Clustering over Multiple Evolving Streams by Events and Correlations

Mi-Yen Yeh, Bi-Ru Dai, and Ming-Syan Chen, *Fellow, IEEE,*

***Abstract*— In applications of multiple data streams such as stock market trading and sensor network data analysis, the clusters of streams change at different time because of the data evolution. The information of evolving cluster is valuable to support corresponding online decisions. In this paper, we present a framework for Clustering Over Multiple Evolving sTreams by CORrelations and Events, which, abbreviated as COMET-CORE, monitors the distribution of clusters over multiple data streams based on their correlation. Instead of directly clustering the multiple data streams periodically, COMET-CORE applies efficient cluster split and merge processes only when significant cluster evolution happens. Accordingly, we devise an event detection mechanism to signal the cluster adjustments. The coming streams are smoothed as sequences of end points by employing piecewise linear approximation. At the time when end points are generated, weighted correlations between streams are updated. End points are good indicators of significant change in streams, and this is a main cause of cluster evolution event. When an event occurs, through split and merge operations we can report the latest clustering results. As shown in our experimental studies, COMET-CORE can be performed effectively with good clustering quality.**

***Index Terms*— Data mining, data clustering, data streams**

I. Introduction

Research about mining in the data stream environment is flourishing recently [1][2][3][4] [5][6][7][8]. In addition to those on considering a data stream at a time, more and more emerging applications involve in monitoring multiple data streams concurrently. Such applications include online stock market trades, call detail records in telecommunication, data collection in sensor network, ATM operations in banks, to name a few. We are able to find out interesting and useful knowledge by analyzing the relationship among these multiple data streams. Therefore, mining multiple data streams has attracted an increasing amount of attention from related researchers. To discover the cross-relationship among streams, one way is to calculate the correlation between streams and report the stream pairs with high correlation [9][10][11]. Another one is to do similarity pattern query between multiple data streams [9][12]. Moreover, several works are reported on applying the clustering technique to multiple data streams [13][14][15][16].

In this paper, we present a framework for monitoring the evolution of groups of correlated data streams by the clustering technique. Explicitly, this framework will trace not only those streams becoming similar to one another but also those becoming dissimilar along with the growing of streams. Clustering is a mining technique which puts the similar objects together and separates dissimilar ones into different clusters. As a result, by clustering the streams dynamically, we can achieve the goal of monitoring the evolution of stream clusters. By observing the changes of cluster numbers and the members of each cluster, we are able to get the useful information for decision making or data management in various applications. The following example applications of clustering over multiple evolving streams:

1) Sensor network data:
   A sensor network is composed of a large number of sensors. Each sensor reads in data continuously as time advances. It is important to know the interrelationships among sensors. By clustering these sensors into groups, the administrator of the sensor network can realize which sensors work together or behave similarly in different time intervals. The sensor data clusters are helpful to awareness of intrusions or abnormalities.
2) Automatic stock exchange monitoring system:
   In the stock market, the price of each stock may vary from time to time and some stocks tend to rise and fall concurrently in some time intervals. The stock monitoring system can show which streams are in the same group and have similar behavior. From such evolving streams, the investors would like to buy a proper set of streams to maximize the profit. According to clusters provided, investors are able to choose a combination of several groups of stocks to reduce the risk of investment.

In addition to the above applications, the clustering framework can also be applied to other applications such as mining gene expression data, protein sequences, climate data, etc.

In [14], an online data summarization framework is designed for offline clustering on multiple data streams when users submit requests. In contrast, we want to provide in this paper a more real-time and automatic system which performs online clustering. The system will report the revolution of clusters as time advances. To achieve this goal, one intuitive solution is to re-cluster these data streams periodically. At the pre-determined time point, streams are updated and clustered directly with an existing clustering algorithm. However, due to the large stream number and the huge data volume, re-clustering streams is very costly. Furthermore, periodical clustering is not able to cope with the data streams with different evolving speeds. If the values of data streams are relatively steady, most of the clustering tasks are unnecessary since the resulting clusters are likely to remain the same. On the other hand, if the values of data streams are relatively fluctuant, we may lose some cluster information when the fixed time period is too long. Concluding from above issues, we need a solution which is able to perform clustering whenever it is necessary.

M.-Y. Yeh is with the Department of Electrical Engineering, National Taiwan University, No. 1, Sec. 4, Roosevelt Road, Taipei, Taiwan, ROC. E-mail: miyen@arbor.ee.ntu.edu.tw

B.-R. Dai is with the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, No. 43, Sec.4, Keelung Road.,Taipei, Taiwan, ROC. E-mail: brdai@csie.ntust.edu.tw

M.-S. Chen is with the Department of Electrical Engineering and the Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan, ROC. E-mail: mschen@cc.ee.ntu.edu.tw

Digital Object Indentifier 10.1109/TKDE.2007.1066

Consequently, a framework named *Clustering Over Multiple Evolving sTreams by CORrelations and Events*, abbreviated as *COMET-CORE*, is proposed in this paper.

For generality, we consider the data on the numerical domain. Our work can be easily extended to the applications with categorical data via proper data transformation. Initially, the streams are divided into several clusters by applying any traditional clustering method. In fact, we can also apply our merge operation, which will be introduced later, to obtain initial clusters. Each stream is growing and evolving when new data points come in. Hence a group of streams may be similar at this moment but become dissimilar to one another later. In order to capture the significant changes of each stream, we use continuous piecewise linear line segments to approximate the original data stream. There are two main reasons to adopt the piecewise linear approximation. Explicitly, the piecewise linear approximation can not only be performed in real time as the data comes in, but also be able to capture the trend of data. Two line segments with different slopes are connected by an end point. The end point represents the significant trend change point of the streaming data. If a stream in a cluster has a significant change, it is possible to cause the split of this cluster. As a result, we can regard each end point of a stream as a "trigger" of the cluster evolution, and call the stream which has a newly encountered end point as "*trigger stream*." For clusters which have trigger streams, the similarities between trigger streams and other streams in the same clusters are updated incrementally. Here we use correlation as our similarity measurement. Two different streams may vary at different numerical level but with similar trends or shapes. Also, it is not easy to normalize all data in advance since in evolving streaming environment the coming data is unknown. Hence, correlations between streams are used as the similarity measure to deal with the aforementioned issues. Correlation is a standard statistical measurement which shows the degree of association and works best with linear relationships between two random variables. This property perfectly meets our requirement. Furthermore, consider the preference for more recent data than obsolete one, a weight factor will be added in the calculation of correlations. In the later section, the way of updating weighted correlation between corresponding streams at necessary time will be described in detail. In the process of similarity updates, if we detect that the weighted correlations related to trigger streams in clusters are below a given threshold, we say an "event" is detected. An event is a signal for the system to make necessary cluster modifications. When an event is found via the event detection mechanism, clusters will be split according to trigger streams. Then, a procedure for checking whether there exist clusters being close enough to be merged together is activated. Since the split and merge processes are very efficient, the event processing procedure is able to handle thousands of streams concurrently.

Consequently, the main contributions of the COMET-CORE framework are as follows:

1) We propose a novel online clustering framework COMET-CORE over multiple data streams. COMET-CORE is efficient in monitoring the relationships between streams concurrently. The system reports the clustering results whenever the clusters encounter significant changes.
2) We devise a weighted correlation measurement for incrementally updating both the similarities among summarized streams and the similarities among clusters online.
3) We propose an efficient split and merge algorithm to perform the cluster modification with good clustering quality.

The remainder of this paper is organized as follows. Preliminaries are given in Section II which includes related studies and the problem model. In Section III, the data summarization method is described. The similarity measurement in COMET-CORE is discussed in Section IV. In Section V, the event detection mechanism and the strategy of modifying the clusters with the split and merge processes are described. The detailed algorithm of event-driven clustering is also provided. Section VI presents the experimental results and finally this paper concludes with Section VII.

II. Preliminaries

A. Related Work

Various research works have been reported to deal with clustering on one data stream [6][8][17][18][19][20]. However, more and more applications will be modeled better if multiple data streams are employed. There is an increasing number of works on dealing with clustering of multiple data streams such as [13][14][15][16]. Among the above works, a COD framework [14] is provided to do online data summarization and clustering offline when users give queries. As for online clustering, without providing details, the work [16] briefly describes a framework to continuously report clusters within the given distance threshold. Another work [13] discusses clustering over parallel data streams. It summarizes the data streams with the Discrete Fourier Transform and adopts weighted Euclidean distance as the distance measurement. By applying a sliding window on streams, it reports clustering results only of the current window. The work [15] provides a time-series system for whole clustering by incrementally constructing a hierarchy of clusters from a divisive point of view. It keeps performing clustering every time when fixed data points of every time series are collected. The scheme of [15] also utilizes the periodical way of checking cluster evolutions. The cluster split or merge processes may be a waste of time if the clusters remain almost the same. On the other hand, the cluster information can be missed due to the fast changing rate of data. In conclusion, to the best of our knowledge, none of the proposed frameworks can achieve the goal of reporting cluster evolutions dynamically with efficient cluster split and merge processes triggered by events.

B. Problem Model

Given an integer n, an n-stream set is denoted as $\Gamma = \{S_1, S_2, ..., S_n\}$ where S_i is the i^{th} stream. A data stream S_i can be represented as $S_i[1, ..., t, ...]$ where $S_i[t]$ is the data value of stream S_i arriving at time t. The data points of each stream arrive continuously and the speed of the growth of data points is rapid. The objective of this paper is that given a set of data streams Γ and the threshold parameters, events of stream clusters are online monitored. Due to the large volume of data streams, we need to summarize each stream. Each stream is represented as a series of end points, and the summary of stream S_i is denoted as $\widehat{S}_i$. The event detection mechanism and clustering task are both based on the summary data. When events occur, cluster modifications will be performed instead of re-clustering all streams and the latest clustering results are reported.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

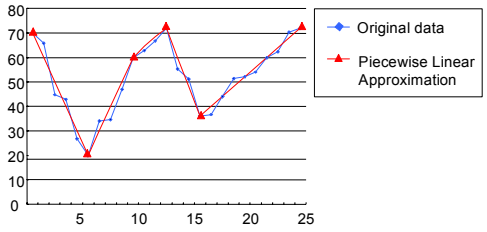


Fig. 1. The data stream is smoothed by a sequence of line segments.

III. Data Summarization

The volume of data in streaming environment is very huge and possibly infinite. As a result, we can only store the summary of data seen thus far. In COMET-CORE, the piecewise linear approximation is used as the summarization method for three reasons. First, in our model, the linear relationships between streams are concerned. Second, in the data streaming environment, we need an online and single linear scan algorithm to process the data. There indeed exists online and single scan algorithm for piecewise linear approximation [21]. Thirdly, we want to detect the significant change of streams during the process of doing summarization. An end point between two approximation line segments is a good indicator of the significant change of a stream [22]. Since the purpose of our framework is to online report the cluster evolutions, we believe that the significant change of streams is a main cause of significant change of stream clusters.

There are some related works about using piecewise linear line segments as the data representation [23][24][25]. Most of the prior works consider data with a fixed length and scan the data multiple times off-line to obtain the best fit line segments. As mentioned in [21], such methods include bottom-up and top-down ways, whose practicality in stream environments needs further justifications. To adapt the characteristics of streaming data, we need an online piecewise linear approximation algorithm. The work in [21] describes the basic concept of online segmenting time series. Using sliding window techniques, we are able to smooth the original data stream as a combination of line segments in linear time. The concept of linear approximation is illustrated in Figure 1.

In Figure 2 we briefly state the generic sliding window algorithm for transforming raw data streams into the linear piecewise representation. $S_i[t_s, ..., t_e]$ denotes the sub-stream from time t_s to t_e of the raw data stream S_i. It is approximated by line segment denoted as $S_i^{app}[t] = a \cdot t + b$, where $t \in [t_s, t_e]$. The parameter a is the slope of the line and b is the intercept. These two parameters can be calculated as follows :

$$a = \frac{S_i[t_e] - S_i[t_s]}{t_e - t_s},$$

$$b = S_i[t_e] - a \cdot t_e = S_i[t_s] - a \cdot t_s.$$

The two parameters in the algorithm, a_i^{now} and b_i^{now}, are the a and b of the latest approximated line segment.

In this piecewise linear algorithm, the *cal_error* function is used to decide when to start a new line segment. One of the feasible error measurement is the residual error illustrated in Figure 2. It computes the square sum of difference between the predicted value $S_i^{app}[t]$ and original value $S_i[t]$. It can be decomposed

Procedure: *Piecewise Linear Approximation*
Input: stream S_i, threshold δ_l
Output: end points of S_i: $\hat{S}_i$

1. $t_s = 1$;
2. while(not finish segmenting the data stream){
3. p=2;
4. calculate a_i^{now}, b_i^{now} with $S_i[t_s]$, $S_i[t_s+p]$;
5. while($cal\_error(S_i[t_s, ..., t_s+p], a_i^{now}, b_i^{now}) < \delta_l$){
6. $p = p + 1$;
7. update a_i^{now}, b_i^{now} of segment $S_i[t_s, \ldots, t_s+p]$;
8. }
9. $\hat{S}_i = \hat{S}_i \cup$ end point ($S_i[t_s+p-1]$, t_s+p-1);
10. calculate a_i^{now}, b_i^{now} with $S_i[t_s+p-1]$, $S_i[t_s+p]$;
11. $t_s = t_s + p - 1$;
12. }

Sub-routine: *cal_error*
Input: Sub-stream $S_i[t_s, ..., t_e]$, slope a, intercept b
Output: error

1. $S_i[t]$ is approximated by $at+b$, where $t \in [t_s, t_e]$;
2. error = $\sum_{t=t_s}^{t_e} (S_i[t] - S_i^{app}[t])^2$;
3. return error;

Fig. 2. Piecewise Linear Approximation Algorithm

into the combination of the summations of t, $S_i[t]$,and $tS_i[t]$ and calculated incrementally:

$$\begin{aligned} error &= \sum_{t=t_s}^{t_e} (S_i[t] - S_i^{app}[t])^2 \\ &= \sum_{t=t_s}^{t_e} (S_i[t] - (a \cdot t + b))^2, \end{aligned}$$

where a is the slope and b is the intercept of the current line segment. This equation describes an kind of absolute error, however, in streaming data environment it may not be easy to give a proper absolute error threshold. In this case, we can use relative error instead. For example, we would like the residual error is less than 2% of the square sum of the original stream.

Assume that at time $t_s + p$, the error between the raw sub-stream $S_i[t_s, ..., t_s + p]$ and the approximated one $S_i^{app}[t_s, ..., t_s + p]$ exceeds δ_l. A segment which is connected by $S_i[t_s]$ and $S_i[t_s + p - 1]$ is obtained, and we record down the latest end point $S_i[t_s + p - 1]$ into the summary structure. This is illustrated in Figure 3(a). After the new end point is recorded, a new line segment is started and the latest slope a_i^{now} and intercept b_i^{now}, are temporarily updated according to $S_i[t_s + p - 1]$ and $S_i[t_s + p]$.

The sliding window method of online piecewise linear approximation is efficient because of its linear time complexity. Many variations are designed to adapt different types of data and error measurement. For example, because of the monotonically non-decreasing property of residual error, we can speed up the algorithm by extending the candidate segment k (k>1) time units each time. In work [21], the authors have proposed an approach which combines sliding window and bottom-up methods. The proposed SWAB algorithm not only takes the advantage of linear time and online fashion of sliding window method, but also

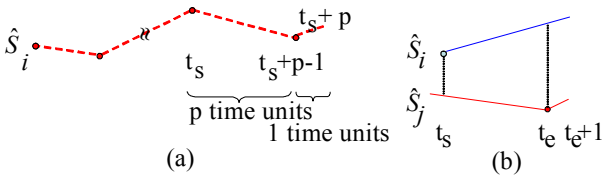


Fig. 3. (a) The streams are smoothed as a sequence of end points. (b) Illustration of incremental distance update.

owns the higher quality of bottom-up method. Another work [12] provides a three-tiered online segmentation and pruning strategy for financial data streams. No matter what kind of optimization is, the choice of error threshold is subjective and data-dependent. The trade of between efficiency and good approximation quality is also an interesting research topic itself, which is however beyond the scope of this paper.

IV. Similarity Measurement on Summary Structure

As mentioned in previous section, we smooth the stream into sequence of line segments and the end points are recorded down. The summary structure of each stream is defined as follows.

Definition 4.1: The summary structure of S_i is defined as $\widehat{S_i}$=$\{(S_i[t_{v1}], t_{v1}),\ (S_i[t_{v2}], t_{v2}),\ ...,\ (S_i[t_{vk}], t_{vk})\}$, where $S_i[t_{vk}]$ is the end point value and t_{vk} denotes the corresponding arrival time of the end point. The end points in the set are not necessarily continuous in time domain.

The two most common measures for streaming data are Euclidean distance and Correlation. The Euclidean distance between two normalized streams reflects the magnitude of difference, whereas the correlation of two streams refers to the degree of resemblance of two streams in shape or pattern. Furthermore, correlation works best with linear relationships and perfectly meets the requirement in our COMET-CORE clustering model. While correlation is a similarity measure rather than a distance measure, it can be converted to a dissimilarity measure (i.e., distance measure) through a straightforward transformation. As a result, we adopt correlation as our similarity measurement.

There are several different correlation techniques and here we focus on the most commonly used one, the Pearson correlation. The correlation coefficient between two random variables, say X, Y, is defined as follows:

$$corr(X,Y) = \frac{E(X,Y) - E(X)E(Y)}{\sigma_X \sigma_Y},$$

where σ_X, σ_Y are the standard deviations of X and Y respectively, and $E(X,Y)$, $E(X)$, $E(Y)$ are the expectations of random variables X, Y and XY.

In the multiple streaming data case, two different streams are regarded as two independent random variables. Therefore, follow the definition, the correlation between two streams S_i and S_j is:

$$corr(S_i,S_j) = \frac{\sum_t S_i[t]\cdot S_j[t] - \sum_t S_i[t] \sum_t S_j[t]}{\sqrt{\sum_t (S_i[t]-\overline{S_i})^2}\sqrt{\sum_t (S_j[t]-\overline{S_j})^2}}.$$

In most streaming applications, the recent data are usually more important compared to the aged data. To reflect the bias toward recent data, a weight function is introduced in the calculation of correlation between two streams.

Definition 4.2: Given two streams S_i, S_j and a weight function $w(t)$, the weighted correlation coefficient between these two streams is defined as

$$wcorr(S_i,S_j) = \frac{(\sum_t w(t)S_i[t]S_j[t] - \frac{\sum_t w(t)S_i[t]\sum_t w(t)S_j[t]}{\sum_t w(t)})}{\sqrt{A(S_i)\times A(S_j)}}, \quad (1)$$

where $A(S_x) = \sum_t w(t)(S_x[t])^2 - \frac{(\sum_t w(t)S_x[t])^2}{\sum_t w(t)}$, and $w(t)$ is a monotonically non-decreasing function of time index t.

$wcorr(S_i,S_j)$ is different from $corr(S_i,S_j)$ in that each term is multiplied by the weight function $w(t)$. To be more efficient, six necessary terms are kept to form a real vector for calculating equation (1).

Definition 4.3: A *WC vector* is defined as a weighted correlation vector, which is used to maintain the statistics for weighted correlation calculation between two sequences of end points. Given two streams S_i, S_j and a weight function $w(t)$, the WC vector of S_i and S_j is defined as $\overline{WC}_{S_iS_j}$=$(WC_1^{t_k}, WC_2^{t_k}, WC_3^{t_k}, WC_4^{t_k}, WC_5^{t_k}, t_k)$, where the first five tuples are accumulation values from the beginning to time t_k: $WC_1^{t_k}$=$\sum_{t=0}^{t_k} w(t)S_i[t]$, $WC_2^{t_k}$=$\sum_{t=0}^{t_k} w(t)(S_i[t])^2$, $WC_3^{t_k}$=$\sum_{t=0}^{t_k} w(t)S_i[t]S_j[t]$, $WC_4^{t_k}$=$\sum_{t=0}^{t_k} w(t)S_j[t]$, and $WC_5^{t_k}$=$\sum_{t=0}^{t_k} w(t)(S_j[t])^2$, and the last tuple t_k records the latest update time.

We set $w(t) = \alpha^{(t_{now}-t)}$, where α is a real number in (0,1] and t_{now} refers to the current time and keeps growing. Obviously, the closer the data is to the current time, the more weight they get. The α parameter controls the fading speed of data weight. When t_{now} becomes larger and larger, the weight of certain data point keeps fading as new data come in continuously.

Now, let us see how our COMET-CORE realizes the similarity measurement. Since each stream is smoothed, the calculation of the distance between two data streams can be reduced to the one between two sequences of end points. Due to the limit of linear scan of data streams, the distance should be updated in an incremental fashion. In other words, we need to update the $\overline{WC}_{S_iS_j}$ vector incrementally. Consider the case shown in Figure 3(b). Assume that $\overline{WC}_{S_iS_j}$ is updated to $((WC_m^{t_s})_{m=1\sim5}, t_s)$ at time t_s+1. At time t_e+1, $\widehat{S}_j$ is detected to have a new end point at time t_e and the WC vector should be updated. The incremental value of mth tuple of the WC vector is denoted as ΔWC_m. In the period from time t_s to t_e, S_i is approximated by $S_i^{app}[t] = a_i^{now}\cdot t + b_i^{now}$, where $t \in [t_s, t_e]$. The two parameters a_i^{now} and b_i^{now} are maintained by the piecewise linear approximation

method. For stream S_j, the line segment between time t_s and t_e is approximated by $S_j^{app}[t] = a_j \cdot t + b_j$ where $t \in [t_s, t_e]$. The two parameters a_j and b_j are calculated by the last two end points in $\widehat{S}_j$, i.e., $(S_j[t_s], t_s)$ and $(S_j[t_e], t_e)$. Then the corresponding ΔWC_m values can be computed as follows:

$$\begin{aligned}
&\Delta WC_1 \\
&= \textstyle\sum_{t=t_s+1}^{t_e} w(t) S_i^{app}[t] \\
&= \textstyle\sum_{t=t_s+1}^{t_e} w(t)(a_i^{now} t + b_i^{now}) \\
&= a_i^{now} \textstyle\sum_{t=t_s+1}^{t_e} tw(t) + b_i^{now} \sum_{t=t_s+1}^{t_e} w(t),
\end{aligned} \tag{2}$$

$$\begin{aligned}
&\Delta WC_2 \\
&= \textstyle\sum_{t=t_s+1}^{t_e} w(t)(S_i^{app}[t])^2 \\
&= \textstyle\sum_{t=t_s+1}^{t_e} w(t)(a_i^{now} t + b_i^{now})^2 \\
&= (a_i^{now})^2 \textstyle\sum_{t=t_s+1}^{t_e} t^2 w(t) \\
&+ 2 a_i^{now} b_i^{now} \textstyle\sum_{t=t_s+1}^{t_e} tw(t) \\
&+ (b_i^{now})^2 \textstyle\sum_{t=t_s+1}^{t_e} w(t),
\end{aligned} \tag{3}$$

$$\begin{aligned}
&\Delta WC_3 \\
&= \textstyle\sum_{t=t_s+1}^{t_e} w(t) S_i^{app}[t] S_j^{app}[t] \\
&= \textstyle\sum_{t=t_s+1}^{t_e} w(t)(a_i^{now} t + b_i^{now})(a_j t + b_j) \\
&= (a_i^{now} a_j) \textstyle\sum_{t=t_s+1}^{t_e} t^2 w(t) \\
&+ (a_i^{now} b_j + a_j b_i^{now}) \textstyle\sum_{t=t_s+1}^{t_e} tw(t) \\
&+ (b_i^{now} b_j) \textstyle\sum_{t=t_s+1}^{t_e} w(t).
\end{aligned} \tag{4}$$

For the incremental values of other two tuples, ΔWC_4 and ΔWC_5 are calculated the same way as equations (2) and (3). Note that parameters a and b should be replaced with a_j and b_j. By observing these equations, we can discover that they are all the linear combinations of $\sum T(t)w(t)$, where $T(t)$ can be t^2, t or some constant. Consider in the weight function and according to equation (2), the 1st tuple of $\overline{WC}_{S_iS_j}$ is updated as shown below:

$$\begin{aligned}
&WC_1^{t_e} \\
&= \textstyle\sum_{t=1}^{t_e} \alpha^{(t_e - t)} S_i^{app}[t] \\
&= \alpha^{(t_e - t_s)} \textstyle\sum_{t=1}^{t_s} \alpha^{(t_s - t)} S_i^{app}[t] \\
&\quad + \textstyle\sum_{t=t_s+1}^{t_e} \alpha^{(t_e - t)} S_i^{app}[t] \\
&= \alpha^{(t_e - t_s)} WC_1^{t_s} \\
&\quad + (a_i^{now} \textstyle\sum_{t=t_s+1}^{t_e} t\alpha^{(t_e - t)} + b_i^{now} \sum_{t=t_s+1}^{t_e} \alpha^{(t_e - t)}) \\
&= \alpha^{(t_e - t_s)} WC_1^{t_s} + \Delta WC_1.
\end{aligned}$$

Induced from above, the WC vector at time t_e is:

$$\overline{WC}_{S_iS_j} = ((\alpha^{(t_e - t_s)} WC_m^{t_s} + \Delta WC_m)_{m=1\tilde{}5},\ t_e). \tag{5}$$

Lemma 4.1: The amortized time complexity of WC vector update is $O(1)$.

Proof: From equation (5), the WC vector can always be updated by adding ΔWC_m to an existing WC vector. It is clear that this equation can be processed in constant time. Next, we observe the time complexity of computing ΔWC_m. The five terms ($\Delta WC_{1\tilde{}5}$) are updated as equation (2)(3)(4) shown and the worst-case cost of each operation is $O(n)$, where n is the length of a stream. Using the aggregate method of amortized analysis, we can obtain the amortized cost of an operation is the average: $O(n)/n = O(1)$. ■

Consequently, we can online incrementally maintain similarity between two streams efficiently by keeping the $\overline{WC}_{S_iS_j}$ vector of two stream summaries. At the end of this section, a brief example is given to illustrate the update procedure.

Example 4.1: Consider the example shown in Figure 3(b). Assume $t_s = 3$, $t_e = 6$ and let $w(t) = \alpha^{(t_{now} - t)}$, where $\alpha = 0.9$. The summaries of two streams are $\widehat{S}_i = \{(80, 1)(60, 3)\}$ and $\widehat{S}_j = \{(50, 1)(0, 6)\}$, where $\widehat{S}_j$ has a new end point $(0, 6)$ at time 6. The last distance updating time of $\widehat{S}_i$ and $\widehat{S}_j$ is $t_s = 3$. Until time 3, we have $\sum t^2 w(t) = 13.41$, $\sum tw(t) = 5.61$, and $\sum w(t) = 2.71$. $\overline{WC}_{S_iS_j}$ = ((187.8, 13194, 7560, 106.5, 4365, 2.71), 3). Therefore, at time 3, the weighted correlation of $\widehat{S}_i$ and $\widehat{S}_j$ is 1. At time 7, $\widehat{S}_j$ has a new end point (0,6). Now, the t_{now} is set to 6. The slope and the intercept for $\widehat{S}_j$ between time 4 and 6 are $a_j = -10$ and $b_j = 60$ respectively. For $\widehat{S}_i$, assume the current parameters $a_i^{now} = 10$ and $b_i^{now} = 30$. As a result, we have $\sum_4^6 t^2 w(t) = 71.46$, $\sum_4^6 tw(t) = 13.74$, and $\sum_4^6 w(t) = 2.71$. The incremental values of $\overline{WC}_{S_iS_j}$ vector in this period are $\Delta\overline{WC}_{S_iS_j}$ = ((218.7, 17829, 1854, 25.2, 414), $*$). By applying equation (5), the final $\overline{WC}_{S_iS_j}$ is: (((187.8, 13194, 7560, 106.5, 4365)$*(0.9)^{(6-3)}$+(218.7, 17829, 1854, 25.2, 414)), 6) = ((355.61, 27447.43, 7365.24, 102.84, 3596.09), 6). From this vector, the weighted correlation coefficient between $\widehat{S}_i$ and $\widehat{S}_j$ is -0.56. ■

At the end of this section, the properties of this incremental similarity update is discussed. Since the weighted correlation between two streams is updated incrementally, the last updated time of two streams is needed when every new end point is generated. This can be done by checking the arrival time of the latest two end points of each stream.

Lemma 4.2: The last updated time between two stream S_i and S_j can certainly be decided by comparing the last two end points of either stream summary.

Proof: Assume at time $(t_e + 1)$ an end point $(S_i[t_e], t_e)$ of $\widehat{S}_i$ is generated, the latest two end points of $\widehat{S}_i$ are {..., $(S_i[t_{i1}]$, $t_{i1})$, $(S_i[t_e]$, $t_e)$}. A stream $\widehat{S}_j$ has the latest two end points {..., $(S_j[t_{j1}], t_{j1}), (S_j[t_{j2}], t_{j2})$}, where $t_{j2} > t_{j1}$. If $t_{j2} = t_e$, i.e., $\widehat{S}_j$ also has an end point at time t_e, then the last updated time between $\widehat{S}_i$ and $\widehat{S}_j$ is $max(t_{i1}, t_{j1})$. If $t_{j2} < t_e$, then the last updated time between $\widehat{S}_i$ and $\widehat{S}_j$ is $max(t_{i1}, t_{j2})$. ■

It is of great advantage that according to Lemma 4.2, the number of end points should be kept for each stream is limited.

Lemma 4.3: The maximal number of end points a stream summary should be kept is 2.

Proof: According to Lemma 4.1, by checking the latest two end points between stream summary pair, we can get the last update time and then complete the similarity update. ■

If there are n streams, from the above lemma, the space for end points is bounded to $O(2n)$. This fact is satisfactory because it is independent of the length of each stream. Therefore, we can limit the space usage of summarized data efficiently.

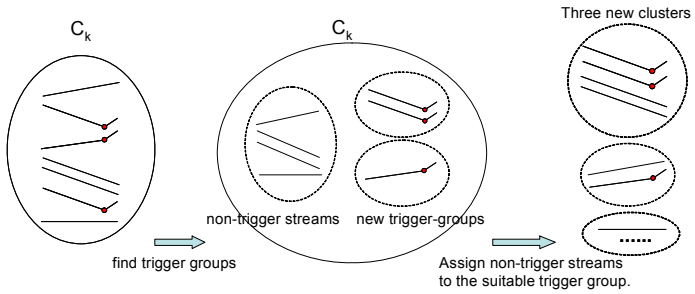


Fig. 4. Illustration of how trigger streams work.

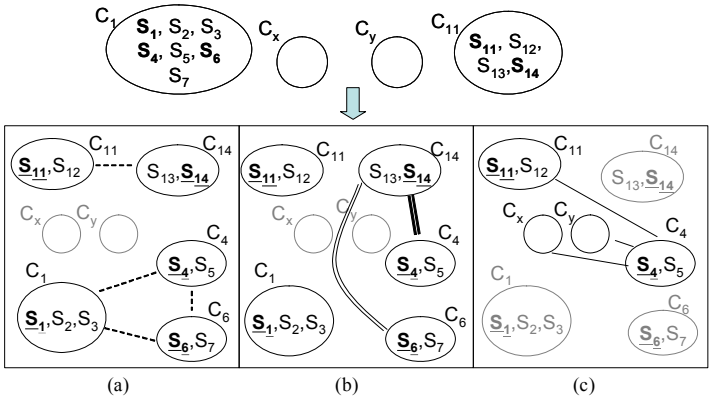


Fig. 5. The illustration of cluster split process.

V. The COMET-CORE Framework

In the beginning of this section, the brief concept of the COMET-CORE framework is given. Instead of re-clustering whole streams, COMET-CORE reports cluster results through efficient cluster splitting and merging when there are significant changes. In essence, a cluster is the set of summarized streams and all the clusters become a cluster set. The number of cluster varies during the process of event detection and stream clustering. Each cluster has a center which is simply the average of its members, i.e., the summarized streams. Therefore, the center of a cluster is also a sequence of end points. As the streams grow, the center of the cluster varies as well. In this way, the similarity between two clusters can also be maintained incrementally with a weighted correlation vector, WC vector, as mentioned in Section IV.

Definition 5.1: Assume the centers of two clusters C_i and C_j are represented by end point sequence $\widehat{S}_i$ and $\widehat{S}_j$ respectively. Then the WC vector of two clusters denoted by $\overline{WC}_{C_iC_j}$ is equal to $\overline{WC}_{S_iS_j}$. The weighted correlation between C_i and C_j denoted by $wcorr(C_i, C_j)$ is equal to $wcorr(S_i, S_j)$.

A. Event Detection

Once a stream generates a new end point, we regard it as a trigger stream. A trigger stream is a trigger of events, where the event here means that some clusters are required to be split or merged. Thus, the weighted correlation between trigger streams and other streams in the same cluster is updated to check if there are streams in the same cluster but getting dissimilar. The procedure of how trigger streams work is illustrated in Figure 4. A cluster may have more than one trigger stream. Among these trigger streams, some of them are similar at current time unit and these trigger streams should be grouped first. Given the user defined threshold δ_a, if the weighted correlation of two trigger streams are no less than this threshold, they will be grouped into the same group, which is called as *trigger group*. Clearly, if the number of trigger-group is larger than one, it means the cluster should be split. For each trigger group, a trigger stream is randomly picked out as the *representative stream* of this group. For the rest of the non-trigger streams, which are the streams in the same cluster but without new end points generated, each of them is compared to the representative trigger streams. According to the weighted correlation between itself and the representative trigger streams, the non-trigger streams will be assigned to the trigger group where the most correlated representative trigger stream is. It is possible that some of the non-trigger streams cannot fit any trigger group, i.e., the weighted correlations between the non-trigger stream itself and all the representative trigger streams are not high enough to reach the user defined threshold δ_a. In this case, these streams form a temporary cluster. Though the streams in the temporary cluster may not be highly correlated, they will be split in the future when next event is detected.

B. Split Clusters

In previous section, we introduce how trigger streams trigger the split of clusters. When new clusters are formed, the corresponding WC vectors of modified clusters should be maintained. Since re-clustering of streams is not permitted here, the new WC vectors should be approximated by existing ones. For these newly generated WC vectors, we can classify them into three types. The first two types consist of the WC vectors between two newly-generated clusters which belong to the same cluster and different clusters originally. The last type to be created is the

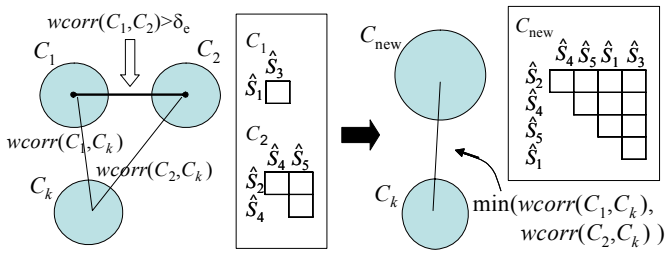


Fig. 6. Illustration of the merge process and the distances to be updated.

WC vectors between newly-generated clusters and the originally existing clusters. To make it easy to understand, let us go through the split process with Figure 5. Originally, there are four clusters, which are C_1, C_x, C_y, and C_{11}. Suppose C_1 is split into three clusters: C_1[1], C_4, and C_6, and cluster C_{11} is split into two clusters: C_{11}, and C_{14}. First of all, the WC vectors among newly generated clusters which originally belong to the same clusters are set. Since the cluster are split according to the representative trigger streams, their WC vectors are used the approximate the WC vectors of the first type WC vectors among clusters.

Type 1. The WC vector between newly generated clusters C_i and C_j, which belong to the same cluster originally, is set to be equal to the WC vector of representative trigger streams S_i and S_j:

$$\overline{WC}_{C_iC_j} = \overline{WC}_{S_iS_j},$$

where S_i and S_j are the representative trigger streams of C_i and C_j respectively. ■

In the example of Figure 5(a), the cluster C_{11} is split to C_{11} and C_{14}, so WC vector of cluster C_{11} and C_{14} is $\overline{WC}_{C_{11}C_{14}} = \overline{WC}_{S_{11}S_{14}}$, and the same, for cluster C_1 split to C_1, C_4, C_6: $\overline{WC}_{C_1C_4} = \overline{WC}_{S_1S_4}$, $\overline{WC}_{C_1C_6} = \overline{WC}_{S_1S_6}$, $\overline{WC}_{C_4C_6} = \overline{WC}_{S_4S_6}$.

Then, consider the WC vector of newly generated clusters which originally belong to different clusters.

Type 2. The WC vector between newly generated clusters C_i and C_j, which originally belong to different clusters is set as:

$$\overline{WC}_{C_iC_j} = \overline{WC}_{C_{io}C_{jo}},$$

where C_{io} and C_{jo} are the original clusters of C_i and C_j respectively. ■

This type of similarity is illustrated in Figure 5(b), where the WC vector between newly generated cluster C_4 and C_{14} is $\overline{WC}_{C_4C_{14}} = \overline{WC}_{C_1C_{11}}$, since C_4 is split from C_1 and C_{14} is split from C_{11}.

Type 3. Assume C_i is the newly generated cluster split from cluster C_{io}. The WC vector between C_i and other originally existing clusters, say C_{oo}, is set as:

$$\overline{WC}_{C_iC_{oo}} = \overline{WC}_{C_{io}C_{oo}}$$

■

[1]For simplicity, each cluster is labeled with the smallest stream number in it. Therefore, if a cluster is split into several sub-clusters, there must be one cluster owns the same label as the original one.

This type of similarity is illustrated in Figure 5(c). The WC vector of C_4 and other existing clusters are: $\overline{WC}_{C_4C_x} = \overline{WC}_{C_1C_x}$, $\overline{WC}_{C_4C_y} = \overline{WC}_{C_1C_y}$ and $\overline{WC}_{C_4C_{11}} = \overline{WC}_{C_1C_{11}}$. For other newly generated clusters C_{14} and C_6, their corresponding vectors are updated likewise.

Note that only the WC vector of type 1 is updated to the latest end point time. It is because that type 1 vector is copied directly from the WC vector of the latest representative trigger streams. For WC vectors of type 2 and type 3, whose 6th tuple t_k remain at the last updated time, they will be updated to the current end point time in the following step.

C. Update Inter-Cluster Similarity

It is assumed that the latest end point occurs at time t_e and we need to update the center of each cluster. The corresponding new end point of the center of each cluster is the average data values of each stream in that cluster at time t_e. For streams which have no end point at time t_e, their values will be approximated by a_i^{now} and b_i^{now}, i.e., $a_i^{now} \times t_e + b_i^{now}$. Then, the new end point of the cluster center can be calculated.

Once we update the center of each cluster, only the inter-cluster WC vectors whose updated time is not equal to t_e will be updated. As aforementioned in the begin of this section, since the center of a cluster can be regarded as another stream summary, the inter-cluster WC vector can be maintained in the same way as shown in section IV. Furthermore, the Lemma 4.3 also holds here. We need only preserve the latest two end points of the center of each cluster.

D. Merge Clusters

After splitting and updating the inter-cluster correlation of each cluster pair, the COMET-CORE framework checks whether there are clusters being close enough to be merged. How close can two clusters be merged is defined by a user given threshold δ_e. If the inter-cluster correlation between any two clusters is equal to or higher than the threshold δ_e, these two clusters are merged. Note that we can apply an appropriate agglomerative hierarchical clustering method in the merge process by setting the stop criteria as the threshold δ_e. In other words, the clusters will be merged repeatedly until no correlations of the existing cluster pairs is higher than or equal to δ_e. The cluster number is relatively small compared to the original number of streams, and thus the execution time is relatively less. The policy of the merge process is as follows. Consider the example shown in Figure 6. Suppose that the weighted correlation of cluster C_1 and C_2 is higher than δ_e, then they will be merged. The WC vector $\overline{WC}_{C_{new}C_k}$ where $k \neq 1$ and $k \neq 2$, is set to either $\overline{WC}_{C_1C_k}$ or $\overline{WC}_{C_2C_k}$ according to who has smaller

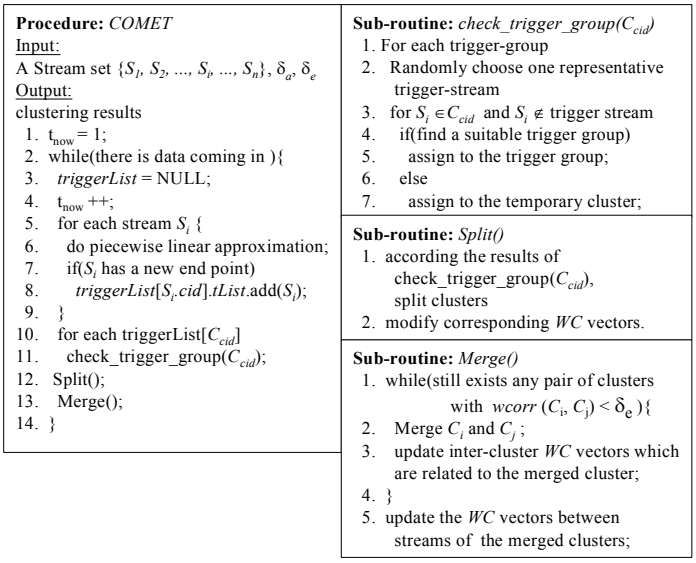


Fig. 7. The algorithm of the COMET framework.

weighted correlations, i.e., $\min(wcorr(C_1, C_k), wcorr(C_2, C_k))$. After updating the corresponding correlations, the merge process will be performed on the rest of clusters until the stop criteria is met.

Finally the algorithm of COMET-CORE is outlined in Figure 7. Note that the $triggerList$ is an array where the index is the cluster id and the corresponding item is the trigger stream list $tList$. $S_i.cid$ represents the id of the cluster to which stream S_i belongs.

E. Analysis of COMET-CORE

In this section, we give analyses of COMET-CORE on parameter usage, space complexity and efficiency.

1) Parameter analysis: In COMET-CORE, several parameters are considered. Due to the nature of invisibility of future data in streaming environment, we need some heuristics on historical data to adjust these parameters. The first one is the threshold used in piecewise linear approximation. As we mentioned earlier in section III, the error threshold used is subjective and data-dependent. In our case, we can set a reasonable error threshold according to the past experience on historical data. Then, as the streaming data come in, we can observe if the trigger (end points) is too frequent or too rare. According to the frequency, we can increase or lower the error threshold dynamically.

Another parameter in COMET-CORE is α. In streaming environment, the more recent data are more attractive and should be sampled more. When α is smaller than 1, the data are biased sampling and of course the cluster results vary. The choice of α value is optional and subjective. How this value affect the clustering results are further studied in the experimental section.

Finally, the thresholds of split and merge processes are discussed. As suggested in [26], a correlation coefficient above 0.8 is regarded as high correlation. Accordingly, the higher the threshold is, the more compact the cluster will be. Users may decide the parameter δ_a and δ_e based on the characteristic of the historical data streams for future use. In addition, without loss of generality, we can set δ_a=δ_e. Notice if δ_a is set to be equal to -1, it means only merge operation is activated. On the other hand, if δ_e is set to be 1, only split process is activated.

2) Space Analysis: In this section, space usage in COMET-CORE is discussed. As we mentioned earlier in Lemma 4.3, the maximal end points a stream should be kept is 2 and the total end points are $O(2n)$=$O(n)$. Next, consider the space usage for WC vectors. For n streams, suppose that there are p clusters and cluster C_k contains m_k streams. The space complexity of WC vectors (intra-WC vector + inter-WC vector) is:

$$\begin{aligned}&\text{space complexity}\\&= \{m_1(m_1-1)/2 + m_2(m_2-1)/2 + ...\\&+m_P(m_P-1)/2\} + \{p(p-1)/2\}\\&< m_1^2 + m_2^2 + ...m_p^2 + p^2\\&< (m_1 + m_2 + ...m_p)^2 + p^2\\&= n^2 + p^2\end{aligned}$$

It can be shown that in most of the time, the space usage is far less than $O(n^2)$. The worst case appears when all streams are in the same cluster or each stream is a cluster itself, but this is a rare case. It is noted that n, the number of stream, is much smaller than the length of endless streams.

3) Efficiency Analysis: The main processes of COMET-CORE are event detection, cluster splitting and merging. In the event-detection process, each trigger stream only updates their correlations with members in the same cluster. The update time is linear to the number of members in a cluster. Furthermore, only the clusters with trigger streams are needed to update the similarity. This saves a lot of time in contrast to re-clustering directly to streams themselves all the time.

Next, we analyze the efficiency of split process. Assume there are p clusters before the split process, and by event detection, q clusters are decided to be newly generated. According to Section V-B, there are three types of WC vector to be generated or updated. For type 1 and type 2 WC vector, they are inter-WC vectors among newly generated q clusters. Hence the number of these WC vectors is $q(q-1)/2$. For type 3 WC vector, it represents the inter-WC vector between newly generated clusters and the existing clusters. Therefore the number is pq. The cost of assigning each new WC vector is $O(1)$. Therefore, the total time

cost of split process is $O(pq+q^2)$. It is noted that when $q << p$, the time cost is reduced to $O(pq)$.

Finally, the complexity of merge process depends on the used agglomerative algorithm. As mentioned in Section V-D, because the cluster number is relatively smaller than stream number in most of the time, the time cost is low.

VI. Empirical Studies

To assess the performance of the COMET-CORE framework, we conducted all the following experiments on a PC with 3 GB RAM and 3GHz CPU which runs the Window XP Professional operating system. We compare the COMET-CORE with two other methods: one is called Basic here and the other is the ODAC algorithm proposed in [15]. The Basic algorithm does clustering once fixed number of examples is collected. The agglomerative hierarchical clustering with complete linkage is always applied directly to streams after updating accumulated statistics. The ODAC algorithm checks cluster split and aggregation periodically as well but with criteria based on Hoeffding bounds. On the contrary, COMET-CORE updates necessary statistics by triggers and adjusts cluster structure by events. All three methods use correlation-like similarity measure: COMET-CORE and Basic both use the weighted correlation as similarity measurement. The difference is that COMET-CORE computes correlations on approximated data as described in former sections while Basic method computes correlations of streams on fact. The ODAC method uses *rooted normalized one-minus-correlation*, which normalizes the one minus correlation value to [0,1], as the distance measurement.

To evaluate the clustering quality, we use a famous cluster validation technique - *Silhouette validation*[27]. The silhouette validation first assigns each stream a quality measure:

$$sil(S_i) = \frac{b(S_i) - a(S_i)}{\max\{a(S_i), b(S_i)\}},$$

where $a(S_i)$ is the average dissimilarity of stream S_i to all other streams in the same cluster and $b(S_i)$ is the average dissimilarity of stream S_i to all other streams in the closest cluster. Follow from the formula, the $sil(S_i)$ value ranges from -1 to 1. If this value is close to 1, it means that S_i is well-clustered. On the other hand, if $sil(S_i)$ is close to -1, it shows that S_i is assigned to an improper cluster. A cluster silhouette for a cluster C_k who owns m members is:

$$sil_{C_k} = \frac{\sum_{s_i \in C_k} sil(S_i)}{m}.$$

Finally, the *Global Silhouette*, *GS*, which is used to asses clustering quality in the following experiments is defined as: $GS = \frac{1}{p}\sum_1^p sil_{C_k}$, where p is the total cluster number. Silhouette Validation consider the cluster structure from the view of relationship between objects. It takes both the compactness and separation of clusters into account.

Note that in the following experiments, the distance used on computing GS value is based on fact instead of on approximated data.

A. Evaluation of COMET-CORE on Real Data

In this section, three real data sets are used to test. The first one is the weather data obtained from Temperature Data Archive of the University of Dayton[2]. From year 1995, the average daily temperatures of 290 cities around the world are collected. Each city is regarded as a data stream and each stream has 3,416 points. The second real data set is S&P 500 stock which can be obtained from a public web site[3]. We collected one-year historical data of 476 stocks. For each stock, we use high and low value for each stock to constitute 952 streams with a length of 252. Finally, the third data set is the training data set for the PDMC competition[4]. There are 9 sensors to be observed and the total observations are 189,961. Assume every stream comes in the rate of one sample per time unit. The parameter setting of three methods are as follows. The stop criterion for Basic is δ_e=0.5. For COMET-CORE, the threshold value is set as δ_a=δ_e=0.5. For ODAC, we set the confidence value as 0.05 for the hoeffding bound as suggested in [15]. Since the Basic and ODAC methods perform clustering periodically, two periods are used to test, which are 10 and 50 time units. COMET-CORE maintains end points with two kinds of relative error: 10% and 20%, and checks necessary cluster splitting or merging by events.

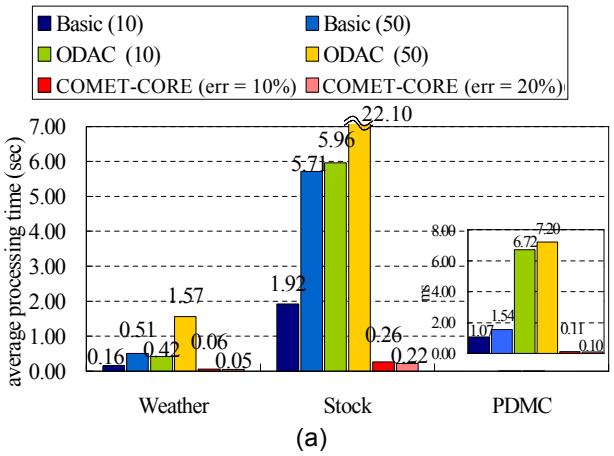
First we evaluate the average processing time for a round of clustering. As shown in Figure 8(a), COMET-CORE is much more efficient than other two methods. For Basic and ODAC methods, the average processing time increases as the period becomes longer. This is because larger amount of data needed to be processed when updating necessary statistics for similarities. However, if the period between two clustering becomes shorter, the frequency of clustering tasks increases and the total processing time will increase significantly as the growing of stream length. Both Basic and ODAC methods spend a lot of time on updating pair-wise similarities between streams. ODAC even spends time on testing cluster splitting and aggregation by computing pair-wise distance between streams in the same cluster for possible clusters. Although ODAC may have good clustering quality, high processing time makes the scalability of ODAC in the environment of multiple streams questionable. On the contrary, COMET-CORE only updates clusters with trigger streams. Under different error threshold of piecewise linear approximation, the number of end points is listed in Figure 8(b). We can find that the threshold value has very small influence on processing time in all three data sets. Conclude from all three methods, we can discover that it is the number of stream which dominates the average processing time. The time cost is almost independent of the length of a stream. It takes much more time for stock data which contains 952 streams with length of 252 than the time for PDMC data which contains 9 streams with length of 189,961 (Note that the time scale for PDMC data set is in milliseconds.)

Next, the cluster quality is discussed. We probe the GS value along stream progression and the average of these value is used to evaluate their performance. The average GS value for three methods of each data set is shown in Figure 8(c). Stock data fluctuates more while the weather and PDMC data are relatively steady. Therefore all methods have lower GS value on stock data set. There are only 9 streams in PDMC data set, therefore the overall GS values are high. Then we discuss the reason for different qualities of each method. For Basic method, the period of clustering does not affect the clustering results since the Basic method always starts merging clusters directly from treating one

[2] http://www.engr.udayton.edu/weather/

[3] http://kumo.swcp.com/stocks/

[4] http://www.cs.utexas.edu/users/sherstov/pdmc/



(a)

Statistics of each data set						
	stream size	stream length	avg. # of end point (err = 10%)		avg. # of end point (err = 20%)	
Weather	290	3,416	939	27.49%	655	19.17%
Stock	952	252	36	14.29%	22	8.73%
PDMC	9	189,961	59482	31.31%	46079	24.26%

(b)

Global Silhouette					
	Basic	ODAC (10)	ODAC (50)	COMET-CORE (err = 10%)	COMET-CORE (err = 20%)
Weather	0.611	0.269	0.201	0.886	0.907
Stock	0.353	0.336	0.264	0.205	0.258
PDMC	0.926	0.638	0.861	0.934	0.932

(c)

Fig. 8. The clustering results of real data sets. (a)Average processing time. (b)The number of end points. (c)The average GS value.

α	0.99	0.98	0.97	0.96
Basic	0.748	0.723	0.715	0.716
COMET-CORE (err = 10%)	0.874	0.810	0.769	0.747

(a)

α	1	0.99	0.98	0.97	0.96
# of exact match / #of clusters	26/28	22/35	45/65	51/71	58/78
proportion	92.9%	62.9%	69.2%	71.8%	74.4%

(b)

Fig. 9. The (a) average GS value and (b) number of exactly match cluster under different α values.

stream as a cluster. Consider the ODAC method, the hoeffding bound depends on the number of samples collected. Therefore, checking cluster splitting and aggregation with different frequency results in different clustering results for the same data streams. It is not easy to choose a good clustering period and confidence value in computing hoeffding bounds. For COMET-CORE, the cluster quality is affected by the following reasons. In our algorithm, to avoid too many 1-stream clusters during clustering, we put all these streams in a temporary clusters and wait for next trigger to split them. Therefore, in this moment, the temporary cluster degrades the cluster quality. Also, the approximation error caused by WC vector has significant influence as well. But this can be remedied by the piecewise linear smoothing. When higher tolerance on piecewise linear approximation error, the resulting number of trigger points degrades. This reduces the frequency of WC vector approximation and hence the cluster quality increases. In the following we discuss the reason GS value of COMET-CORE is superior to other two methods. The Basic method applies agglomerative hierarchical clustering and ODAC applies mainly divisive clustering. Once the order of merging or splitting is decided for hierarchical clustering, it cannot be changed later. However, in COMET-CORE, the combination of split and merge processes provides streams flexibility to jump among clusters for a better clustering structure. Furthermore, the advantage that COMET-CORE captures cluster evolutions dynamically by events is revealed here: periodical clustering method such as Basic and ODAC cannot capture cluster evolutions which occur at some time during two clustering tasks.

Finally, take the weather data for example, the clustering results under different α are shown in Figure 9(a). Since there provides no bias option in ODAC algorithm, it is not considered here. For different α value, the data weight varies hence we will get different clustering results. Again, COMET-CORE is superior to Basic in that it captures the clustering evolution dynamically instead of clustering every fixed time period. In addition, we compare the final number of cluster which has exactly the same members for two methods. From 9(b), we can see that the proportion of exact match is quite high, which supports that COMET-CORE is of high reliability.

B. Evaluation of COMET-CORE on Synthetic Data

In this section, two kinds of synthetic data sets are used for evaluation.

1) Cylinder-bell-funnel data set: We use cylinder-bell-funnel data to demonstrate that COMET-CORE can significantly cluster data in an online fashion and explain the function of α parameter. The generation of this data was proposed in [28] and [29]. Three classes of data: cylinder, bell and funnel, and examples are illustrated in Figure 10(a). Each class of data has a length of 128. In our test data, we concatenate three distinct classes of data in different order as shown in Figure 10(b). The different combinations of data classes constitute six types of 384-long streams. For each type, 100 streams are generated and normal distribution number ranges from 0 to 1 are randomly added on

each of them. Consequently we have a total of 600 streams to test.

We set $\delta_a = \delta_e = 0.80$, which is quite a strict criterion to make sure that streams inside the same cluster are highly correlated. The α value is set from 0.98 to 1. In Figure10(c), the column shows different time points and the row represents different α values. At certain time points and α values, the cluster number and the error rate are recorded. Here, the error rate is calculated as: ($\sum_{C_i}$(number of streams$\notin C_i$/ total stream number inside C_i))/(number of real cluster), where real cluster means cluster with member number > 1. If the class of a stream, say S_i, is of minority in the cluster C_i where the stream is assigned, we say $S_i \notin C_i$. According to Figure 10(c), around time 92, there should be three clusters and the COMET-CORE successfully reflects the results under all α value. Around time 128, which is the end of the first class period of each stream, we found that when α is 0.99, the cluster number becomes to 2. One cluster contains purely the whole F-B-C and F-C-B data (200 streams), while the other cluster contains the whole B-C-F, B-F-C, C-B-F, C-F-B data (totally 400 streams). This is because the data near time 128 are weighted more than the data at the beginning, which makes the bell and cylinder data alike and both of them are dissimilar to the funnel data. Also, the error rate of 25% results from the reason that 200 out of 400 streams in the same cluster are judged as those which do not belong to this cluster. For $\alpha = 0.98$, the cluster number increases. Due to the faster decay rate and the domination of noise data, some correlations between the same-type streams fall below 0.8. Around time 226, the cluster number becomes 6 for $\alpha = 1$. This is because each data point is treated with equal weight and therefore different types of streams are separated into different clusters. For $\alpha = 0.99$ and 0.98, both of them have three clusters. This time the clusters are decided by the second part of each stream, the weight before time 128 is almost 0. Hence, the B-C-F and F-C-B are in one cluster, B-F-C and C-F-B are in another cluster, and the rest of two types are in the other cluster. The cases of the following time points are the same as previous ones and are hence omitted here. In this experiment, we show that COMET-CORE indeed performs significant clustering and can obtain desired clustering results under different α values.

2) Random Walk Data Set: To evaluate the scalability of three methods, the synthetic data used are as follows. First, we generate four random walk data sets with the number of stream ranging from 100 to 2,000. Each stream has a length of 20,000 points. For Basic and ODAC methods, the clustering task is performed every 200 points. The confidence value of hoeffding bounds for ODAC is set as 0.05. The threshold of merge process for Basic is δ_e=0.7 and for COMET-CORE we set δ_a=δ_e=0.7. In addition, COMET-CORE does not apply piecewise linear approximation on stream data here. The average processing time is showed in Figure 11(a). Here the time axis (y-axis) is plotted in log scale for clarity because the processing time of each method is different in several order. Again, COMET-CORE is manifestly more efficient than other two periodical clustering methods. Basic and ODAC spend too much time on computing pair-wise similarities between streams, while COMET-CORE only checks clusters with trigger streams for possible splitting and merging. Then we see the average GS value for each method in figure 11(c). These values are obtained by averaging the GS value every 400 points for each method. In most cases, Basic has the highest value because each time it merges streams starting from one stream cluster and stops when the criterion is met. ODAC has lower GS value for two possible reasons. One is as mentioned in the empirical study of real data, it is hard to decide a good clustering period and confidence value for hoeffding bound. Second, ODAC splits a cluster into only two clusters each time when the splitting condition is met. The cluster may not be split good enough. Finally, for COMET-CORE, the main degradation on cluster quality is caused by the approximation of WC vector when splitting and merging clusters. This is a trade-off between efficiency and quality. However, from the clustering results, we show that COMET-CORE still keeps an acceptable cluster quality with high efficiency.

Next we show the performance of three methods while the number of cluster varies. The stream number is fixed at 500 and each of them contains 20,000 points. For different data sets, streams are randomly distributed in different number of cluster. The cluster number varies from 50 to 200. From Figure 11(b), we can find that the processing time is almost independent of the cluster number for all three methods. This again proves that it is the number of stream which dominates the time cost. As for the clustering quality shown in figure 11(d), the GS value for each method keeps steady itself under different cluster number. The reasons for better or worse GS value are the same as aforementioned.

Finally, we further analyze how the number of cluster which contains trigger streams affect the processing time of COMET-CORE. In the following, we use *trigger cluster* to mean the cluster with trigger streams. In the empirical study of real data, we have shown that the number of trigger points is not a magnificent factor of average processing time. Because many trigger streams may exist in only few clusters, and COMET-CORE only updates similarities of trigger clusters. To show this, we use the same data set which is 200 clusters with 500 streams. As the progression of streaming clustering along 20,000 points, we record the number of trigger cluster and its corresponding processing time. As shown in Figure 12, the processing time indeed has a trend that it is increasing with the number of trigger cluster.

VII. Conclusion

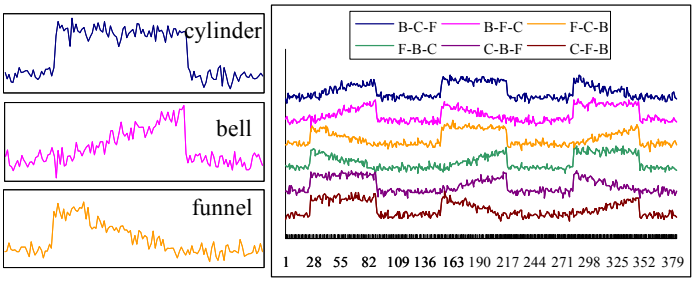
In this paper, we proposed the COMET-CORE framework for online monitoring clusters over multiple evolving streams by correlations and events. The streams are smoothed by piecewise linear approximation and we can regard each end point of the line segment as a trigger point. At each trigger point, for clusters which have trigger streams, we update the weighted correlations related to trigger streams in clusters. Whenever an event happens, i.e., clusters change, the clusters are modified through efficient split and merge processes. As validated in the experimental results, COMET-CORE is shown to be efficient and of good scalability while producing cluster results of good quality.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *Proc. of Principles of Database Systems*, 2002.

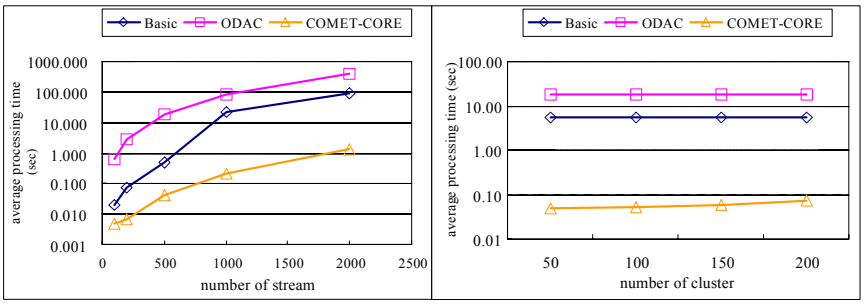
[2] A. Bulut and A. K. Singh, "SWAT: Hierarchical stream summarization in large networks," in *Proc. of Int'l Conf. on Data Engineering*, 2003.

[3] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. of ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 2000.



time / ▪	~92	~128	~226	~256	~352	~384
1	3	3	6	6	6	6
	0%	0%	0%	0%	0%	0%
0.99	3	2	3	3	3	3
	0%	25%	0%	0%	0%	11%
0.98	3	7	3	6	3	6
	0%	10%	0%	17%	0%	17%

Fig. 10. (a)Examples of cyliner-bell-funnel data. (b)Six types of our testing data set. (c)number of cluster and error rate along stream progression.



# of stream / method	100	200	500	1000	2000
Basic	0.679	0.735	0.720	0.722	0.653
ODAC	0.408	0.336	0.282	0.379	0.403
COMET-CORE	0.524	0.445	0.355	0.604	0.529

(c)

# of cluster / method	50	100	150	200
Basic	0.656	0.601	0.556	0.587
ODAC	0.250	0.262	0.232	0.174
COMET-CORE	0.515	0.533	0.591	0.621

(d)

Fig. 11. The average processing time while the number of (a)stream and (b)cluster varies, and the clustering quality while the number of (c)stream and (d)cluster varies.

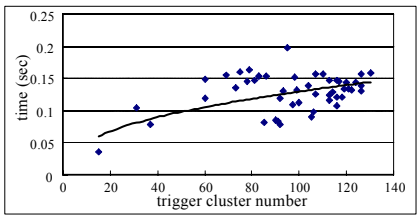


Fig. 12. The average processing time versus the number of trigger cluster.

[4] M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky, "Cost-efficient mining techniques for data streams," in *Proc. of Australasian Workshop on Data Mining and Web Intelligence*, 2004.

[5] V. Ganti, J. Gehrke, and R. Ramakrishnan, "Demon: Mining and monitoring evolving data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, pp. 50–63, 2001.

[6] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams," in *the Annual Symposium on Foundations of Computer Science*, 2000.

[7] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. of ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 2001.

[8] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-data algorithms for high-quality clustering," in *Proc. of Int'l Conf. on Data Engineering*, 2002.

[9] A. Bulut and A. K. Singh, "A unified framework for monitoring data streams in real time," in *Proc. of Int'l Conf. on Data Engineering*, 2005.

[10] X. Liu and H. Ferhatosmanoglu, "Efficient k-nn search on streaming data series," in *Proc. of Symposium on Spatial and Temporal Databases*, 2003, pp. 83–101.

[11] Y. Zhu and D. Shasha, "Statstream: Statistical monitoring of thousands of data streams in real time," in *Proc. of Very Large Data Bases*, 2002, pp. 358–369.

[12] H. Wu, B. Salzberg, and D. Zhang, "Online event-driven subsequence matching over financial data streams," in *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, 2004, pp. 23–34.

[13] J. Beringer and E. Hullermeier, "Online clustering of parallel data streams," *Data and Knowledge Engineering*, vol. 58, no. 2, pp. 180–204, 2005.

[14] B.-R. Dai, J.-W. Huang, M.-Y. Yeh, and M.-S. Chen, "Adaptive clustering for multiple evolving streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 9, pp. 1166–1180, 2006.

[15] P. P. Rodrigues, J. Gama, and J. P. Pedroso, "Odac: Hierarchical clustering of time series data streams," in *Proc. of the Sixth SIAM Int'l Conf. on Data Mining*, Bethesda, Maryland, USA, April 2006, pp. 499–503.

[16] J. Yang, "Dynamic clustering of evolving streams with a single pass," in *Proc. of Int'l Conf. on Data Engineering*, 2003, pp. 695–697.

[17] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *Proc. of Very Large Data Bases*, 2003.

[18] ——, "On high dimensional projected clustering of data streams," *Data Mining and Knowledge Discovery*, vol. 10, no. 3, pp. 251–273, 2005.

[19] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proc. of the SIAM Conference on Data Mining*, 2006.

[20] Z. He, X. Xu, S. Deng, and J. Z. Huang, "Clustering categorical data streams," *Computational Methods in Science and Engineering*, 2004.

[21] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani, "An online algorithm for segmenting time series," in *Proc. of Int'l Conf. on Data Mining*, 2001.

[22] V. Guralnik and J. Srivastava, "Event detection from time series data," in *Proc. of the fifth ACM SIGKDD Int'l Conf. on Knowledge discovery and data mining*. ACM Press, 1999, pp. 33–42.

[23] D. P. Kacso, "Approximation by means of piecewise linear functions," *Results in Mathematics*, vol. 35, pp. 89–102, January 1999.
[24] E. J. Keogh, "A fast and robust method for pattern matching in time series databases," in *Proc. of Int'l Conf. on Tools with Artificial Intelligence*, 1997.
[25] E. J. Keogh and M. J. Pazzani, "An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback," in *Proc. of Int'l Conf. on Knowledge Discovery and Data Mining*, 1998, pp. 239–243.
[26] A. Franzblau, "A primer of statistics for non-statisticians," in *Harcourt, Brace and World*, 1958.
[27] P. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.
[28] N. Saito, "Local feature extraction and its applications using a library of bases," Ph.D. dissertation, Department of Mathematics, Yale University, 1994.
[29] S. Manganaris, "Supervised classification with temporal data," Ph.D. dissertation, Department of Computer Science, Vanderbilt University, 1997.

Mi-Yen Yeh received the B.S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2002. She is currently a Ph.D. candidate in the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan. Her research interests include data mining, data clustering, and data stream management.

Bi-Ru Dai received the B.S. and Ph.D. degrees in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2000 and 2006, respectively. She is currently an assistant professor in the Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan. Her research interests include data mining, data stream management, and bioinformatics.

Ming-Syan Chen received the B.S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, and the M.S. and Ph.D. degrees in Computer, Information and Control Engineering from The University of Michigan, Ann Arbor, MI, USA, in 1985 and 1988, respectively. He is now a Distinguished Professor jointly appointed by Electrical Engineering Department, Computer Science and Information Engineering Department, and also Graduate Institute of Communication Eng. at National Taiwan University. He was a research staff member at IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA from 1988 to 1996. His research interests include database systems, data mining, mobile computing systems, and multimedia networking, and he has published more than 240 papers in his research areas. In addition to serving as program chairs/vice-chairs and keynote/tutorial speakers in many international conferences, Dr. Chen was an associate editor of IEEE TKDE and also JISE, is currently on the editorial board of Very Large Data Base (VLDB) Journal, Knowledge and Information Systems (KAIS) Journal, and International Journal of Electrical Engineering (IJEE), and is a Distinguished Visitor of IEEE Computer Society for Asia-Pacific from 1998 to 2000, and also from 2005 to 2007. He holds, or has applied for, eighteen U.S. patents and seven ROC patents in his research areas. He is a recipient of the NSC (National Science Council) Distinguished Research Award, Pan Wen Yuan Distinguished Research Award, Teco Award, Honorary Medal of Information, and K.-T. Li Research Breakthrough Award for his research work, and also the Outstanding Innovation Award from IBM Corporate for his contribution to a major database product. He also received numerous awards for his research, teaching, inventions and patent applications. Dr. Chen is a Fellow of ACM and a Fellow of IEEE.