Proceedings of the 10th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2008), Turin, Italy, 1-5 September, 2008

Continuous Trend-Based Clustering in Data Streams*

Maria Kontaki, Apostolos N. Papadopoulos, and Yannis Manolopoulos

Department of Informatics, Aristotle University
54124 Thessaloniki, GREECE
{kontaki,apostol,manolopo}@delab.csd.auth.gr

Abstract. Trend analysis of time series is an important problem since trend identification enables the prediction of the near future. In streaming time series the problem is more challenging due to the dynamic nature of the data. In this paper, we propose a method to continuously clustering a number of streaming time series based on their trend characteristics. Each streaming time series is transformed to a vector by means of the Piecewise Linear Approximation (PLA) technique. The PLA vector comprises pairs of values (timestamp, trend) denoting the starting time of the trend and the type of the trend (either UP or DOWN) respectively. A distance metric for PLA vectors is introduced. We propose split and merge criteria to continuously update the clustering information. Moreover, the proposed method handles outliers. Performance evaluation results, based on real-life and synthetic data sets, show the efficiency and scalability of the proposed scheme.

1 Introduction

The study of query processing and data mining techniques for data stream processing has recently attracted the interest of the research community [3, 6], due to the fact that many applications deal with data that change frequently with respect to time. Examples of such application domains are network monitoring, financial data analysis, sensor networks, to name a few.

A class of algorithms for stream processing focuses on the recent past of data streams by applying a *sliding window*. In this way, only the last W values of each streaming time series is considered for query processing, whereas older values are considered obsolete and they are not taken into account. As it is illustrated in Figure 1, streams that are non-similar for a window of length W (left), may be similar if the window is shifted in the time axis (right). Note that, in a streaming time-series data values are ordered with respect to the arrival time. New values are appended at the end of the series.

Trend analysis has been used in the past in static and streaming time series [9, 12, 8]. We use trends as a base to cluster streaming time series for two reasons. First, the trend is an important characteristic of a streaming time series.

* Research supported by the PENED 2003 program, funded by the GSRT, Ministry of Development, GREECE.

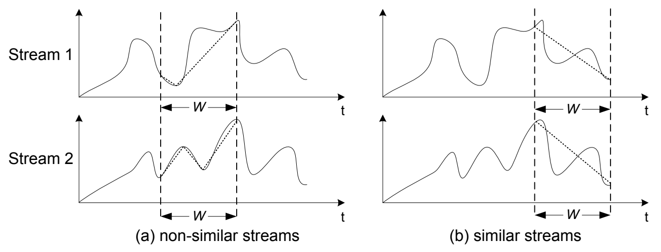


Fig. 1. Similarity using a sliding window of length W.

In several applications the way that stream values are modified is considered important, since useful conclusions can be drawn. For example, in a stock data monitoring system it is important to know which stocks have an increasing trend and which ones have a decreasing one. Second, trend-based representation of time series is closer to human intuition. In the literature, many papers [5] use the values of the data streams and a distance function, like Euclidean distance, to cluster streams. Although the distance between a pair of streams may be large, the streams may be considered similar, if their plots are examined. Thus, distance functions on raw values are not always appropriate to cluster or to classify objects.

In this paper, we focus on the problem of continuous clustering of streaming time series based on the trends of the series as time progresses. The sliding window model is used for the clustering, i.e., the last W values of each stream are considered. Each streaming time series is represented by a Piecewise Linear Approximation (PLA). The PLA of a stream is calculated based on incremental trend identification. An appropriate distance function is introduced to quantify the dissimilarity between PLAs.

Recently, a number of methods have been proposed to attack the problem of data stream clustering [1, 5]. The fundamental characteristic of the proposed methods is that they attack the problem of incremental clustering the data values of a single streaming time series, whereas our work focuses on incremental streaming time series clustering using multiple streams.

The majority of the aforementioned contributions apply variations of k-median clustering technique and therefore, the desired number of clusters must be specified. Our clustering algorithm automatically detects the number of clusters, by using the proposed *split* and *merge* criteria. As time progresses, the values, and probably the trends of streaming time series, are modified. It is possible, a split of a cluster in two different new clusters to be necessary to capture the clustering information. The proposed split criterion identifies such situations. In addition, the proposed merge criterion identifies a possible merge between two different clusters. Moreover, the proposed method handles outliers. The contributions of our work are summarized as follows:

– The PLA technique is used based on incremental trend identification, which enables the continuous representation of the time series trends.

– A distance function between PLAs is introduced.
– Continuous trend-based clustering is supported. Split and merge criteria are proposed in order to automatically detect the number of clusters.

The rest of the article is organized as follows. In Section 2, we discuss the incremental trend identification process. Section 3 presents the proposed method for continuous clustering, whereas Section 4 reports the experimental results based on real-life and synthetic data sets. Finally, Section 5 concludes the work.

2 Trend Detection

In this section, we study the problem of the incremental determination of each stream synopsis, to enable stream clustering based on trends. The basic symbols, used throughout the study, are summarized in Table 1.

Trend detection has been extensively studied in statistics and related disciplines [4, 7]. In fact, there are several indicators that can be used to determine trend in a time series. Among the various approaches we choose to use the TRIX indicator [7] which is computed by means of a triple moving average on the raw stream data. We note that before trend analysis is performed, a smoothing process should be applied towards removing noise and producing a smoother curve, revealing the time series trend for a specific time interval. This smoothing is facilitated by means of the TRIX indicator, which is based on a triple exponential moving average (EMA) calculation of the logarithm of the time series values.

The EMA of period p over a streaming time series S is calculated by means of the following formula:

$$EMA_p(t) = EMA_p(t-1) + \frac{2}{1+p} \cdot (S(t) - EMA_p(t-1)) \qquad (1)$$

Symbol	**Description**
S, S_i	a streaming time series
PLA_x	PLA of streaming time series S_x
$PLA(i)$, $PLA_x(i)$	the i-th segment of a PLA
$PLA(i).t_{start}$,$PLA(i).t_{end}$	the starting and ending time of segment $PLA(i)$
$PLA(i).v_{start}$,$PLA(i).v_{end}$	the values of the starting and ending time of $PLA(i)$
$PLA(i).slope$	the slope of segment $PLA(i)$
cs, cs_i	a common segment between two PLAs
C, C_i	a cluster
$C.n$, $C_i.n$	the number of streaming time series of a cluster
$centroid_i$	the centroid of cluster C_i
$C.avg$, $C_i.avg$	the average DPLA distance of the streaming time series of the cluster and its centroid
nC_i	the nearest cluster of cluster C_i
W	sliding window length

Table 1. Basic notations used throughout the study.

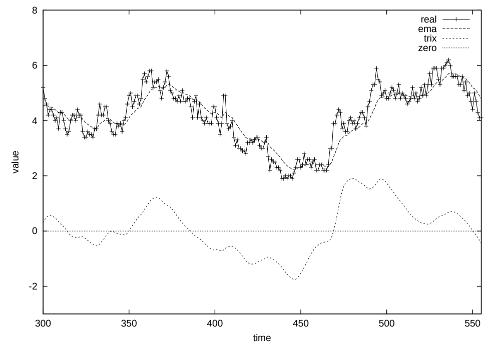


Fig. 2. Example of a time series and the corresponding $TRIX(t)$ signal.

The TRIX indicator of period p over a streaming time series S is calculated by means of the following formula:

$$TRIX(t) = 100 \cdot \frac{EMA3_p(t) - EMA3_p(t-1)}{EMA3_p(t-1)} \tag{2}$$

where $EMA3_p$ is a signal generated by the application of a triple exponential moving average of the input time series.

The signal $TRIX(t)$ oscillates around the zero line. Whenever $TRIX(t)$ crosses the zero line, it is an indication of trend change. This is exactly what we need in order to perform a trend representation of an input time series. Figure 2 illustrates an example. Note that the zero line is crossed by the $TRIX(t)$ signal, whenever there is a trend change in the input signal. Figure 2 also depicts the smoothing achieved by the application of the exponential moving average.

Definition 1
The PLA representation of a streaming time series S for a time interval of W values is a sequence of at most W-1 pairs of the form $(t, trend)$, where t denotes the starting time of the segment and *trend* denotes the trend of the stream in the specified segment (UP or DOWN).

Each time a new value arrives, the PLA is updated. Three operations (ADD, UPDATE and EXPIRE) are implemented to support incremental computation of PLAs. The ADD operation is applied when a trend change is detected and adds a new PLA point. The UPDATE operation is applied when the trend is stable and updates the timestamp of the last PLA point. The EXPIRE operation deletes the first PLA point when the first segment of the PLA expires.

3 Continuous Clustering

3.1 Distance Function

The literature is rich in distance metrics for time series. The most popular family of distance functions is the L_p norm, which is known as city-block or Manhattan norm when p=1 and Euclidean norm when p=2. A significant limitation of L_p norms is that they require the time series to have equal length. In our proposal, we compute distances between the PLAs of streaming time series, which may have different lengths. Therefore, L_p norm distance metrics cannot be used. In order to express similarity between time series of different lengths, other more sophisticated distance measures have been proposed. One such distance measure is Time Warping (TW) that allows time series to be stretched along the time axis. The disadvantage of TW is that it is computationally expensive and therefore, its use is impractical in a streaming scenario. In [10], an incremental computation of TW has been proposed, but it is limited in the computation of the distance between a static time series and a streaming time series, thus it is not suitable for our problem.

We propose the $DPLA$ distance function to overcome the above shortcomings. $DPLA$ splits the PLAs in common segments and computes the distance between each segment. The sum of distances of all segments gives the overall distance between two PLAs. A distance function for PLAs should take into account specific characteristics of the time series: 1) the trend of the segment: segments with different trends should have higher distance than segments with similar trends, and 2) the length of the segment: long segments should influence more the distance than short ones.

Before we proceed with the definition of $DPLA$, let us define the *slope* of a segment. Assume the i-th segment of a PLA starting at time $PLA(i).t_{start}$ and ending at time $PLA(i).t_{end}$. The values of $PLA(i)$ at the start and the end point are denoted as $PLA(i).v_{start}$ and $PLA(i).v_{end}$ respectively.

Definition 2
The *slope* of a segment $PLA(i)$ is the fraction of the difference of the values of the segment to the length of the segment:

$$PLA(i).slope = \frac{PLA(i).v_{end} - PLA(i).v_{start}}{PLA(i).t_{end} - PLA(i).t_{start}} \tag{3}$$

Generally, PLAs have a different number of segments, each one of different length. Thus, in order to compare two PLAs, we use the notion of *common segment.* A common segment of PLA_x and PLA_y is defined between $max(PLA_x(i).t_{start}, PLA_y(j).t_{start})$ and $min(PLA_x(i).t_{end}, PLA_y(j).t_{end})$, where i and j are initialized to 1 and assume values up to the number of segments of PLA_x and PLA_y respectively.

For example, assume the two PLAs of Figure 3. We start with $i = j = 1$. The first common segment is defined by the maximum starting timestamp (t_1) and the minimum ending timestamp (t_2). Since we have reached the ending point of the segment belonging to PLA_2, we increase j by one. We inspect now the first segment of PLA_1 ($i = 1$) and the second segment of PLA_2 ($j = 2$). By observing Figure 3 we realize that the next common segment of PLA_1 and PLA_2 is defined by the timestamps t_2 and t_3. This process continues until we reach the end of the PLAs.

The distance in a common segment cs defined by the i-th segment of the first PLA and the j-th segment of the second PLA is given by the following formula:

$$Dcs = |PLA_x(i).slope - PLA_y(j).slope| \cdot (cs.t_{end} - cs.t_{start}) \tag{4}$$

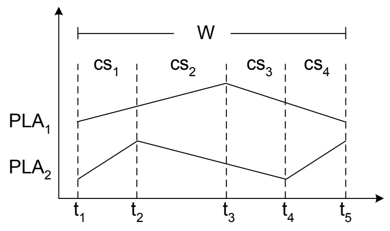


Fig. 3. Common segments of PLAs.

Definition 3
The distance between two PLAs PLA_x and PLA_y with n common segments is given by the sum of distances of common segments:

$$DPLA(PLA_x, PLA_y) = \sum_{i=0}^{n} Dcs_i \tag{5}$$

Notice that $DPLA$ function takes into account both the trend and the length of the segment and it can be computed incrementally.

3.2 Clustering Algorithm

Each cluster C_i has an id and a centroid which is the PLA of a streaming time series belonging to this cluster. Moreover, a cluster C stores a $C.n \times C.n$ matrix with the distances of streaming time series of the cluster. A streaming time series S_x belongs to cluster C_i, if: $\forall j \neq i,\ DPLA(PLA_x, centroid_i) \leq DPLA(PLA_x, centroid_j)$. Additionally, we keep a two-dimensional matrix with the distances of the centroids of all clusters.

First, we present the merge criterion. The average distance $C_i.avgD$ of a cluster C_i is the average $DPLA$ distance of the streaming time series of the cluster and its centroid.

Merge criterion
Two clusters C_i and C_j are merged if the sum of their average distances is higher than the half of the distance between their centroids:

$$c_i.avgD + c_j.avgD > DPLA(centroid_i, centroid_j)/2 \tag{6}$$

To merge two clusters, we have to decide the centroid of the new cluster. We compute the distances of all PLAs of the two clusters with their centroids. The PLA which has the minimum sum of these two distances is chosen as the centroid of the new cluster.

The split criterion is more complicated. For each PLA_x of cluster C_i, we partition all other PLAs in two subsets A and B. Subset A comprises the PLAs that are close to PLA_x and subset B comprises the remaining PLAs. We separate close from distant streaming time series by using a threshold. Therefore, PLAs that their distance to PLA_x is below the threshold are assumed to be close to PLA_x. In our experiments, this threshold is set to half of the maximum distance of PLA_x to all other PLAs of C_i. The average distance between PLA_x and PLAs belonging to subset A is denoted as $PLA_x.close$, whereas the average distance between PLA_x and PLAs belonging to subset B is denoted as $PLA_x.distant$.

Split criterion
A cluster C_i splits in two different clusters if:

$$\frac{1}{C_i.n}\sum_{x=1}^{C_i.n}\frac{PLA_x.distant - PLA_x.close}{max(PLA_x.distant, PLA_x.close)} > \delta \tag{7}$$

The above definition tries to approximate the silhouette coefficient of the new clusters. The silhouette coefficient [11] is a well-known metric for clustering evaluation, and ranges between [-1,1]. Values close to 1 indicate the existence of a good clustering whereas values below 0 indicate the absence of a clustering. The intuition for the above definition is that, if two clusters exist, then for each PLA the fraction $\frac{PLA_x.distant-PLA_x.close}{max(PLA_x.distant,PLA_x.close)}$ should be high. The parameter δ can affect the clustering significantly. Values below 0.5 may reduce the number of produced clusters, whereas large values (above 0.7) can cause consecutive splits resulting in a large number of clusters. Essentially, parameter δ controls the clustering quality and therefore, it is not necessary to change at runtime. In our experiments, we have used $\delta = 0.6$.

The centroids of the new clusters are chosen to be the PLAs that complies to the following rules: 1) $\frac{PLA.distant-PLA.close}{max(PLA.distant,PLA.close)} > \delta$ for both PLAs and 2) the $DPLA$ distance between them is the highest distance between the PLAs survived the first rule.

The outline of CTCS algorithm (<u>C</u>ontinuous <u>T</u>rend-based <u>C</u>lustering of <u>S</u>treaming time series) is depicted in Figure 4. Lines 3-10 describe the update of the clusters, whereas lines 11-17 show how CTCS adapts to the number of clusters. Notice that CTCS does not require new values for all streaming time series to update the clustering. In line 2, only the PLAs of streaming time series that have a new value, are updated.

```
Algorithm CTCS
Input
    new values of streaming time series
Output
    set of clusters

1.  updC = ∅ //set of changed clusters
2.  update PLAs of streaming time series
3.  for (each $PLA_i$)
4.      $C_k$ = cluster that $PLA_i$ belongs to
5.      find its new nearest cluster $C_j$
6.      if ($C_j \neq C_k$)
7.          move $PLA_i$ to cluster $C_j$
8.          insert $C_j$ and $C_k$ to updC
9.      end
10. end
11. for (each cluster $C_i$ of updC)
12.     remove $C_i$ from updC
13.     apply merge criterion
14.     if (merge occurs) insert the new cluster to updC
15.     apply split criterion
16.     if (a split occurs) insert the new clusters to updC
17. end
18. report the clusters;
```

Fig. 4. Outline of CTCS algorithm.

Additionally, an outlier detection scheme could be applied. A PLA belongs to a cluster, if its distance from the centroid of this cluster is minimized. Let PLA_x be a PLA belonging to cluster C_i. PLA_x will be declared as outlier if the $DPLA$ distance between PLA_x and the centroid of the cluster C_i, is higher than the $DPLA$ distance between the centroids of the cluster C_i and its nearest cluster nC_i. In Figure 4, we can apply the outlier detection before line 6. If PLA_x is an outlier, we insert it into outliers and we continue with the next PLA, omitting the computations of lines 6-9.

Table 2 shows the worst case complexity of the basic operations of CTCS algorithm. These complexities are easily determined by a careful examination of the corresponding operations. Stream update refers to the update of cluster data due to the update of a stream S. k is the current number of clusters, C_{old} is the previous cluster containing S, whereas C_{new} is the new cluster containing

operation	worst case complexity
PLA update	$O(1)$
stream update	$O(k) + O(C_{old}.n) + O(C_{new}.n)$
split test	$O(C.n)$
split process	$O((C.n)^2) + O(k)$
merge test	$O(1)$
merge process	$O((C_1.n + C_2.n)^2) + O(k)$

Table 2. Complexity analysis of CTCS

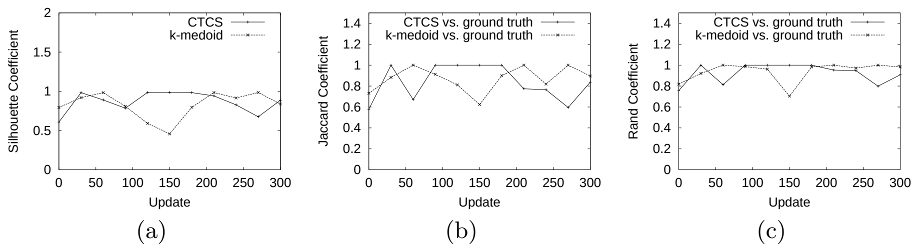


Fig. 5. Quality comparison: a) Silhouette coefficient, b) Jaccard coefficient and c) Rand coefficient for SYNTH.

S. Notice that, split and merge processes have a quadratic complexity on the number of streams per cluster and therefore, they are more computationally intensive. However, these two operations are executed less frequently than the rest, and thus the overall cost is not affected significantly.

4 Performance Study

In this section, we report the experimental results. We have conducted a series of experiments to evaluate the performance of the proposed method. Algorithm k-medoid is used as a competitor. k-medoid is modified to use the proposed distance function to handle PLAs of streaming time series. Notice that incremental implementations of k-medoid are not applicable due to the evolution of clusters, i.e., the number of clusters varies over time. However, in favor to k-medoid, we assume that the number of clusters is known, and we seed the algorithm with the previously determined medoids. All methods are implemented in C++ and the experiments have been conducted on a Pentium IV system at 3.0GHz, with 1GB of main memory running Windows XP.

We use both real and synthetic data sets. STOCK contains stock prices obtained from http://finance.yahoo.com. The data set consists of 500 time series, and the maximum length of each one is set to 3000. SYNTH is a synthetic data set and it is used in order to evaluate the quality of our method. The data set generator takes as parameters the number of streaming time series, the size of the sliding window and the number of clusters in different time instances. In this way, the number of clusters are varied over time and therefore, we can validate the performance of split and merge criteria.

First, we examine the quality of the results. We use the synthetic data set which consists of 500 streams. The window size is 30. We apply 300 updates and every 30 time instances we measure the silhouette coefficient of the clustering produced by CTCS and k-medoid. Parameter k of k-medoid is set to the actual number of clusters in each update. Figure 5(a) depicts the results. CTCS achieves silhouette coefficients more than 0.6 in all cases. Moreover, we compare the actual clustering with the clusterings of CTCS and k-medoid by using the Jaccard and

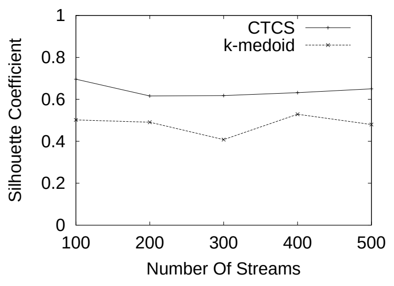
Update	0	30	60	90	120	150	180	210	240	270	300
No. Clusters	6	7	5	6	5	4	6	7	8	7	6
CTCS	3	6	6	4	5	4	6	7	8	4	6

Table 3. Number of clusters over time (SYNTH)

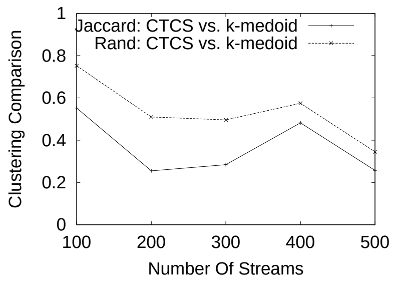
Rand coefficients [11]. These coefficients range from 0 up to 1. Values close to 1 indicate high correlation between the two clusterings, whereas values close 0 indicate low correlation. Figures 5(b) and (c) depict the results for the Jaccard and Rand coefficient respectively. Jaccard and Rand coefficients are 1 in some cases which means that CTCS gives the actual clustering (ground truth).

To better comprehend the results, we study the number of clusters that CTCS detects. Table 3 shows the number of actual clusters and the number of clusters determined by CTCS. Associating the results of Figure 5 and Table 3, we observe that when CTCS detects the number of clusters, the silhouette coefficient of the clustering is more than 0.85 and the Jaccard and Rand Coefficient is more than 0.8. In cases where CTCS misses one or two clusters, silhouette, Jaccard and Rand coefficient are good (more than 0.78, 0.67 and 0.81 respectively) which means that CTCS has recognized two clusters as one or the opposite. The results of k-medoid are as good as CTCS but notice that CTCS algorithm automatically detects the number of clusters.

In the next experiment, we examine the quality of the results in a real data set. Figure 6 shows the results with respect to the number of streams. For each run, a number of updates are applied and the average results are given. We set the parameter k of k-medoid equal to the number of clusters of CTCS in each update. As the number of streams increases, the two clusterings have less correlation (Figure 6 (b)). However the silhouette coefficient of CTCS is better than that of k-medoid and it is above 0.6 in all cases which indicates a good clustering.



(a)



(b)

Fig. 6. Quality comparison: a) Silhouette coefficient and b) Clustering comparison vs number of streams for STOCK.

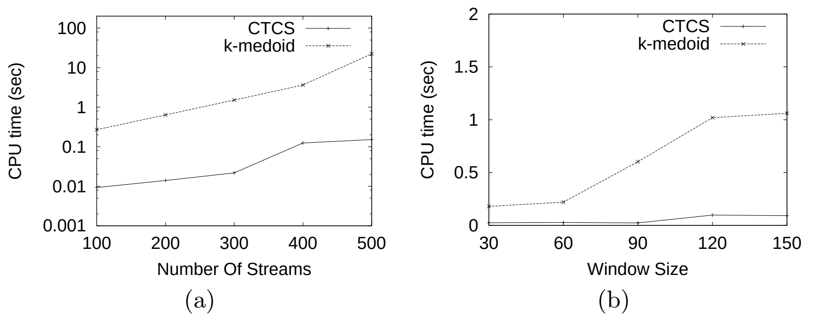


Fig. 7. CPU cost vs a) number of streams and b) window size for STOCK.

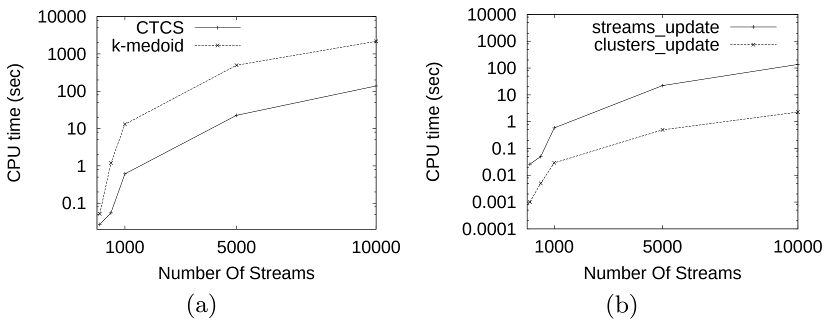


Fig. 8. CPU cost vs number of streams for SYNTH.

Next, we study the CPU cost of the proposed method with respect to the number of streaming time series and the window size (Figure 7(a) and (b) respectively). The average CPU cost is given. It is evident that CTCS outperforms k-medoid. Especially, k-medoid is highly affected by the number of streams (the CPU axis is shown in logarithmic scale), whereas CTCS can handle a large number of streams in less than 1 second.

Finally, we examine the scalability of the proposed method. Figure 8(a) depicts the CPU cost with respect to the number of streams for the SYNTH data set. The number of streams varies between 100 and 10000. CTCS outperforms k-medoid in all cases. CTCS algorithm has two basic steps: a) the incremental computation of the PLA of a stream and the update of cluster data that the stream belongs to before and after the update (streams update) and b) the continuous update of the clustering (clusters update). Figure 8(b) shows the CPU cost of the two steps separately. It is evident, that the major overhead of the method is the first step, since the clustering update requires 2.5 sec at most. Notice that in each update all the streaming time series are updated and this is the worst scenario, thus the time of the streams update is expected to be smaller in a realistic scenario.

5 Conclusions

In this paper, a novel method has been proposed towards efficient continuous clustering of streaming time series. The proposed algorithm, CTCS, uses the PLAs of the streaming time series in order to achieve a trend-based clustering. Trends are automatically detected and PLAs are updated incrementally. Moreover, a new distance function, DPLA, is proposed. Additionally, CTCS does not require the number of clusters, since split and merge criteria are used to adjust the number of clusters automatically. Performance evaluation results illustrate the superiority of the proposed method against the k-medoid algorithm concerning the CPU cost and the quality of the produced clustering. Moreover, it demonstrates the capability of the proposed method to detect the number of clusters. Future work may include the use of a distance function that obeys the triangular inequality, towards exploiting indexing schemes to improve performance.

References

1. M. Charikar, L. O'Callaghan and R. Panigrahy: "Better Streaming Algorithms for Clustering Problems", *Proceedings of STOC*, pp.30-39, 2003.
2. M. Datar, A. Gionis, P. Indyk, R. Motwani: "Maintaining stream statistics over sliding windows", *Proceedings of ACM-SIAM SODA*, pp.635-644, 2002.
3. P. Domingos, G. Hulten: "Mining High-Speed Data Streams",*Proceedings of ACM SIGKDD*, pp.71-80, 2000.
4. G. P. C. Fung, J. X. Yu, W. Lam: "News Sensitive Stock Trend Prediction", *Proceedings of PAKDD*, pp. 481-493, 2002
5. S. Guha, A. Meyerson, N. Mishra, R. Motwani, L. OCallaghan: "Clustering Data Streams: Theory and Practice", *IEEE TKDE*, Vol. 15, No. 3, pp. 515-528, May/June 2003.
6. G. Hulten, L. Spencer, P. Domingos: "Mining Time Changing Data Streams", *Proceedings of ACM KDD*, pp. 97-106, 2001.
7. J. K. Hutson: "TRIX - Triple Exponential Smoothing Oscillator", *Technical Analysis of Stocks and Commodities*, pp. 105-108, July/August 1983.
8. M. Kontaki, A.N. Papadopoulos, Y. Manolopoulos: "Continuous Trend-Based Classification of Streaming Time Series", *Proceedings of ADBIS*, pp. 294-308, Tallinn, Estonia, 2005.
9. L. Sacchi, R. Bellazzi, C. Larizza, P. Magni, T. Curk, U. Petrovic and B. Zupan: "Clustering and Classifying Gene Expressions Data through Temporal Abstractions", *Proceedings of IDAMAP*, Protaras, Cyprus, 2003.
10. Y. Sakurai, C. Faloutsos and M. Yamamuro: "Stream Monitoring Under the Time Warping Distance", *Proceedings of ICDE*, pp. 1046-1055, Istanbul, Turkey, 2007.
11. P.-N. Tan, M. Steinbach, V. Kumar: "Introduction to Data Mining", *Addison-Wesley*, 2006.
12. J.P. Yoon, Y. Luo and J. Nam: "A Bitmap Approach to Trend Clustering for Prediction in Time-Series Databases", *Proceedings of Data Mining and Knowledge Discovery: Theory, Tools, and Technology II*, Florida, USA, 2001.