomaly detection approach for dyslexia diagnosis using EEG signals. In: Vicente, J.M. F., Álvarez-Sánchez, J.R., de la Paz, F., López, J.T., Moreo, H.A. (eds.) Understanding the Brain Function and Emotions. LNCS, vol. 11486, pp. 369–378. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19591-5_38
4. Munir, M., Erkel, S., Dengel, A., Ahmed, S.: Pattern based contextual anomaly detection in HVAC systems. In: Proceedings of IEEE International Conference on Data Mining Workshops (ICDMW), pp. 1066–1073 (2017)
5. Keogh, E., Lin, J., Fu, A.: HOT SAX: efficiently finding the most unusual time series subsequence. In: Proceedings of the Fifth IEEE International Conference on Data mining, Houston, Texas, pp. 226–233 (2005)
6. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: Symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, San Diego, CA, 13 June (2003)
7. Bu, Y., Leung, T.W., Fu, A.W.C., Keogh, E., Pei, J., Meshkin, S.: WAT: finding top-k discords in time series database. In: Proceedings of the 2007 SIAM International Conference on Data Mining, April, pp. 449–454 (2007)
8. Oliveira, A.L.I., Neto, F.B.L. and Meira, S.R.L.: A method based on RBF-DAA neural network for improving Novelty detection in time series. In: Proc. of 17th International FLAIRS Conference, AAAI Press, Miami Beach (2004)
9. Sanchez, H., Bustos, B.: Anomaly detection in streaming time series based on bounding boxes. In: Traina, A.J.M., Traina, C., Cordeiro, R.L.F. (eds.) Similarity Search and Applications. LNCS, vol. 8821, pp. 201–213. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11988-5_19
10. Huang, T., Zhu, Y. Wu, Y. and Shi, W.: J-distance discord: an improved time series dis-cord definition and discovery method. In: Proceedings of 15th International Conference on Data Mining Workshops, pp. 303–310 (2015)
11. Thuy, H.T.T., Anh, D.T., Chau, V.T.N.: Some efficient segmentation-based techniques to improve time series discord discovery. In: Vinh, P.C., Barolli, L. (eds.) Nature of Computation and Communication. LNICSSITE, vol. 168, pp. 179–188. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46909-6_17
12. Vy, N.D.K., Anh, D.T.: Detecting variable length anomaly patterns in time series data. In: Tan, Y., Shi, Y. (eds.) Data Mining and Big Data. Lecture Notes in Computer Science, vol. 9714, pp. 279–287. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-40973-3_28
13. Buda, T.S., Caglayan, B., Assem, H.: DeepAD: a generic framework based on deep learning for time series anomaly detection. In: Phung, D., Tseng, V.S., Webb, G.I., Ho, B., Ganji, M., Rashidi, L. (eds.) PAKDD 2018. LNCS (LNAI), vol. 10937, pp. 577–588. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93034-3_46

14. Rakthanmanon, T., et al.: Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping. ACM Trans. Knowl. Discov. Data (TKDD) **7** (3), 10 (2013)
15. The UCR Time Series Dataset Archive for Discord Detection. https://www.cs.ucr.edu/~eamonn/discords/. Accessed 2019
16. NVIDIA: CUDA Programming Guide Version 8.0 (2017). https://docs.nvidia.com/cuda/index.html
17. NVIDIA: CUDA Toolkit Documentation Version 8.0 (2017). https://docs.nvidia.com/cuda/index.html
18. Wani, M.A., Quadri, S.M.K.: Accelerated dynamic programming on GPU: a study of speed up and programming approach. Int. J. Comput. Appl. **69**(3), 18–21 (2013)
19. Berger, K.E., Galea, F.: An efficient parallelization strategy for dynamic programming on GPU. In: Proceedings of IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, pp. 1797–1806 (2013)
20. Zhang, C., Liu, H., Yin, A.: Research of detection algorithm for time series abnormal subsequence. In: Proceedings of International Conference of Pioneering Computer Scientists, Engineers and Educations, 16 September, pp. 12–26 (2017)