A Continuous Segmentation Algorithm for Streaming Time Series

Yupeng Hu[1,2], Cun Ji[1], Ming Jing[1], Yiming Ding[1], Shuo Kuai[1], and Xueqing Li[1](✉)

[1] School of Computer Science and Technology, Shandong University, Jinan, China
{huyupeng, jicun, jingming, xqli}@sdu.edu.cn, dingyiming@mail.sdu.edu.cn, kuaishuo_sdu@163.com

[2] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China

Abstract. Along with the arrival of Industry 4.0 era, massive numbers of detecting instruments in various fields are continuously producing a plenty number of time series stream data. In order to efficiently and effectively analyze and mine the high-dimensional streaming time series, the segmentation which provides more accurate representation to the raw time series data, should be done as the first step. In this paper, we propose a novel online segmentation approach based on the turning points to partition the time series into some continuous subsequences and maintain a high similarity between the processed subsequences and the raw data. It achieves the best overall performance on the segmentation results compared with other baseline methods. Extensive experiments on all kinds of typical time series datasets have been conducted to demonstrate the advantages of our method.

Keywords: Data mining · Time series · Online segmentation · Algorithms

1 Introduction

Along with the arrival of Industry 4.0 era, the evolving of IoT (Internet of Things) has stimulated the deployment of massive numbers of detecting instruments in various fields including business, finance, medicine, astronomy, aviation and so on, which are continuously producing a plenty number of time series stream data [1–3]. Time series stream data can be described as an ordered collection of elements, where the elements are continuously generated in a high speed and potentially forever [4, 5]. Due to the large amount, high-dimensional, continuous and other related properties, it is not capable to do in-depth researches for the streaming time series.

In consideration of the above situation, the segmentation of the streaming time series should be done as the first step to reduce both the space and the computational cost of storing and transmitting such data. In this paper, we are concerned with the online linear segmentation approach for the streaming time series, which is aimed for not only producing approximate representation conformed the temporal features, but also being effective in continuous segmentation. An efficient online segmentation

S. Wang and A. Zhou (Eds.): CollaborateCom 2016, LNICST 201, pp. 140–151, 2017.
DOI: 10.1007/978-3-319-59288-6_13

algorithm for streaming time series would be a useful tool for other data mining tasks, for instance:

1. The similarity search and pattern recognition in time series first need several "primitive shapes" [6] and "frequent patterns" [7] subsequences, which can be used for the next similarity measure steps.
2. In time series **classification** tasks, the typical prototypes should be created for the predefined classes, which could be generated by the segmentation approach as a preprocessing step.
3. In time series **clustering** tasks, some renowned heuristic methods such as K-means need to use several meaningful temporal feature sequences rather than random points to improve its convergence ability [8].

At present, most existing representation-based segmentation approaches do not work well for our problems. Some methods [9, 10] process segmentation on static data sets, and may incur a high cost if applied directly to streaming data. There do exist works that address the segmentation problem for data streams [11], which is called Sliding Windows (SW) but the accuracy of the segmentation is in low level, due to the lack of a more holistic view of segmentation. To solve this problem, Keogh et al. [12] consider combining the online nature of Sliding Windows and the superiority of Bottom-Up, which is called Sliding Window and Bottom-up (SWAB). However, this algorithm treats every point of the time series equally and exists some computation redundancy. In order to speed up the efficiency of the online segmentation algorithm, Liu et al. [13] propose the Feasible Space Window (FSW) method, which introduces the concept of feasible space to find the farthest segmenting point of each segment. This method greatly enhances the efficiency of the segmentation, but the partitioned subsequences are unable to make more accurate representation than SWAB.

In this paper, we propose a novel segmentation approach based on the turning points to piecewise linear representation (PLR) for streaming time series, and our method can provide a more accurate segmentation than all of the baseline methods previously used. In summary, we have achieved the following contributions in this paper.

1. We propose an online time series segmentation algorithm called the continuous segmentation algorithm based on turning points (CS_TP) which partitions the time series by a set of temporal feature points and maintains a high similarity between the processed subsequences and the raw data.
2. We adopt two segmentation criteria to refine the FSW-based segmentation results by standing on a more holistic view with the temporal features, and propose an optimal merging algorithm to reduce computation redundancy.
3. We compare the CS_TP with other baseline methods on both real open source data sets and some kinds of the typical industry time series datasets to demonstrate the superiority of our approach.

The remainder of the paper is organized as follows. In Sect. 2 we present the related work. Section 3 describes some preliminaries. The online segmentation approach is described in detail in Sect. 4. Section 5 presents the experiment results and analyses. Finally, we conclude this paper in Sect. 6.

2 Related Work

Segmentation which supplies more accurate and compact representations of time series can be considered as a discretization problem. Piecewise Linear Representation (PLR) [14] has been one of the most widely used segmentation methods for many practical applications [15, 16], which divides a time series into segments and uses a linear function to approximate each segment.

Compared with other methods, PLR is more consistent with human visual experience. In addition, PLR has some advantages of low index dimension for storing, fast speed for calculating and similarity searching. For this, PLR method is more fit for segmenting and detecting, so we use PLR to divide time series data into linear segments, which can be subdivided into three main segmentation strategies: Top-Down (TD), Bottom-Up (BU) and Sliding Windows (SW).

1. **The PLR based on Top-Down algorithm**

The PLR based on Top Down (PLR_TD) algorithm, which starts with an unsegmented sequence and introduces one cutting point at a time, repeating this process until some stopping criteria are met, in that case, the raw time series data would be partitioned into some straight lines. According to the temporal features of time series data, Zhou et al. [17] propose a novel time series segmentation based on series importance point (PLR_SIP). However, the algorithm is mainly designed to investigate the error of segments, the error of single point is not considered in this algorithm. To solve this problem, Ji et al. [18] propose a new piecewise linear representation based on importance data point (PLR_IDP) for time series, which finds the important points by calculating the fitting error of single point and piecewise segments. As a consequence of this, the fitting error of PLR_IDP is much smaller than any other PLR_TD methods.

2. **The PLR based on Bottom-Up algorithm**

The PLR based on Bottom-Up (PLR_BU) algorithm, starting with the n-1 segments, and then, two adjacent segments are greedily merged into one by choosing the lowest cost of merging pairs, iterating this process until some stopping criteria are met. Keogh et al. [19] present an improved method, which not only produces segments, but also predetermines the number of segments represented the time series.

3. **The PLR based on Sliding Windows algorithm**

The PLR based on Sliding Window (PLR_SW) algorithm, initializing the first data point of time series as the left endpoint of a segment and then trying to find the right endpoint of a segment by sequential scanning time series data. Keogh et al. [12] consider combining the online nature of sliding window and the superiority of bottom-up, which is called Sliding Window and Bottom-up (SWAB). However, this algorithm treats every point of the time series equally and exists some computation redundancy. In order to speed up the efficiency of the online segmentation algorithm, Liu et al. [13] propose the Feasible Space Window (FSW) and Stepwise FSW (SFSW) method, which introduce the concept of feasible space to find the farthest segmenting point of each segment. FSW method greatly enhances the efficiency of the segmentation, but the partitioned subsequences are unable to make more accurate representation, due to the lack of a more holistic view of segmentation.

According to the different requirements, the PLR segmentation can also be categorized into two classes: online and offline. The offline methods segment the whole data sequence, and the online methods can continuously partition these streaming data as long as the data is continuously producing (potentially forever). We can see that the PLR_TD methods (e.g., PLR_SIP, PLR_IDP et al.) and the PLR_BU methods should be grouped into the offline segmentation approaches, the PLR_SW methods including SWAB, FSW, SFSW and so on should be grouped into the online segmentation approaches.

Through the comparative analysis of the above algorithms, the major advantage of the PLR_SW methods is their ability to continuous segmentation, which meets our requirement for streaming time series segmentation. The main problem of the PLR_SW is the accuracy of representation which can not be guaranteed, compared with its offline counterparts. The PLR_TD and PLR_BU methods can produce more accurate approximation, but the entire data set should be scanned first, which is impossible for the streaming time series segmentation.

3 Preliminaries

3.1 The Turning Points Definition

When we plan on dividing the streaming time series into some continuous subsequences, it is advisable for us to segment the stream according to their temporal features. The temporal feature is constructed by a sequence of data points and each point actually has the different influence on the variation trend. In our paper, we focus on some data points indicating the change trend of time series, which can be called as the Turning Points (TPs).

Judging from the intuition, these local maximum and minimum points seemingly can be defined as the TPs since they indicate the change in the trend of the time series which has been proposed by Yin et al. [20] on the financial time series. According to the analysis on the different types of time series data sets, we draw the conclusion: The TPs are not only composed by the local extreme points (e.g., inflection points, step points etc.) and not all local extreme points should be defined as TPs in consideration of the noise and disturbance. For such reason, we provide our definition of TPs as follow:

1. For a time series $T = (\ldots, a_i, \ldots, a_j, \ldots)$ where $a_i = (x_i, y_i)$ is a raw data point in time series. If a_i meets one of the following two inequations, it can be defined as the **Level 1 TP**:

$$y_{i-1} < y_i < y_{i+1} \; or \; y_{i-1} < y_i = y_{i+1} \; or \; y_{i-1} = y_i < y_{i+1} \tag{1}$$

$$y_{i-1} > y_i > y_{i+1} \; or \; y_{i-1} > y_i = y_{i+1} \; or \; y_{i-1} = y_i > y_{i+1} \tag{2}$$

2. For a time series $T = (. .,a_i, . ., a_j, ...)$ where $a_i = (x_i,y_i)$ and $a_j = (x_j,y_j)$ are **TP**s in the **level 1**. If a_j meets the following condition, it can be defined as the **Level 2 TP**:

$$x_j - x_i \geq \mu \, and \, \frac{|y_j - y_i|}{(|y_j| + |y_i|)/2} \geq \rho \tag{3}$$

The intuitive idea of the **Level 2** is to discard unimportant fluctuations and keep major peaks and valleys with parameter µ and ρ, which can be specified by users. In order to make our intention clearer, we select one subset of data from the Fig. 1(a) to magnify in the Fig. 1(b). As shown in Fig. 1(b), if the value µ = 3 and $\rho = 1$, two TPs in the **Level 1** (i, j) can be defined as the TPs in the **Level 2** and two TPs (k, m) can be smoothed. In other words, we can directly draw a straight line from the point i to point j so as to discard unimportant fluctuations in the streaming time series.

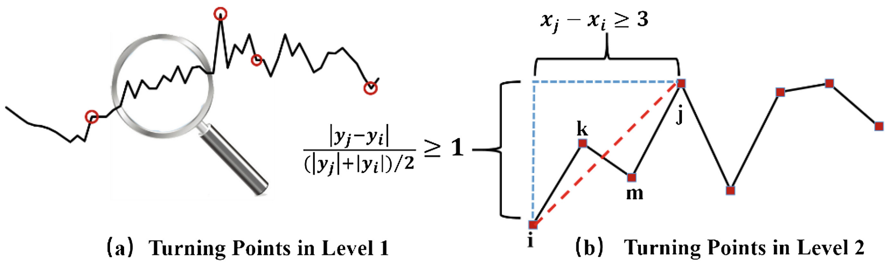

$x_j - x_i \geq 3$
$\frac{|y_j - y_i|}{(|y_j|+|y_i|)/2} \geq 1$
k
j
m
i
(a) Turning Points in Level 1
(b) Turning Points in Level 2

Fig. 1. The definitions of turning points

3.2 The Segmentation Criteria

To ensure the accuracy and the efficiency of the segmentation algorithm, we need to introduce two segmentation criteria for a potential segment.

1. The maximum error for single point (**ME_SP**)

The ME_SP is used to evaluate the fitting error of the single data in a segment. In the classic PLR_SW method, a segment continues to grow until the maximum vertical distance (MVD) for certain data point exceeds the ME_SP. The FSW method greatly simplifies the calculation of the MVD by the slope calculation (SC) [13], and therefore, we will adopt this SC measure instead of the direct MVD calculation and combine with the discovery of the TPs for our online segmentation. In order to make this concept clearly, we will provide some definitions and instructions of the SC measure (more details in reference 13) in Table 1: for a time series $T = (a_i, \ldots a_j, \ldots a_k)$ and the σ (the ME_SP specified by users) >0, if $sline(a_i, a_k)$ satisfies the inequation: $slow(a_i, a_j)$

Table 1. The definitions of the slope calculation

Definitions	Instructions
$line(a_i, a_j)$	The straight line from point a_i to a_j
$sline(a_i, a_j)$	The slope of the straight line from point a_i to a_j
$slow(a_i, a_j)$	The slope of the straight line from point a_i to $a_i - \sigma$
$sup(a_i, a_j)$	The slope of the straight line from point a_i to $a_i + \sigma$

$\leq sline(a_i, a_k) \leq sup(a_i, a_j)$, the MVD between a_j and line(a_i, a_k) will not exceed the σ(ME_SP). Along with the growth of a segment, the values of *slow*() and s*up*() are constantly changing, and we define the $maxslow_{(i:j)}$ and $minsup_{(i:j)}$ to find the maximum value of *slow*() and the minimum value of s*up*(). The definition is as follow:

$$maxslow_{(i:j)} = max_{i<t<j} slow(a_i, a_t) \tag{4}$$

$$minsup_{(i:j)} = min_{i<t<j} sup(a_i, a_t) \tag{5}$$

When such condition: $maxslow_{(i:j)} > minsup_{(i:j)}$ is satisfied, the current segmentation will be ended and then to repeat the above operation until the entire stream data has been processed.

2. The maximum error for entire segment (**ME_ES**)

The ME_ES is used to evaluate the fitting error for the entire segment, and we use this segmentation criterion to eliminate the insufficiency of only relying on the ME_SP to segment the streaming time series, in other words, the segmentation criterion ME_SP only guarantees the fitting error of each point under certain threshold, but fails to control the fitting error of the entire segment in a reasonable range.

For this reason, we will introduce an additional reference condition ME_ES to guarantee the accurate representation for online segmentation.

4 Algorithm Description

In this section, we first describe our online segmentation algorithm CS_TP, and then, we will provide the time complexity analysis of our algorithm. The CS_TP algorithm can be divided into three major steps, as follow:

1. **Initial segmentation by the ME_SP and SC**

The initial segmentation can divide the streaming time series into several segments by the **SC** measure (proposed in Sect. 3) and ensure the fitting error of each point is under the ME_SP. More importantly, the initial segmentation will find all of the TPs in segment by the definition of the **TP**s (**in Level 2**), which is prepared for optimizing the results of segmentations. Figure 2(a) and (b) describe this process. The red rhombus points denote the segmenting points of three segments and the blue dots between two rhombus points denote the TPs in a segment.

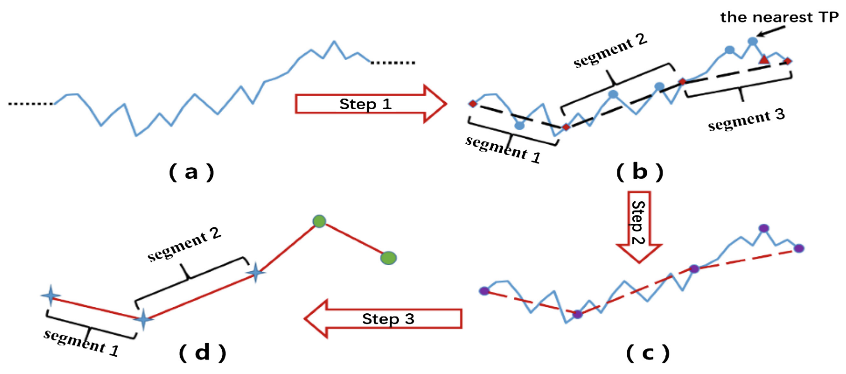


Fig. 2. The major steps of the segmentation

2. **Segmentation error evaluation by the ME_ES and TPs in Level 2**

After the above process, we will use the ME_ES to evaluate the fitting error of the segments by accumulating the fitting error of the single point until the last point of segment has been accumulated or the accumulative value exceeds the ME_ES. We will record the position of the current point and find the nearest TP, Fig. 2(b) and (c) describes this process. In these pictures, it is obvious that, in Fig. 2(b), the accumulation of the segment 1 and segment 2 do not exceed the MP_ES, however when the accumulative calculation of the segment 3 at the red triangular point, the value of the accumulation has exceeded the MP_ES, in that case, we will select the nearest TP in Fig. 2(b). The accumulative calculation will restart at the new selected point. As shown in Fig. 2(c), When the evaluation of the segmentation error is complete, we can get all of the original segmenting points and the nearest TPs (violet dots in Fig. 2(c)).

3. **Merging the subsequences by standing on holistic view**

After the above two steps, we can make use of the segmenting points to refine the FSW-based segmentation results by iteratively merging the lowest cost of the consecutive pairs of the sequences. The process of our method is similar with the SWAB methods, however we optimize the process from two aspects:

On one hand, the merging process begins with the consecutive sequences instead of connecting two adjacent points, in other words, the merging process can retain the segmentation results in the first two steps and keep the number of the segments as low as possible. On the other hand, not only the leftmost segment would be removed from the buffer, the consecutive segments behind the leftmost one can also be removed as long as the segmenting points are exactly the initial segmenting points defined in the first step. With the above optimization, the efficiency of our approach can be significantly improved compared with the SWAB. As shown in Fig. 2(d), the segment 1 and the segment 2 can be removed from the buffer immediately, and segment 3 which has been divided into two subsequences, will be merged along with the next arriving segments. The summary of notations and the pseudo code is described as follows (Table 2):

Table 2. The summary of notations

Definitions	Instructions
nsp_qu	The queue for storing the segmentation points which are process by step 1 and step 2
m_cost	The merging cost of the consecutive pair of segments
cost_priqu	The priority queue for storing the cost of the merge
calc_merg()	The calculation of the cost of the merge
rsp_qu	The queue for storing the segmentation points of segments which have been considered to complete the segmentation

Function merge_seg()

```
Input: nsp_qu, ME_ES
Output: rsp_qu
Begin
    for i=1 :2 : nsp_qu.length
        m_cost = calc_merg();
        cost_priqu.enqueue(m_cost);
    end for
        m_cost = cost_priqu.dequeue();
    while m_cost < ME_ES
        merge the two adjacent sequences;
        m_cost = calc_merg();    // update the merging cost
        cost_priqu.enqueue(m_cost);
    end while
    while nsp_qu.top() is the original segmenting point
        rsp_qu.enqueue(nsp_qu.dequeue());
    end while
    remove the segments whose segmenting points are stored in the rsp_qu queue
    return rsp_qu
End
```

According to the above description of the algorithm, in the first step, the slope calculation and the TPs discovery can be done by scanning the whole sequence in linear time, so the time cost of this operation is *O(n)*. In the second step, the evaluation of the fitting error of the entire segments should consider the number of the segmenting points and the number of the TPs. In our method the buffer can store about 6 segments (7 segmenting points), and the number of TPs (k) in one segment is far less than the number of points (n) in the segment, which can be represented by $k \ll n$, the time complexity of the second step is *O(7*k*n)*. In the third step, the operation is similar with the PLR_BU, but our method optimizes the process from two aspects, which has been described above. So the time complexity of the third step is no worse than *O(Ln)*, (L is the average segment length). So the total time complexity of our method is *O(O(n) + O(kn) + O(Ln))*, that is *O(Ln)*. Finally, we summarize the time complexities for all of the online segmentation methods mentioned in the Table 3.

Table 3. The time complexities of online segmentation methods

Algorithm	Complexity
CS_TP	*O(Ln)* L is the average segment length
SWAB	*O(Ln)* L is the average segment length
FSW	*O(Mn)* M is the number of segmenting points
SFSW	*O(Mn^2)* M is the number of segmenting points

5 Experiment and Analysis

In this section, we evaluate the performance of our online segmentation algorithm: CS_TP by compared with the SWAB, FSW and SFSW, which are nearly the most highly cited online segmentation algorithm based on the PLR up to date. We are concerned not only the representation accuracy of these techniques, but also the efficiency by using different datasets.

5.1 Dataset and Evaluation Metrics

In order to accomplish the experiment, we select some kinds of typical time series datasets which contain an average of 10,000 records from different fields, including medicine, finance, industry provided by the UCR homepage [21], and we also choose some representative industrial streaming time series including the monitoring data of Jinan municipal steam heating system (JMSHSD) from December 2013 to March 2016 (i.e.,100,532 data points), the monitoring data of Dong Fang Hong satellite (DFHSD) from January 2015 to June 2015 (i.e.,320,675 data points), which is the Chinese satellite dataset provided by China Academy of Space Technology.

In our experiment, the goal of our segmentation is to minimize the holistic representation error and obtain as few segments as possible in a more efficient manner. Therefore, in order to evaluate the segmentation for a given streaming time series, we consider the representation error, as well as the number of segments by two parameters: the ME_SP and ME_ES.

5.2 Comparison with Existing Methods

We compare our methods with the three baseline methods (FSW, SFSW, SWAB) by varying the ME_SP and ME_ES.

The SWAB [12] combines the main features of the PLR_SW and PLR_BU, which segments the streaming time series by iteratively using the BU merging method to the new sliding window and removing the leftmost segment as a finished one. The FSW and SFSW [13] belong to the PLR_SW, the former adopts the slope calculation instead of the direct MVD calculation to segment streaming time series, the latter is carried out corresponding optimization on the basis of the former. When two consecutive segments have been segmented, the SFSW would adopt backward segmenting strategy to refine the previously segmenting result.

Finally, we provide the overall performances of segmentation methods on all of the above-mentioned datasets. Before the experiments, considering the variety and complexity of the datasets, some conditions need to be defined in advance.

First of all, we adopt the Maximum Error Percentage for Single Point (MEP_SP) to substitute the absolute ME_SP, which is proposed by Liu et al. [10] and MEP_SP can eliminate the influences on different data sets by specifying the percentage of the value range on different data sets. What's more, with the change of the MEP_SP, the ME_ES in CS_TP should also be changed simultaneously to avoid that the ME_SP exceeds the ME_ES, so we will adopt the Maximum Error Percentage for Entire Segment (MEP_ES) to substitute ME_ES, and the MEP_ES is set as an integral multiple of N which is an integer greater than 2. Last but not least, we consider the result of the SWAB with MEP_SP whose value is 10% as the benchmark (set as 1), and we can normalize the results of other methods with the benchmark.

When we vary the MEP_SP from 10% to 50% in the Fig. 3(a), we can see that the CS_TP has the lowest the normalized representation error, which means this method can provide more accurate representation than other methods, and the error of all methods gradually increase with the rising of MEP_SP. However, the error of CS_TP grows more slowly than the other three methods because of the restriction of the MEP_ES, in other words, the CS_TP can guarantee a relative accurate representation even though the single point fitting error is constantly increased. In the Fig. 3(b), we can also find that the normalized number of segments of all methods gradually decrease with the rising of MEP_SP, and when the value of MEP_SP is up to 40%, the number of CS_TP is less than the SWAB because of the optimal merging step in our method.

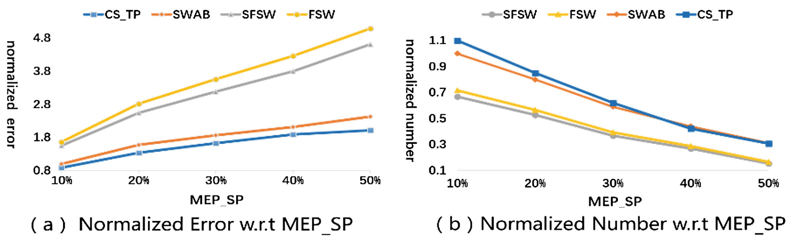
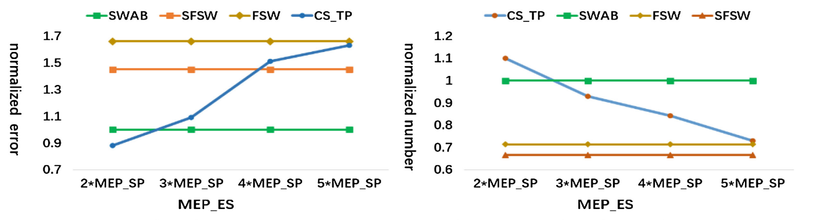


Fig. 3. The MEP_SP analysis

When we vary the MEP_ES from 2*MEP_SP to 5*MEP_SP in the Fig. 4(a), the three baseline methods are shown as three straight lines parallel to the X axis, because these methods do not use MEP_ES to segment the time series. However, the CS_TP relies on the MEP_ES to refine the result of initial segmentation. We can see that the normalized representation error of CS_TP constantly increases with the gradually loosened restriction of the MEP_SP and the error of CS_TP is close unlimitedly to the FSW. In the Fig. 4(b), the normalized number of segments of CS_TP continues to decrease with the change of the MEP_SP, finally the number of segments of CS_TP is nearly the same as the FSW, which means that the segmentation effect of CS_TP is no lower than the FSW in the worst case.



(a) Normalized Error w.r.t MEP_ES (b) Normalized Number w.r.t MEP_ES

Fig. 4. The MEP_EP analysis

At last, we provide the representation error for all methods on the above-mentioned datasets, the result of the SWAB is set as the benchmark. The results are listed in the Table 4. We can find that the CS_TP can provide more accurate segmentation than the FSW, SFSW and SWAB.

Table 4. Normalized representation error of different methods

Normalized representation error of different methods				
Data sets	Method			
	SWAB	CS_TP	FSW	SFSW
DFHSD	1	0.79	2.43	2.39
JMSHSD	1	0.89	2.21	2.17
ECG	1	0.97	1.97	2.03
Phone1	1	1.01	1.98	1.73
Powerplant	1	0.91	2.15	1.83
Wind	1	0.95	1.24	1.01
Average	1	0.92	2.00	1.86

6 Conclusion

In this paper, we propose a new segmentation algorithm (CS_TP) which performs well on segmenting the streaming time series, and holds the main characteristic of time series by refining he FSW-based segmentation result and reducing computation redundancy. The extensive numeric experiments demonstrate the advantages of our algorithm. The CS_TP can produce lower representation error to guarantee more accurate representation for online segmentation. In future, we plan to use this algorithm as a useful tool for time series classification, clustering and anomaly detection.

Acknowledgment. The authors would like to acknowledge the support provided by the Novel Software Technology Project(KFKT2015B02) and the Science & Technology Development Project of Shandong Province (2015GGX101009).

References

1. Lin, J., Keogh, E., Lonardi, S.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of ACM SIGMOD (2003)
2. Chandra, R.: Competition and collaboration in cooperative coevolution of Elman recurrent neural networks for time-series prediction. IEEE Trans. Neural Netw. Learn. Syst. **26**(12), 3123–3136 (2015)
3. Jamali, S., Jönsson, P., Eklundh, L., Ardö, J., Seaquist, J.: Detecting changes in vegetation trends using time series segmentation. Remote Sens. Environ. **156**, 182–195 (2015)
4. Palpanas, T., Vlachos, M., Keogh, E.: Online amnesic approximation of streaming time series. In: Proceedings of IEEE ICDE 2004 (2004)
5. Luo, G., Yi, K., Cheng, S.W., Li, Z., Fan, W., He, C., Mu, Y.: Piecewise linear approximation of streaming time series data with max-error guarantees. In Proceedings of IEEE ICDE 2015(2015)
6. Chiu, B., Keogh, E., Lonardi, S.: Probabilistic discovery of time series motifs proceedings. In: Proceedings of ACM SIGKDD 2003 (2003)
7. Lin, J., Keogh, E., Patel, P., Lonardi, S.: Finding motifs in time series. In: Proceedings of ACM SIGKDD 2002 (2002)
8. Fayyad, U., Reina, C., Bradley, P.: Initialization of iterative refinement clustering algorithms. In: Proceedings of ACM SIGKDD 1998 (1998)
9. Yi, B., Faloutsos, B.: Fast time sequence indexing for arbitrary Lp-norms. In: Proceedings of VLDB International Conference 2000 (2000)
10. Lazaridis, I., Mehrotra, S.: Capturing sensor-generated time series with quality guarantees. In: Proceedings of IEEE ICDE 2003 (2003)
11. Wang, C., Wang, S.: Supporting content-based searches on time Series via approximation. In: Proceedings of IEEE SSDBM 2000 (2000)
12. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An online algorithm for segmenting time series. In: Proceedings of IEEE ICDM 2001 (2001)
13. Liu, X., Lin, X., Wang, H.: Novel online methods for time series segmentation. TKDE **20** (12), 1616–1626 (2008)
14. Shatkay, H., Zdonik, S.B.: Approximate queries and representations for large data sequences. In: Proceedings of IEEE ICDE 1996 (1996)
15. Chen, Y., Nascimento, M.A., Ooi, B.C., Tung, A.K.H.: SpADe: on shape-based pattern detection in streaming time series. In: Proceedings of IEEE ICDE 2007 (2007)
16. Li, Q., Lopez, I.F.V., Moon, B.: Skyline index for time series data. IEEE Trans. Knowl. Data Eng. **16**(6), 669–684 (2004)
17. Zhou, D.: Time Series Segmentation Based on Series Importance Point. Computer Engineering, 2008(34) (2008)
18. Ji, C., Liu, S., et al.: A piecewise linear representation method based on importance data points for time series data. In: Proceedings of IEEE CSCWD 2016 (2016)
19. Keogh, E., et al.: Fast similarity search in the presence of longitudinal scaling in time series databases. In: Proceedings of IEEE ICTAI 1997 (1997)
20. Yin, J., Si, Y., Gong, Z.: Financial time series segmentation based on turning points. In: IEEE ICSSE 2011 (2011)
21. Keogh, E., Folias, T.: The UCR Time Series Data Mining Archive, Computer Science and Engineering Department, University of California (2002). www.cs.ucr.edu/～eamonn/TSDMA/index.html