# Stream Data Clustering Based on Grid Density and Attraction

Li Tu
Nanjing University of Aeronautics and Astronautics
and
Yixin Chen
Washington University in St. Louis

Clustering real-time stream data is an important and challenging problem. Existing algorithms such as CluStream are based on the $k$-means algorithm. These clustering algorithms have difficulties to find clusters of arbitrary shapes and to handle outliers. Further, they require the knowledge of $k$ and user-specified time window. To address these issues, this paper proposes **D-Stream**, a framework for clustering stream data using a density-based approach.

Our algorithm uses an online component that maps each input data record into a grid and an offline component that computes the grid density and clusters the grids based on the density. The algorithm adopts a density decaying technique to capture the dynamic changes of a data stream and a attraction-based mechanism to accurately generate cluster boundaries.

Exploiting the intricate relationships between the decay factor, attraction, data density and cluster structure, our algorithm can efficiently and effectively generate and adjust the clusters in real time. Further, a theoretically sound technique is developed to detect and remove sporadic grids mapped by outliers in order to dramatically improve the space and time efficiency of the system. The technique makes high-speed data stream clustering feasible without degrading the clustering quality. The experimental results show that our algorithm has superior quality and efficiency, can find clusters of arbitrary shapes, and can accurately recognize the evolving behaviors of real-time data streams.

Categories and Subject Descriptors: D.2.8 [**Database Management**]: Database Applications—*data mining*

General Terms: Algorithms, management

Additional Key Words and Phrases: Stream data, data mining, clustering, density-based algorithms

## 1. INTRODUCTION

Clustering high-dimensional stream data in real time is a difficult and important problem with ample applications such as network intrusion detection, weather monitoring, emergency response systems, stock trading, electronic business, telecommunication, plan-

L. Tu, Institute of Information Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.
Y. Chen, One Brookings Drive, Campus Box 1045, Washington University, St. Louis, MO 63130. chen@cse.wustl.edu.

etary remote sensing, and web site analysis. In these applications, large volume of multi-dimensional data streams arrive at the data collection center in real time. Examples such as the transactions in a supermarket and the phone records of a mobile phone company illustrate that, the raw data typically have massive volume and can only be scanned once following the temporal order [9; 12]. Recently, there has been active research on how to store, query and analyze data streams.

Clustering is a key data mining task. In this paper, we consider clustering multi-dimensional data in the form of a stream, i.e. a sequence of data records stamped and ordered by time. Stream data clustering analysis causes unprecedented difficulty for traditional clustering algorithms. There are several key challenges. First, the data can only be examined in one pass. Second, viewing a data stream as a long vector of data is not adequate in many applications. In fact, in many applications of data stream clustering, users are more interested in the evolving behaviors of clusters.

Recently, there have been different views and approaches for stream data clustering. Earlier clustering algorithms for data stream use a single-phase model which treats data stream clustering as a continuous version of static data clustering [13]. These algorithms uses divide and conquer schemes that partition data streams into segments and discover clusters in data streams based on a $k$-means algorithm in finite space [14; 17]. A limitation of such schemes is that they put equal weights to outdated and recent data and cannot capture the evolving characteristics of stream data. Moving-window techniques are proposed to partially address this problem [2; 4].

Another recent data stream clustering paradigm proposed by Aggarwal et al. uses a **two-phase scheme** [1] which consists of an online component that processes raw data stream and produces summary statistics and an offline component that uses the summary data to generate clusters. Strategies for dividing the time horizon and manage the statistics are studied. The design leads to the CluStream system [1]. Many recent data stream clustering algorithms are based on CluStream's two-phase framework. Wang et al. proposed an improved offline component using an incomplete partitioning strategy [22]. Extensions of this work include clustering multiple data streams [8], parallel data streams [5], and distributed data steams [3], and applications of data stream mining [16; 21; 18].

A number of limitations of CluStream and other related work lie in the $k$-means algorithm used in their offline component. First, a fundamental drawback of $k$-means is that it aims at identifying spherical clusters but is incapable of revealing clusters of *arbitrary* shapes. However, nonconvex and interwoven clusters are seen in many applications. Second, the $k$-means algorithm is unable to detect noise and outliers. Third, the $k$-means algorithm requires multiple scans of the data, making it not directly applicable to large-volume data streams. For this reason, the CluStream architecture uses an online processing which compresses raw data stream into micro-clusters, which are used as the basic elements in the offline phase.

Density-based clustering has been long proposed as another major clustering algorithm [19; 20]. We find the density-based method a natural and attractive basic clustering algorithm for data streams, because it can find arbitrarily shaped clusters, it can handle noises and is an one-scan algorithm that needs to examine the raw data only once. Further, it does not demand a priori knowledge of the number of clusters $k$ as the $k$-means algorithm does.

In this paper, we propose **D-Stream**, a density-based clustering framework for data streams. It is not a simple switch-over to use density-based instead of $k$-means algorithms

for data streams. There are three main technical challenges.

First, it is not desirable to treat the data stream as a long sequence of static data since we are interested in the evolving temporal feature of the data stream. To capture the dynamic changing of clusters, we propose an innovative scheme that associates a *decay factor* to the density of each data point. Unlike the CluStream architecture which asks the users to input the target time duration for clustering, the decay factor provides a novel mechanism for the system to dynamically and automatically form the clusters by placing more weights on the most recent data without totally discarding the historical information. In addition, D-Stream does not require the user to specify the number of clusters $k$. Thus, D-Stream is particularly suitable for users with little domain knowledge on the application data.

Second, due to the large volume of stream data, it is impossible to retain the density information for every data record. Therefore, we propose to partition the data space into discretized fine grids and map new data records into the corresponding grid. Thus, we do not need to retain the raw data and only need to operate on the grids. However, for high-dimensional data, the number of grids can be large. Therefore, how to handle high dimensionality and improve scalability is a critical issue. Fortunately, in practice, most grids are empty or only contain few records. A memory-efficient technique for managing such a sparse grid space is developed in D-Stream.

Finally, in traditional density-based clustering algorithms, when each data item maps to a grid, the positional information of the data in that grid is lost, leading to possibly poor clustering results. In this paper, we propose a novel concept, the *attraction* of grids, that characterizes the positional information of the data in each grids. By considering both density and attraction of grids, we generate better clustering results.

By addressing the above issues, we propose D-Stream, a density-based stream data clustering framework. We study in depth the relationship between time horizon, decay factor, attraction, and data density to ensure the generation of high quality clusters, and develop novel strategies for controlling the decay factor and detecting outliers. D-Stream automatically and dynamically adjusts the clusters without requiring user specification of target time horizon and number of clusters. The experimental results show that D-Stream can find clusters of arbitrary shapes. Comparing to CluStream, D-Stream shows better in terms of both clustering quality and efficiency on several datasets. D-Stream also exhibits good scalability for large-scale stream data.

In this paper we use decay factor to address the concept drift in stream data. The decay factor method is a popular method for stream data mining (for example, [24; 15; 11]). There are several other important methods. The stochastic modelling method [23] is very useful for ensemble-based classification, but it is not clear how to apply the idea to clustering. The pyramidal time frame method used by CluStream and some other works [6; 10] is another interesting method. Unlike our decay factor method which considers all the historical data, the pyramidal time frame method uses a more sparse sampling for older data.

The rest of the paper is organized as follows. In Section 2, we overview the overall architecture of D-Stream. In Section 3, we present the concept and theory on the proposed density grid and attraction analysis. In Section 4, we give the algorithmic details and theoretical analysis for D-Stream. We conduct experimental study of D-Stream in Section 5 and conclude the paper in Section 6.

```
1.  procedure D-Stream
2.      t_c = 0;
3.      initialize an empty red-black tree for grid_list;
4.      while data stream is active do
5.          read record x = (x_1, x_2, ..., x_d);
6.          determine the density grid g that contains x;
7.          if (g not in grid_list) insert g to grid_list;
8.          update the characteristic vector of g;
9.          if t_c == gap then
10.             call initial_clustering(grid_list);
11.         end if
12.         if t_c mod gap == 0 then
13.             detect and remove sporadic grids from grid_list;
14.             call adjust_clustering(grid_list);
15.         end if
16.         t_c = t_c + 1;
17.     end while
18. end_procedure
```

Fig. 1. The overall process of D-Stream.

## 2. OVERALL ALGORITHM OF D-STREAM

We overview the overall architecture of D-Stream, which assumes a discrete time step model, where the time stamp is labelled by integers $0, 1, 2, 3, \ldots$. Like CluStream [1], D-Stream has an online component and an offline component. The overall algorithm is outlined in Figure 1.

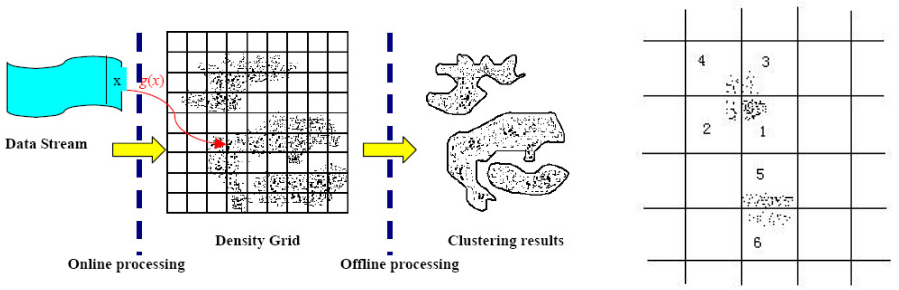
For a data stream, at each time step, the online component of D-Stream continuously reads a new data record, places the multi-dimensional data into a corresponding discretized *density grid* in the multi-dimensional space, and updates the *characteristic vector* of the density grid (Lines 5-8 of Figure 1). The density grid and characteristic vector are to be described in detail in Section 3. The offline component dynamically adjusts the clusters every $gap$ time steps, where $gap$ is an integer parameter. After the first $gap$, the algorithm generates the initial cluster (Lines 9-11). Then, the algorithm periodically removes sporadic grids and regulates the clusters (Lines 12-15).

We note that here we assume only one data record is processed at one time and we update the the characteristic vector before the next record comes. This is a standard model that has also been used in other streaming mining methods such as CluStream [1]. Of course, it is a simplified model that facilitates our analysis. In practice, we can always introduce a buffer and make the online component read from the buffer one data each time.

## 3. DENSITY GRIDS

In this section, we introduce the concept of density grid and other associated definitions, which form the basis for the D-Stream algorithm.

Since it is impossible to retain the raw data, D-Stream partitions the multi-dimensional data space into many density grids and forms clusters of these grids. This concept is schematically illustrated in Figure 2a.




a) Use of density grids. b) Distribution of data records

Fig. 2. Illustration of density grids and data distribution

### 3.1 Attraction of grids

In this paper, we assume that the input data has $d$ dimensions, and each input data record is defined within the space

$$S = S_1 \times S_2 \times \cdots \times S_d, \tag{1}$$

where $S_i$ is the definition space for the $i^{th}$ dimension.

In D-Stream, we partition the $d-$dimensional space $S$ into **density grids**. Suppose for each dimension, its space $S_i$, $i = 1, \ldots, d$ is evenly divided into $p_i$ partitions as

$$S_i = S_{i,1} \bigcup S_{i,2} \bigcup \cdots \bigcup S_{i,p_i}, \tag{2}$$

then the data space $S$ is partitioned into $N = \prod_{i=1}^{d} p_i$ density grids. For a density grid $g$ that is composed of $S_{1,j_1} \times S_{2,j_2} \cdots \times S_{d,j_d}$, $j_i = 1, \ldots, p_i$, we denote it as

$$g = (j_1, j_2, \ldots, j_d). \tag{3}$$

A data record $x = (x_1, x_2, \ldots, x_d)$ can be **mapped** to a density grid $g(x)$ as follows:

$$g(x) = (j_1, j_2, \ldots, j_d) \text{ where } x_i \in S_{i,j_i}.$$

Most density-based clustering algorithms simply map each data record to a grid to compute the density. Such a mapping sometimes can lose certain positional information of the data, although having a finer resolution of the grids can alleviate this issue. An example is shown in Figure 2b, in which the data in grid 1 locate at the left upper corner and have high density, the data in grid 5 locate at the bottom of the grid and have high density, while the density of grids 2, 3, 4, 6 are low. The usual density-based clustering algorithm will group grids 1 and 5 as one cluster while labeling grids 2, 3, 4, 6 as sparse grids. Such a result is obviously undesirable as we see that grids 1-4 and grids 5-6 form two natural clusters. To address this problem, we propose a better scheme that considers the attraction of neighboring grids and integrate it into our density-based clustering algorithm.

DEFINITION 3.1. **(Initial Attraction)** For a grid $g$, let its width at the $i^{th}$ dimension be $2r_i$ and let $c_i$ be the location of the middle point of the $i^{th}$ dimension. For each data record $x = (x_1, \ldots, x_d)$ that maps to $g$, construct a hypercube $Cube(x)$ centered at $x$ whose width is $2\epsilon_i$, $0 \leq \epsilon_i \leq r_i$, at each of the $d$ dimensions. Let $V(x, h)$ be the volume of the intersection of $Cube(x)$ and a grid $h$, the initial attraction between $x$ and $h$ $attr_{ini}(x, h)$ is

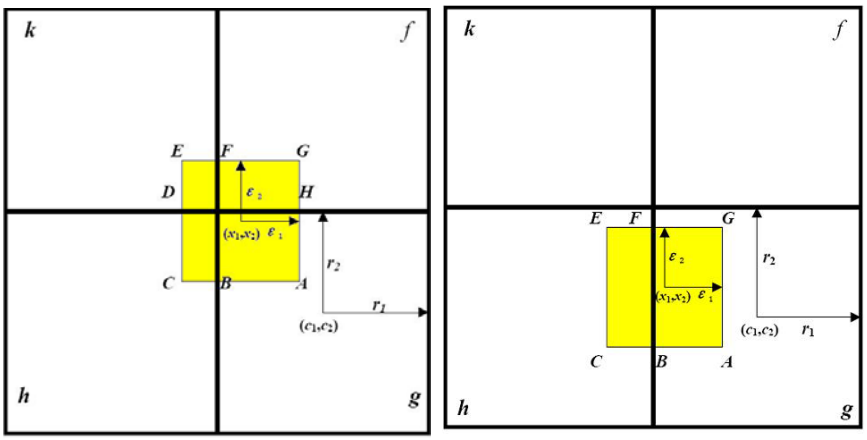



Fig. 3. Illustration of initial attraction between a data record $x$ and some grids.

defined as the ratio of $V(x,h)$ to the volume of $Cube(x)$. That is,

$$attr_{ini}(x,h) = \frac{V(x,h)}{\prod_{i=1}^{d} 2\epsilon_i}. \tag{4}$$

We illustrate the initial attraction in Figure 3. As shown in Figure 3a, the data $x = (x_1, x_2)$ maps to a grid $g$ centered at $(c_1, c_2)$. $Cube(x)$ is the yellow rectangle ACEG, sized $2\epsilon_1 \times 2\epsilon_2$. $Cube(x)$ intersects with grids $g, h, f$ and $k$.

From Figure 3a, we see that the area of the intersection of $Cube(x)$ and $g$ is

$$\begin{aligned} \overline{AB} \cdot \overline{AH} &= [(x_1 + \epsilon_1) - (c_1 - r_1)] \cdot [(c_2 + r_2) - (x_2 - \epsilon_2)] \\ &= [(\epsilon_1 + r_1) - |c_1 - x_1|] \cdot [(\epsilon_2 + r_2) - |c_2 - x_2|]. \end{aligned} \tag{5}$$

Therefore, the initial attraction between $x$ and $g$ is

$$\begin{aligned} attr_{ini}(x,g) &= \frac{[(\epsilon_1 + r_1) - |c_1 - x_1|] \cdot [(\epsilon_2 + r_2) - |c_2 - x_2|]}{4\epsilon_1\epsilon_2} \\ &= \frac{(\epsilon_1 + r_1) - |c_1 - x_1|}{2\epsilon_1} \cdot \frac{(\epsilon_2 + r_2) - |c_2 - x_2|}{2\epsilon_2} = b_1(x,g) b_2(x,g), \end{aligned}$$

where $b_i(x,g) = \frac{(\epsilon_i + r_i) - |c_i - x_i|}{2\epsilon_i} = \frac{1}{2} + \frac{r_i}{2\epsilon_i} - \Delta_i$, and $\Delta_i = \frac{|c_i - x_i|}{2\epsilon_i}$.

Similarly, for grid $h$, we can find that

$$\overline{CB} \cdot \overline{CD} = [(\epsilon_1 - r_1) + |c_1 - x_1|] \cdot [(\epsilon_2 + r_2) - |c_2 - x_2|], \tag{6}$$

$$attr_{ini}(x,h) = \frac{(\epsilon_1 - r_1) + |c_1 - x_1|}{2\epsilon_1} \cdot \frac{(\epsilon_2 + r_2) - |c_2 - x_2|}{2\epsilon_2} = b_1(x,h) b_2(x,h), \tag{7}$$

where $b_1(x,h) = \frac{(\epsilon_1 - r_1) + |c_1 - x_1|}{2\epsilon_1} = \frac{1}{2} - \frac{r_1}{2\epsilon_1} + \Delta_1$ and $b_2(x,h) = \frac{(\epsilon_2 + r_2) - |c_2 - x_2|}{2\epsilon_2} = \frac{1}{2} + \frac{r_2}{2\epsilon_2} - \Delta_2$.

We can further find that $attr_{ini}(x,f) = b_1(x,f)b_2(x,f)$ where $b_1(x,f) = \frac{1}{2} + \frac{r_1}{2\epsilon_1} - \Delta_1$ and $b_2(x,f) = \frac{1}{2} - \frac{r_2}{2\epsilon_2} + \Delta_2$; $attr_{ini}(x,k) = b_1(x,k)b_2(x,k)$ where $b_1(x,k) = \frac{1}{2} - \frac{r_1}{2\epsilon_1} - \Delta_1$ and $b_2(x,k) = \frac{1}{2} - \frac{r_2}{2\epsilon_2} - \Delta_2$.

In Figure 3a, $x_i$ and $c_i$ satisfy $|x_i - c_i| > r_i - \epsilon_i$. When $|x_i - c_i| < r_i - \epsilon_i$, $Cube(x)$ will be completely within grid $g$ in the $i^{th}$ dimension. In this case, if grids $h$ and $g$ correspond to the same partition in the $i^{th}$ dimension, then we set $b_i(x,h) = 1$ when computing $attr_{ini}(x,h)$; if they differ in the $i^{th}$ dimension, we set $b_i(x,h) = 0$.

For example, for the data $x$ shown in Figure 3b, we have the following results.

For grid $g$, $b_1(x,g) = \frac{1}{2} + \frac{r_1}{2\epsilon_1} - \Delta_1$, $b_2(x,g) = 1$, $attr_{ini}(x,g) = b_1(x,g)b_2(x,g) = \frac{1}{2} + \frac{r_1}{2\epsilon_1} - \Delta_1$.

For grid $h$, $b_1(x,h) = \frac{1}{2} - \frac{r_1}{2\epsilon_1} + \Delta_1$, $b_2(x,h) = 1$, $attr_{ini}(x,h) = b_1(x,h)b_2(x,h) = \frac{1}{2} - \frac{r_1}{2\epsilon_1} + \Delta_1$.

For grid $f$, $b_1(x,f) = \frac{1}{2} + \frac{r_1}{2\epsilon_1} - \Delta_1$, $b_2(x,f) = 0$, $attr_{ini}(x,f) = b_1(x,f)b_2(x,f) = 0$.

For grid $k$, $b_1(x,k) = \frac{1}{2} - \frac{r_1}{2\epsilon_1} + \Delta_1$, $b_2(x,k) = 0$, $attr_{ini}(x,k) = b_1(x,k)b_2(x,k) = 0$.

The following result generalizes to the case of $d$ dimensions. In a $d$-dimensional space, for a grid $g$ centered at $c_1, \ldots, c_d$. We define the **neighborhood of** $g$, $NB(g)$ as the set of grids whose center differs from $g$ in at most one dimension.

LEMMA 3.1. For a $d$-dimensional data record $x = (x_1, \ldots, x_d)$, let $x$ map to a grid $g$ centered at $c_1, \ldots, c_d$. Let $g' \in NB(g)$ be a neighboring grid of $g$ centered at $(g'_1, \ldots, g'_d)$, $c_k \neq g'_k$, and $c_i = g'_i, i = 1, \ldots, d, i \neq k$. The attraction between $x$ and $g'$ is

$$attr_{ini}(x,g') = \prod_{i=1}^{d} b_i(x,g'), \tag{8}$$

where $b_i(x,g')$, which denotes the attraction between $x$ and $h$ in the $i^{th}$ dimension, is

$$b_i(x,g') = \begin{cases} \frac{1+\omega}{2}, & \text{if } |x_i - c_i| < r_i - \epsilon_i; \\ \frac{1}{2} + \omega(\frac{r_i}{2\epsilon_i} - \Delta_i), & \text{otherwise.} \end{cases} \tag{9}$$

where $\omega = -1$ if $i \neq k$ and $\omega = 1$ if $i = k$, and $\Delta_i = \frac{|x_i - c_i|}{2\epsilon_i}$.

We have the following result about the total attraction in the neighboring grids of a data.

LEMMA 3.2. For a $d$-dimensional data record $x = (x_1, \ldots, x_d)$, let $x$ maps to a grid $g$, then we have

$$\sum_{h \in NB(g)} attr_{ini}(x,h) = 1. \tag{10}$$

**Proof.** Lemma 3.2 can be seen from the definitions of the attraction and the neighborhood. Since $x$ maps to $g$ and since the hypercube $Cube(x)$ has a width of $2\epsilon_i$, $\epsilon_i < r_i$, in the $i^{th}$ dimension, we have that $Cube(x)$ only intersects with the grids in $NB(g)$ and the total volume of these intersections equals the volume of $Cube(x)$. Thus, let $V(x,h)$ be volume of the intersection of $Cube(x)$ and a grid $h$, we have $\sum_{h \in NB(g)} V(x,h) = \prod_{i=1}^{d} 2\epsilon_i$. Dividing each side of the above equation by $prod_{i=1}^{d} 2\epsilon_i$ and using Eq.(4), we have

$$\sum_{h \in NB(g)} \frac{V(x,h)}{\prod_{i=1}^{d} 2\epsilon_i} = \sum_{h \in NB(g)} attr_{ini}(x,h) = 1.$$

■

### 3.2 Density of grids

For each data record $x$, we assign it a **density coefficient** that decreases as $x$ ages. In fact, if $x$ arrives at time $t_c$, we define its **time stamp** $T(x) = t_c$, and its density coefficient $D(x,t)$ at time $t$ is

$$D(x,t) = \lambda^{t-T(x)} = \lambda^{t-t_c}, \tag{11}$$

where $\lambda \in (0,1)$ is a constant called the **decay factor**.

DEFINITION 3.2. **(Grid Density)** For a grid $g$, at a given time $t$, let $E(g,t)$ be the set of data records that are map to $g$ at or before time $t$, its density $D(g,t)$ is defined as the sum of the density coefficients of all data records that map to $g$. Namely, the density of $g$ at $t$ is:

$$D(g,t) = \sum_{x \in E(g,t)} D(x,t).$$

The density of any grid is constantly changing. However, we have found that it is unnecessary to update the density values of all data records and grids at every time step. Instead, it is possible to update the density of a grid only when a new data record is mapped to that grid. For each grid, the time when it receives the last data record should be recorded so that the density of the grid can be updated according to the following result when a new data record arrives at the grid.

THEOREM 3.1. Suppose a grid $g$ receives a new data record at time $t_n$, and suppose the time when $g$ receives the last data record is $t_l$ ($t_n > t_l$), then the density of $g$ can be updated as follows:

$$D(g,t_n) = \lambda^{t_n - t_l} D(g,t_l) + 1. \tag{12}$$

**Proof.** Let $X = \{x_1, \ldots, x_m\}$ be the set of all data records in $g$ at time $t_l$, we have:

$$D(g,t_l) = \sum_{i=1}^{m} D(x_i,t_l). \tag{13}$$

According to (11), we have that:

$$\begin{aligned} D(x_i,t_n) &= \lambda^{t_n - T(x_i)} = \lambda^{t_n - t_l}\lambda^{t_l - T(x_i)} \\ &= \lambda^{t_n - t_l} D(x_i,t_l), \text{ for } i = 1,\ldots,m. \end{aligned} \tag{14}$$

Therefore, we have:

$$\begin{aligned} D(g,t_n) &= \sum_{i=1}^{m} D(x_i,t_n) + 1 = \sum_{i=1}^{m} \lambda^{t_n - t_l} D(x_i,t_l) + 1 \\ &= \lambda^{t_n - t_l} \sum_{i=1}^{m} D(x_i,t_l) + 1 = \lambda^{t_n - t_l} D(g,t_l) + 1. \end{aligned}$$

■

Theorem 3.1 allows us to save a huge amount of computational time. To update all grids at each time step requires $\Theta(N)$ computing time for density update at each time step. In contrast, using Theorem 3.1 allows us to update only one grid, leading to a $\Theta(1)$ running

time. The efficiency improvement is significant since the number of grids $N$ is typically large.

Moreover, Theorem 3.1 saves memory space. We find that we do not need to save the time stamps and densities of all the data records in a grid. Instead, for each grid, it suffices to store a characteristic vector defined as follows. We will explain the use of each element in the vector later.

DEFINITION 3.3. **(Characteristic vector of a grid)** We define the characteristic vector of a grid $g$ as a tuple
$(t_g, t_m, C, D, label)$, where $t_g$ is the last time when $g$ is updated, $t_m$ is the last time when $g$ is removed from $grid\_list$ as a sporadic grid (if ever), $C$ is a $2d$-vector denoting the attraction from $g$ to its $2d$ neighbors, $D$ is the grid density at the last update, and $label$ is the class label of the grid.

### 3.3 Density-based grid clusters

We now need to decide how to derive clusters based on the density information. Our method is based on the following observation.

THEOREM 3.2. Let $X(t)$ be the set of all data records that arrive from time $0$ to $t$, we have:

1) $\sum_{x\in X(t)} D(x,t) \leq \frac{1}{1-\lambda}$, for any $t = 1, 2, \ldots$;

2) $\lim_{t\to\infty} \sum_{x\in X(t)} D(x,t) = \frac{1}{1-\lambda}$.

**Proof.** For a given time $t$, $\sum_{x\in X(t)} D(x,t)$ is the sum of density coefficient of the $t+1$ data records that arrive at time steps $0, 1, \ldots, t$, respectively. For a data record $x$ arriving at time $t'$, $0 \leq t' \leq t$ ($T(x) = t'$), its density is $D(x,t) = \lambda^{t-t'}$. Therefore, the sum over all the data records is:

$$\sum_{x\in X(t)} D(x,t) = \sum_{t'=0}^{t} \lambda^{t-t'} = \frac{1-\lambda^{t+1}}{1-\lambda} \leq \frac{1}{1-\lambda}.$$

Also, it is clear that:

$$\lim_{t\to\infty} \sum_{x\in X(t)} D(x,t) = \lim_{t\leftarrow\infty} \frac{1-\lambda^{t+1}}{1-\lambda} = \frac{1}{1-\lambda}. \qquad \blacksquare$$

Theorem 3.2 shows that the sum of the density of all data records in the system will never exceed $\frac{1}{1-\lambda}$. Since there are $N = \prod_{i=1}^{d} p_i$ grids, the **average density** of each grid is no more than but approaching $\frac{1}{N(1-\lambda)}$. This observation motivates the following definitions.

At time $t$, for a grid $g$, we call it a **dense grid** if

$$D(g,t) \geq \frac{C_m}{N(1-\lambda)} = D_m, \tag{15}$$

where $C_m > 1$ is a parameter controlling the threshold. For example, we set $C_m = 3$. We require $N > C_m$ since $D(g,t)$ cannot exceed $\frac{1}{1-\lambda}$.

At time $t$, for a grid $g$, we call it a **sparse grid** if

$$D(g,t) \leq \frac{C_l}{N(1-\lambda)} = D_l, \tag{16}$$

where $0 < C_l < 1$. For example, we set $C_l = 0.8$.

At time $t$, for a grid $g$, we call it a **transitional grid** if

$$\frac{C_l}{N(1-\lambda)} < D(g,t) < \frac{C_m}{N(1-\lambda)}. \tag{17}$$

In the multi-dimensional space, we consider connecting neighboring grids, defined below, in order to form clusters.

DEFINITION 3.4. **(Neighboring Grids)** For two density grids $g_1 = (j_1^1, j_2^1, \ldots, j_d^1)$ and $g_2 = (j_1^2, j_2^2, \ldots, j_d^2)$, if there exists $k$, $1 \leq k \leq d$, such that:
1) $j_i^1 = j_i^2, i = 1, \ldots, k-1, k+1, \ldots, d$; and
2) $|j_k^1 - j_k^2| = 1$,
then $g_1$ and $g_2$ are neighboring grids in the $k^{th}$ dimension, denoted as $g_1 \sim g_2$.

DEFINITION 3.5. **(Grid Group)** A set of density grids $G = (g_1, \ldots, g_m)$ is a grid group if for any two grids $g_i, g_j \in G$, there exist a sequence of grids $g_{k_1}, \ldots, g_{k_l}$ such that $g_{k_1} = g_i$, $g_{k_l} = g_j$, and $g_{k_1} \sim g_{k_2}$, $g_{k_2} \sim g_{k_3}$, ..., and $g_{k_{l-1}} \sim g_{k_l}$.

DEFINITION 3.6. **(Inside and Outside Grids)** Consider a grid group $G$ and a grid $g \in G$, suppose $g = (j_1, \ldots, j_d)$, if $g$ has neighboring grids in $G$ in every dimension $i = 1, \ldots, d$, then $g$ is an inside grid of $G$. Otherwise $g$ is an outside grid of $G$.

Now we are ready to define how to form clusters based on the density of grids.

DEFINITION 3.7. **(Grid Cluster)** A set of grids $G = (g_1, \ldots, g_m)$ is a grid cluster if it is a grid group, every inside grid of $G$ is a dense grid, and every outside grid of $G$ is either a dense grid or a transitional grid.

Intuitively, a grid cluster is a connected grid group which has higher density than the surrounding grids. Note that we always try to merge clusters whenever possible, so the resulting clusters are surrounded by sparse grids.

### 3.4 Attraction over time

For each data record $x$, let its arrival time be $t_c$, the attraction between $x$ and a grid $g$ at the current time $t$ $attr(x, g, t)$ is defined as

$$attr(x, g, t) = \lambda^{t-t_c} attr_{ini}(x, g), \tag{18}$$

where $\lambda$ is the decay factor. If $t_1 > t_2$, then we see that

$$attr(x, g, t_1) = \lambda^{t_1-t_2} attr(x, g, t_2), \tag{19}$$

DEFINITION 3.8. **(Grid Attraction)** Let $g$ and $h$ be two neighboring grids, for a given time $t$, let $E(g,t)$ be the set of data records that are mapped to $g$ at or before time $t$, the attraction from $g$ to $h$ is defined as

$$attr(g, h, t) = \sum_{x \in E(g,t)} attr(x, h, t). \tag{20}$$

It should be noted that the grid attraction is asymmetric. That is, typically we have $attr(g, h, t) \neq attr(h, g, t)$ because $attr(g, h, t)$ represents how close the data in $g$ is to $h$.

LEMMA 3.3. If $t_1 > t_2$, then $attr(g, h, t_1) = \lambda^{t_1-t_2} attr(g, h, t_2)$.

**Proof.** We have that

$$\begin{aligned} attr(g, h, t_1) &= \sum_{x \in E(g,t)} attr(x, h, t_1) = \sum_{x \in E(g,t)} \lambda^{t_1-t_2} attr(x, h, t_2) \\ &= \lambda^{t_1-t_2} attr(g, h, t_2) \end{aligned}$$

■

LEMMA 3.4. We have the following relationship between the density and attraction of a grid. For each grid $g$, we have

$$D(g,t) = \sum_{h \in NB(g)} attr(g,h,t). \tag{21}$$

**Proof.** Let the data records in grid $g$ at time $t$ be $x_1, \ldots, x_m$. Let their arriving time be $t_1, \ldots, t_m$, respectively. We have

$$\begin{aligned}\sum_{h \in NB(g)} attr(g,h,t) &= \sum_{h \in NB(g)} \sum_{i=1}^{m} attr(x_i,h,t) = \sum_{h \in NB(g)} \sum_{i=1}^{m} \lambda^{t-t_i} attr_{ini}(x_i,h) \\ &= \sum_{i=1}^{m} \left( \lambda^{t-t_i} \sum_{h \in NB(g)} attr_{ini}(x_i,h) \right) = \sum_{i=1}^{m} \lambda^{t-t_i} = D(g,t)\end{aligned}$$

■

THEOREM 3.3. Let $X(t)$ be the set of all data records that arrive from time 0 to $t$ and $S$ be the set of all grids, we have that,

1) $\lim_{t\to\infty} \sum_{g\in S} \sum_{h\in NB(g)} attr(g,h,t) = \frac{1}{1-\lambda}$ ;
2) $\lim_{t\to\infty} \sum_{x\in X(t)} \sum_{g\in S} attr(x,g,t) = \frac{1}{1-\lambda}$ .

**Proof.**

1) From Lemma 3.4, we have

$$\lim_{t\to\infty} \sum_{g\in S} \sum_{h\in NB(g)} attr(g,h,t) = \lim_{t\to\infty} \sum_{g\in S} D(g,t) = \frac{1}{1-\lambda}.$$

2) We have:

$$\begin{aligned}\lim_{t\to\infty} \sum_{x\in X(t)} \sum_{g\in S} attr(x,g,t) &= \lim_{t\to\infty} \sum_{g\in S} \sum_{x\in X(t)} attr(x,g,t) \\ &= \lim_{t\to\infty} \sum_{g\in S} \sum_{h\in NB(g)} attr(g,h,t) = \frac{1}{1-\lambda}.\end{aligned}$$

■

We will later use Theorem 3.3 to enhance our clustering algorithm based on grid attraction information.

## 4. COMPONENTS OF D-STREAM

We now describe in detail the key components of D-Stream outlined in Figure 1. As we have discussed in the last section, for each new data record $x$, we map it to a grid $g$ and use (12) to update the density of $g$ (Lines 5-8 of Figure 1). We then periodically (every $gap$ time steps) form clusters and remove sporadic grids. In the following, we describe our strategies for determining $gap$, managing the list of active grids, and generating clusters.

### 4.1 Grid inspection and time interval gap

To mine the dynamic characteristics of data streams, our density grid scheme developed in Section 3 gradually reduces the density of each data record. A dense grid may degenerate to a transitional or sparse grid if it does not receive new data for a long time. On the other hand, a sparse grid can be upgraded to a transitional or dense grid after it receives some new data records. Therefore, after a period of time, the density of each grid should be inspected and the clusters adjusted.

A key decision is the length of the time interval for grid inspection. It is interesting to note that the value of the time interval $gap$ cannot be too large or too small. If $gap$ is too large, dynamical changes of data streams will not be adequately recognized. If $gap$ is too small, it will result in frequent computation by the offline component and increase the workload. When such computation load is too heavy, the processing speed of the offline component may not match the speed of the input data stream.

We propose the following strategy to determine the suitable value of $gap$. We consider the minimum time needed for a dense grid to degenerate to a sparse grid as well as the minimum time needed for a sparse grid to become a dense grid. Then we set $gap$ to be the minimum of these two intervals in order to ensure that the inspection is frequent enough to detect the density changes of any grid.

THEOREM 4.1. For any dense grid $g$, the minimum time needed for $g$ to become a sparse grid is

$$\delta_0 = \left\lfloor \log_\lambda \left( \frac{C_l}{C_m} \right) \right\rfloor. \tag{22}$$

**Proof.** According to (15), if at time $t$, a grid $g$ is a dense grid, then we have:

$$D(g,t) \geq D_m = \frac{C_m}{N(1-\lambda)}. \tag{23}$$

Suppose after $\delta_t$ time, $g$ becomes a sparse grid, then we have:

$$D(g,t+\delta_t) \leq D_l = \frac{C_l}{N(1-\lambda)}. \tag{24}$$

On the other hand, let $E(g,t)$ be the set of data records in $g$ at time $t$, we have $E(g,t) \subseteq E(g,t+\delta_t)$ and

$$\begin{aligned} D(g,t+\delta_t) &= \sum_{x \in E(g,t+\delta_t)} D(x,t+\delta_t) \geq \sum_{x \in E(g,t)} D(x,t+\delta_t) \\ &= \sum_{x \in E(g,t)} \lambda^{\delta_t} D(x,t) = \lambda^{\delta_t} D(g,t). \end{aligned} \tag{25}$$

Combining (24) and (25) we get:

$$\lambda^{\delta_t} D(g,t) \leq D(g,t+\delta_t) \leq \frac{C_l}{N(1-\lambda)}. \tag{26}$$

Combining (23) and (26) we get:

$$\lambda^{\delta_t} \frac{C_m}{N(1-\lambda)} \leq \lambda^{\delta_t} D(g,t) \leq \frac{C_l}{N(1-\lambda)}, \tag{27}$$

which yields:

$$\delta_t \geq \log_\lambda \left( \frac{C_l}{C_m} \right). \tag{28}$$

■

THEOREM 4.2. For any sparse grid $g$, the minimum time needed for $g$ to become a dense grid is

$$\delta_1 = \left\lfloor \log_\lambda \left( \frac{N - C_m}{N - C_l} \right) \right\rfloor. \tag{29}$$

**Proof.** According to (16), if at time $t$, a grid $g$ is a sparse grid, then we have:

$$D(g,t) \leq D_l = \frac{C_l}{N(1-\lambda)}. \tag{30}$$

Suppose after $\delta_t$ time, $g$ becomes a dense grid, then we have:

$$D(g,t+\delta_t) \geq D_m = \frac{C_m}{N(1-\lambda)}. \tag{31}$$

We also know that:

$$D(g,t+\delta_t) = \sum_{x \in E(g,t+\delta_t)} D(x,t+\delta_t). \tag{32}$$

$E(g,t+\delta_t)$ can be divided into those points in $E(g,t)$ and those come after $t$. The least time for a sparse grid $g$ to become dense is achieved when all the new data records are mapped to $g$. In this case, there is a new data record mapped to $g$ for any of the time steps from $t+1$ until $t+\delta_t$. The sum of the density of all these new records at time $t+\delta_t$ is $\sum_{i=0}^{\delta_t - 1} \lambda^i$. Therefore we have:

$$\begin{aligned} D(g,t+\delta_t) &\leq \sum_{x \in E(g,t)} D(x,t+\delta_t) + \sum_{i=0}^{\delta_t - 1} \lambda^i \\ &= \sum_{x \in E(g,t)} \lambda^{\delta_t} D(x,t) + \frac{1-\lambda^{\delta_t}}{1-\lambda} = \lambda^{\delta_t} D(g,t) + \frac{1-\lambda^{\delta_t}}{1-\lambda} \end{aligned} \tag{33}$$

Now we plug (31) and (30) into (33) to obtain

$$\frac{C_m}{N(1-\lambda)} \leq D(g,t+\delta_t) \leq \lambda^{\delta_t} D(g,t) + \frac{1-\lambda^{\delta_t}}{1-\lambda} \leq \frac{\lambda^{\delta_t} C_l}{N(1-\lambda)} + \frac{1-\lambda^{\delta_t}}{1-\lambda}. \tag{34}$$

Solving (34) yields:

$$\lambda^{\delta_t} \leq \frac{N - C_m}{N - C_l}, \tag{35}$$

which results in:

$$\delta_t \geq \log_\lambda \left( \frac{N - C_m}{N - C_l} \right). \tag{36}$$

Note $N - C_m > 0$ since $C_m < N$ according to (15). ■

Based on the two theorems above, we choose $gap$ to be small enough so that any change of a grid from dense to sparse or from sparse to dense can be detected. Thus, in D-Stream we set:

$$\begin{aligned} gap &= \min\{\delta_0, \delta_1\} = \min\left\{\left\lfloor \log_\lambda \frac{C_l}{C_m}\right\rfloor, \left\lfloor \log_\lambda \frac{N-C_m}{N-C_l}\right\rfloor\right\} \\ &= \left\lfloor \log_\lambda \left(\max\left\{\frac{C_l}{C_m}, \frac{N-C_m}{N-C_l}\right\}\right)\right\rfloor. \end{aligned} \tag{37}$$

## 4.2 Detecting and removing sporadic grids

A serious challenge for the density grid scheme is the large number of grids, especially for high-dimensional data. For example, if each dimension is divided into 20 regions, there will be $20^d$ possible grids.

A key observation is that most of the grids in the space are empty or receive data very infrequently. In our implementation, we allocate memory to store the characteristic vectors for those grids that are not empty, which form a very small subset in the grid space. Unfortunately, in practice, this is still not efficient enough due to the appearance of outlier data that are made from errors, which lead to continual increase of non-empty grids that will be processed during clustering. We call such grids **sporadic grids** since they contain very few data. Since a data stream flows in by massive volume in high speed and it could run for a very long time, due to noises and exceptions in the data stream, more and more grids will be activated during the process, although many of them contain only very few data. If we do not control these sporadic grids, the total number of grids in the *grid list* will keep increasing and become exceedingly large. Therefore, it is imperative to detect and remove such sporadic grids periodically. This is done in Line 13 of the D-Stream algorithm in Figure 1.

We define a density threshold function to differentiate these two classes of sparse grids.

DEFINITION 4.1. **(Density Threshold Function)** Suppose the last update time of a grid $g$ is $t_g$, then at time $t$ ($t > t_g$), the density threshold function is

$$\pi(t_g, t) = \frac{C_l}{N}\sum_{i=0}^{t-t_g}\lambda^i = \frac{C_l(1-\lambda^{t-t_g+1})}{N(1-\lambda)} \tag{38}$$

LEMMA 4.1. The density threshold function $\pi(t_g, t)$ has the following properties.

(1) If $t_1 \le t_2 \le t_3$, then

$$\lambda^{t_3-t_2}\pi(t_1, t_2) + \pi(t_2+1, t_3) = \pi(t_1, t_3).$$

(2) If $t_1 \le t_2$, then $\pi(t_1, t) \ge \pi(t_2, t)$ for any $t > t_1, t_2$.

**Proof.** (1) We see that:

$$\begin{aligned} \lambda^{t_3-t_2}\pi(t_1, t_2) + \pi(t_2+1, t_3) &= \frac{C_l}{N}\sum_{i=0}^{t_2-t_1}\lambda^{t_3-t_2+i} + \frac{C_l}{N}\sum_{i=0}^{t_3-t_2-1}\lambda^i \\ = \frac{C_l}{N}\sum_{i=t_3-t_2}^{t_3-t_1}\lambda^i + \frac{C_l}{N}\sum_{i=0}^{t_3-t_2-1}\lambda^i &= \frac{C_l}{N}\sum_{i=0}^{t_3-t_1}\lambda^i = \pi(t_1, t_3). \end{aligned}$$

(2) Let $\Delta t = t_2 - t_1$, we have

$$\begin{aligned}\pi(t_1,t) &= \frac{C_l}{N}\sum_{i=0}^{t-t_1}\lambda^i = \frac{C_l}{N}\sum_{i=0}^{t-t_2+\Delta t}\lambda^i \\ &= \frac{C_l}{N}\sum_{i=0}^{t-t_2}\lambda^i + \frac{C_l}{N}\sum_{i=t-t_2+1}^{t-t_2+\Delta t}\lambda^i = \pi(t_2,t) + \frac{C_l}{N}\sum_{i=t-t_2+1}^{t-t_2+\Delta t}\lambda^i \geq \pi(t_2,t).\end{aligned}$$

■

We use $\pi(t_g,t)$ to detect sporadic grids from sparse grids. In the periodic inspection in Line 13 of Figure 1, at time $t$, we judge that a sparse grid is a sporadic grid if $D(g,t) < \pi(t_g,t)$. During the periodic inspection in Line 13 of Figure 1, all sporadic grids are deleted from $grid\_list$.

In D-Stream, we maintain a $grid\_list$ which includes the grids that are under consideration for clustering analysis. The $grid\_list$ is implemented as a red-black tree, which allows for fast lookup, update, and deletion. The key of the tree is the grid coordinates, while the associated data for each grid entry is the characteristic vector. Later we show that the size of $grid\_list$ in our system is only $O(\log_{1/\lambda} N)$, where $N$ is the total number of grids and $\lambda$ is the decay factor. The time for lookup and update in the red-black tree is only $O(\log\log_{1/\lambda} N)$, which is very small.

It should be noted that once a sporadic grid is deleted, its density is in effect reset to zero since its characteristic vector is deleted. A deleted grid may be added back to $grid\_list$ if there are new data records mapped to it later, but its previous records are discarded and its density restarts from zero. Such a dynamic mechanism maintains a moderate size of the grids in memory, saves computing time, and prevents infinite accumulation of sporadic grids in memory.

Although deleting sporadic grids is critical for the efficient performance of D-Stream, an important issue for the correctness of this scheme is whether the deletions affect the clustering results. In particular, since a sporadic grid may receive data later and become a transitional or dense grid, we need to know if it is possible that the deletion prevents this grid from being correctly labelled as a transitional or dense grid. We have designed the density threshold function $\pi(t_g,t)$ and the deletion rules in such a way that a transitional or dense grid will never be falsely deleted due to the removal of sporadic grids.

Consider a grid $g$, whose density at time $t$ is $D(g,t)$. Suppose that it has been deleted several times before $t$ (the density is reset to zero each time) because its density is less than the density threshold function at various times. Suppose these density values are not cleared and suppose all historic data are kept, the density of grid $g$ would be $D_a(g,t)$. We call $D_a(g,t)$ the **complete density function** of the grid $g$.

Now we present several theoretical properties of $\pi(t_g,t)$ which ensure the proper functioning of the D-Stream system. We will show that, if a grid can later become a transitional or dense grid, deleting it as a sporadic grid will not affect its later upgrades.

The first question we investigate is, if a grid $g$ is detected as a sporadic grid, is it possible that $g$ can be non-sporadic if it has not been previously deleted from $grid\_list$? It is answered in the following result.

THEOREM 4.3. Suppose the last time a grid $g$ is deleted as a sporadic grid is $t_m$ and the last time $g$ receives a data record is $t_g$. If at current time $t$, we have $D(g,t) < \pi(t_g,t)$, then we also have $D_a(g,t) < \pi(0,t) < D_l$.

**Proof.** Suppose the grid $g$ has been previously deleted for the periods of $(0, t_1)$, $(t_1+1, t_2)$, ..., $(t_{m-1}+1, t_m)$, then the density value $D(g, t_i), i = 1..m$ satisfies (let $t_0 = -1$):

$$D(g, t_i) < \pi(t_{i-1} + 1, t_i). \tag{39}$$

Thus, if all these previous data are not deleted, the complete density function satisfies:

$$D_a(g, t) = \sum_{i=1}^{m} D(g, t_i)\lambda^{t-t_i} + D(g, t) \ < \ \sum_{i=1}^{m} \pi(t_{i-1} + 1, t_i)\lambda^{t-t_i} + \pi(t_g, t). \tag{40}$$

Since $t_g \geq t_m + 1$, from property (2) in Lemma 4.1, we know

$$\begin{aligned} D_a(g, t) \ &< \ \sum_{i=1}^{m} \pi(t_{i-1} + 1, t_i)\lambda^{t-t_i} + \pi(t_m + 1, t) \\ &= \ \sum_{i=1}^{m-1} \pi(t_{i-1} + 1, t_i)\lambda^{t-t_i} + \pi(t_{m-1} + 1, t) \\ &= \ \sum_{i=1}^{m-2} \pi(t_{i-1} + 1, t_i)\lambda^{t-t_i} + \pi(t_{m-2} + 1, t) \\ &\cdots \\ &= \ \pi(0, t) = \frac{C_l(1 - \lambda^{t+1})}{N(1 - \lambda)} < D_l. \end{aligned} \tag{41}$$

The last equalities are based on successive applications of property (1) in Lemma 4.1. ■

Theorem 4.3 is important since it shows that deleting a sporadic grid will not cause transitional or dense grid to be falsely deleted. It shows that, if $g$ is deleted as a sporadic grid at $t$ since $D(g, t) < \pi(t_g, t)$, then even if all the previous deletions have not occurred, it is still sporadic and cannot be a transitional or dense grid since $D_a(g, t) < D_l$.

We explain the design of the threshold function as follows. Suppose a grid $g$ has been deleted at time steps $t_1, t_2, \ldots, t_k = t$. To avoid false-positive, we need to ensure that $g$ cannot become a transitional grid even without the deletions. That is, we should have:

$$D_a(g, t) \leq \frac{C_l(1 - \lambda^{t+1})}{N(1 - \lambda)}.$$

One the other hand, we know $D_a(g, t) \leq \sum_{i=1}^{k} \pi(t_{i-1}, t_i)\lambda^{t-t_i}$. Hence, we require:

$$\sum_{i=1}^{k} \pi(t_{i-1}, t_i)\lambda^{t-t_i} \leq \frac{C_l(1 - \lambda^{t+1})}{N(1 - \lambda)}. \tag{42}$$

Since we have:

$$\begin{aligned} \frac{C_l(1 - \lambda^{t+1})}{N(1 - \lambda)} \ &= \ \frac{C_l}{N}(1 + \lambda + \cdots + \lambda^t) = \frac{C_l}{N}\sum_{i=1}^{k}(1 + \lambda + \cdots + \lambda^{t_i - t_{i-1}})\lambda^{t-t_i} \\ &= \sum_{i=1}^{k} \frac{C_l(1 - \lambda^{t_i - t_{i-1}+1})}{N(1 - \lambda)}\lambda^{t-t_i}. \end{aligned} \tag{43}$$

Comparing (43) with (42), we see that we should set

$$\pi(t_{i-1}, t_i) = \frac{C_l(1-\lambda^{t_i - t_{i-1}+1})}{N(1-\lambda)}.$$

We have the following results regarding the memory requirement when sporadic grids are removed.

THEOREM 4.4. The size of $grid\_list$ will never exceed $L = \log_\lambda \frac{C_l}{N+C_l}$, where $N$ is the total number of grids and $\lambda$ is the decay factor.

**Proof.** First, we see that the density of any grid $g$ at time $t$ satisfies $D(g,t) = D(g, t-t_g)\lambda^{t-t_g}$, where $t_g$ is the time when $g$ last receives a data.

Consider the case when $t - t_g > L = \log_\lambda \frac{C_l}{N+C_l}$, then we have

$$\begin{aligned} D(g,t) &= D(g,t-t_g)\lambda^{t-t_g} < D(g,t-t_g)\lambda^L \\ &< \frac{\lambda^L}{1-\lambda} \left(\text{since total density never exceeds } \frac{1}{1-\lambda}\right) \end{aligned}$$

Since

$$N\lambda^L + C_l\lambda^{t-t_g+1} < N\lambda^L + C_l\lambda^{t-t_g} < N\lambda^L + C_l\lambda^L = C_l,$$

we know that

$$N\lambda^L < C_l - C_l\lambda^{t-t_g+1}.$$

Therefore, we have:

$$D(g,t) < \frac{\lambda^L}{1-\lambda} = \frac{N\lambda^L}{N(1-\lambda)} < \frac{C_l - C_l\lambda^{t-t_g+1}}{N(1-\lambda)} = \frac{C_l(1-\lambda^{t-t_g+1})}{N(1-\lambda)} = \pi(t_g, t).$$

That is to say, when $t - t_g > L$, the grid will be deleted and will not be kept in *grid_list*. Note that all the entries in $grid\_list$ have different $t_g$ values. Since at a time step $t$, any grid with $t - t_g > L$ will be deleted, there are most $L$ entries in *grid_list*.

Theorem 4.4 shows the importance of removing sporadic grids. It shows that our algorithm has a space complexity of $O(\log_{1/\lambda} N)$. Thus, although $N$ is exponential to the number of dimensions, the size of $grid\_list$ is small when sporadic grids are removed.

### 4.3 Clustering algorithms

We describe the algorithms for generating the initial clusters and for adjusting the clusters every $gap$ steps. The procedure initial_clustering (used in Line 10 of Figure 1) is illustrated in Figure 4. We use the attraction between grids to decide the merging of grids. In initial_clustering, the standard density-based algorithm merges neighboring grids to form clusters. In our algorithm, however, we merge two neighboring grids only if they are strongly correlated. We define that two grids are **strongly correlated** if their attractions in both directions are higher than a threshold $\theta > 0$. That is, two grids $g$ and $h$ are strongly correlated if $attr(g,h,t) > \theta$ and $attr(h,g,t) > \theta$.

To set the appropriate value of $\theta$, we note from Theorem 3.3 that the summation of all pairs of neighboring grids approaches $\frac{1}{1-\lambda}$. Define the set $\mathcal{P}$ of grid pairs as

$$\mathcal{P} = \left\{ (g,h) \middle| g \in S, h \in NB(g) \right\}. \tag{44}$$

```
1.  procedure initial_clustering (grid_list)
2.      update the density of all grids in grid_list;
3.      assign each dense grid to a distinct cluster;
4.      label all other grids as NO_CLASS;
5.      repeat
6.          foreach cluster c
7.              foreach outside grid g of c
8.                  foreach neighboring grid h of g
9.                      if (g and h are strongly correlated) and (h belongs to cluster c')
10.                         if (|c| > |c'|) label all grids in c' as in c;
11.                         else label all grids in c as in c';
12.                     else if (g and h are strongly correlated) and (h is transitional) label h as in c;
13.     until no change in the cluster labels can be made
14. end_procedure
```

Fig. 4. The procedure for initial clustering.

The result in Theorem 3.3 can be written as

$$\lim_{t\to\infty} \sum_{(g,h)\in\mathcal{P}} attr(g,h,t) = \frac{1}{1-\lambda}. \tag{45}$$

Therefore, the total attraction is approaching but no more than $\frac{1}{1-\lambda}$ and the average attraction between each grid pair $(g,h)$ is approaching but no more than $\frac{1}{|\mathcal{P}|(1-\lambda)}$, where $|\mathcal{P}|$, the size of $\mathcal{P}$, can be easily calculated given the number of partitions in each dimension. Following the same reason as that for defining the dense grids, we set $\theta = \frac{C_m}{|\mathcal{P}|(1-\lambda)}$, where $C_m > 1$ is a parameter. That is, two neighboring grids $g$ and $h$ are strongly correlated if $attr(g,h,t) > \frac{C_m}{|\mathcal{P}|(1-\lambda)}$ and $attr(h,g,t) > \frac{C_m}{|\mathcal{P}|(1-\lambda)}$.

The procedure adjust_clustering (used in Line 14 of Figure 1) is illustrated in Figure 5. It first updates the density of all active grids to the current time. Once the density of grids are determined at the given time, the clustering procedure differs from the standard method used by density-based clustering in that we use the information of grid attraction and merge only strongly correlated grids. In adjust_clustering(), it only needs to adjust those grids whose density attribute (dense/sparse/transitional) are changed since the last call to adjust_clustering(). Since at most $gap$ grids are changed during each cycle, the main loop of adjust_clustering() will be executed at most $gap$ times. The cost of executing the main loop once is $O(1)$ and the cost of adjust_clustering() is $O(gap)$.

Since the D-Stream algorithm uses an online/offline framework, the online task of processing data and the offline work in adjust_clustering() are performed in parallel, in a pipelined fashion. The procedure adjust_clustering() is executed every $gap$ time steps. Therefore, adjust_clustering() will not affect the overall efficiency of the algorithm as long as its running time is less than the time needed for the online component to process $gap$ data records.

It should be noted that, during the computation, whenever we update grids or find neighboring grids, we only consider those grids that are maintained in $grid\_list$. Therefore, although the number of possible grids is huge for high-dimensional data, most empty or infrequent grids are discarded, which saves computing time and makes our algorithm very fast without deteriorating clustering quality.

```
1.  procedure adjust_clustering (grid_list)
2.      update the density of all grids in grid_list;
3.      foreach grid g whose attribute (dense/sparse/transitional)
                is changed since last call to adjust_clustering()
4.          if (g is a sparse grid)
5.              delete g from its cluster c, label g as NO_CLASS;
6.              if (c becomes unconnected) split c into two clusters;
7.          else if (g is a dense grid)
8.              among all neighboring grids of g that are strongly correlated to g,
                    find out the grid h whose cluster c_h has the largest size;
9.              if (h is a dense grid)
10.                 if (g is labelled as NO_CLASS) label g as in c_h;
11.                 else if (g is in cluster c and |c| > |c_h|)
12.                     label all grids in c_h as in c;
13.                 else if (g is in cluster c and |c| ≤ |c_h|)
14.                     label all grids in c as in c_h;
15.             else if (h is a transitional grid)
16.                 if ((g is NO_CLASS) and (h is an outside
                        grid if g is added to c_h)) label g as in c_h;
17.                 else if (g is in cluster c and |c| ≥ |c_h|)
18.                     move h from cluster c_h to c;
19.         else if (g is a transitional grid)
20.             among neighboring grids of g that are dense and strongly correlated to g,
                    find the grid whose cluster c' has the largest size;
21.             label g as in c';
22.     end for
23. end_procedure
```

Fig. 5. The procedure for dynamically adjusting clusters.

## 5. EXPERIMENTAL RESULTS

We evaluate the quality and efficiency of D-Stream and compare it with CluStream [1]. The algorithm we propose in this paper extends the D-Stream framework in our KDD paper [7] by introducing the use of grid attraction. Therefore, we report the performance of both the D-Stream system without attraction (**denoted as** $DS0$) and with attraction (**denoted as** $DS1$). All of our experiments are conducted on a PC with 1.7GHz CPU and 256M memory. We have implemented D-Stream in VC++ 6.0 with a Matlab graphical interface. In all experiments, we use $C_m = 3.0$, $C_l = 0.8$, $\lambda = 0.998$, and $\beta = 0.3$.

We use two testing sets. The first testing set is a real data set used by the KDD CUP-99. It contains network intrusion detection stream data collected by the MIT Lincoln laboratory [1]. This data set contains a total of five clusters and each connection record contains 42 attributes. As in [1], all the 34 continuous attributes are used for clustering. In addition, we also use some synthetic data sets to test the scalability of D-Stream. The synthetic data sets have a varying base size from 30K to 85K, the number of clusters is set to 4, and the number of dimensions is in the range of 2 to 40. In the experiments below, we normalize all the attributes of the data sets to [0, 1]. Each dimension is evenly partitioned into multiple segments, each with length $len$. We will show results with different $len$.

### 5.1 Evolving data streams with many outliers

We find that the temporal order of data stream can make great effect on the clustering results. In order to validate the effectiveness of D-Stream, we generate some synthetic data sets with two different orders.

First, we randomly generate 30K 2-dimensional data with 4 clusters, including 5K outlier data that are scattered in the space. The distribution of the original data set is shown in Figure 6a. These clusters have nonconvex shapes and some are interwoven. We generate the data sequentially at each time step. At each time, any data point that has not been generated is equally likely to be picked as the new data record. Therefore, data points from different clusters and those outliers *alternately* appear in the data stream. The final test result by D-Stream is shown in Figure 6b. we set $len = 0.05$. From Figure 6b, we can see that the algorithm can discover the four clusters without user supply on the number of clusters. It is much more effective than the $k$-means algorithm used by CluStream since $k$-means will fail on such data sets with many outliers. We can also see that our scheme for detecting sporadic grids can effectively remove most outliers.

In the second test, we aim to show that D-Stream can capture the dynamic evolution of data clusters and can remove real outlier data during such an adaptive process. To this end, we order the four classes and generate them sequentially one by one. In this test, we generate 85K data points including 10K random outlier data. The data distribution is shown in Figure 6c. The speed of the data stream is 1K/second, which means that there are 1K input data points coming evenly in one second and the whole stream is processed in 85 seconds. We check the clustering results at three different times, including $t_1 = 25$, $t_2 = 55$, and $t_3 = 85$. The clustering results are shown from Figure 6d to 6f. It clearly illustrates that D-Stream can adapt timely to the dynamic evolution of stream data and is immune to the outliers.

### 5.2 Clustering quality comparison

We first compare DS0 and DS1 on a synthetic data set to see the effects of using grid attraction. The results are shown in Figure 7. We see that DS1 using attraction can generate better clusters. Without using attraction, the DS0 algorithm merges several clusters together since it only uses the density information but not attraction.

Next, we compare DS0, DS1, and CluStream on the synthetic data set in Figure 6 and KDD CUP-99 data set described above. We also test DS0 and DS1 under different grid granularity. The correct rates of clustering results at different times are shown in Figure 8a. In the figures, $len$ indicates the size of each partitioned segment in the normalized dimensions. For example, when $len = 0.02$, there are 50 segments in each dimension. From Figure 8a, the average correct rates on the synthetic data set by D-Stream is above 96.5%. We see that DS1 is significantly better than DS0 when $len = 0.5$, which shows the benefit of using grid attraction. When $len$ is larger, the grids have a coarser resolution and the benefit of using grid attraction is more salient.

Figure 8b shows results on the KDD CUP-99 data. The dataset contains around five million network connections. Each connection is labelled as either normal, or as an attack, with exactly one specific attack type out of four possible types. The precision is the ratio of correctly labelled connections. From Figure 8b, we see that the average correct rate of DS0 and DS1 on KDD CUP-99 is above 92.5%.

We also compare the qualities of the clustering results by D-Stream and those by CluS-

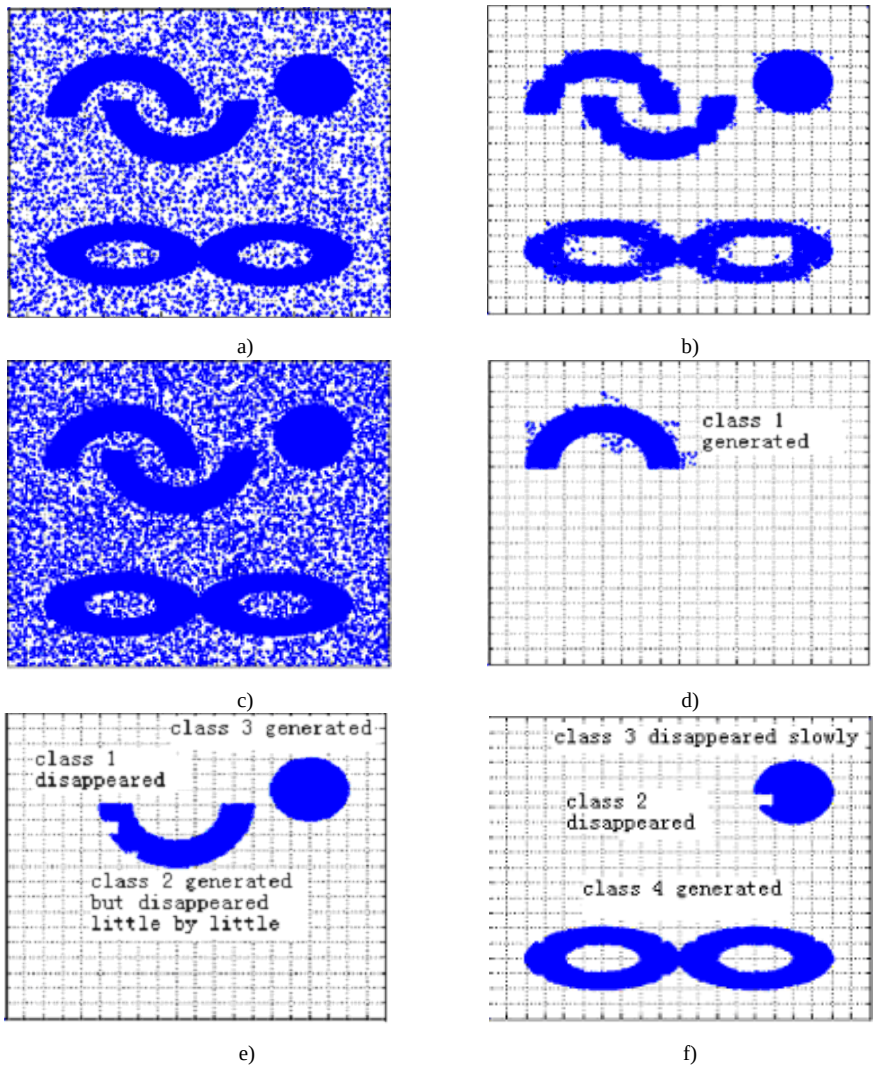
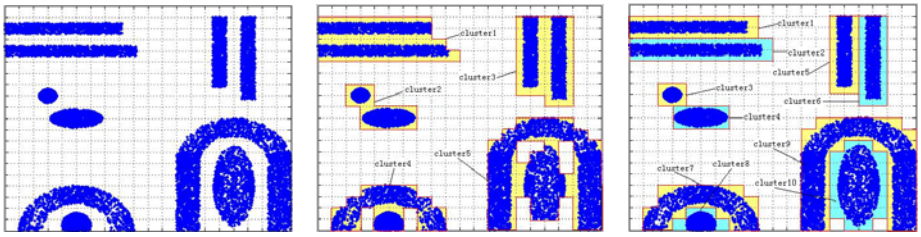



Fig. 6. a) Original distribution of the 30K data. b) Final clustering results on the 30K data. c) Original distribution of the 85K data. d) Clustering results at $t_1 = 25$ on the 85K data. e) Clustering results at $t_2 = 55$ on the 85K data. f) Clustering results at $t_3 = 85$ on the 85K data.

tream. Due to the non-convexity of the synthetic data sets, CluStream cannot get correct results on them. Thus, its quality cannot be compared to that of D-Stream. Therefore, we only compare the precision and the sum of squared distance (SSQ) of the two algorithms on the network intrusion data from KDD CUP-99. Figure 8b and Figure 8c show the results. We can see that the average quality of both DS0 and DS1 at various times are always better than that of CluStream. We also see that DS1 outperforms DS0, validating the effectiveness of using grid attraction in density-based clustering.




a) b) c)

Fig. 7. a) The distribution of the data. b) Clustering result of DS0. c) Clustering result of DS1.

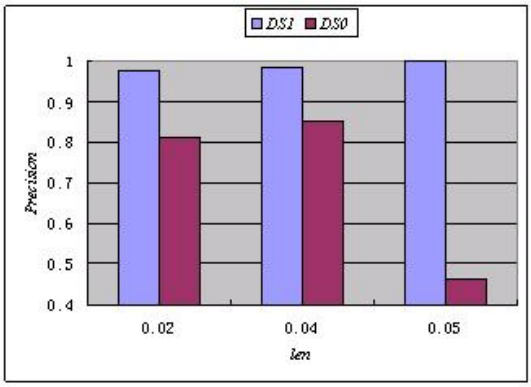
### 5.3 Time and memory performance comparison

We test and compare the clustering speed of D-Stream and CluStream. First, both algorithms are tested on the KDD CUP-99 data with different sizes. The time is the total clock time used by the algorithms to process all data. For D-Stream, the online task of processing data and the offline work of adjust_clustering() are performed in parallel, in a pipelined fashion. The results are shown in Figure 9a. We can see that the two versions of D-Stream, DS0 and DS1, have very close time efficiency. We also see that CluStream requires four to six times more clustering time than both DS0 and DS1. D-Stream is efficient since it only puts each new data record to the corresponding grid by the online component without computing distances as CluStream does. Furthermore, the dynamic detection and deletion of sporadic grids save much time. It can also be seen that D-Stream has better scalability since its clustering time grows slower with an increasing data size. We also the compare the memory used by D-Stream, with and without deleting the sporadic grids, in Figure 10. We see that, without removing sporadic grids, the memory usage of D-Stream grows very fast. When sporadic grids are used, the memory usages scales up very slowly.

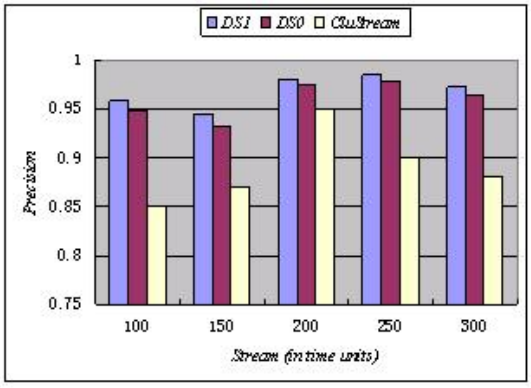
Next, all the algorithms are tested on the KDD CUP-99 data with different dimensionality. We set the size of data set as 100K and vary the dimensionality from 2 to 40. We list the time costs under different dimensionality by all the algorithms in Figure 9b. DS0 and DS1 are 3.5 to 11 times faster than CluStream and scale better.

## 6. CONCLUSIONS

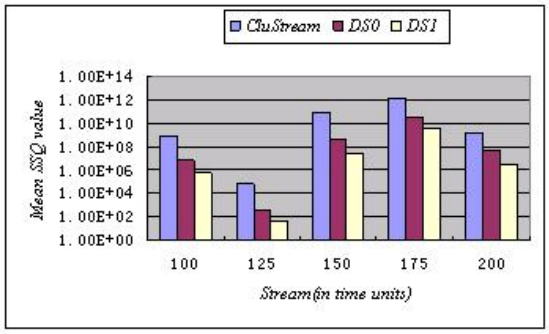
In this paper, we propose D-Stream, a new framework for clustering stream data. The algorithm maps each input data into a grid, computes the density of each grid, and clusters the grids using a density-based algorithm. In contrast to previous algorithms based on $k$-means, the proposed algorithm can find clusters of arbitrary shapes. The algorithm also proposes a density decaying scheme that can effectively adjust the clusters in real time and capture the evolving behaviors of the data stream. We introduce a new concept on the attraction between grids in order to improve the quality of density-based clustering. Further, a sophisticated and theoretically sound technique is developed to detect and remove the sporadic grids in order to dramatically improve the space and time efficiency without affecting the clustering results. The technique makes high-speed data stream clustering feasible without degrading the clustering quality.




a)




b)

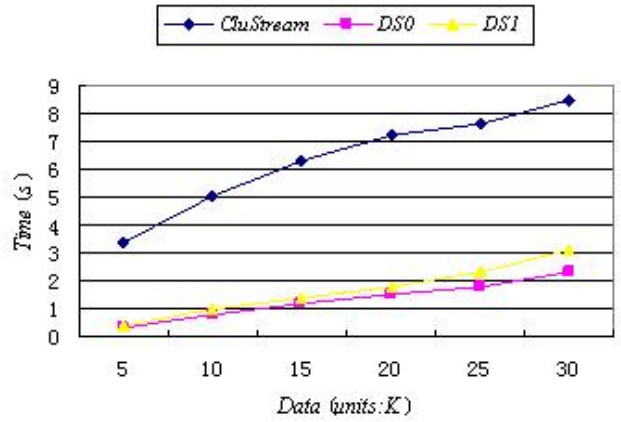



c)

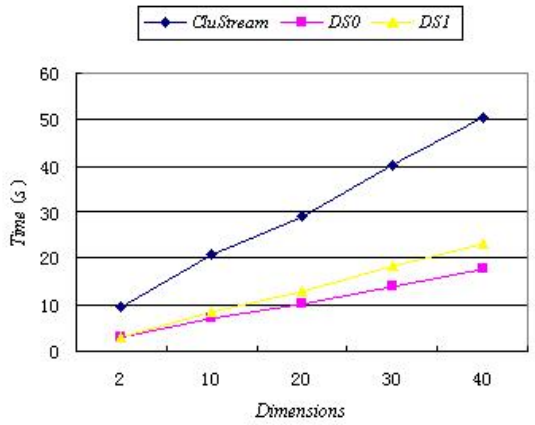
Fig. 8. a) Correct rates of D-Stream (DS0 and DS1) on synthetic data with different $len$. b) Correct rates of D-Stream and CluStream on KDD CUP-99 data. c) SSQ of D-Stream and CluStream on KDD CUP-99 data.

ACKNOWLEDGMENTS

This work is partly supported by a Microsoft Research New Faculty Fellowship, NSF grant IIS-0713109, Department of Energy grant ER25737, Chinese National Natural Science Foundation grant No. 60673060, and Natural Science Foundation of Jiangsu Province




a)




b)

Fig. 9. Efficiency comparison with a) varying sizes of data sets and b) varying dimensionality.

grant No. BK2008206.

## REFERENCES

C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *Proc. VLDB*, pages 81–92, 2003.

B. Babcock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining variance and k-medians over data stream windows. In *Proceedings of the twenty-second ACM symposium on Principles of database systems*, pages 234–243, 2003.

S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu, and S. Datta. Clustering distributed data streams in peer-to-peer environments. *Information Sciences*, 176(14):1952–1985, 2006.

D. Barbará. Requirements for clustering data streams. *SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.

J. Beringer and E. Hüllermeier. Online-clustering of parallel data streams. *Data and Knowledge Engineering*, 58(2):180–204, 2006.

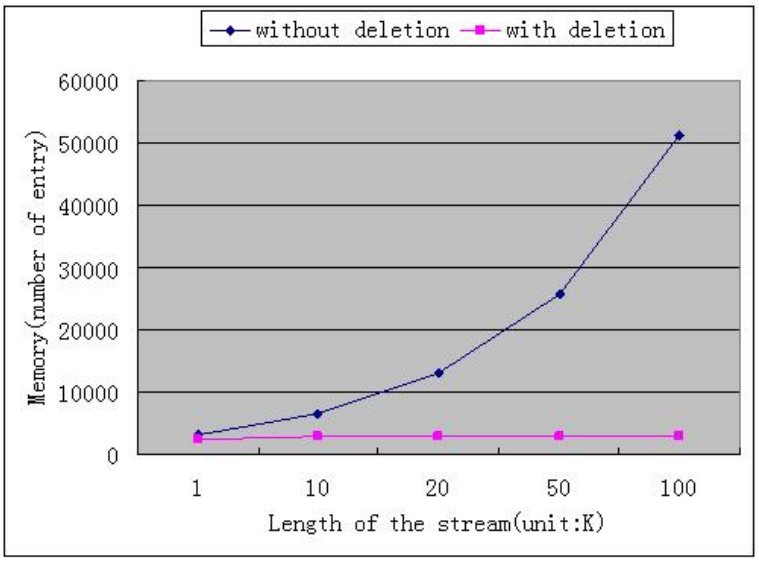



Fig. 10. Comparison of the memory requirement of D-Stream with and without the deletion of sporadic grids on the KDD CUP'99 data.

Y. Chen, G. Dong, J. Han, J. Pei, B. W. Wah, and J. Wang. Olaping stream data: Is it feasible? In *Proc. Workshop on Research Issues in Data Mining and Knowledge Discovery, ACM SIGMOD, (DMKD 02)*, pages 53–58, 2002.

Y. Chen and L. Tu. Density-Based Clustering for Real-Time Stream Data. *Proc. ACM SIGKDD*, 2007.

B.R. Dai, J.W. Huang, M.Y. Yeh, and M.S. Chen. Adapative clustering for multiple evolving streams. *IEEE Transaction On Knowledge and data engineering*, 18(9), 2006.

M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: you only get one look. In *Proc. ACM SIGMOD*, pages 635–635, 2002.

C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha, editors, *Next Generation Data Mining*, pages 191–212, 2003.

A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. VLDB*, pages 79–88, 2001.

L. Golab and M. T. Özsu. Issues in Data Stream Management. *SIGMOD Record*, 32(2):5–14, 2003.

S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *Trans. Know. Eng.*, 15(3):515–528, 2003.

S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Annual IEEE Symp. on Foundations of Comp. Sci.*, pages 359–366, 2000.

D. Lee and W. Lee. Finding maximal frequent itemsets over online data streams adaptively. In *Proc. ICDM*, pages 266–273, 2005.

O. Nasraoui, C. Rojas, and C. Cardona. A framework for mining evolving trends in web data streams using dynamic learning and retrospective validation. *Computer Networks*, 50(10):1488–1512, 2006.

L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proc. of 18th International Conference on Data Engineering*, pages 685–694, 2002.

S. Oh, J. Kang, Y. Byun, G. Park, and S. Byun. Intrusion detection based on on clustering a data stream. In *Third ACIS International Conference on Software Engineering Research, Management and Applications*, pages 220–227, 2005.

J. Sander, M. Ester, H. Kriegel, and X. Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Min. Knowl. Discov.*, 2(2):169–194, 1998.

S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online outlier detection in sensor data using non-parametric models. In *Proc. VLDB*, pages 187–198, 2006.

H. Sun, G. Yu, Y. Bao, F. Zhao, and D. Wang. S-tree: an effective index for clustering arbitrary shapes in data streams. In *Research Issues in Data Engineering: Stream Data Mining and Applications, 15th International Workshop on*, pages 81–88, 2005.

Z. Wang, B. Wang, C. Zhou, , and X. Xu. Clustering Data streams on the Two-tier structure. *Advanced Web Technologies and Applications*, pages 416–425, 2004.

H. Wang, J. Yin, J. Pei, P. S. Yu, and J. X. Xu. Suppressing model overfitting in mining concept-drifting data streams. *Proc. ACM SIGKDD*, 2006.

S. Zhong. Efficient stream text clustetring. *Neural Networks*, 18(6):790–798, 2006.

Received June 10, 2008; revised September 16, 2008; accepted December 9, 2008.