ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.7, July 2009, pp.1756−1767
doi: 10.3724/SP.J.1001.2009.00566
© by *Institute of Software, the Chinese Academy of Sciences*. All rights reserved.

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于相关分析的多数据流聚类*

屠 莉[1+], 陈 崚[2,3], 邹凌君[2]

[1](南京航空航天大学 信息科学与技术学院,江苏 南京 210093)
[2](扬州大学 计算机科学与工程系,江苏 扬州 225009)
[3](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093)

Clustering Multiple Data Streams Based on Correlation Analysis

TU Li[1+], CHEN Ling[2,3], ZOU Ling-Jun[2]

[1](Institute of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210093, China)
[2](Department of Computer Science and Engineering, Yangzhou University, Yangzhou 225009, China)
[3](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)
+ Corresponding author: E-mail: yzutuli@yahoo.com.cn, lchen@yzcn.net

Tu L, Chen L, Zou LJ. Clustering multiple data streams based on correlation analysis. *Journal of Software*, 2009,20(7):1756−1767. http://www.jos.org.cn/1000-9825/566.htm

Abstract: This paper proposes a compression scheme which quickly compresses the raw data from multiple streams into a compressed synopsis. The synopsis allows to incrementally reconstruct the correlation coefficients without accessing the raw data. A modified *k*-means algorithm is developed to generate clustering results and dynamically adjust the number of clusters in real time so as to detect the evolving changes in the data streams. Finally, the framework is extended to support clustering on demand (COD), where a user can query for clustering results over an arbitrary time horizon. A theoretically sound time-segment partitioning scheme is developed so that any demand time horizon can be fulfilled by a combination of those time-segments. Experimental results on synthetic and real data sets show that the algorithm has higher clustering quality, speed and stability than other methods and can detect the evolving changes of the data streams in real time.

Key words: clustering; data stream; correlation analysis

摘 要: 提出基于相关分析的多数据流聚类算法.该算法将多数据流的原始数据快速压缩成一个统计概要.根据这些统计概要,可以增量式地计算相关系数来衡量数据间的相似度.提出了一种改进的 *k*-平均算法来生成聚类结果.改进的 *k*-平均算法可以动态、实时地调整聚类数目,并及时检测数据流的发展变化.还将算法应用到按照用户要求的聚类问题(COD),使得用户可以在任意的时间区间上查询聚类结果.提出了一种合理的时间片断划分机制,使得用户指定的任意时间区间都可以由这些时间片断组合而成.在模拟和真实数据上的实验结果都表明,该算法比其他方法具有更好的聚类质量、速度和稳定性,能够实时地反映数据流的变化.

关键词: 聚类;数据流;相关分析

* Supported by the National Natural Science Foundation of China under Grant Nos.60673060, 60773103 (国家自然科学基金); the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2008206 (江苏省自然科学基金)
Received 2007-08-11; Revised 2008-01-10, 2008-03-11; Accepted 2008-07-02

中图法分类号: TP18 文献标识码: A

1 Introduction

Extensive research has been done for mining data streams[1], including those on the stream data classification[2,3], mining frequent patterns[4−6], and clustering stream data[4,7−16]. There are various applications where it is desirable to cluster the streams themselves rather than the individual data records within them. In this paper, we study the clustering of multiple and parallel data streams. Our goal is to group multiple streams with similar behaviors and trends together, instead of clustering the data records within one data stream. Therefore, the methods designed for clustering static data sets[17−19] cannot be directly applied to multiple data streams. There has been some previous works on clustering multiple data streams[20−23]. Yang[20] used the weighted aggregation of snapshot deviations as the distance measure between two streams, which can observe the similarity of data values but ignore the trends of streams. Beringer *et al*.[21] proposed a method which used a discrete Fourier transforms (DFT) approximation of the original data, and with that method the distance between two streams was computed by using their low-frequency coefficients. Since DFT transformation preserves the Euclidean distances, the DFT distance is equivalent to the Euclidean distance of the data streams.

A serious limitation of the above previous works is that they are all based on the Euclidean distance of data records, but the important trend information contained in data streams is typically discarded by clustering methods based on Euclidean distance. This is true because data streams with similar trends may not be close in their Euclidean distance. As an illustration, Figure 1 shows the trends of three stocks on Nylon, chemical fiber, and CPU chip respectively. Although the two stocks on CPU chip and chemical fiber are closer in their data values and hence their Euclidean distance, the trends of the two stocks on Nylon and Chemical fiber are obviously closer. However, such trends of stocks can be suggested by their higher correlation coefficient. If with Euclidean distance, we may conclude that the two stocks on CPU chips and chemical fiber are more similar.

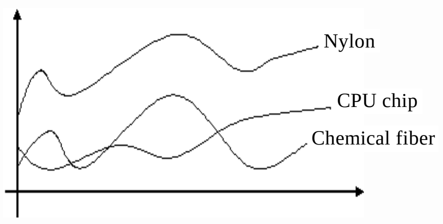

Fig.1 Price of stocks on nylon, chemical fiber, and CPU chips

In this paper, we propose a multiple streams clustering algorithm based on the correlation analysis. We propose a novel compression scheme that quickly compresses the raw data from multiple streams into a compressed synopsis. The synopsis allows us to incrementally reconstruct the correlation coefficients without accessing the raw data. A modified *k*-means algorithm is developed to generate the clustering results. The *k*-means algorithm is modified so that it can dynamically adjust the number of clusters in real time, and therefore can detect the evolving changes in the data streams. Finally, we extend the framework to support clustering on demand (COD), where a user can query for clustering results over an arbitrary time horizon. A theoretically sound time-segment partitioning scheme is developed so that any demand time horizon can be fulfilled by a combination of those time-segments. Experimental results on synthetic and real data sets show that our algorithm is higher in clustering speed, quality and stability than other methods, and can detect the evolving changes in the data streams in real time.

2 Background

2.1 Clustering data streams

A data stream X is a sequence of data items $x_1,\ldots,x_k$ arriving at discrete time steps $t_1,t_2,\ldots,t_k$. We assume that there are n data streams $\{X_1,...,X_n\}$ at each time step m, where $X_i=\{x_{i1},...,x_{im}\}$, $1\le i\le n$, and $x_{ij}(j=1,...,m)$ is the value of stream X_i at the time j.

The problem of clustering multiple data streams is defined as follows. Given the time horizon for clustering L and the number of clusters k, the clustering algorithm partitions n data streams into k clusters $C(L)=\{C_1(L),...,C_k(L)\}$ that minimizes some objective function measuring the quality of clustering in the period $[t-L+1,t]$, where t is the time when the analysis is performed. The given clusters $C_j(L)$, $j=1,...,k$, should satisfy

$$\bigcap_{j=1}^{k} C_j(L)=\varnothing, \bigcup_{j=1}^{k} C_j(L)=\{X_1(L),...,X_n(L)\},\ \text{where } X_i(L)=\{x_{i(t-L+1)},...,x_{it}\},\ i=1,...,n \tag{1}$$

2.2 Attenuation coefficient

In the stream data analysis, in order to recognize the evolving changes of data streams, newer data records are often given more weights than the older ones. Therefore, we use an *attenuation coefficient* $\lambda\in[0,1]$ to gradually lessen the significance of each data record over time. Suppose t is the current time and a data point x_i is received at time i, then we replace the original value of x_i by $x_i(t)=\lambda^{t-i}x_i$ in our analysis.

2.3 Segment

We first consider the fixed-length clustering and then extend the algorithm to the arbitrary-length clustering of COD in Section 4. Given a fixed length L, at any time t, we report in real time the clustering results for the data streams in the time horizon $[t-L+1,t]$. To support efficient processing, we partition the data streams of length L into m time segments of equal length $l=L/m$. Whenever a new segment of length l accumulates (Fig.2), we re-compute the clustering results.

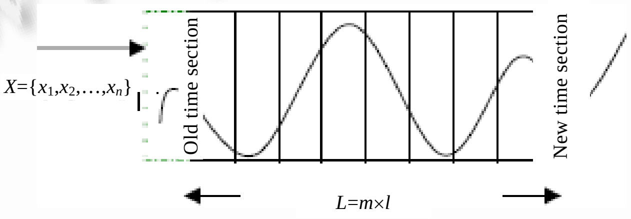

Fig.2 A fixed length L is divided into m segments of size l

3 The Correl-Cluster Algorithm

The framework of the proposed CORREL-cluster algorithm is shown in Fig.3.

The algorithm keeps detecting and reporting the clusters for the most recent data streams of the fixed length L. For every l time steps, it first computes the compressed correlation representation (CCR) and the correlation coefficients of the data streams, and then clusters the streams based on their correlation coefficients. Since the number of clusters k may be changing, CORREL-cluster also employs a new algorithm to dynamically adjust k in order to recognize the evolving behaviors of the data streams (Line 12). The algorithm is schematically shown in Fig.4.

```
1.   procedure CORREL-cluster
2.   $t$=0;
3.   while data stream is not terminated
4.       set $t$=$t$+1;
5.       read new data record $x_k(t)$, $k$=1,…,$n$, one from
             each of the $n$ data streams;
6.       if ($t$ mod $l$=0) then
7.           calculate the CCR of the new time segment;
8.           update CCRs of the existing segments;
9.           if $t$=$L$ then compute initial CCR
                 for time window [1,$L$];
10.          else incrementally update CCR
                 for the new time window
11.          call correlation-$k$-means();
12.          call adjust_$k$();
13.          output the clustering result;
14.      endif
15.  end while
16.  end procedure
```

Fig.3 Overall process of CORREL-cluster

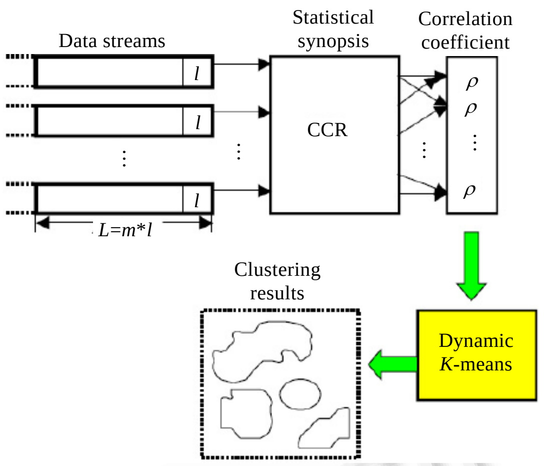

Fig.4 Illustration of CORREL-cluster

3.1 Correlation analysis of data streams

Before describing the details of our algorithm, we first overview the concepts of correlation analysis.

We first define the correlation coefficients for two data streams. For data segments of two streams $X=(x_1,...,x_n)$ and $Y=(y_1,...,y_n)$, their correlation coefficient is defined as

$$\rho_{XY}=\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2\sum_{j=1}^{n}(y_i-\bar{y})^2}},$$

where $\bar{x}=(\sum_{i=1}^{n}x_i)/n$ and $\bar{y}=(\sum_{i=1}^{n}y_i)/n$.

From the definition, we can see that $|\rho_{XY}|\le 1$. A large value of $|\rho_{XY}|$ indicates a strong correlation between streams X and Y, and $\rho_{XY}=0$ means X and Y are uncorrelated. Since it is often impossible to store all the past raw data in the stream, we need to compress the raw data and only retain a synopsis for each time segment of a data stream.

Theorem 3.1. The correlation coefficient ρ_{XY} between two sequences X and Y can be calculated with the information $\sum x_i$, $\sum x_i^2$, $\sum y_i$, $\sum y_i^2$, and $\sum x_i y_i$.

The proof of Theorem 3.1 is omitted due to the limited space. Based on Theorem 3.1, in CORREL-cluster, for any time segment, we store a compressed synopsis for the multiple data streams instead of the whole raw data.

Definition 3.1 (**compressed correlation representation**). Given n data streams $X_1,...,X_n$, suppose the current time is t, the time segment length is l, then the compressed correlation representation of this time segment is defined as $(\vec{S}(t,l),\vec{Q}(t,l),\vec{C}(t,l))$, where vectors $\vec{S}(t,l)=(S_1,S_2,...,S_n)$, $\vec{Q}(t,l)=(Q_1,Q_2,...,Q_n)$ and matrix $C(t,l)=[C_{ij}]$. Components of the vectors $\vec{S}(t,l)$, $\vec{Q}(t,l)$ and matrix $C(t,l)$ are defined as:

$$S_i=\sum_{k=t-l+1}^{t}x_{ik}(t),\ Q_i=\sum_{k=t-l+1}^{t}x_{ik}^2(t),\ C_{ij}=\sum_{k=t-l+1}^{t}x_{ik}(t)x_{jk}(t),\ i,j=1,...,n,i<j \qquad (2)$$

here, $x_{ik}(t)$ is the kth datum of the ith stream at time t.

The compressed correlation representation provides enough information to compute the correlation coefficients between any two streams in n data streams $X_1,...,X_n$. Based on Theorem 3.1 and the definition of correlation coefficient, we can easily get the following theorem.

Theorem 3.2. Let X_i and X_j, $i,j=1,...,n$, be data segments of two streams, their correlation coefficients $\rho_{X_iX_j}$.

can be computed by

$$\rho_{X_iX_j}=\frac{C_{ij}-\frac{1}{n}S_iS_j}{\sqrt{\left(Q_i-\frac{1}{n}S_i^2\right)\left(Q_j-\frac{1}{n}S_j^2\right)}} \tag{3}$$

Example 1. Let l=10 and t=1, data streams X_1=(18,23,21,17,24,22,25,19,21,20) and X_2=(19,22,24,20,22,23,24,20,16,18) in the time period 1 to 10. Since S_1=210, S_2=208, Q_1=4470, Q_2=4390, C_{12}=4402, the compressed correlation representation are [(210,208),(4470,4390),4420]. By Theorem 3.2, the correlation coefficient of X_1 and X_2 is $\rho_{X_1X_2}$=0.9713.

3.2 Update of the compressed correlation representation

Let $t-t_c=\Delta t$ and λ be the attenuation coefficient, at any time $t>t_c$, the data values of $x_{ik}(t_c)$ are replaced by $x_{ik}(t)$:

$$x_{ik}(t)=\lambda^{t-i}x_{ik}=\lambda^{t-t_c+t_c-i}x_{ik}=\lambda^{t-t_c}\lambda^{t_c-i}x_{ik}=\lambda^{\Delta t}x_{ik}(t_c).$$

Since the values of $x_{ik}(t_c)$ are updated, the compressed correlation representation needs to be modified accordingly. Let $\overrightarrow{S_i'}(t_c,l_c)$, $\overrightarrow{Q_i'}(t_c,l_c)$ and $C_{ij}'(t_c,l_c)$ be the updated values of $\overrightarrow{S_i}(t_c,l_c)$, $\overrightarrow{Q_i}(t_c,l_c)$ and $C_{ij}(t_c,l_c)$, then $\overrightarrow{S_i'}(t_c,l_c)=\lambda^{\Delta t}\overrightarrow{S_i}(t_c,l_c)$, $\overrightarrow{Q_i'}(t_c,l_c)=\lambda^{2\Delta t}\overrightarrow{Q_i}(t_c,l_c)$ and $C_{ij}'(t_c,l_c)=\lambda^{2\Delta t}C_{ij}(t_c,l_c)$. We notice that such an update happens not at every time step, but every l step (Line 8 in Fig.3). Whenever a new time segment comes in, we compute the new compressed correlation representation.

Example 2. In Example 1, let λ=0.9, then at time t=3, since Δt=2, therefore $S_1=210\times0.9^2$=170.10, $S_2=208\times0.9^2$=168.48, $Q_1=4470\times0.9^4$=2932.767, $Q_2=4390\times0.9^4$=2880.279, $C_{12}=4402\times0.9^4$=2888.1522.

3.3 Aggregation of compressed correlation representation

In CORREL-cluster, for a user specified clustering length $L=ml$, we need to compute the correlation coefficients for data segments in time window $[t-L+1,t]$ so as to cluster the data streams. Since we compute a compressed correlation representation for each time segment with length l, we need to combine them with the compressed correlation representation for the time window $[t-L+1,t]$.

In each time window, there are m time segments and m compressed correlation representations $[\vec{S}(t-vl+1,l),\vec{Q}(t-vl+1,l),C(t-vl+1,l)]$ for v=1,...,m. Similarly, we denote the compressed correlation representation for time window $[t-L+1,t]$ as $[\vec{S}(t-L+1,L),\vec{Q}(t-L+1,L),C(t-L+1,L)]$. Let the ith component of $\vec{S}(t-L+1,L)$ and $\vec{Q}(t-L+1,L)$ be $\overrightarrow{S_i}(t-L+1,L)$ and $\overrightarrow{Q_i}(t-L+1,L)$ respectively, then we have $\overrightarrow{S_i}(t-L+1,L)=\sum_{v=1}^{m}\overrightarrow{S_i}(t-vl+1,l)$, $\overrightarrow{Q_i}(t-L+1,L)=\sum_{v=1}^{m}\overrightarrow{Q_i}(t-vl+1,l)$ and $C(t-L+1,L)=\sum_{v=1}^{m}C(t-vl+1,l)$.

We use the above equations to compute the compressed correlation representation when we receive the first m time segments (Line 9 of Fig.3). For later updates (Line 10), we do not need to redo the summation. In fact, we can obtain compressed correlation representation for the new time window $[t-L+l+1,t+l]$ by

$$\vec{S}(t-L+l+1,L)=\vec{S}(t-L+1,L)+\vec{S}(t+l,l)-\vec{S}(t-ml+1,l) \tag{4}$$

$$\vec{Q}(t-L+l+1,L)=\vec{Q}(t-L+1,L)+\vec{Q}(t+l,l)-\vec{Q}(t-ml+1,l) \tag{5}$$

$$C(t-L+l+1,L)=C(t-L+1,L)+C(t+l,l)-C(t-ml+1,l) \tag{6}$$

3.4 Dynamic *k*-means algorithm

We propose a dynamic k-means algorithm to cluster the data streams in the user-specified window. In the algorithm, the distance between two data streams X and Y is measured by using the reciprocal of the correlation coefficient $d(X,Y)=1/\rho_{XY}$. The **correlation-*k*-means** algorithm is shown in Fig.5.

The fundamental idea behind this algorithm is that two consecutive clustering results are not changing very fast over a short time gap l, and may significantly overlap with each other. The algorithm could take very few steps to converge if it starts from the clustering result on the data segment in the previous time window.

To capture the dynamic evolution of data streams, we continuously update the number of clusters k every time a new data segment with length l is received. Noticing that the number of the clusters could not change abruptly and frequently in a short time gap l, we will only consider the cases where the number of clusters increases or decreases by one. The **adjust_*k*** algorithm is shown in Fig.6. In the algorithm, the number of the clusters is adjusted by splitting or emerging some current clusters, and the clustering quality is measured by an objective function $G=\sum_{i=1}^{k}\sum_{j=1}^{n_i}(1/\rho_{X_jC_i})$, where $\rho_{X_jC_i}$ is the correlation coefficient between the data stream X_j and the ith cluster center C_i, which consists of streams from X_1 to X_{n_i}.

```
Algorithm correlation-k-means(k,Center_k,R_k)
Input: number of the clusters k, sets of the center
          points, Center_k; current clustering result R_k;
Output: updated clustering results R_k and its
          objective function value G_k.
begin
  1.    repeat
  2.     for i=1 to n
            calculate the correlation distances between
            stream X_k and centers of k clusters and assign
            X_k to the cluster with the shortest distance;
          end for
  3.     compute the new center of each cluster,
         update the set of centers Center_k;
  4.    until no change of clustering result
  5.    calculate the objective function G_k.
end
```

Fig.5 Algorithm for clustering

```
Algorithm adjust_k(k,Center_k,R_k)
Input: number of the clusters k, sets of the center
        points Center_k, current clustering result R_k;
Output: updated number of clusters k′, updated
          clustering result R_k′ and its objective
          function value G_k.
begin
1   Calculation of R_{k+1}
      Among all the clusters, choose the data stream X which is
      the farthest from its cluster center, set a new cluster with X
      as its center;
      Center_{k+1}=Center_k∪{X}
      correlation-k-means(k+1,Center_{k+1},R_{k+1})
2   Calculation of R_{k−1}
          Choose two closest clusters, suppose their centers are
          C_1 and C_2 respectively,
          combine these two clusters into a new cluster,
          compute the center C_3 of the new cluster
          Center_{k−1}=Center_k∪{C_3}−{C_1,C_2}
          correlation-k-means(k−1,Center_{k−1},R_{k−1})
3   choose the one with best G form R_{k−1},R_k,R_{k+1},
    set k′ and R_k′ accordingly;
end
```

Fig.6 Algorithm for adjusting k

4 Clustering on Demand

In some applications, the length of the time window depends on users' demands[23]. Here, we extend our clustering algorithm to support clustering on demand (COD). We call the extended algorithm CORREL-COD. The CORREL-COD algorithm consists of online and offline processes which are performed in a pipelined fashion. The online component calculates the summary information in correlation parameters, whereas the offline part performs clustering.

We assume that the maximum time horizon over which the user will demand is L. Namely, we only need to preserve information for the time period $[t-L+1,t]$.

We assemble a combination of partitioned time segments in such a way that minimizes the difference between the user-specified time horizon r and the best approximate time horizon r' over which we can get compressed correlation representation synopsis.

Let $L=2^d$ and m be the maximum number of segments the memory can store. The basic idea of our scheme is that the segments containing newer data will have shorter lengths, leading to higher clustering accuracy for newer data so as to make the difference $|r-r'|$ as small as possible. We assume $m>\log L$. In our scheme, we first arrange segments with lengths of $1,2,2^2,2^3,...,2^{d-1}$, respectively. Let $m'=m-\log L=m-d$, and S_i denotes the segment with length 2^i. For the m' unassigned segments, we assign them according to the following rule.

We first remove S_{d-1}, replace the region of S_{d-1} by two more S_{d-2}, and then reduce m' by one. If $m'>0$, we will keep splitting a larger segment into two smaller segments. When there are still S_{d-2} segments, we split the most recent S_{d-2} into two S_{d-3}, and then reduce m' by one, until all S_{d-2} segments are removed or $m'=0$. If all S_{d-2} segments are removed and $m'>0$, we will split the most recent S_{d-3} into two S_{d-4} and reduce m' by one. We repeat this process until $m'=0$.

Given L, m, and $m'=m-\log L$, it is easy to show that with our scheme, the maximum length of any segment is 2^k, where

$$k=\text{argmax}_i(m'\le C_i)-1, C_i=2^{d-i+2}-(d-i+2) \quad (7)$$

Let T_i be the number of segments of type S_i, we can prove that (the proof is omitted):

$$T_k=C_{k+1}-m'+1,\ T_{k-1}=2(2^{d-k}-T_k)-1,\ T_i=1, i=0,1,\dots,k-2,\ T_i=0 \text{ for } i>k \quad (8)$$

Example 3. Let $L=1024$, $m=20$, $d=10$. Then $m'=m-\log L=10$. By Eq.(7), we get $C_9=3$, $C_8=10$, $C_7=25$. This gives $k=7$ since $\text{argmax}_i(m'\le C_i)=8$. Then by Eq.(8), we get $T_7=10-10+1=1$, $T_6=2(2^{(10-7)}-1)-1=13$, and $T_5=T_4=T_3=T_2=T_1=T_0=1$.

Theorem 4.1. With the above partitioning scheme, we have m segments in total.

Proof: The number of segments is

$$\sum_{i=0}^{k}T_i=T_k+2(2^{d-k}-T_k)-1+(k-1)=2^{d-k+1}-T_k+k-2=2^{d-k+1}-(C_{k+1}-m'+1)+k-2$$
$$=2^{d-k+1}-2^{d-k+1}+(d-k+3)+m'+k-3=m'+d=m. \qquad \square$$

Theorem 4.2. Using the above partitioning scheme, the total length of the m segments is $L-1$.

Proof: The total length of all segments is

$$\sum_{i=0}^{k}2^iT_i=2^kT_k+2^{k-1}T_{k-1}+...+2^0T_0=2^kT_k+2^{k-1}[2(2^{d-k}-T_k)-1]+2^{k-1}-1=2^d-1=L-1. \qquad \square$$

Example 4. In Example 3,

$$\sum_{i=0}^{k}T_i=1+13+1+1+1+1+1+1=20=m,\ \sum_{i=0}^{k}2^iT_i=2^7\times1+2^6\times13+2^5\times1+2^4\times1+2^3\times1+2^2\times1+2^0\times1=1023=L-1.$$

Theorem 4.3. Suppose that the user demands a query for a horizon of length $r\le L$. With the above partitioning scheme, suppose r' is the total length of a set of selected segments closest to r, then we can get $r'-r\le2^k$, where $k=\text{argmax}_i(m'\le C_i)-1$.

Proof: To form a set of the most recent segments with a total length closest to r, we can incrementally add S_1, S_2,… to the set until the total length of the selected segments r' is larger than r. Suppose the longest segment in the resulting set is Sg, since $r<L$ we have $g\le k$, and $r'-r\le2^g\le2^k$, here $k=\text{argmax}_i(m'\le C_i)-1$ by Eq.(7). $\square$

The above results show that our partitioning scheme achieves two goals. First, we assign shorter segments to newer data and thus give newer data higher precision. Second, the largest possible difference between the length of the approximate combination of segments and the user requested length is lowered to 2^g, where $g<k$. It can easily be seen from Eq.(7) that $k<\log L-1$, therefore the difference $r'-r$ is much smaller than $L/2$.

5 Experimental Results

To evaluate the performance of our algorithms, we test them by using both synthetic data and real data on a PC with 1.7GHz CPU and 512 MB memory running Window XP. The algorithms are coded by Visual C++ 6.0.

5.1 Testing data

We generate the synthetic data in the same way as in Ref.[21]. For each cluster, we first define a prototype $p(\cdot)$ which is a stochastic process defined by a second-order difference equation:

$$\begin{aligned} p(t+\Delta t) &= p(t)+p'(t+\Delta t) \\ p'(t+\Delta t) &= p'(t)+u(t),\ t=0,\Delta t,2\Delta t,\ldots \end{aligned} \tag{9}$$

where $u(t)$ are independent random variables uniformly distributed in an interval $[-a,a]$. The data streams in the cluster are then generated by "distorting" the prototype, both horizontally (by stretching the time axis) and vertically (by adding noise). The formulation for a data stream $x(\cdot)$is defined as:

$$x(t)=p(t+h(t))+g(t) \tag{10}$$

where $h(\cdot)$ and $g(\cdot)$ are stochastic processes generated in the same way as the prototype $p(\cdot)$. The constant a that determines the smoothness of a process can be different for $p(\cdot)$, $h(\cdot)$, and g(·), such as 0.04, 0.04, 0.5, respectively.

We can then generate different clusters by generating different prototype functions $p(\cdot)$. For each prototype function, we randomly distort the prototype to generate multiple data streams in that cluster.

The real data set (Fig.7) that we use contains the average daily temperatures of 169 cities around the world, recorded from Jan.1, 1995 to the present. Each city is regarded as a data stream and each stream has 3 416 points.

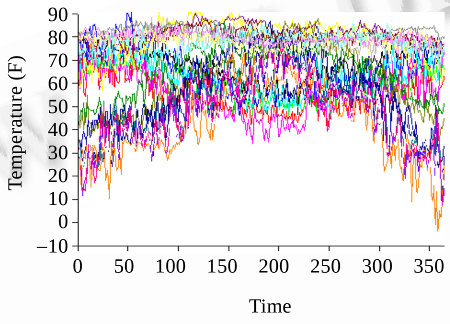

Fig.7 Daily temperatures for 169 cities around the world

5.2 Performance analysis on CORREL-cluster

5.2.1 Clustering results

We ran CORREL-cluster on the real data set downloaded from the website http://www.engr.udayton.edu/weather/default.htm to cluster cities based on the recorded daily temperatures. We set L=360 and l=30. The input of the algorithm is the daily temperatures of cities around the world. CORREL-cluster gave five clusters each of which contains cities mostly in the same continent and belonging to the same temperature zone. The correct rate is around 85% to 89%. The results are shown in Figs.8~12, where each figure illustrates one cluster.

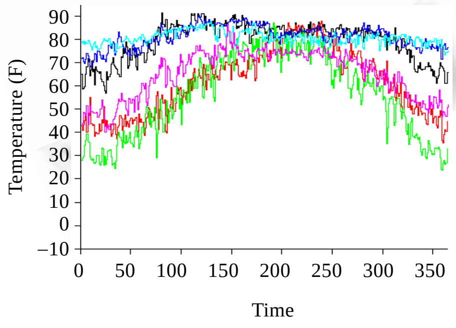

Fig.8 Cluster 1: Cities in Asia

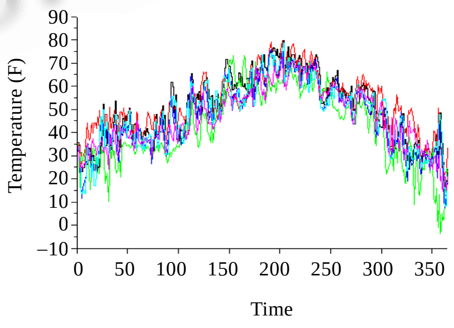

Fig.9 Cluster 2: Cities in Europe

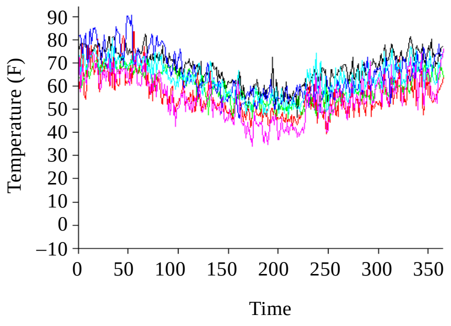

Fig.10 Cluster 3: Cities in Oceania

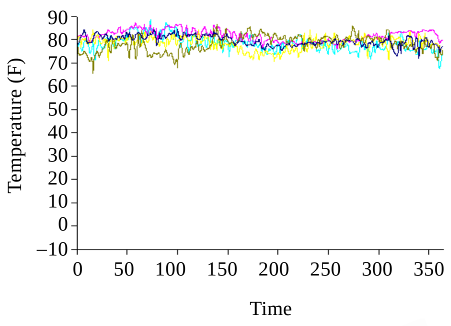

Fig.11 Cluster 4: Cities in Africa

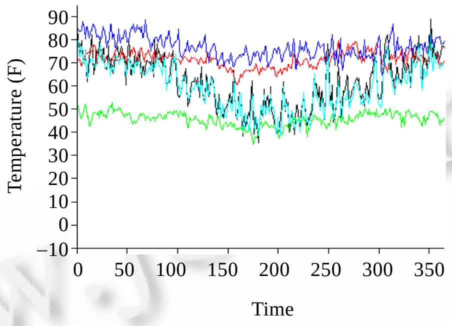

Fig.12 Cluster 5: Cities in South America

5.2.2 Quality

We evaluate the quality of the clustering from CORREL-cluster by comparing with that from DFT-cluster[21] (30 DFT coefficients). Figure 13 shows a comparison of the clustering quality on the real city-temperature data set by CORREL-cluster and DFT-cluster for various numbers of segments. The quality is measured by the correct rate, the ratio of the number of cities that are correctly labeled to the total number of cities.

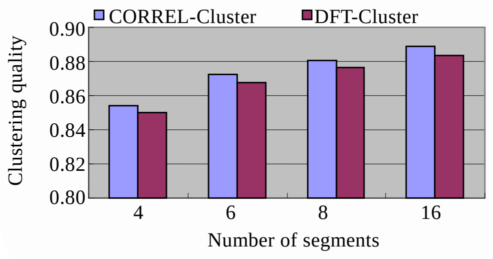

Fig.13 Clustering quality of CORREL-cluster and DFT-cluster on real data

Since we use a fixed time horizon L=360, the larger the number of segments is, the more frequently clustering is executed. Thus, for both algorithms, the quality improves when the number of segments increases. However, as we can see from Fig.13, CORREL-cluster always has a better quality than DFT-cluster.

5.2.3 Speed

Since the clustering on the real data set is too fast, we use synthetic data sets to test the processing speed of CORREL-cluster. We generate 6 synthetic data sets each of which contains 100 data streams. Each data stream has 65 536 data elements. Again, we compare it with DFT-cluster (250 DFT coefficients). The experimental results show that the executing time for CORREL-cluster is shorter than that of DFT-cluster for every synthetic data set. Figure 14 shows that the average processing time per segment for CORREL-cluster is 0.928 seconds, whereas 1.2 seconds for DFT-cluster with 250 DFT coefficients. DFT-cluster needs even longer processing time when more coefficients are used. With 1 500 DFT coefficients, DFT-cluster takes in an average over 7 seconds. Reducing the number of DFT coefficients can save time but will lead to worse quality. As we see in Fig.15, DFT-cluster with 250 DFT coefficients has much worse quality than CORREL-cluster on these synthetic data sets.

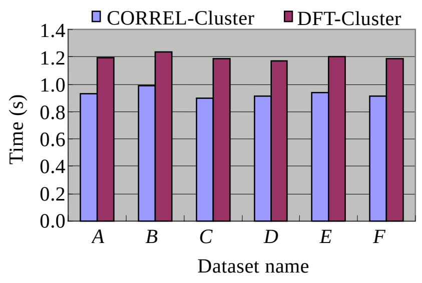

Fig.14 Computation time of CORREL-cluster and DFT-cluster on synthetic data

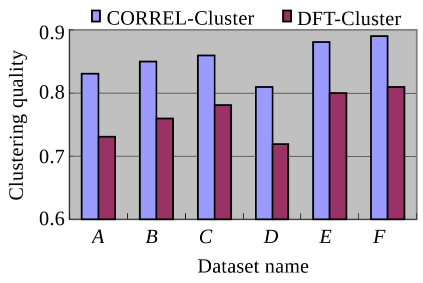

Fig.15 Quality of CORREL-cluster and DFT-cluster on synthetic data

5.2.4 Dynamic number of clusters

CORREL-cluster requires an initial value *k* for the number of clusters. We study its sensitivity to *k* by trying different initial values of *k*. Figure 16 shows the clustering results of CORREL-cluster on a synthetic data set with three clusters for different initial values of *k*, including 2, 3,4, 6, 8, 10 and 20. We can see that the number of clusters given by CORREL-cluster soon becomes the same regardless its initial value, which indicates that the initial value of *k* has little influence on the clustering performance.

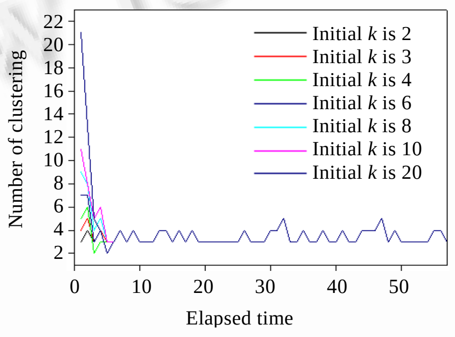

Fig.16 Number of clusters found by CORREL-cluster for different initial values of *k*

5.3 Performance analysis on CORREL-COD

5.3.1 Scalability

To evaluate the scalability of the online processing, we test our CORREL-COD algorithm and ADAPTIVE-cluster[23] with several randomly generated synthetic data sets of sizes varying from 1 000 to 10 000. As we see in Fig.17, the execution time for both algorithms increases linearly with the number of data points, but CORREL-COD is always faster than ADAPTIVE-cluster.

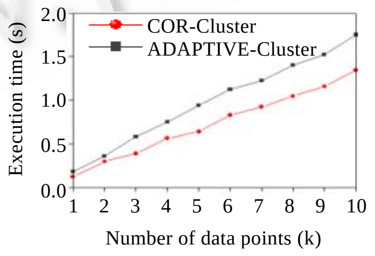

Fig.17 Comparison of the scalability of CORRL-COD and ADAPTIVE-cluster

5.3.2 Quality

We measure the quality of clustering with $G_{rawdata}/G_{COD}$, where $G_{rawdata}$ denotes the objective function obtained by clustering the raw data directly without segmentation and G_{COD} denotes the objective function by clustering based on the summary information retrieved by the online processor.

Figure 18 shows the quality of clustering on two simulated data sets for different number of segments. We see that the larger the number of segment is, the more precise our algorithm achieves. This is because the difference between the user-specified length and the length of the approximated statistical information becomes smaller when the number of segments increases. From Fig.18, we can see the clustering quality is always above 95%, which means that results by our COD algorithm are close to the optimal results that can be obtained from raw data.

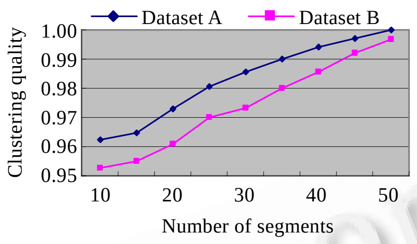

Fig.18 Clustering quality for different numbers of segments

6 Conclusions

When our aim is to mine the similarity on the trends of data streams, correlation coefficient is a more appropriate measure for similarity between data streams than Euclidean distance in previous data stream clustering methods. In this paper, we have developed algorithms CORREL-cluster and CORREL-COD for clustering multiple data streams based on correlation coefficients, which supports online clustering analysis over both the fixed and flexible time horizons. Since data streams have a high speed and massive volume, we cannot retain the raw data to perform the correlation analysis. We have proposed a compression scheme that supports an one-scan algorithm for computing the correlation coefficients. We have developed an adaptive algorithm to dynamically determine the number of clusters so that CORREL-cluster can detect the evolving behaviors of data streams. Moreover, we have developed a novel partitioning algorithm to support the clustering of arbitrary length at the user's request. Experimental results on real and synthetic data sets show that our algorithms are high in clustering quality, speed and scalability.

References:

[1] Babcock B, Datar M, Motwani R, O'Callaghan L. Maintaining variance and *k*-medians over data stream windows. In: Proc. of the 22nd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems. San Diego, 2003. 234–243.

[2] Aggarwal CC, Han J, Wang J, Yu PS. On demand classification of data streams. In: Proc. of the 2004 Int'l Conf. on Knowledge Discovery and Data Mining (KDD 2004). Seattle, 2004.

[3] Wang H, Fan W, Yu PS, Han J. Mining concept-drifting data streams using ensemble classifiers. In: Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining. Washington: ACM Press, 2003. 226–235.

[4] Giannella C, Han J, Pei J, Yan X, Yu PS. Mining frequent patterns in data streams at multiple time granularities. In: Kargupta H, Joshi A, Sivakumar K, Yesha Y, eds. Proc. of the Next Generation Data Mining. AAAI/MIT, 2003.

[5] Manku G, Motwani R. Approximate frequency counts over data streams. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases. Hong Kong: Morgan Kanfmann Publishers, 2002. 346–357.

[6] Metwally A, Agrawal D, El Abbadi A. Efficient computation of frequent and top-*k* elements in data streams. In: Proc. of the 10th ICDT Int'l Conf. on Database Theory. California, 2005. 398–412.

[7] Aggarwal CC, Han J, Wang J, Yu PS. A framework for clustering evolving data streams. In: Proc. of the 29th Very Large Databases Conf. Berlin: Morgan Kaufmann Publishers, 2003. 81−92.

[8] Aggarwal CC, Han J, Wang J, Yu PS. A framework for projected clustering of high dimensional data streams. In: Proc. of the 2004 Int'l Conf. Very Large Data Bases (VLDB 2004). Toronto, 2004. 852−863.

[9] Domingos P, Hulten G. A general method for scaling up machine learning algorithms and its application to clustering. In: Proc. of the 18th Int'l Conf. on Machine Learning. Williamstown: Morgan Kaufmann Publishers, 2001. 106−113.

[10] Guha S, Mishra N, Motwani R, O'Callaghan L. Clustering data streams. In: Proc. of the 41st Annual Symp. on Foundations of Computer Science. Washington: IEEE Computer Society, 2000. 359−366.

[11] Guha S, Rastogi R, Shim K. CURE: An efficient clustering algorithm for large databases. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Seattle: ACM Press, 1998. 73−84.

[12] Henzinger MR, Raghavan P, Rajagopalan S. Computing on data streams. Technical Report, SRC Technical Note 1998-011, Palo Alto: Digital Systems Research Center, 1998.

[13] Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: Proc. of the 7th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. San Francisco: ACM Press, 2001. 97−106.

[14] Keogh E, Kasetty S. On the need for time series data mining benchmarks: A survey and empirical demonstration. In: Proc. of the 8th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. Edmonton, 2002. 102−111.

[15] Madden S, Franklin M. Fording the stream: An architecture for queries over streaming sensor data. In: Proc. of the 18th Int'l Conf. on Data Engineering. 2002. 555−566.

[16] O'Callaghan L, Mishra N, Meyerson A, Guha S, Motwani R. Streaming-Data algorithms for high-quality clustering. In: Proc. of the 18th Int'l Conf. on Data Engineering. Washington: IEEE Computer Society, 2002. 685−694.

[17] Zhang T, Ramakrishnan R, Livny M. BIRCH: An efficient data clustering method for very large databases. In: Proc. of the ACM SIGMOD Int'l Conf. on Management of Data. Montreal, 1996. 103−114.

[18] Ester M, Kriegel HP, Sander J, Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. of the Int'l Conf. on Knowledge Discovery and Data Mining (KDD'96). Portland: AAAI Press, 1996.

[19] Agrawal R, Gehrke J, Gunopulos D, Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In: Proc. of the ACM SIGMOD Conf. Seattle, 1998. 94−105.

[20] Yang J. Dynamic clustering of evolving streams with a single pass. In: Proc. of the 19th IEEE Int'l Conf. on Data Engineering (ICDE 2003). Bangalore, 2003. 695−697.

[21] Beringer J, Hüllermeier E. Online-Clustering of parallel data streams. Data and Knowledge Engineering, 2006,58(2):180−204.

[22] Sakurai Y, Panadimitriou S, Sun J, Faloutsos C. Braid: Stream mining through group lag correlations. In: Proc. of the ACM SIGMOD. Baltimore, 2005. 180−201.

[23] Dai BR, Huang JW, Yeh MY, Chen MS. Adaptive clustering for multiple evolving streams. IEEE Trans. on Knowledge and Data Engineering, 2006,18(9):1166−1180.

TU Li was born in 1980. She is a Ph.D. candidate at the Institute of Information Science and Technology, Nanjing University of Aeronautics and Astronautics and a teacher at the Department of Computer Science, Jiangyin Polytechnic College. Her research areas are data mining and parallel computation.

CHEN Ling was born in 1951. He is a professor and Ph.D. supervisor at the Department of Computer Science and Engineering, Yangzhou University and a CCF senior member. His research areas are data mining, bioinformatics and parallel computation.

ZOU Ling-Jun was born in 1984. She is a master candidate at the Department of Computer Science and Engineering, Yangzhou University. Her research areas are optimization algorithm and parallel computation.