Knowl Inf Syst (2008) 15:181–214
DOI 10.1007/s10115-007-0070-x

Knowledge and Information Systems

REGULAR PAPER

Aoying Zhou · Feng Cao · Weining Qian · Cheqing Jin

Tracking clusters in evolving data streams over sliding windows

Received: 5 October 2005 / Revised: 2 September 2006 / Accepted: 17 January 2007 /
Published online: 9 March 2007

Abstract Mining data streams poses great challenges due to the limited memory availability and real-time query response requirement. Clustering an evolving data stream is especially interesting because it captures not only the changing distribution of clusters but also the evolving behaviors of individual clusters. In this paper, we present a novel method for tracking the evolution of clusters over sliding windows. In our SWClustering algorithm, we combine the exponential histogram with the temporal cluster features, propose a novel data structure, the Exponential Histogram of Cluster Features (EHCF). The exponential histogram is used to handle the in-cluster evolution, and the temporal cluster features represent the change of the cluster distribution. Our approach has several advantages over existing methods: (1) the quality of the clusters is improved because the EHCF captures the distribution of *recent* records precisely; (2) compared with previous methods, the mechanism employed to adaptively maintain the in-cluster synopsis can track the cluster evolution better, while consuming much less memory; (3) the EHCF provides a flexible framework for analyzing the cluster evolution and tracking a specific cluster efficiently without interfering with other clusters, thus reducing the consumption of computing resources for data stream clustering. Both the theoretical analysis and extensive experiments show the effectiveness and efficiency of the proposed method.

Keywords Cluster tracking · Evolving · Data streams · Sliding windows

A. Zhou (✉) · F. Cao · W. Qian · C. Jin
Department of Computer Science and Engineering, Fudan University, Shanghai 200433, P.R. China
E-mail: ayzhou@fudan.edu.cn

F. Cao
IBM China Research Lab, Beijing 100094, P.R. China

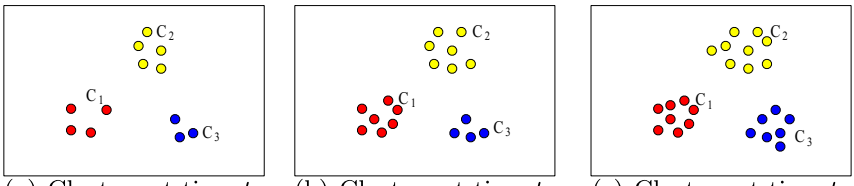
C. Jin
Department of Computer Science, East China University of Science and Technology, Shanghai 200237, P.R. China

1 Introduction

Recently, substantial amount of researches are devoted to mining continuous data streams [6, 18, 23, 33]. The typical applications include network traffic monitoring, credit card fraud detection, and sensor network data processing, etc. As a common data mining task, clustering is widely studied to reveal similar features among data records. In data stream scenario, the clustering techniques are applied to group multidimensional records such that the intracluster similarity is maximized, and the intercluster similarity is minimized. Due to the restriction of the computing resources, clustering data stream has to be performed online with limited memory requirement and one-pass data scan.

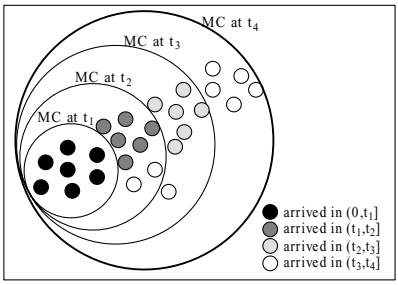
Clustering evolving data streams has been an interesting research topic because of its wide and potential applications. Generally speaking, clustering over an entire data stream is dominated by the outdated historic information of the stream. The resulting clusters are of less use from an application point of view. Therefore, clustering evolving data streams over sliding windows becomes a natural choice, since the most recent N records are considered to be more critical and preferable in many applications [5, 12, 17]. However, since data stream clustering is applied with limited memory availability (relative to the size of the sliding window), it is impossible to load the entire data set into memory. Thus, approaches to approximate the clustering results are necessary. Fortunately, a small deviation from the exact value in a sliding window is acceptable in most practical applications. For example, in network traffic monitoring the administrator may often query: "What is the distribution of the most recent 10,000,000 network connections with the tolerance of 10,000 connections?"

Furthermore, the formation of current clusters can help users better understand the evolving behavior of the clusters. Previous clustering methods often adopt a global uniform strategy to maintain the synopses. However, it is preferable to adaptively maintain separate information for each cluster. Since every cluster has its own evolving behavior, the uniform strategy may lose a lot of valuable information. Characteristics of a cluster (e.g., the number of objects, the center and radius of the cluster) often change as data streams proceed. Within a certain period of time, the evolving behaviors of different clusters are usually different. For instance, in Fig. 1, there exist three clusters C_1, C_2, and C_3 in the data stream at time t_1. At time t_2, the numbers of records in clusters C_1, C_2, and C_3 increase by 3, 2, and 1, respectively. At time t_3, the numbers of newly arrived records in those clusters change to 1, 2, and 3, respectively. Conceivably, the evolving behavior of each cluster may also change significantly as new records arrive continuously.

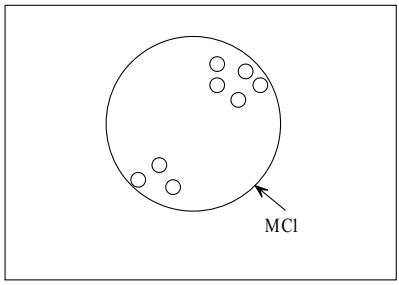


(a) Clusters at time t_1 (b) Clusters at time t_2 (c) Clusters at time t_3

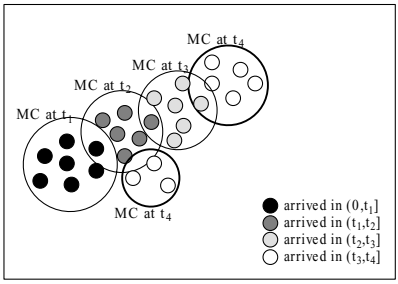
Fig. 1 The evolving clusters in a data stream



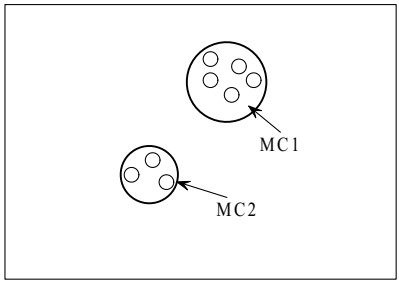
(a) The formation of micro clusters in CluStream



(b) The micro clusters formed by recent records in CluStream



(c) The formation of microclusters in our sliding-window-based method



(d) The micro clusters formed by recent records in our sliding-window-based method

Fig. 2 The forming of microclusters

Motivated by this observation, we propose an in-cluster maintenance strategy to track the clusters in evolving data streams over sliding windows. Sliding windows conveniently eliminate the outdated records. Previously proposed clustering algorithms cannot be easily extended to this scenario. For example, a direct extension of CluStream [1] to sliding windows requires that the intermediate clustering results (i.e., current set of microclusters, called a snapshot in CluStream) be maintained simultaneously whenever a new record arrives. Such an expensive operation will result in large overhead for online processing even if all the snapshots can be maintained in memory.

Designed for a long-term clustering analysis, CluStream fails to capture the precise distribution of *recent* records. The old records have great influence on the formation of the microcluster, which is illustrated in Fig. 2a. In CluStream, old records arriving before t_3 are not eliminated. To answer a clustering request for *recent* records (i.e., the records arriving within (t_3, t_4), CluStream subtracts the microcluster at t_4 from that at t_3, resulting in only one microcluster for *recent* records, as shown in Fig. 2b.

In general, when the center of a microcluster shifts gradually, CluStream always maintains the microcluster with growing radius, instead of splitting it into multiple microclusters. Since it introduces more memory consumption, splitting operation is not implemented in CluStream. CluStream creates a new microcluster if and only if arriving records cannot be incorporated into existing microclusters.

When a record is within the maximum boundary of a microcluster (e.g., a factor of the Root Mean Square (RMS) deviation of the records from the microcluster centroid), it will be absorbed by the microcluster, which may result in a larger boundary of the microcluster. A microcluster with large boundary will absorb more records, which leads to further increase on the boundary of microclusters, as shown in Fig. 2b.

If we promptly eliminate the influence of the old records in the microcluster when new records arrive, a new microcluster can be created in the interval $(t_3, t_4]$. The formation of the two microclusters is shown in Fig. 2c. Taking the history into account, it is natural to put the newly arriving records into one microcluster as shown in Fig. 2a. However, the final resulting clusters with two microclusters in Fig. 2d better capture the distribution of *recent* records. Our experimental results (refer to Figs. 7 and 9) support this claim.

In a sliding window, the old records expire while the new records arrive continuously. It is very important to provide an efficient mechanism to eliminate the effect of the expired records and incorporate new records into the synopses. However, this is not a trivial task. Once two synopses have been merged, there is no way to split them, but the splitting is required when old records expire from the sliding window.

The contributions of this paper are summarized as follows:

- We propose a novel synopsis called *Exponential Histogram of Cluster Feature* (EHCF). EHCFs maintain the cluster features over a sliding window and bound the number of expired records within ϵN, where N is the length of the sliding window. This in-cluster structure adaptively updates the Temporal Cluster Feature based on the new records of each cluster. It is suitable for capturing the distribution of recent records as well as the evolving behavior of each cluster.
- Based on EHCFs, we present a new clustering algorithm, SWClustering, for data streams over sliding windows. SWClustering is capable of analyzing not only the clusters but also the evolution of the individual clusters. It effectively eliminates the influence of old records while incorporating new records. This mechanism leads to better clustering quality than previous ones.
- The memory consumption of SWClustering is bounded. The optimization techniques for further reducing the consumption of computing resource are discussed as well. Comprehensive empirical study shows that SWClustering can achieve high quality clustering with low overhead, illustrating the effectiveness and efficiency of the proposed algorithm.

The remainder of this paper is organized as follows. Section 2 surveys related work. Section 3 presents the synopsis EHCF along with the technique for merging two EHCFs. SWClustering algorithm is described in Section 4. The experiment setting and the analysis of experimental results are presented in Section 5. Section 6 concludes this paper.

2 Related work

Previous algorithms on clustering data streams can be classified into two categories: one-pass approach and evolving approach.

2.1 One-pass approach

The one-pass approach clusters data stream by scanning the data stream only once. Traditional clustering methods such as the k-means were extended to their data stream versions under the assumption that the data objects arrived in chunks [11, 23, 24].

Guha et al. [23, 24] proposed a k-means based algorithm which requires $O(kN)$ time and $O(N^{\epsilon})$ space, where k is the number of centers, N is the stream length, and $\epsilon < 1$. It is proved that the lower bound of the run time is $O(kN)$ for any k-median algorithm achieving constant factor approximation. The algorithm starts with clustering the first $O(M/k)$ points into $2k$, where M/k is the sample size determined by the available memory size, then a local search algorithm is used to cluster $O(M)$ medians of level i to $2k$ medians of level $i+1$. This process is repeated until the $O(k)$ medians is clustered into k groups.

Charikar et al. [11] proposed another k-median based algorithm which requires $O(kpoly\log n)$ space. They theoretically addressed the problem of increasing approximation factors in algorithm [23] where the increase in the number of levels leads to the increasing approximation factors in the final results.

Chalaghan et al. [10] proposed the STREAM algorithm to cluster data streams. STREAM first determines the size of sample. If the size of data chunk exceeds the sample size, a LOCALSEARCH procedure is invoked to obtain the clusters of the chunk. Finally, the LOCALSEARCH is applied to all the cluster centers generated in the previous iterations.

Domingos et al. [19] extended the k-means algorithm and proposed the VFKM algorithm. It is guaranteed that the model produced does not differ significantly from the one that would be obtained with infinite data. The Hoeffding bound is adopted to determine the number of examples needed in each step of k-means algorithm.

Ordonez [37] addressed the problem of clustering binary data streams. A variant of the k-means algorithm, incremental k-means, was proposed to obtain high quality solutions. Incremental k-means requires $O(kNT)$, where T is the average size of the records, N is the number of records, and k is the number of clusters. Sparse matrix operations and simple sufficient statistics were designed for speedup. He et al. [25] proposed another method for clustering the categorical data streams based on the squeeze algorithm [26]. A Lossy Counting [33] procedure is adopted to maintain the histogram. Aggarwal et al. [4] presented an online approach for clustering massive text and categorical data streams with the use of a cluster feature [40] summarization methodology.

The empirical study in Keogh et al. [30, 31] shows that many time series data streams clustering algorithms come up with meaningless results in subsequence clustering, so a solution using k-motif to choose the subsequences was proposed in their paper. Rodrigues et al. [38] proposed a time-series clustering system which incrementally constructed a hierarchy of clusters. The correlation between time-series is used as similarity measure. Cluster splitting or aggregation is carried out at each time step.

2.2 Evolving approach

The evolving approach [1, 16] views the behavior of streams as an evolving process over time. For this approach, there are three kinds of window models [42], i.e., landmark window, sliding window, and fading window. Among them, the sliding window model is widely adopted in stream mining [5, 12, 13, 17, 32].

Cao et al. [9] proposed a density-based method for discovering arbitrary-shape clusters in an evolving data stream with noise. A core-microcluster structure is introduced to summarize the clusters with arbitrary shape, while the potential core-microcluster and outlier microcluster structures are proposed to maintain and distinguish the potential clusters from outliers. A novel pruning strategy is designed based on these concepts, and the precision of the weights of the microclusters is guaranteed with limited memory. TECNO-STREAMS, an artificial immune system (AIS) based clustering approach, was proposed by Nasraoui [35]. The system is based on a dynamic weighted B-cell model to improve the learning ability and scalability of traditional AIS learning methods. Its sample usage includes mining user profiles in noisy Web click stream data [36]. Different from these approaches aiming at discovering arbitrary-shape clusters, our method focuses on the clustering problem over sliding windows.

Aggarwal et al. [2, 3] developed a projected data stream clustering method, HPStream. The main contributions are the fading cluster structure and the methodology of projection-based clustering. HPStream is effective in finding clustering in subspace.

Babcock et al. [7] theoretically studied the problem of clustering data in sliding windows based on the previous work [23]. They focused on the theoretical bound of the performance. In contrast, our work is devoted to the problem of analyzing the evolution of clusters in sliding windows.

The work most similar to our SWClustering algorithm is CluStream [1] which is designed to cluster data streams over different time horizons in an evolving environment. Since the snapshots of clustering results over the landmark window have to be stored, CluStream consumes more space, thus not suitable for the clustering problem over sliding windows. In addition, the influence of expired records cannot be promptly eliminated during online clustering, while such a prompt elimination is required in sliding window clustering.

There are some other related work on clustering multiple data streams. Beringer et al. [8] developed an online version of k-means algorithm for clustering multiple data streams based on scalable online transformation of the raw data. The transformation allows a fast computation of approximate distances between streams. COD, a general framework of clustering multiple data streams, was introduced by Dai et al. [15]. COD dynamically clusters multiple data streams and offers support to meet flexible mining requirements. The problem of clustering data streams with increasing dimensionality over time is addressed in Yang [39], where a weighted distance metric between two streams is used, and an incremental algorithm is proposed to produce stream clusters.

3 Exponential histogram of cluster feature

3.1 EHCF synopsis

Assume that a data stream consists of a set of multidimensional records $x_1, \ldots, x_i, \ldots$ arriving at time stamps $t_1, \ldots, t_i, \ldots, x_i = (x_i^1 \ldots x_i^d)$. In sliding window model, only the most recent N records are considered at any time. The most recent N records are called *active* records, and the rest are called *expired* records which no longer contribute to the clustering.

To construct cluster features based on the most recent N records, we propose a novel synopsis data structure, called *Exponential Histogram of Cluster Feature* (EHCF). Every bucket in an EHCF is a *Temporal Cluster Feature* (TCF) for a set of records.

Definition 1 (Temporal Cluster Feature (TCF)) A *Temporal Cluster Feature* (TCF) for a set of d-dimensional records $x_1 \ldots x_n$ with time stamps $t_1 \ldots t_n$ is defined as a $(2 \cdot d + 2)$-dimension vector $(\overrightarrow{CF2^x}, \overrightarrow{CF1^x}, t, n)$, where n is the number of records; $\overrightarrow{CF2^x}$ is the squared sum of the n records, i.e., $\overrightarrow{CF2^x} = \sum_{i=1}^{n} x_i^2$; $\overrightarrow{CF1^x}$ is the linear sum of the n records, i.e., $\overrightarrow{CF1^x} = \sum_{i=1}^{n} x_i$; and t is the time stamp of the most recent record, i.e., $t = t_i$.

Note that a TCF is a temporal extension of the *cluster feature vector* in Zhang et al. [40]. Aggarwal et al. [1] extends the *cluster feature vector* by summing up the time stamps in CluStream algorithm. However, with the time stamp of the most recent record in a TCF, it is possible to determine the time span of records rather than approximating it as CluStream does.

The *Temporal Cluster Feature* for a set of records C is represented as $\overrightarrow{TCF}(C)$. Let $V(C)$ be the number of records in C, a TCF is called an l-level TCF iff $V(C) = 2^l$. TCFs can be formed in an additive way, as shown later.

Property 1 $\overrightarrow{TCF}(C_1 \cup C_2)$ can be constructed from $\overrightarrow{TCF}(C_1)$ and $\overrightarrow{TCF}(C_2)$, where C_1 and C_2 are two sets of records.

By Definition 1, fields $\overrightarrow{CF1^x}$, $\overrightarrow{CF2^x}$ and n of $\overrightarrow{TCF}(C_1 \cup C_2)$ are just the sum of the corresponding fields in $\overrightarrow{TCF}(C_1)$ and $\overrightarrow{TCF}(C_2)$. Note that the most recent record in $C_1 \cup C_2$ must be one of the most recent record in C_1 or C_2, i.e., the field t of $\overrightarrow{TCF}(C_1 \cup C_2)$ is $\max(\overrightarrow{TCF}(C_1).t, \overrightarrow{TCF}(C_2).t)$.

Definition 2 (Exponential Histogram of Cluster Feature (EHCF)) Given a user-defined parameter ϵ $(0 < \epsilon < 1)$, an EHCF is defined as a collection of TCFs on a set of records C_i with the following constrains. (1) All records in C_i arrive earlier than records in C_j for $i < j$. (2) C_1, the first set, contains only one record, while any other set C_i $(i > 1)$ contains either the same or as twice much number of records as its prior set, i.e., $V(C_i) = V(C_{i-1})$ or $V(C_i) = 2 \cdot V(C_{i-1})$. (3) $\frac{1}{\epsilon}$ or $\frac{1}{\epsilon} + 1$ l-level TCFs can be found for each l $(0 \leq l < L)$ except for the L-level, where L is the highest level of TCFs in the EHCF.

Notice that ϵ limits the number of expired records in an EHCF within $[0, \epsilon n]$, where n is number of records in the EHCF. This is because only the last TCF in the EHCF may contain the expired records, it contains at most ϵn records.[1]

Exponential Histogram technology is a widely adopted approximate structure [17, 22]. Datar et al. [17] proposed an EH-based method to maintain aggregates over sliding windows. In our EHCF structure, we adopt the Exponential Histogram technique to maintain the TCFs.

Definition 3 (Center of EHCF) The center c of an EHCF H with m TCFs is defined as the mean of the $\overrightarrow{CF1^x}$ of all the TCFs in H, i.e., $c = \frac{\sum_{i=1}^{m} \overrightarrow{CF1^x}_i}{\sum_{i=1}^{m} n_i}$, where $TCF_i = (\overrightarrow{CF2^x}_i, \overrightarrow{CF1^x}_i, t_i, n_i)$, $1 \leq i \leq m$.

The center of an EHCF is used to determine the distance between the EHCF and a newly arrived record.

3.2 Updating an EHCF

When a record x_p arrives, an EHCF synopsis can be updated incrementally as follows:

1. Generate $\overrightarrow{TCF}(C)$ according to Definition 1, where C is a data set containing only x_p.
2. Append $\overrightarrow{TCF}(C)$ to the EHCF synopsis. If there are $\lceil \frac{1}{\epsilon} \rceil + 2$ 0-level TCFs, merge two oldest 0-level TCFs into a new 1-level TCF. The number of 0-level TCFs is reduced to $\lceil \frac{1}{\epsilon} \rceil$. The merge operation will cascade to level $l = 1, 2, \ldots$ if the number of TCFs in level l becomes $\lceil \frac{1}{\epsilon} \rceil + 2$. The Steps 1 and 2 form the merging process.
3. Check the field t of the last TCF in the EHCF. If the time stamp t is not one of the most recent N time stamps any more, the TCF is dropped, and the occupied memory is reclaimed.

Example 1 (Merging Process of an EHCF) Figure 3 illustrates the merging process of an EHCF synopsis for 10 records, $x_1, \ldots, x_{10}$, arriving at time stamps $1, \ldots, 10$. The user-defined parameter ϵ is set to be 0.5. In real applications, the ϵ should be far less than 1. At time stamp 4, a new TCF for record x_4 is generated, which results in 4 $(= \frac{1}{\epsilon} + 2)$ 0-level TCFs in the EHCF. A new synopsis $\overrightarrow{TCF}(\{x_1, x_2\})$ is generated by merging $\overrightarrow{TCF}(\{x_1\})$ and $\overrightarrow{TCF}(\{x_2\})$. Similar merge operations occur at time stamps 6 and 8. At time stamp 10, the arrival of record x_{10} triggers the merge of $\overrightarrow{TCF}(\{x_7\})$ and $\overrightarrow{TCF}(\{x_8\})$, which further triggers the merge of $\overrightarrow{TCF}(\{x_1, x_2\})$ and $\overrightarrow{TCF}(\{x_3, x_4\})$.

EHCF bounds the number of expired records within $[0, \epsilon n]$ with limited memory consumption $(\frac{1}{\epsilon} + 1)(\log(\epsilon n + 1) + 1)$. Following theorems give the memory usage of EHCF synopsis.

[1] For the EHCF contains n records, the last TCF contains s records, and $\frac{1}{\epsilon}(1+2+4+\cdots+s) \leq n$. Therefore, $s \leq \epsilon n$ holds.

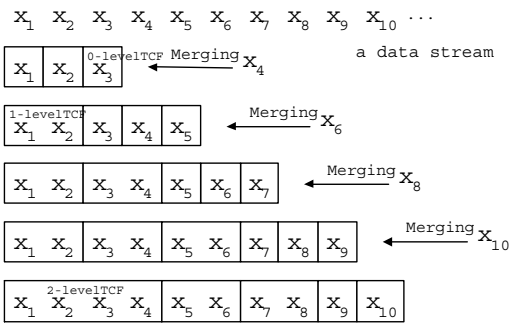


Fig. 3 Merging processes of an EHCF: $\epsilon = 0.5$

Theorem 1 (Datar et al. [17]) *Exponential Histogram (EH) Structure computes an ϵ-deficient synopsis using at most $(\frac{1}{\epsilon}+1)(\log(\epsilon N+1)+1)$ buckets, where N denotes the size of the sliding window.*

Theorem 2 *Given an EHCF synopsis of n records, there are at most $(\frac{1}{\epsilon}+1)(\log(\epsilon n+1)+1)$ TCF synopses in the EHCF.*

Proof Given any EHCF of n records and a user-defined parameter ϵ, we can construct an EH structure with window size n and parameter ϵ. As each TCF in the EHCF can be mapped to a bucket in the EH structure and vice versa, from Theorem 1, there are at most $(\frac{1}{\epsilon}+1)(\log(\epsilon n+1)+1)$ TCF synopses in the EHCF. □

By observing the process of updating EHCF, we obtain the following theorem:

Theorem 3 *Assume that an EHCF synopsis is constructed using n records. For any TCF F_i in the EHCF, F_i can be constructed from its following 2^l records if there are $(\lceil\frac{1}{\epsilon}\rceil(2^l-1)$ records before F_i.*

See Appendix A for the proof.

3.3 Merge two EHCFs

Merging two EHCFs is essential to bound the space complexity while keeping the correctness of newly generated EHCFs. Assuming that H_1 and H_2 denote two nearest EHCF synopses containing m_1 and m_2 records, respectively, two cases need to be considered when merging H_1 and H_2.

First consider the case when H_1 and H_2 are nonoverlapping, i.e., all records in H_1 arrive before all records in H_2. Merging H_1 and H_2 in this case can be done by simply placing the TCFs in H_1 followed by placing the TCFs in H_2, then sweeping from right to left to merge the TCFs.

Theorem 4 *Assume that H_1 and H_2 are two nonoverlapping EHCF synopses containing m_1 and m_2 records, respectively. The time complexity of merging H_1 and H_2 is $O(\frac{1}{\epsilon}\log(\epsilon(m_1+m_2)))$. The new EHCF synopsis consists of $O(\frac{1}{\epsilon}\log(\epsilon(m_1+m_2)))$ TCF synopses.*

See Appendix B for the proof.

Second, consider when H_1 and H_2 are overlapping, i.e., the arrival time of the records in H_1 and H_2 are interleaved. Here H_1 and H_2 cannot be merged in a straightforward way. Instead, a new EHCF H_{new} is created where two pointers are maintained to point to H_1 and H_2. Because no record is absorbed at the very beginning of the merging, H_{new} does not contain any TCFs initially. The center of H_{new} is initialized to be the mean of the $\overrightarrow{CF1^x}$'s of all TCFs in H_1 and H_2.

In the latter case, H_{new} is created conceptually to merge H_1 and H_2 with H_1 and H_2 being temporarily kept, while in the first case, a new EHCF is created to replace H_1 and H_2 directly .

The EHCF being merged is called a *merging-EHCF* (e.g., H_1 or H_2). In order to keep the EHCF structures of merging-EHCFs, we create new TCFs (called *void-TCF*s) conceptually in merging-EHCFs. In fact, the *void-TCF*s does not need to be created. Only a counter is maintained instead.

H_1, H_2, and H_{new} form an EHCF-tree which can be used to merge overlapping EHCFs. Each entry in the tree is an EHCF synopsis. The records in a parent entry arrive after all the records in child entries. For example, the records in H_{new} arrive after all the records in H_1 and H_2.

EHCF-trees may merge with another one as well. For example, H_{new} may merge with an overlapping EHCF H_3. This merging operation consists of two steps: (1) H_{new} is merged into H_1(or H_2) directly, because H_{new} does not overlap with H_1 or H_2. After this step, there is no TCF in H_{new}. (2) H_{new} is merged with H_3, and a new EHCF H'_{new} is created.

When a new record is absorbed by the root, the maintenance of the EHCF-tree is as follows:

1. The root creates a new TCF and maintains its own structure, as discussed in Sect. 3.2.
2. One of the child nodes of the root creates a 0-level *void-TCF* and maintains its structure as if the new record is absorbed by itself (called *virtual absorbing*). The *virtual absorbing* processes propagate from root to leaves. For the nodes on the same level, the probability of virtually absorbing a record is equal, unless it (with its descendant EHCFs) contains only one TCF, in which case the probability is 0, since the *virtual absorbing* does not reduce the number of TCFs in the tree.
3. Assume V_{H_l} and V_{H_r} denote the number of records contained in the left and right child, respectively. If a node H_i absorbs $\lceil \frac{1}{\epsilon} \rceil (2^{\lceil \log(V_{H_l}+V_{H_r}) \rceil} - 1)$ records, then all the TCFs in its child EHCFs can be merged into a new TCF and appended to the end of H_i. The child EHCFs are deleted accordingly.

Figure 4 illustrates an EHCF-tree whose root absorbs four new records. If an EHCF has absorbed $2(\lceil \frac{1}{\epsilon} \rceil)(2^l - 1)$ records, according to Theorem 3, all the TCFs in their two child *merging-EHCF*s can be merged into new TCFs with level $\geq l$. Thus, the number of TCFs in *merging-EHCF*s is gradually reduced. Furthermore, when H_{new} absorbs $\lceil \frac{1}{\epsilon} \rceil 2^{\lceil \log(V_{H_1}+V_{H_2}) \rceil}$ records, H_1 and H_2 can be merged into one TCF and appended to H_{new}. Since the real merging operation is performed after the very beginning of merging, such process is called *late merging*. The following example demonstrates the *late merging*.

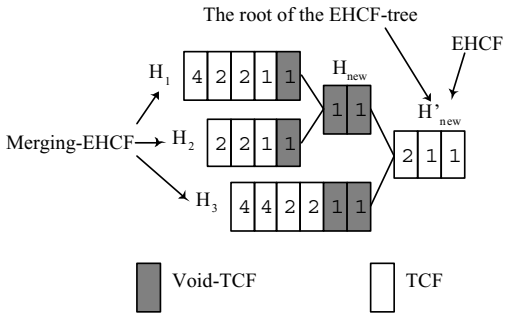


Fig. 4 An EHCF-tree

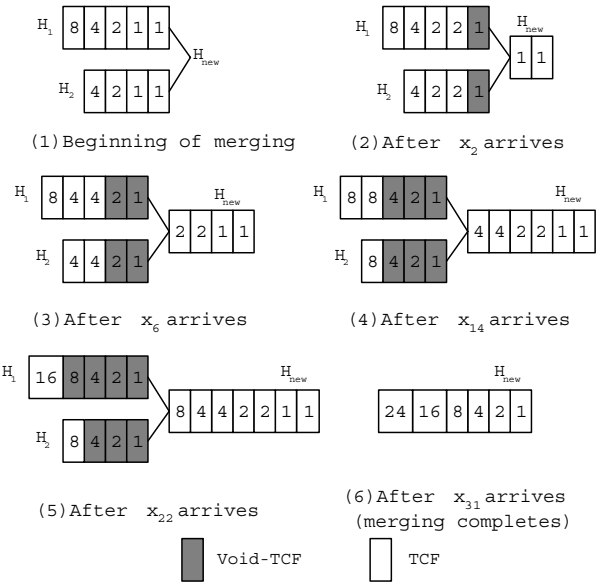


Fig. 5 Late merging

Example 2 (Late Merging of EHCFs) Assume that two overlapping EHCFs (H_1 and H_2) are to be merged. $V_{H_1} = 16$, $V_{H_2} = 8$, and H_{new} is the newly created EHCF. The *late merging* of H_1 and H_2 is illustrated in Fig. 5. The shadowed boxes in Fig. 5 represent the *void-TCF*s. The number in the box indicates the number of records contained in the TCF. For the sake of simplicity, we assume $\epsilon = 1$ here. (ϵ can be set to a positive value far less than 1 in a real application.) So, there are at least $\lceil \frac{1}{\epsilon} \rceil = 1$ and at most $\lceil \frac{1}{\epsilon} \rceil + 1 = 2$ i-level TCFs for each i. New records $x_1, x_2, x_3, \ldots$ are continuously absorbed by H_{new} after the very beginning of merging. We do not consider the expiring process here.

Step (1) illustrates the initial state of merging. H_1 contains 5 TCFs, while H_2 contains 4 TCFs. A new EHCF H_{new} is initialized to contain 0 TCF. In Step(2), x_1 and x_2 have been absorbed by H_{new}, so there are two 0-level TCFs in H_{new}, while one 0-level *void-TCF* is created in H_1 and H_2, respectively. Because the

0-level TCFs is 3 ($= \frac{1}{\epsilon} + 2$), the two oldest 0-level TCFs are merged into a new 1-level TCFs, and the numbers of TCFs in H_1 and H_2 are reduced to 4 and 3, respectively. Step(3) illustrates the state after x_6 arrives. Because there are $2(\lceil \frac{1}{\epsilon} \rceil)(2^2 - 1) = 6$ records in H_{new}, the 1-level TCFs in H_1 and H_2 become 2-level TCFs. Step(4) shows the state when H_2 becomes a 3-level TCF, since there are $2(\lceil \frac{1}{\epsilon} \rceil)(2^3 - 1) = 14$ records in H_{new}. In Step(5), after record x_{22} arrives, H_1 contracts to one TCF. Step(6) represents the final merging. When x_{31} arrives, there are $\lceil \frac{1}{\epsilon} \rceil (2^{\lceil \log(V_{H_1} + V_{H_2}) \rceil} - 1) = 31$ records in H_{new}, so H_1 and H_2 are merged into one new 5-level TCF which is then appended to H_{new}. Notice that the number of records incorporated in the last TCF of H_{new} is 24, slightly smaller than 32. But it has no impact on the whole EHCF structure.

3.4 Comparing EHCF with pyramidal time frame

The EHCF is originally designed for sliding window applications, while the pyramidal time frame and microcluster in CluStream [1] are proposed for landmark windows. The differences between them are discussed in the following.

3.4.1 Granularity of microclusters

As the clusters in an evolving data stream always change, the synopses used in clustering methods have to capture the distribution of recent records at any given time. The EHCF synopsis employs a mechanism for eliminating old records in online clustering, which prevents the radius of the EHCF from becoming larger and larger when the center of the cluster shifts. In such sense, an EHCF is a fine granularity "microcluster".

In case of cluster drifting, the radius of the microcluster in CluStream monotonically increases. Although subtraction is performed to reduce the influence of old records when processing a clustering request, the influence is not totally eliminated when incorporating new records. As a result, the microcluster becomes coarse as its radius increases, which may result in undistinguishable clusters.

3.4.2 Maintenance of synopses and the impact of outliers

EHCF synopsis adaptively adjusts the frequency of creating new TCFs. As each EHCF maintains its own histogram, a rapidly changing cluster will maintain more TCFs than a slowly changing one. Since the pyramidal time frame is designed for all possible time horizons, it has to store a snapshot for every microcluster reaching the pyramidal time. However, this strategy ignores the property of individual clusters. It is also difficult to coordinate the different frequency of storage required by different clusters.

During processing a data stream, a new synopsis need be created for each new record if it can not be absorbed by any existing microcluster. Such a record is probably an outlier. In CluStream, a new microcluster is created for such an outlier, and the storage of the microcluster is repeatedly allocated until it is eliminated. By contrast, the new EHCF creates only 1 TCF for the outlier. Therefore, EHCF saves a lot of memory at the presence of outlier.

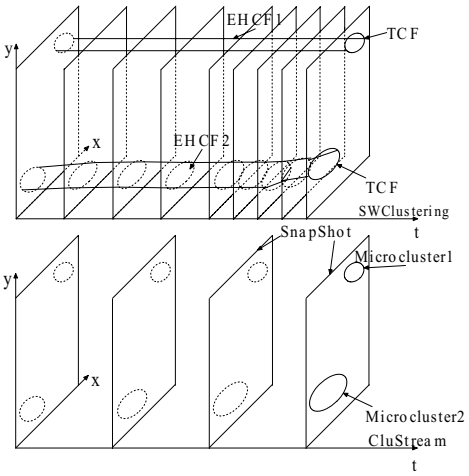


Fig. 6 EHCF versus pyramidal time frame and microcluster

3.4.3 Record-level analysis

Because EHCF synopsis adaptively creates a TCF for every newly arrived record, it supports record-level clustering analysis. This property is very important for time-critical applications with rapidly changing streams, such as financial data analysis and network intrusion detection. A simple extension of CluStream would require storing a snapshot whenever a new record arrives, which results in heavy I/O cost and memory consumption.

Figure 6 illustrates the aforementioned differences.

4 Sliding windows clustering

4.1 SWClustering algorithm

In this section, we describe the SWClustering algorithm on a data stream over a sliding window. SWClustering consists of two parts as shown in Algorithm 1. The first part maintains a group of EHCF synopses (Lines 1–18). The second part calculates the clustering result based on the collection of synopses (Lines 19–22).

Algorithm 1 *SWClustering(DS, ϵ, NC) for clustering an evolving data stream DS in a sliding window with a relative error parameter ϵ. At most NC EHCFs can be kept in memory.*

Procedure SWClustering(DS, ϵ, NC)
1: count:= 0;
2: **for**(each record x in DS)**do**
3: Get h, the nearest EHCF to record x;
4: **if** (x can be absorbed by h)
5: Insert x into h;
6: **else**

```
7:        if(count = NC)
8:          Merge the two nearest EHCFs;
9:          count:= count - 1;
10:       end if
11:       Generate a EHCF containing only TCF({x});
12:       count:= count + 1;
13:     end if
14:     Update h_e, the EHCF containing expired records;
15:     if (all TCFs of h_e are expired)
16:       Delete h_e;
17:       count:= count - 1;
18:     end if
19:     if(clustering request arrived)
20:       Calculate clusters upon all of EHCFs;
21:     end if
22:   end for
```

4.1.1 Maintenance of EHCFs

There are three parameters in the SWClustering Algorithm: DS, the data stream to be processed; $\epsilon (0 < \epsilon < 1)$, the relative error of windows; and NC, the maximal number of EHCFs kept in memory. The value of NC is determined by the amount of memory available. Our experiments show that even if NC is set to be a small value, the accuracy of the algorithm is still very high.

The distance between a record x and an EHCF synopsis h is defined as the distance between x and the center of h. For each x, the nearest EHCF synopsis from x is obtained first.

The algorithm then checks whether the record x can be absorbed by the nearest h or not. A simple approach is to compare the distance of x and h, $dist(x, h)$, with the radius R of h. The radius R of h is defined as the radius of the set of the records absorbed by h. It can be easily calculated by using the cluster feature vector of C [40]. We can define β ($\beta > 0$), a radius threshold, to determine whether to absorb the record x or not. When $dist(x, h) > \beta \cdot R$, x is viewed as far away from h; otherwise, x is absorbed by h. One problem left is how to determine the value of β. As well known, if the distance between the data points and the centroid follows Gaussian distribution, then $\beta = 2$ results in that more than 95% of the data points lie within the corresponding cluster boundary. Therefore, β is set to 2 in our experiments. For an EHCF with only one record, the radius is heuristically defined as the distance to the nearest EHCF.

If record x can be absorbed by h, it is merged into h. The details are discussed in Sect. 3.2. Lines 7–12 show the operations in case record x can not be absorbed by h. If the number of EHCF synopses reaches the maximum value NC, two nearest EHCFs must be merged to generate a new EHCF synopsis, and the *count* is decreased by 1. A new EHCF synopsis is then generated for record x, and the *count* is increased by 1.

Line 14 updates the EHCF containing expired records. Because this operation is invoked repeatedly whenever a new record arrives, at most one EHCF (denoted as h_e if existed) contains expired records at any time. Remember that an EHCF

synopsis is actually a collection of TCF synopses, each of which represents a set of records. The oldest TCF synopsis in h_e is removed when its time stamp does not belong to the most recent N time stamps. If the final TCF in h_e expires, h_e must be removed, and the *count* is decreased by 1.

4.1.2 Cluster generation

When a clustering request arrives, clusters are calculated from all EHCF synopses immediately, as shown on Lines 19–22. For each EHCF synopsis h_i, all the TCF synopses in h_i are summed up to generate a large TCF synopsis F_i, from which the number of records in h_i and the center of h_i can be obtained. Calculating the clusters from cluster feature vectors has been widely studied [1, 40]. The basic idea is to treat the EHCF h_i as a pseudopoint locating at the center of h_i with weight m_i, where m_i is the number of records contained in h_i. The k-means algorithm [28] can be employed to produce clusters of all pseudopoints.

4.1.3 Optimization

Algorithm 1 is a space-efficient method, but the processing of each new record is time consuming. Therefore, some optimization techniques are introduced to speed up the computation.

On Lines 3 and 20, the algorithm merges all TCF synopses in each EHCF h to get the center of h. According to Theorem 2, the number of TCF synopses can reach $O(\frac{1}{\epsilon}\log(\epsilon n))$. Thus, the cost of a merge operation is $O(\frac{1}{\epsilon}\log(\epsilon n))$.

It is better to maintain incrementally an additional TCF synopsis (denoted as F) for each EHCF. This will reduce the cost of the merge operations to $O(1)$. When a new TCF synopsis (denoted as F') is generated and inserted into an EHCF, F is updated by $F + F'$, as shown on Line 5. On Line 14, an EHCF must be updated when some records expire. F can also be incrementally maintained due to the following property.

Property 2 Assume that an EHCF h contains m TCF synopses, $F_1, F_2, \ldots, F_m$, and F denotes the TCF synopsis over all records in h. When F_1 expires, a new TCF synopsis F' over all active records in h can be incrementally generated.

In fact, the fields $\overrightarrow{CF2^x}$, $\overrightarrow{CF1^x}$, and n of F' are just the subtraction results of the corresponding fields in F and F_1, and field t of F' equals to the t of F.

On Line 14, the algorithm searches for an EHCF synopsis containing expired records. An index can be built on field t of these special synopses F_m, then the oldest field is checked to see whether it is expired or not.

4.2 Analysis of SWClustering

First, we discuss why *late merging* operation can efficiently reduce the memory consumption with guaranteed correctness.

Assume that H_{new} is the new EHCF resulting from the late merging of H_1 and H_2, and $V_{H_1} = m_1$, $V_{H_1} = m_2$, $V_{H_{\text{new}}} = n$, where n increases as new records

are absorbed by H_{new}. From Theorem 1, the total number of TCFs in H_1 is $(\frac{1}{\epsilon}+1)(\log(\epsilon(m_1+\frac{n}{2})+1)+1)$. There are $(\frac{1}{\epsilon}+1)(\log((\frac{\epsilon n}{2})+1)+1)$ *void-TCFs* in H_1, since H_1 has virtually absorbed $\frac{n}{2}$ records. It follows that the number of TCFs in H_1 is:

$$B_{H_1}=\left(\frac{1}{\epsilon}+1\right)\left(\log\left(\epsilon\left(m_1+\frac{n}{2}\right)+1\right)-\log\left(\frac{\epsilon n}{2}+1\right)\right). \tag{1}$$

Similarly, the number of TCFs in H_2 is $\left(\frac{1}{\epsilon}+1\right)\left(\log\left(\epsilon\left(m_2+\frac{n}{2}\right)+1\right)-\log\left(\frac{\epsilon n}{2}+1\right)\right)$, and the number of TCFs in H_{new} is $\left(\frac{1}{\epsilon}+1\right)(\log(\epsilon n+1)+1)$. So, the total number of TCFs in H_1, H_2, and H_{new} is: $B_{\text{com}}=\left(\frac{1}{\epsilon}+1\right)\left(\log\left(\epsilon\left(m_1+\frac{n}{2}\right)+1\right)+\log\left(\epsilon\left(m_2+\frac{n}{2}\right)+1\right)-2\log\left(\frac{\epsilon n}{2}+1\right)+\log\left(\epsilon n+1\right)+1\right)$. As $n\gg 1$, we have

$$B_{\text{com}}\approx\left(\frac{1}{\epsilon}+1\right)\left(\log\left(\epsilon\left(m_1+\frac{n}{2}\right)\right)+\log\left(\epsilon\left(m_2+\frac{n}{2}\right)\right)-\log\left(\frac{\epsilon n}{2}\right)+2\right). \tag{2}$$

Assuming that H_0 is another EHCF with $V_{H_0}=m_1+m_2+n$, according to Theorem 1, the number of TCFs in H_0 is:

$$B_{H_0}=\left(\frac{1}{\epsilon}+1\right)(\log(\epsilon(m_1+m_2+n))+1). \tag{3}$$

The difference between B_{com} and B_{H_0} is:

$$\begin{aligned}B_{\text{com}}-B_{H_0}&<\left(\frac{1}{\epsilon}+1\right)\left(\log\left(m_{\max}+\frac{n}{2}\right)-\log\left(\frac{n}{2}\right)+1\right)\\&=\left(\frac{1}{\epsilon}+1\right)\left(\log\left(1+\frac{2m_{\max}}{n}\right)+1\right).\end{aligned} \tag{4}$$

where $m_{\max}=\max(m_1,m_2)$. As n increases, the difference diminishes gradually. For example, when $n=2m_{\max}$, the difference is only $2(\frac{1}{\epsilon}+1)$. When the number of newly absorbed records exceeds $\frac{1}{\epsilon}2^{\lceil\log(m_1+m_2)\rceil}$, the two *merging-EHCF* can be merged into a new EHCF according to Theorem 3, and there is no difference between the *late merging* EHCFs and H_0.

There exists a reasonable upper bound of memory consumption for the SWClustering algorithm. Because the impact of *late merging* on memory diminishes gradually, it is ignored in the following analysis.

Theorem 5 *The SWClustering algorithm uses at most* $(\frac{1}{\epsilon}+1)(\log\left(\prod_{i=1}^{k}(\epsilon N_i+1)\right)+k)$ *TCFs, where* N_i *denotes the size of* H_i, $\sum_{i=1}^{k}N_i=N$, *and* k *is the number of EHCFs.*

Proof From Theorem 1, it is known that for each EHCF H_i the space needed is at most $(\frac{1}{\epsilon}+1)(\log(\epsilon N_i+1)+1)$. Thus, the memory requirement of the algorithm is at most $\sum_{i=1}^{k}(\frac{1}{\epsilon}+1)(\log(\epsilon N_i+1)+1)=(\frac{1}{\epsilon}+1)\sum_{i=1}^{k}(\log(\epsilon N_i+1)+1)=(\frac{1}{\epsilon}+1)(\log(\prod_{i=1}^{k}(\epsilon N_i+1))+k)$ □

According to Theorem 5, the memory requirement of SWClustering depends on ϵ, $N_i (1 \leq i \leq k)$, and k. The following lemma can be used to reduce the dependent variables.

Lemma 1 *If k natural numbers $N_1, N_2, \ldots, N_k$ satisfy $\sum_{i=1}^{k} N_i = N$, then $\prod_{i=1}^{k} N_i \leq \lceil \frac{N}{k} \rceil^k$.*

See Appendix C for the proof.

Theorem 6 *The SWClustering algorithm uses $O(\frac{k}{\epsilon} \log(\epsilon \lceil \frac{N}{k} \rceil))$ TCFs, where N denotes the window size, and k is current number of EHCFs.*

Proof From Lemma 1, $\prod_{i=1}^{k} N_i \leq \lceil \frac{N}{k} \rceil^k$. Thus, $O(\frac{1}{\epsilon} \log(\prod_{i=1}^{k} \epsilon N_i)) = O(\frac{1}{\epsilon} \log(\epsilon^k \lceil \frac{N}{k} \rceil^k)) = O(\frac{k}{\epsilon} \log(\epsilon \lceil \frac{N}{k} \rceil))$ Therefore, from Theorem 5, the algorithm uses $O(\frac{k}{\epsilon} \log(\epsilon \lceil \frac{N}{k} \rceil))$ TCFs. □

4.3 Evolutionary analysis

With the EHCFs maintained by SWClustering, we are able to track the evolving behavior of clusters. The evolutionary analysis of data streams has been a hot topic because of its practical applications. For example, people may concern "how is (are) current cluster(s) formed" or "what is the special time in the formation of the cluster(s)" while getting the clustering results. Fortunately, the proposed EHCF synopsis can provide sufficient information to answer such questions off-line.

To answer the first question, assuming $C_i (1 \leq i \leq k)$ is the user-specified cluster, there are m_i EHCFs associated with cluster C_i. In an EHCF, every TCF contains the number of arrived records during the time interval between itself and the next TCF. Based on these m_i EHCFs, a curve can be drawn to reveal the relation between the number of records in the cluster and the time interval from current time T_c to $T_c - N$ intuitively. Such curve can be analyzed further by domain experts to reveal other characteristics of the cluster.

To answer the second question, we divide the time axis into several equal segments, and the percentage of the number of arriving records (*arriving ratio*) in every segment is then computed. The time when the *arriving ratio* exceeds some threshold δ_{high} always corresponds to the ascending phase, while the time when the *arriving ratio* is below some threshold δ_{low} corresponds to the descending phase.

5 Experimental results

5.1 Experimental setting

All experiments are conducted on a 2.4 GHz PentiumIV PC with 512MB memory, running Microsoft Windows 2000 Professional. To demonstrate the accuracy and efficiency of the SWClustering algorithm, the well-known CluStream algorithm [1] is used as the comparing algorithm. Both algorithms are implemented in Microsoft Visual C++.

To evaluate the clustering quality, scalability, and sensitivity of the SWClustering algorithm, both real and synthetic data sets are used in our experiments.

5.1.1 Real life data sets

In order to demonstrate the effectiveness of SWClustering, the KDD-CUP'99 Network Intrusion Detection data set is used. It has been used in Aggarwal et al. [1] to evaluate the accuracy of CluStream with respect to STREAM [24]. The data set consists of raw TCP connection records from a local area network. Features collected for each connection include the duration of the connection, the number of bytes transmitted from source to destination, etc. Each record in the data set corresponds to either a normal connection or one of four attack types: denial of service, unauthorized access from a remote machine (e.g., guessing password), unauthorized access to root, and probing (e.g., port scanning). Most of the connections in this data set are normal, but occasionally there could be a burst of attacks at certain times. As in Aggarwal et al. [1] and Chalaghan et al. [10], all 34 out of the total 42 continuous attributes available are used for clustering with one outlier point being removed.

A relatively stable real-life data set KDD-CUP'98 Charitable Donation data set is also used. It has been used to evaluate several clustering algorithms [1, 21]. This data set contains 95,412 records about people who made charitable donations in response to direct mailing requests. Clustering can be used to group donors with similar donation behaviors. As in Aggarwal et al. [1] and Farnstrom et al. [21], total 56 out of 481 fields are used, and the data set is converted into a stream by taking the data input order as the order of streaming.

5.1.2 Synthetic data sets

Synthetic data sets are generated to test the scalability of SWClustering. The data sets have between 100K and 800K points each and vary in the number of natural clusters from 5 to 40. The dimensionality ranges from 10 to 80. The points in each synthetic data set follow a series of Gaussian distributions. As in Aggarwal et al. [1], the mean and variance of the current Gaussian distribution are changed for every 10K points during the synthetic data generation. The following notations are used to characterize the synthetic data sets: "B" indicates the number of data points in the data set, "C" and "D" indicate the number of natural clusters and the dimensionality of each point, respectively. For example, B100C20D30 means the data set contains 100K data points of 30-dimensions, belonging to 20 different clusters.

The sum of squared distance (SSQ) is widely adopted in stream clustering [1, 10, 24] to evaluate the accuracy, as defined below. For each point x_i in current window, the nearest centroid C_{x_i} is first obtained for it, then the distance between x_i and C_{x_i}, $d(x_i, C_{x_i})$, is computed. Finally, current SSQ is calculated as equal to the sum of $d^2(x_i, C_{x_i})$ for all the points in current window.

We also evaluate the clustering accuracy by purity defined as follows: $pur = \frac{\sum_{i=1}^{k} \frac{|C_i^d|}{|C_i|}}{k} \times 100\%$, where k denotes the number of clusters. $|C_i^d|$ denotes the number of points with the dominant class label in cluster i. $|C_i|$ denotes the number of points in cluster i.

In the experiments, SWClustering maintains the same number of EHCF structures as that of microclusters used by CluStream. The parameters for CluStream

are chosen to be the same as those adopted in Aggarwal et al. [1]. If the relative stamp of a microclusters in CluStream is not one of the most recent N stamps, then the microcluster is removed. Unless mentioned otherwise, the parameters for SWClustering are set to $\epsilon = 0.1$, window size $N = 10000$, and radius threshold $\beta = 2$.

5.2 Clustering quality evaluation

At first, the clustering quality of SWClustering is compared with that of CluStream for different window sizes at different time stamps. In order to make the results more accurate, these two algorithms are executed five times and their average SSQs are calculated. The results on the Network Intrusion data set show that SWClustering produces better clustering output than CluStream in the scenario of sliding windows.

Figure 7 gives the results when window size N is set to 10,000. It shows that SWClustering is usually better than CluStream by several orders of magnitude. For example, at time stamp 200,000, the average SSQ of SWClustering is about five orders of magnitude smaller than that of CluStream. The quality of clustering with the Charitable Donation data set is examined as well. It is assumed that CluStream should get fairly better results, because the data set is a relatively stable data stream. Figures 10 and 8 show that the result of SWClustering is still much better than that of CluStream for such a relatively stable stream.

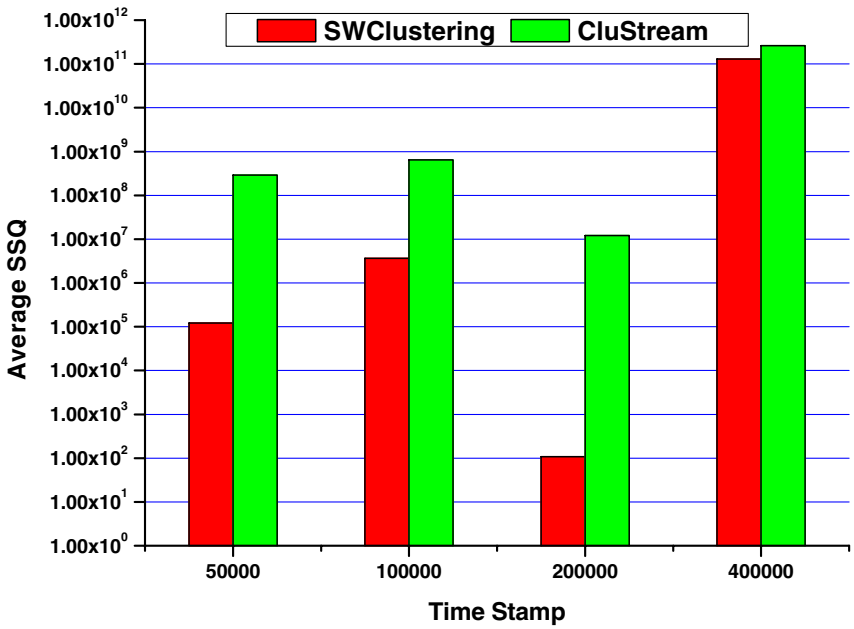


Fig. 7 Quality comparison (Network Intrusion data set, window size $N = 10,000$). SWClustering is almost always better than CluStream by several orders of magnitude for this rapidly evolving stream

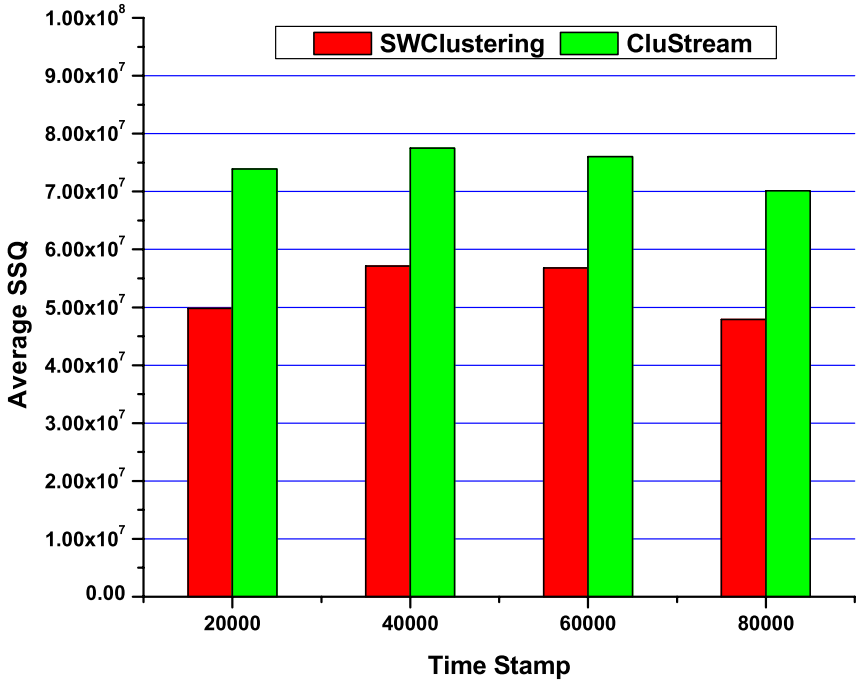


Fig. 8 Quality comparison (Charitable Donation data set, window size $N = 12,000$). SWClustering is much better than CluStream in a large window for this relatively stable stream

We also evaluate the clustering quality by purity. Figure 9 shows the results when $N = 1000$. We test the data set at selected time points when some attacks happen. For example, at time stamp 80,000 there are 752 "satan" attacks, 12 "warezmaster" attacks, 10 "guess-passwd" attacks, and 226 normal connections. It can be seen that SWClustering clearly outperforms CluStream and the purity of SWClustering is always above 90%. For example, at time stamp 80,000 the purity of SWClustering is about 91% and 19% higher than that of CluStream.

The quality clustering of SWClustering originates from EHCF's ability of precisely capturing the distribution of current records. The old records in EHCFs are promptly eliminated in online clustering. This expiring mechanism keeps the small radius of an EHCF when its cluster center drifts, which leads to finer granularity than CluStream, as illustrated in Fig. 2. In particular, the clusters in SWClustering are always formed by the most recent N records with error bound ϵN. However, in CluStream the time stamps of the records in a microcluster are summed up, a microcluster is deleted when its *relevance stamp* is below the threshold. Such strategy eliminates the microcluster as a whole, not only the individual records. Furthermore, the radius of a microcluster may continuously increase when the cluster center drifts, which may mess up different clusters and reduce the clustering quality. Therefore, SWClustering can separate different attacks into different clusters, while CluStream may merge different attacks into one attack (or normal connections).

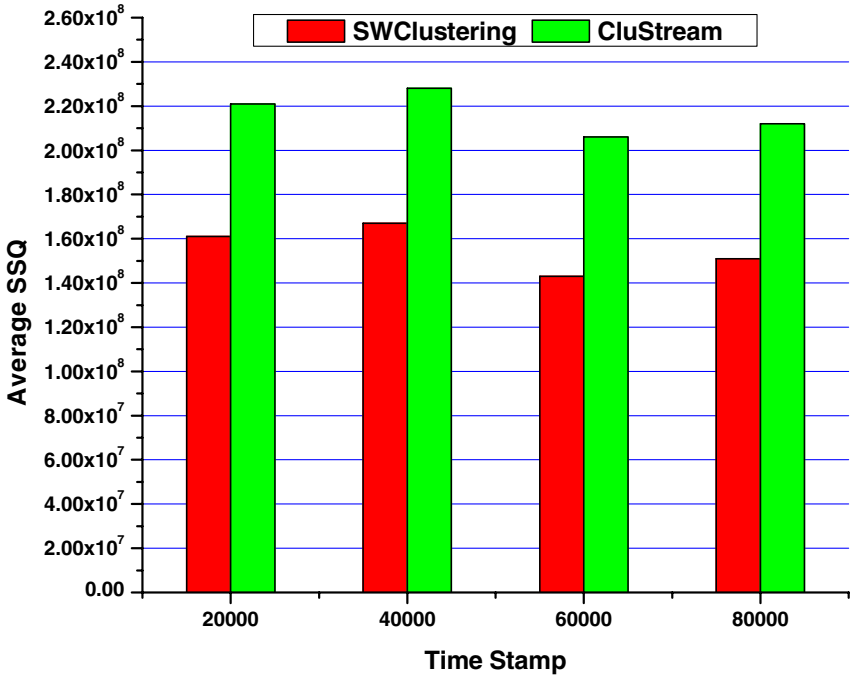


Fig. 9 Quality comparison (Network Intrusion data set, window size $N = 1,000$). The purity of SWClustering is always above 90% and clearly outperforms CluStream

Note that the quality difference between the two algorithms is not a constant. This is because: (1) the underlying distribution of the data stream is always changing, and the SSQs over sliding windows are different at different time stamps in the streams. (2) CluStream may occasionally capture the distribution of recent records, and get a relatively precise result, when a lot of old records in CluStream happen to be eliminated as a whole. For example, in Figure 7, the difference between CluStream and SWClustering is relatively small at time stamp 400,000. By checking the execution of CluStream, it is found that a lot of old microclusters are eliminated at that time.

5.3 Scalability results

The following experiments are designed to evaluate the scalability of SWClustering. The first part is used to evaluate the execution time. The second part is used to study the memory usage.

5.3.1 Execution time

As we know, the CluStream algorithm needs to periodically store the current snapshot of microclusters under the Pyramidal Time framework. In the implementation of the CluStream algorithm, the snapshots of microclusters are stored in memory to save the execution time.

Fig. 10 Quality comparison (Charitable Donation data set, window size $N = 4,000$). SWClustering is much better than CluStream in a small window for this relatively stable stream

We first compare the efficiency of maintaining EHCFs of SWClustering with that of the on-line component of CluStream on the real data sets. In CluStream, the frequency of storing snapshot affects the processing throughput and the precision. High storing frequency can improve the precision of CluStream at the cost of the throughput. In order to get the same precision as SWClustering, CluStream has to store a snapshot whenever a new record arrives. This inevitably leads to large overhead. To compare these two algorithms, we lower the precision requirement for CluStream 100 times, and the snapshot storing frequency is set to 1 snapshot per 100 records. Figures 11 and 12 show the execution time increases linearly when the data stream proceeds, and the curves are of different slopes. Notice that the curve with lower slope implies higher processing throughput. Hence, it can be concluded that SWClustering is much more efficient than CluStream.

Then, the execution time of SWClustering is evaluated on data streams with various dimensionality and different numbers of natural clusters. Synthetic data sets are used for these evaluations because any combination of the number of natural clusters and dimensions can be obtained during the generating of data sets.

The first series of data sets are generated by varying the number of natural clusters from 10 to 40, while fixing the size and dimensionality of the data streams. Figure 13 shows that the execution time of SWClustering is linear with respect to the number of natural clusters. For example, when the number of clusters increases from 10 to 40 for data set series B400D40, the running time increases from 30 to 54 s with nearly 8 s per 10 clusters.

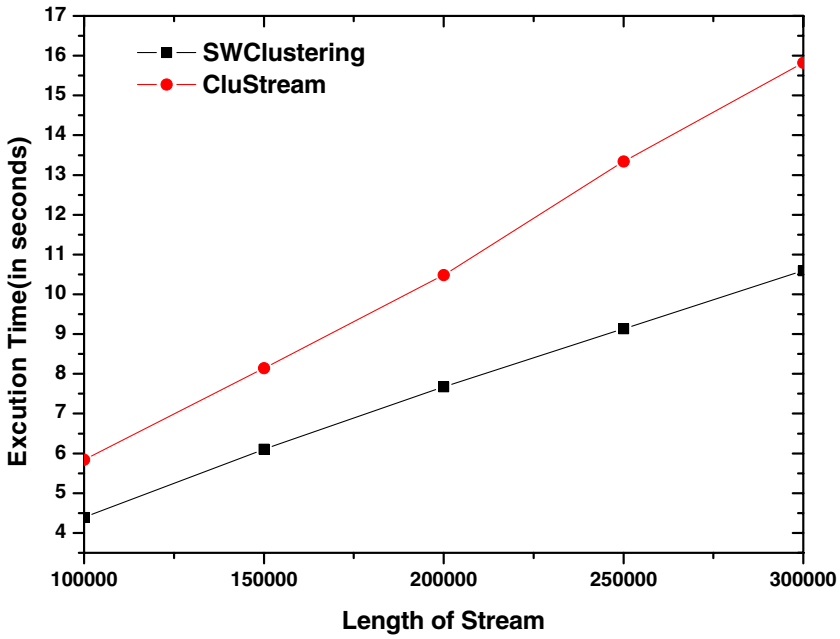


Fig. 11 Execution time versus length of stream (Network Intrusion data set). Notice that the precision requirement for CluStream is lowered 100 times, SWClustering is much faster than the CluStream in this rapidly evolving stream

The other three series of data sets are generated by varying the dimensionality from 10 to 80, while fixing the stream size and number of natural clusters. Figure 14 shows that the execution time increases linearly with respect to the dimensionality.

5.3.2 Memory usage

One common feature for the algorithms applied to data stream is their limited upper bounds for the memory usage. Since the memory usage may fluctuate in the progress of data streams, the maximum memory usage is used as the measurement. The entity used in CluStream is a microcluster, and in SWClustering it is a TCF. Since these two kinds of entities require similar memory space, the number of entities can be used to evaluate the memory usage.

For the comparison of memory usage, the window size is set in the range from 10,000 to 100,000 for the Network Intrusion data set, and from 10,000 to 60,000 for the Charitable Donation data set. As the snapshot storing frequency will affect the memory usage of CluStream, different frequencies (from 1 snapshot per 1 record to 1 snapshot per 100 records) are set to check the fluctuation of memory usage. As shown in Figures 15 and 16, the memory usage of SWClustering is consistently much lower than CluStream as the window size increases. In all cases, SWClustering outperforms CluStream by a factor of 2–5.

The error parameter ϵ is an important parameter in SWClustering and has a significant impact on memory usage. In the experiment with Network Intrusion

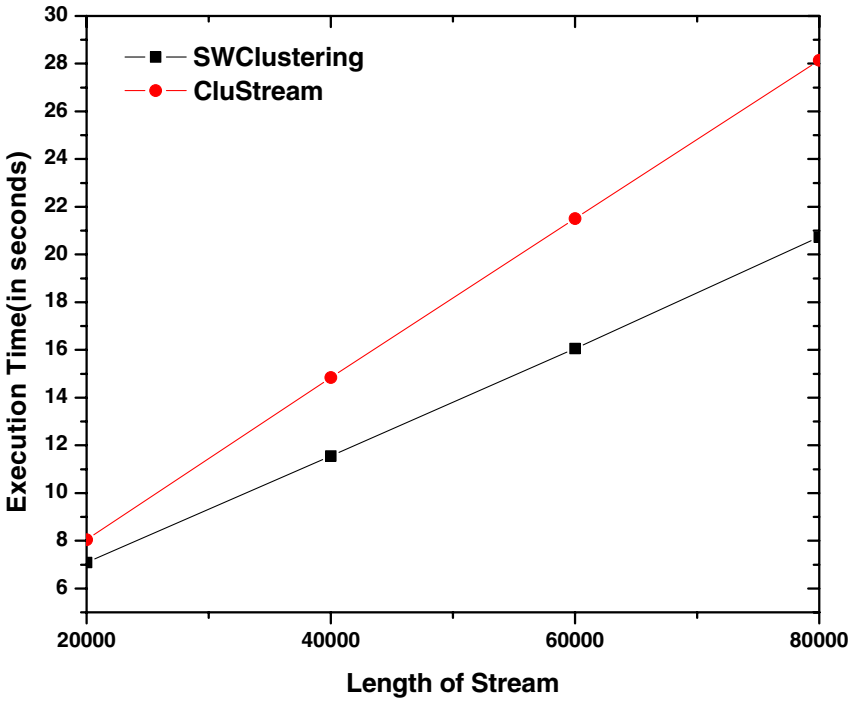


Fig. 12 Execution time versus length of stream (Charitable Donation data set). Notice that the precision requirement of CluStream is lowered 100 times, SWClustering is also much faster than CluStream in this relatively stable stream

data set, ϵ varies from 0.01 to 0.1 (and the number of snapshots in the same level of pyramidal time frame varies from 100 to 10 accordingly). The goal is to investigate the fluctuation of memory usage. The storing frequency of CluStream is fixed to be 1 snapshot per 100 records and window size is set to 100,000. Figure 17 illustrates the results. It can be seen that when the value of ϵ increases, the memory usage decreases significantly.

5.4 Sensitivity analysis

In Sect. 4.1, it is known that the number of EHCFs should be larger than the number of natural clusters in order to perform effective clustering. However, maintaining a large number of EHCFs leads to the increase of memory usage and slowdown of execution. We define *EHCF-ratio* as the number of EHCFs divided by the number of natural clusters.

The real-life data sets are used to evaluate the clustering quality by varying the *EHCF-ratio*. The current time stamp T_c is set to be 100,000 for the Network Intrusion data set and 50,000 for the Charitable Donation data set. Figure 18 shows that if *EHCF-ratio* $= 1$, i.e., the number of EHCFs is exactly the same as the natural clusters, the clustering quality is quite poor. When the *EHCF-ratio* increases, the average SSQ reduces. The average SSQ becomes stable when *EHCF-ratio* is

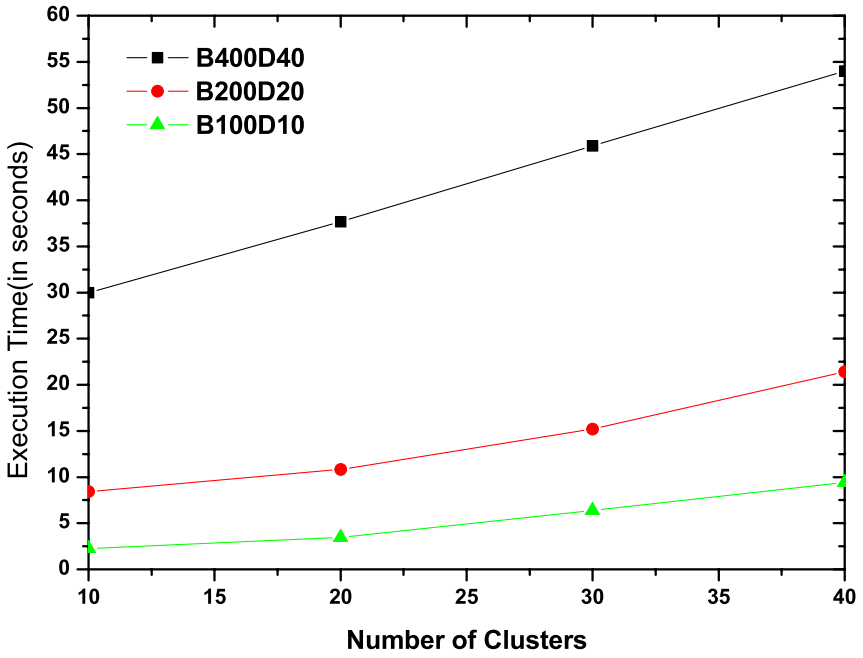


Fig. 13 Execution time versus number of natural clusters

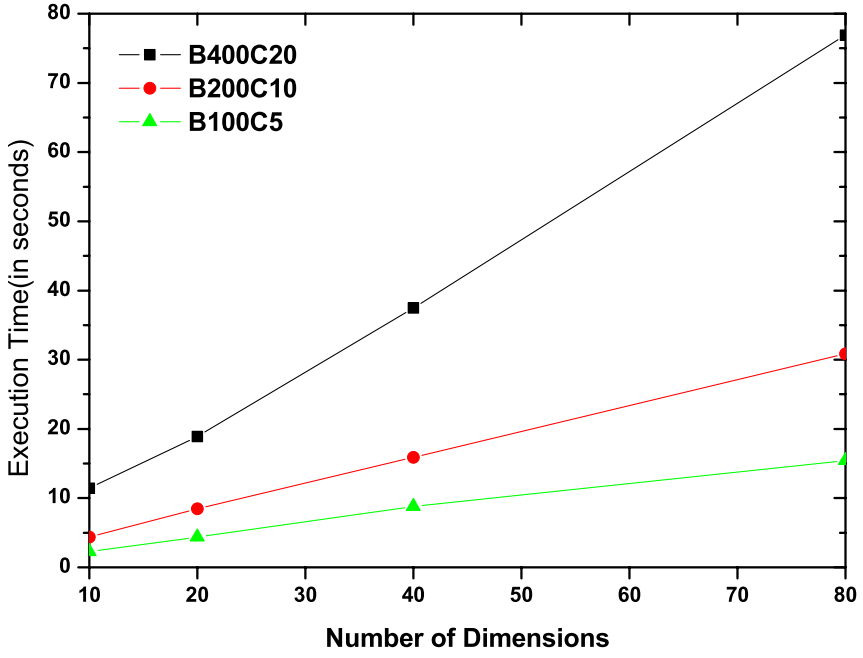


Fig. 14 Execution time versus data dimensionality

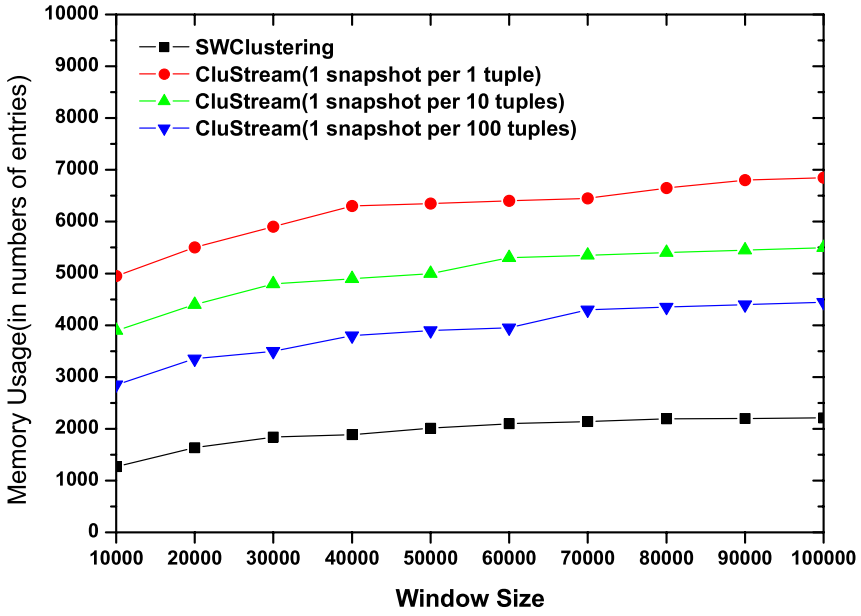


Fig. 15 Memory usage versus window size (Network Intrusion data set). SWClustering constantly costs much fewer than CluStream as the window size increasing

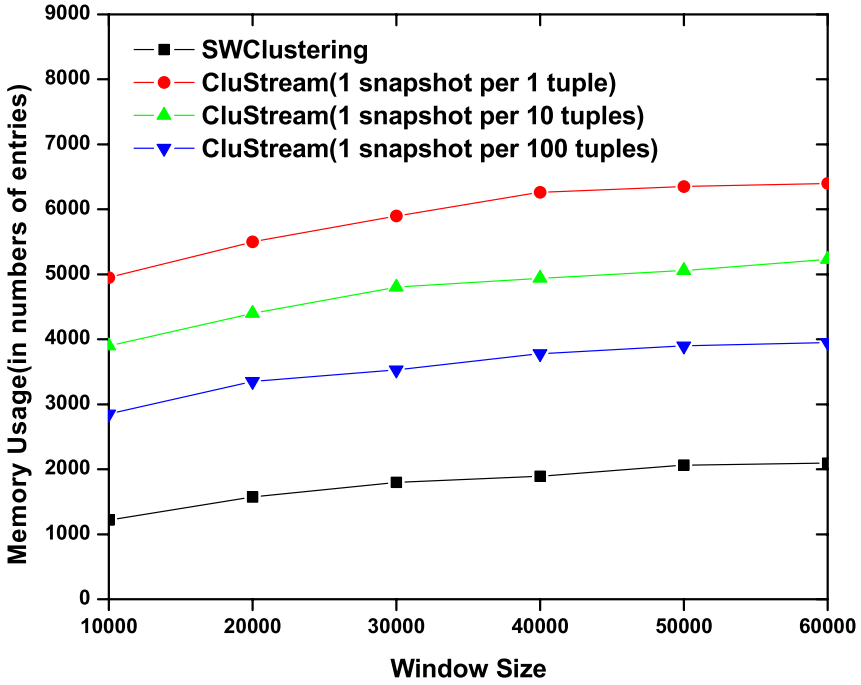


Fig. 16 Memory usage versus window size (Charitable Donation data set). SWClustering constantly costs much fewer than CluStream as the window size increasing

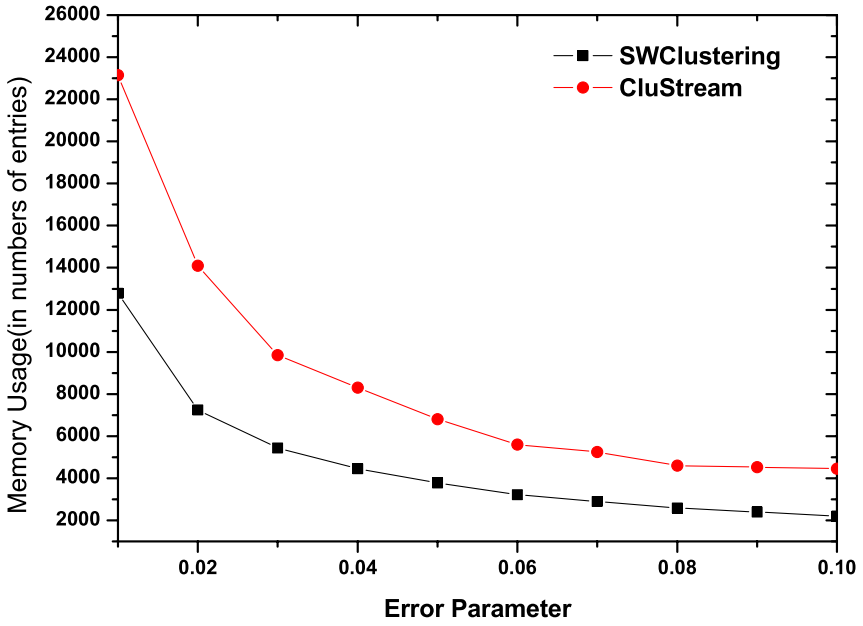


Fig. 17 Memory usage versus error parameter ϵ (Network Intrusion data set). When ϵ increases, the memory usage decreases significantly

about 10. This implies that to achieve quality clustering, *EHCF-ratio* needs not to be too large.

Another important parameter which may affect the clustering quality is the error parameter ϵ. ϵ is used to control the number of expired records in sliding windows. The SSQ is evaluated by varying the ϵ. For the Network Intrusion data set, the window size N is set to 20,000 with current time stamp $T_c = 100,000$, and for the Charitable Donation data set, N is set to 5000 with $T_c = 80,000$. Figure 19 shows that if $0.005 \leq \epsilon \leq 0.16$, the quality is relatively stable. However, when ϵ becomes a large value like 0.32, the quality of SWClustering deteriorates quickly. We note that the choice of $\epsilon = 0.32$ represents a case in which the clusters are determined by the data set with more than 30% outdated data.

5.5 Analysis of tracking clusters

Our SWClustering algorithm is able to track clusters in evolving data streams. The experiment reported here is to demonstrate that the clusters can be tracked at different time stamps based on the results of SWClustering.

We use the Network Intrusion data set as an example. Here, the window size is set to be 4000 with current time stamp T_c being 10,000. SWClustering finds 58 EHCFs. Three of them are displayed in Figure 20, each of which corresponds to one type of real-life network attacks. It can be seen that a `snmpget` attack ends at 8200, an `ipsweep` attack begins at 7700 and ends at 8100, while a `smurf` attack starts at 8400 and lasts till the end of the window. The evolving of these attacks can be fully recorded. Furthermore, different characteristics of the attacks are illustrated by the corresponding curves. For example, it can be seen that the `snmpget` attack goes up slowly, the `ipsweep` attack seems relatively scattered

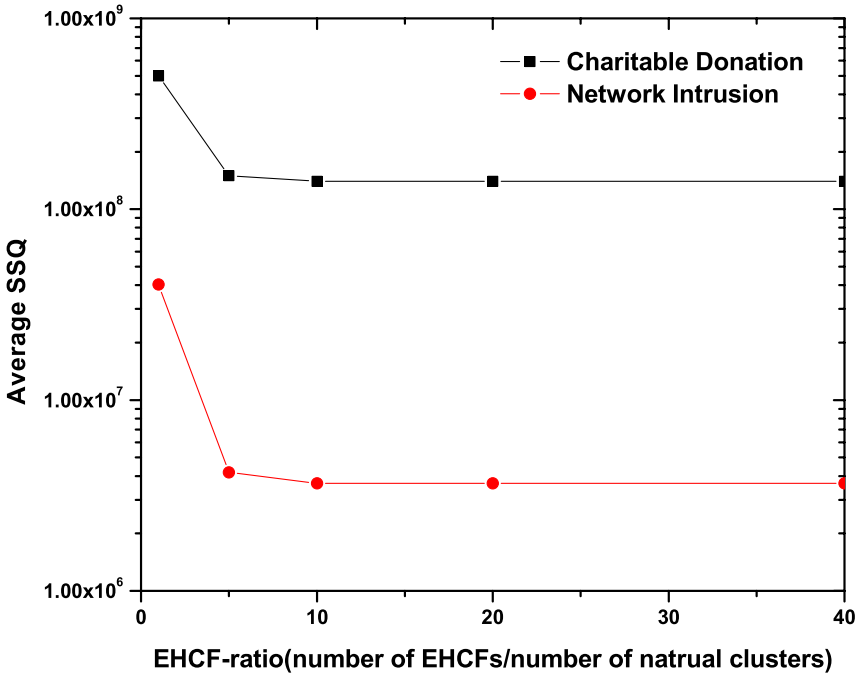


Fig. 18 Accuracy impact of *EHCF-ratios*. The average SSQ for each data set becomes stable when *EHCF-ratio* is about 10

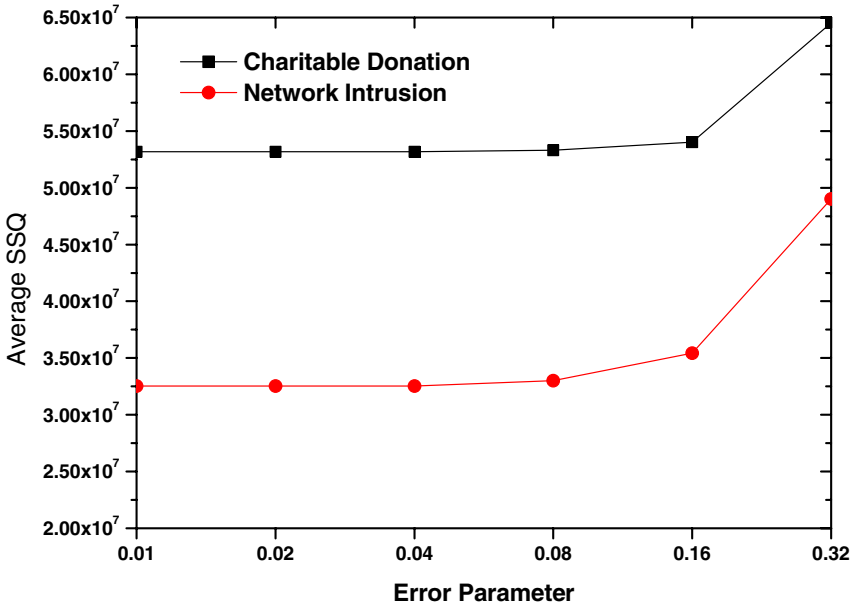


Fig. 19 Accuracy impact of error parameter ϵ

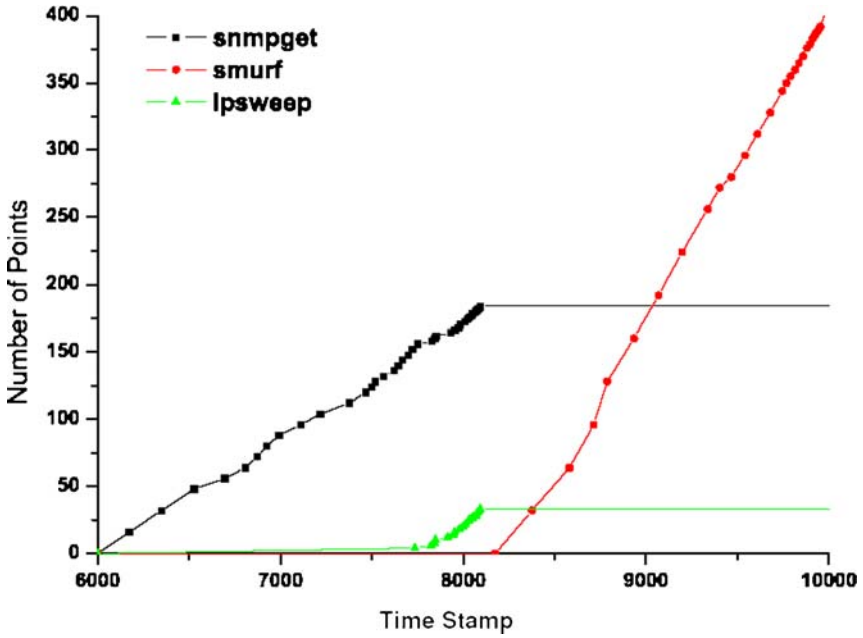


Fig. 20 Tracking clusters

and the `smurf` attack goes up sharply. Further analysis on the evolving clusters provides a deep understanding of the nature of different network attacks. This can be used to predict new attacks in the future.

6 Conclusions

An effective and efficient clustering approach for analyzing evolving data streams over sliding windows is proposed in this paper. An EHCF structure is introduced by combining Exponential Histogram with Cluster Feature to record the evolution of each cluster and to capture the distribution of recent records. Comparing to CluStream with batch updating and storing the whole snapshot, an EHCF is updated only when new records are collected. This approach has several advantages: (1) the quality of clustering is improved due to the fine granularity of EHCFs; (2) theoretical analysis shows that the memory consumption is limited and bounded, and the experiments show that its performance is better than CluStream in sliding window scenario; (3) it provides a novel framework for analyzing cluster evolution, which is very useful for online mining tasks over data streams. Moreover, several innovative ideas such as the late merging of EHCFs are proven to be useful for further improvement of the performance. Future work will focus on applying EHCF to other data mining tasks such as outlier detection, and providing more evolution analysis functionalities based on SWClustering.

Acknowledgements This work was done while the first author was visiting University of California at Berkeley as a Berkeley Scholar. The work is partially supported by National Natural

Science Foundation of China (Grant Nos. 60496325 and 60496327). The third author is supported by Shanghai Rising Star Program (04QMX1404).

Appendix

A Proof Theorem 3 of in Sect. 3.2

Theorem 3 *Assuming that an EHCF synopsis is constructed using n records, for any TCF F_i in the EHCF, if there are $(\lceil \frac{1}{\epsilon} \rceil (2^l - 1)$ records before F_i, F_i could be constructed upon its following 2^l records.*

Proof Since F_i can be any one TCF in an EHCF, there are two cases: 1) If F_i is not the last TCF, it keeps the structure of the EHCF. 2) If F_i is the last TCF, the absolute error of the number of expired records is 2^l. Because only the last TCF in an EHCF may contain expired records. And F_i contains 2^l records. The relative error is $\frac{2^l}{(\lceil \frac{1}{\epsilon} \rceil (2^l - 1)} \approx \epsilon$, which also keeps the structure of the EHCF. Therefore, F_i could be constructed upon its following 2^l records. □

B Proof Theorem 4 of in Sect. 3.3

Theorem 4 *Let H_1 and H_2 denote two non-overlapping EHCF synopses, containing m_1 and m_2 records, respectively. The merge operator upon them takes $O(\frac{1}{\epsilon} \log(\epsilon(m_1 + m_2)))$ time, and the new EHCF synopsis uses $O(\frac{1}{\epsilon} \log(\epsilon(m_1 + m_2)))$ TCF synopses.*

Proof Assume all m_1 records in H_1 arrive before all those m_2 records in H_2, the last TCF in H_1 is a m-level TCF, and there are $\lceil \frac{1}{\epsilon} \rceil = 2^l$ TCFs for each level from 0 to i in H_2, where $i \geq m$.

First, what should be proved is these $i * \lceil \frac{1}{\epsilon} \rceil$ TCFs can be merge into TCFs larger than 2^m. The TCFs in H_2 whose sizes are larger than 2^m meet the constraint naturally. Thus, the only consideration is on the TCFs with sizes from 2^{m-1} to 2^0 i.e., $\frac{1}{\epsilon}$ TCFs of $(m-1)$-level, $\frac{1}{\epsilon}$ TCFs of $(m-2)$-level, ..., and $\frac{1}{\epsilon}$ TCFs of 0-level. These TCFs are merged by level, and $\frac{1}{2\epsilon}$ TCFs of m-level , $\frac{1}{2^2\epsilon}$ TCFs of m-level, ..., and $\frac{1}{2^{m-1}\epsilon}$ TCFs of m-level are generated. As $\lceil \frac{1}{\epsilon} \rceil = 2^l$, only the TCFs equal to or after the $\frac{1}{2^l\epsilon}$ TCFs of m-level are whole m-level TCFs. So all the TCFs before the $\frac{1}{2^l\epsilon}$ m-level TCFs are merged into $(\frac{1}{2^{l+1}\epsilon} + \frac{1}{2^{l+2}\epsilon} + \cdots + \frac{1}{2^{m-1}\epsilon}) = (\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^{m-1-l}}) \approx 1$ m-level TCF. Second, these m-level TCFs are inserted into H_2 and keep at most $\lceil \frac{1}{\epsilon} \rceil + 1$, i-level TCFs for each $i \geq m$. At last, H_2 could add to H_1 directly after transformation. H_{new} holds the ϵ error bound. In the merge process, only one scan over H_1 and H_2 is needed, while the total number of TCFs in H_1 and H_2 is $O(\frac{1}{\epsilon} \log(\epsilon(m_1 + m_2)))$. So, the time complexity is $O(\frac{1}{\epsilon} \log(\epsilon(m_1 + m_2)))$. As the new EHCF contains $m_1 + m_2$ records, the number of TCFs in H_{new} is $O(\frac{1}{\epsilon} \log(\epsilon(m_1 + m_2)))$ according to Theorem 2. □

C Proof Lemma 1 of in Sect. 4.2

Lemma 1 *If k natural numbers $N_1, N_2, \ldots N_k$ satisfy $\sum_{i=1}^{k} N_i = N$, then $\prod_{i=1}^{k} N_i \leq \lceil \frac{N}{k} \rceil^k$.*

Proof Let $N = km + r$, $0 \leq r < k$, k and m be natural numbers. If k nature numbers N_1, N_2, ..., N_k satisfy $\sum_{i=1}^{k} N_i = N$, then $\prod_{i=1}^{k} N_i \leq m^{k-r}(m+1)^r$ holds, which could be found in a number of theory textbooks. Depending on $r = 0$ or not, there are two cases: 1) If $r = 0$, then $m = \frac{N}{k}$. So, $m^{k-r}(m+1)^r = (\frac{N}{k})^k \leq \lceil \frac{N}{k} \rceil^k$. 2) If $r \neq 0$, then $m = \frac{N-r}{k} = \lceil \frac{N}{k} \rceil - 1$. So, $m^{k-r}(m+1)^r = (\lceil \frac{N}{k} \rceil - 1)^{k-r}(\lceil \frac{N}{k} \rceil)^r < \lceil \frac{N}{k} \rceil^k$. In these two cases, the inequality $m^{k-r}(m+1)^r \leq \lceil \frac{N}{k} \rceil^k$ holds. Therefore, $\prod_{i=1}^{k} N_i \leq \lceil \frac{N}{k} \rceil^k$ holds.

References

1. Aggarwal CC, Han J, Wang J, Yu PS (2003) A framework for clustering evolving data streams. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A (eds) Proceedings of 29th international conference on very large data bases, Berlin, Germany, pp 81–92
2. Aggarwal CC, Han J, Wang J, Yu PS (2004) A framework for projected clustering of high dimensional data streams. In: Nascimento MA, Özsu MT, Kossmann D, Miller RJ, Blakeley JA, Schiefer KB (eds) Proceedings of the 30th international conference on very large data bases, Toronto, Canada, pp 852–863
3. Aggarwal CC, Han J, Wang J, Yu PS (2005) On high dimensional projected clustering of data streams. Data Min Knowl Discovery 10(3):251–273
4. Aggarwal CC, Yu P (2006) A framework for clustering massive text and categorical data streams. In: Proceedings of the ACM SIAM conference on data mining, 2006 (Text and categorical clustering of high dimensional data streams) pp 479–483
5. Arasu A, Manku GS (2004) Approximate counts and quantiles over sliding windows. In: Deutsch A (ed) Proceedings of the 23th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Paris, France, pp 286–296
6. Babcock B, Babu S, Datar M, Motwani R, Widom J (2002) Models and issues in data stream systems. In: Popa L (ed) Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, Madison, WI, pp 1–16
7. Babcock B, Datar M, Motwani R, Callaghan LO' (2003) Maintaining variance and k-medians over data stream windows. In: Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, San Diego, CA, pp 234–243
8. Beringer J, Hullermeier E (2006) Online clustering of parallel data streams. Data Knowl Eng 58(2):180–204
9. Cao F, Ester M, Qian W, Zhou A (2006) Density-based clustering over an evolving data stream with noise. In: Proceedings of the 2006 SIAM conference on data mining (SDM), Bethesda, MD, pp 328–339
10. Chalaghan LO, Mishra N, Meyerson A, Guha S (2002) Streaming data algorithms for high-quality clustering. In: Proceedings of the 18th international conference on data engineering, San Jose, CA, pp 685–694
11. Charikar M, Callaghan LO', Panigrahy R (2003) Better streaming algorithms for clustering problems. In: Proceedings of the 35th annual ACM symposium on theory of computing, San Diego, CA, pp 30–39
12. Chi Y, Wang H, Yu PS, Muntz RR (2004) Moment: maintaining closed frequent itemsets over a stream sliding window. In: Proceedings of the 2004 IEEE international conference on data mining, ICDM 2004, Brighton, UK, pp 59–66
13. Chi Y, Wang H, Yu PS, Muntz RR (2006) Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. Knowl Inf Syst 10(3):265–294
14. Cormode G, Muthukrishnan S (2003) What's hot and what's not: tracking most frequent items dynamically. In: Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, San Diego, CA, pp 296–306
15. Dai B, Huang J, Yeh M, Chen M (2004) Clustering on demand for multiple data streams. In: Proceedings of the 4th IEEE international conference on data mining (ICDM 2004), Brighton, UK, pp 367–370
16. Dai BR, Huang JW, Yeh MY, Chen MS (2006) Adaptive clustering for multiple evolving streams. IEEE Trans Knowl Data Eng 18(9):1166–1180
17. Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. In: Proceedings of the 13th annual ACM-SIAM symposium on discrete algorithms, San Francisco, CA, pp 635–644
18. Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining, Boston, MA, pp 71–80
19. Domingos P, Hulton G (2001) A general method for scaling up machine learning algorithms and its application to clustering. In: Brodley CE, Danyluk AP (eds) Proceedings of the 18th international conference on machine learning (ICML 2001), Williams College, Williamstown, MA, pp 106–113

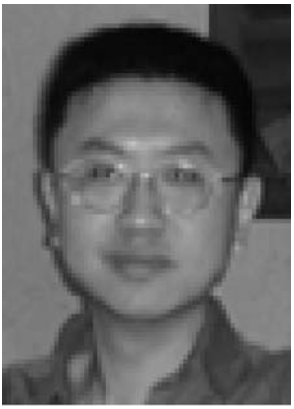
20. Domingos P, Hulton G (2001) Catching up with the data: research issues in mining data streams. In: SIGMOD workshop on research issues in data mining and knowledge discovery, Santa Barbara, CA
21. Farnstrom F, Lewis J, Elkan C (2000) Scalability for clustering algorithms revisited. SIGKDD Explor 2(1):51–57
22. Guha S, Koudas N (2001) Data-streams and histograms. In: Proceedings of the thirty-third annual ACM symposium on theory of computing, STOC 2001, New York, pp 471–475
23. Guha S, Mishra N, Motwani R, Callaghan LO' (2000) Clustering data stream. In: Proceedings of the 41st annual symposium on foundations of computer science, FOCS 2000, Redondo Beach, CA, pp 359–366
24. Guha S, Meyerson A, Mishra N, Motwani R, Callaghan LO' (2003) Clustering data streams: theory and practice. IEEE Trans Knowl Data Eng (TKDE) 3(15):515–528
25. He Z, Xu X, Deng S, Huang JZ (2004) Clustering categorical data streams. J Comput Methods Sci Eng (JCMSE) URL http://arxiv.org/ftp/cs/papers/0412/0412058.pdf
26. He Z, Xu X, Dend S (2002) Squeezer: an efficient algorithm for clustering categorical data. J Comput Sci Technol 17(5):611–624
27. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the 7th ACM SIGKDD international conference on knowledge discovery and data mining, San Francisco, CA, pp 97–106
28. Jain A, Dubes R (1998) Algorithms for clustering data. Prentice-Hall, Englewood Cliffs, NJ
29. Jin C, Qian W, Sha C, Yu J, Zhou A (2003) Dynamically maintaining frequent items over a data stream. In: Proceedings of the 12nd ACM CIKM international conference on information and knowledge management, New Orleans, LA, pp 287–294
30. Keogh E, Lin J (2005) Clustering of time-series subsequences is meaningless: implications for previous and future research. Knowl Inf Syst 8(2):154–177
31. Keogh E, Lin J, Truppel W (2003) Clustering of time series subsequences is meaningless: implications for past and future research. In: Proceedings of the 3rd IEEE international conference on data mining, Melbourne, FL
32. Lin M, Lee S (2005) Efficient mining of sequential patterns with time constraints by delimited pattern growth. Knowl Inf Syst 7(4):499–514
33. Manku GS, Motwani R (2002) Approximate frequency counts over data streams. In: Proceedings of 28th international conference on very large data bases, Hong Kong, China, pp 346–357
34. Muthukrishnan S, Shah R, Vitter JS (2004) Mining deviants in time series data streams. In: Proceedings of the 16th international conference on scientific and statistical database management (SSDBM 2004), Santorini Island, Greece, pp 41–50
35. Nasraoui O, Cardona C, Rojas C, Gonzalez F (2003) TECNO-STREAMS: tracking evolving clusters in noisy data streams with a scalable immune system learning model. In: Proceedings of the 3rd IEEE international conference on data mining (ICDM 2003), Melbourne, FL, pp 235–242
36. Nasraoui O, Cardona C, Rojas C, Gonzalez F (2003) Mining evolving user profiles in noisy web clickstream data with a scalable immune system clustering algorithm. In: Proceeding of WebKDD 2003CKDD workshop on web mining as a premise to effective and intelligent Web applications, Washington DC
37. Ordonez C (2003) Clustering binary data streams with k-means. In: Zaki MJ, Aggarwal CC (eds) Proceedings of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, DMKD 2003, San Diego, CA pp 12–19
38. Rodrigues P, Gama J, Pedroso JP (2004) Hierarchical time-series clustering for data streams. In: Jesus Aguilar-Ruiz, Joäo Gama (eds) Proceedings of the first international workshop on knowledge discovery in data streams, Pisa, Italy, pp 22–31
39. Yang J (2003) Dynamic clustering of evolving streams with a single pass. In: Dayal U, Ramamritham K, Vijayaraman TM (eds) Proceedings of the 19th international conference on data engineering, Bangalore, India, pp 695–697
40. Zhang T, Ramakrishnan R, Livny M (1996) BIRCH: an efficient data clustering method for very large databases. In: Jagadish HV, Mumick IS (eds) Proceedings of the 15th ACM SIGMOD international conference on management of data, Montreal, Quebec, Canada, pp 103–114

41. Zhou A, Cai Z, Wei L, Qian W (2003) M-kernel merging: towards density estimation over data streams. In: Proceeeding of 8th international conference on database systems for advanced applications (DASFAA'03), Kyoto, Japan, pp 285–292
42. Zhu Y, Shasha D (2002) StatStream: Statistical monitoring of thousands of data streams in real time. In: Proceeding of very large data bases conference, Hong Kong, China, pp 358–369

Author Biographies

Aoying Zhou is currently a Professor in Computer Science at Fudan University, Shanghai, P.R. China. He won his Bachelor and Master degrees in Computer Science from Sichuan University in Chengdu, Sichuan, P.R. China in 1985 and 1988, respectively, and Ph.D. degree from Fudan University in 1993. He served as the member or chair of program committee for many international conferences such as WWW, SIGMOD, VLDB, EDBT, ICDCS, ER, DASFAA, PAKDD, WAIM, and etc. His papers have been published in ACM SIGMOD, VLDB, ICDE, and several other international journals. His research interests include Data mining and knowledge discovery, XML data management, Web mining and searching, data stream analysis and processing, peer-to-peer computing.

Feng Cao is currently an R&D engineer in IBM China Research Laboratories. He received a B.E. degree from Xi'an Jiao Tong University, Xi'an, P.R. China, in 2000 and an M.E. degree from Huazhong University of Science and Technology, Wuhan, P.R. China, in 2003. From October 2004 to March 2005, he worked in Fudan-NUS Competency Center for Peer-to-Peer Computing, Singapore. In 2006, he received his Ph.D. degree from Fudan University, Shanghai, P.R. China. His current research interests include data mining and data stream.



Weining Qian is currently an Assistant Professor in computer science at Fudan University, Shanghai, P.R. China. He received his M.S. and Ph.D. degree in computer science from Fudan University in 2001 and 2004, respectively. He is supported by Shanghai Rising-Star Program under Grant No. 04QMX1404 and National Natural Science Foundation of China (NSFC) under Grant No. 60673134. He served as the program committee member of several international conferences, including DASFAA 2006, 2007 and 2008, APWeb/WAIM 2007, INFOSCALE 2007, and ECDM 2007. His papers have been published in ICDE, SIAM DM, and CIKM. His research interests include data stream query processing and mining, and large-scale distributed computing for database applications.

Cheqing Jin is currently an Assistant Professor in Computer Science at East China University of Science and Technology. He received his Bachelor and Master degrees in Computer Science from Zhejiang University in Hangzhou, P.R. China in 1999 and 2002, respectively, and the Ph.D. degree from Fudan University, Shanghai, P.R. China. He worked as a Research Assistant at E-business Technology Institute, the Hong Kong University from December 2003 to May 2004. His current research interests include data mining and data stream.