Density-Based Clustering for Real-Time Stream Data

Yixin Chen
Department of Computer Science and Engineering
Washington University in St. Louis
St. Louis, USA
chen@cse.wustl.edu

Li Tu
Institute of Information Science and Technology
Nanjing University of Aeronautics and Astronautics
litu@nuaa.edu.cn

ABSTRACT

Existing data-stream clustering algorithms such as CluStream are based on k-means. These clustering algorithms are incompetent to find clusters of arbitrary shapes and cannot 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. The algorithm uses an online component which maps each input data record into a grid and an offline component which 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. Exploiting the intricate relationships between the decay factor, 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 to 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

H.2.8 [**Database Management**]: Database Applications—*data mining*

General Terms

Algorithms, Experimentation, Performance, Theory

Keywords

Stream data mining, density-based clustering, D-Stream, sporadic grids

KDD'07, August 12–15, 2007, San Jose, California, USA.

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, planetary 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 [7, 8]. 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 to stream data clustering. Earlier clustering algorithms for data stream uses a single-phase model which treats data stream clustering as a continuous version of static data clustering [9]. 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 [10, 12]. 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 [17]. Extensions of this work including clustering multiple data streams [6], parallel data streams [5], and

distributed data steams [3], and applications of data stream mining [11, 16, 13].

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 stream. For this reason, the CluStream architecture uses an online processing which compresses raw data stream in 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 [14, 15]. 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 prior 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 two 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 with high dimensionality and improve scalability is a critical issue. Fortunately, in practice, most grids are empty or only contain few records and a memory-efficient technique for managing such a sparse grid space is developed in D-Stream.

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, 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 is better in terms of both clustering quality and efficiency and it exhibits high scalability for large-scale and high-dimensional stream data.

```
1.  procedure D-Stream
2.      t_c = 0;
3.      initialize an empty hash table 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
```

Figure 1: The overall process of D-Stream.

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 decay factor. In Section 4, we give the algorithmic details and theoretical analysis for D-Stream. We conduct experimental study of D-Stream and compare D-Stream to CluStream on real-world and synthetic data sets in Section 5 and conclude the paper in Section 6.

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 $0, 1, 2, \cdots, n, \cdots$. 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, place the multi-dimensional data into a corresponding discretized *density grid* in the multi-dimensional space, and update 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).

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 2.

3.1 Basic definitions

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

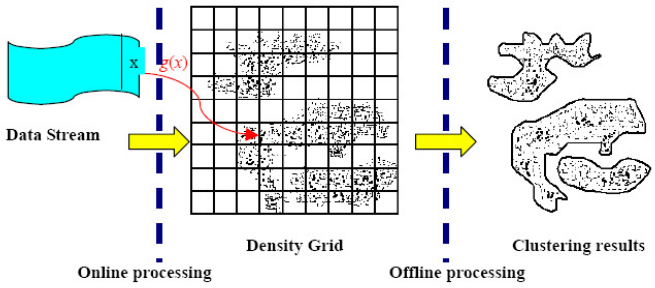

Figure 2: Illustration of the use of density grid.

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, $i = 1, \cdots, d$ is 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

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

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

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

For each data record x, we assign it a **density coefficient** which decreases with 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

$$D(x,t) = \lambda^{t-T(x)} = \lambda^{t-t_c}, \tag{4}$$

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

Definition 3.1. **(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 mapped 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.

Proposition 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:

$$D(g,t_n) = \lambda^{t_n - t_l} D(g,t_l) + 1 \tag{5}$$

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

$$D(g,t_l) = \sum_{i=1}^{m} D(x_i,t_l). \tag{6}$$

According to (4), we have that:

$$\begin{aligned} D(x_i,t_n) &= \lambda^{t_n - T(x_i)} = \lambda^{t_n - t_l}\lambda^{t_l - T(x_i)} \\ &= \lambda^{t_n - t_l} D(x_i,t_l), \text{ for } i = 1, \cdots, m. \end{aligned} \tag{7}$$

Therefore, we have:

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

■

Proposition 3.1 saves huge amount of computation time. To update all grids at each time step requires $\Theta(N)$ computing time for density update at each time step. In contrast, using Proposition 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, Proposition 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 3.2. **(Characteristic Vector)** The characteristic vector of a grid g is a tuple $(t_g, t_m, D, label, status)$, 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), D is the grid density at the last update, *label* is the class label of the grid, and *status* = {SPORADIC, NORMAL} is a label used for removing sporadic grids.

3.2 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.

Proposition 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 $t = 1, 2, \cdots$;

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 $0, 1, \cdots, t$, 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:

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

■

Proposition 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

$$D(g,t) \geq \frac{C_m}{N(1-\lambda)} = D_m, \tag{8}$$

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

$$D(g,t) \leq \frac{C_l}{N(1-\lambda)} = D_l, \tag{9}$$

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

$$\frac{C_l}{N(1-\lambda)} \leq D(g,t) \leq \frac{C_m}{N(1-\lambda)}. \tag{10}$$

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

Definition 3.3. **(Neighboring Grids)** Consider two density grids $g_1 = (j_1^1, j_2^1, \cdots, j_d^1)$ and $g_2 = (j_1^2, j_2^2, \cdots, j_d^2)$, if there exists k, $1 \leq k \leq d$, such that:
1) $j_i^1 = j_i^2, i = 1, \cdots, k-1, k+1, \cdots, d$; 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 3.4. **(Grid Group)** A set of density grids $G = (g_1, \cdots, g_m)$ is a grid group if for any two grids $g_i, g_j \in G$, there exist a sequence of grids $g_{k_1}, \cdots, g_{k_l}$ such that $g_{k_1} = g_i$, $g_{k_l} = g_j$, and $g_{k_1} \sim g_{k_2}$, $g_{k_2} \sim g_{k_3}$, $\cdots$, and $g_{k_{l-1}} \sim g_{k_l}$.

Definition 3.5. **(Inside and Outside Grids)** Consider a grid group G and a grid $g \in G$, suppose $g = (j_1, \cdots, j_d)$, if g has neighboring grids in every dimension $i = 1, \cdots, d$, then g is an inside grid in G. Otherwise g is an outside grid in G.

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

Definition 3.6. **(Grid Cluster)** Let $G = (g_1, \cdots, g_m)$ be a grid group, if every inside grid of G is a dense grid and every outside grid is either a dense grid or a transitional grid, then G is a grid cluster.

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.

4. COMPONENTS OF D-STREAM

We now describe in detail the key components of D-Stream outline in Figure 1. As we have discuss in the last section, for each new data record x, we map it to a grid g and use (5) 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 and grid. A dense grid may degenerate to a transitional or sparse grid if it does not receive no 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 minimum of these two minimum times in order to ensure that the inspection is frequent enough to detect the density changes of any grid.

Proposition 4.1. For any dense grid g, the minimum time needed for g to become a sparse grid from being a dense grid is

$$\delta_0 = \left\lfloor \log_\lambda \left(\frac{C_l}{C_m} \right) \right\rfloor. \tag{11}$$

Proof. According to (8), if at time t, a grid g is a dense grid, then we have:

$$D(g,t) \geq D_m = \frac{C_m}{N(1-\lambda)}. \tag{12}$$

Suppose after δ_t time, g becomes a sparse grid, then we have:

$$D(g, t+\delta_t) \leq D_l = \frac{C_l}{N(1-\lambda)}. \tag{13}$$

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:

$$\begin{aligned} D(g, t+\delta_t) &= \sum_{x \in E(g, t+\delta_t)} D(x, t+\delta_t) \\ &\geq \sum_{x \in E(g,t)} D(x, t+\delta_t) \\ &= \sum_{x \in E(g,t)} \lambda^{\delta_t} D(x,t) = \lambda^{\delta_t} D(g,t) \end{aligned} \tag{14}$$

Combining (13) and (14) we get:

$$\lambda^{\delta_t} D(g,t) \leq D(g, t+\delta_t) \leq \frac{C_l}{N(1-\lambda)} \tag{15}$$

Combining (12) and (15) we get:

$$\lambda^{\delta_t} \frac{C_m}{N(1-\lambda)} \leq \lambda^{\delta_t} D(g,t) \leq \frac{C_l}{N(1-\lambda)} \tag{16}$$

which yields:

$$\delta_t \geq \log_\lambda \left(\frac{C_l}{C_m} \right) \tag{17}$$

■

PROPOSITION 4.2. For any sparse grid g, the minimum time needed for g to become a dense grid from being a sparse grid is

$$\delta_1 = \left\lfloor \log_\lambda \left(\frac{N - C_m}{N - C_l} \right) \right\rfloor. \quad (18)$$

Proof. According to (9), if at time t, a grid g is a sparse grid, then we have:

$$D(g,t) \leq D_l = \frac{C_l}{N(1-\lambda)}. \quad (19)$$

Suppose after δ_t time, g becomes a dense grid, then we have:

$$D(g,t+\delta_t) \geq D_m = \frac{C_m}{N(1-\lambda)}. \quad (20)$$

We also know that:

$$D(g,t+\delta_t) = \sum_{x \in E(g,t+\delta_t)} D(x,t+\delta_t) \quad (21)$$

$E(g,t+\delta_t)$ 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:

$$\begin{aligned} D(g,t+\delta_t) &\leq \sum_{x \in E(g,t)} D(x,t+\delta_t) + \sum_{i=0}^{\delta_t - 1} \lambda^i \\ &= \sum_{x \in E(g,t)} \lambda^{\delta_t} D(x,t) + \frac{1-\lambda^{\delta_t}}{1-\lambda} \\ &= \lambda^{\delta_t} D(g,t) + \frac{1-\lambda^{\delta_t}}{1-\lambda} \end{aligned} \quad (22)$$

Now we plug (20) and (19) into (22) to obtain:

$$\begin{aligned} \frac{C_m}{N(1-\lambda)} &\leq D(g,t+\delta_t) \leq \lambda^{\delta_t} D(g,t) + \frac{1-\lambda^{\delta_t}}{1-\lambda} \\ &\leq \frac{\lambda^{\delta_t} C_l}{N(1-\lambda)} + \frac{1-\lambda^{\delta_t}}{1-\lambda} \end{aligned} \quad (23)$$

Solving (23) yields:

$$\lambda^{\delta_t} \leq \frac{N - C_m}{N - C_l}, \quad (24)$$

which results in:

$$\delta_t \geq \log_\lambda \left(\frac{N - C_m}{N - C_l} \right). \quad (25)$$

Note $N - C_m > 0$ since $C_m < N$ according to (8). ■

Based on the two propositions 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 recognized. Thus, in D-Stream we set:

$$\begin{aligned} gap &= \min\{\delta_0, \delta_1\} \\ &= \min\left\{ \left\lfloor \log_\lambda \frac{C_l}{C_m} \right\rfloor, \left\lfloor \log_\lambda \frac{N - C_m}{N - C_l} \right\rfloor \right\} \\ &= \left\lfloor \log_\lambda \left(\max\left\{ \frac{C_l}{C_m}, \frac{N - C_m}{N - C_l} \right\} \right) \right\rfloor \end{aligned} \quad (26)$$

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, sporadic grids accumulate and their number can become exceedingly large, causing the system to operate more and more slowly. 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.

Sparse grid with $D \leq D_l$ are candidates for sporadic grids. However, there are two reasons for the density of a grid to be less than D_l. The first cause is that it has received very few data, while the second cause is that the grid has previously received many data but the density is reduced by the effect of decay factor. Only the grids in the former case are true sporadic grids that we aim to remove. The sparse grids in the latter case should not be removed since they contain many data records and are often upgraded to transitional or dense grids. We have found through extensive experimentation that wrongly removing these grids in the latter case can significantly deteriorate the clustering quality.

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

$$\pi(t_g,t) = \frac{C_l}{N} \sum_{i=0}^{t-t_g} \lambda^i = \frac{C_l(1-\lambda^{t-t_g+1})}{N(1-\lambda)} \quad (27)$$

PROPOSITION 4.3. There are the following properties of the function $\pi(t_g,t)$.

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

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

(2) If $t_1 \leq t_2$, then $\pi(t_1,t) \geq \pi(t_2,t)$ 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 all 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:

(S1) $D(g,t) < \pi(t_g,t)$; and

(S2) $t \geq (1+\beta)t_m$ if g has been delete before (at time t_m), where $\beta > 0$ is a constant.

Note that t_m and t_g are stored in the characteristic vector.

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 hash table using doubly-linked lists to resolve collision. The hash table allows for fast lookup, update, and deletion. The key of the hash table are the grid coordinates, while the associated data for each grid entry is the characteristic vector.

We use the following rules to delete sporadic grids from *grid_list*.

(D1) During the periodic inspection in Line 13 of Figure 1, all grids satisfying (S1) and (S2) are marked as SPORADIC but wait until the next periodic inspection to be considered for deletion.

(D2) In the next periodic inspection, if a grid g marked as SPORADIC has not received any data since last inspection, we remove g from *grid_list*. Otherwise, check if g satisfies (S1) and (S2): if yes, we keep g marked as SPORADIC but do not remove it; otherwise, we reset the label to NORMAL.

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 for clustering, 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 not 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 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 strong theoretical properties of the $\pi(t_g,t)$ 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.

PROPOSITION 4.4. 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), $\cdots$, $(t_{m-1}+1,t_m)$, then the density value $D(g,t_i), i = 1..m$ satisfies (let $t_0 = -1$):

$$D(g,t_i) < \pi(t_{i-1}+1,t_i). \tag{28}$$

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

$$\begin{aligned}D_a(g,t) &= \sum_{i=1}^{m} D(g,t_i)\lambda^{t-t_i} + D(g,t) \\ &< \sum_{i=1}^{m}\pi(t_{i-1}+1,t_i)\lambda^{t-t_i} + \pi(t_g,t) \qquad (29)\end{aligned}$$

Since $t_g \geq t_m + 1$, by property (2) in Proposition 4.3, we know

$$\begin{aligned}D_a(g,t) &< \sum_{i=1}^{m}\pi(t_{i-1}+1,t_i)\lambda^{t-t_i} + \pi(t_m+1,t) \\ &= \sum_{i=1}^{m-1}\pi(t_{i-1}+1,t_i)\lambda^{t-t_i} + \pi(t_{m-1}+1,t) \\ &= \sum_{i=1}^{m-2}\pi(t_{i-1}+1,t_i)\lambda^{t-t_i} + \pi(t_{m-2}+1,t) \\ &\cdots \\ &= \pi(0,t) = \frac{C_l(1-\lambda^{t+1})}{N(1-\lambda)} < D_l. \qquad (30)\end{aligned}$$

The last equalities are based on successive applications of property (1) in Proposition 4.3. ■

Proposition 4.4 is important since it shows that deleting a sporadic grid will not cause transitional or dense grid 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 occured, it is still sporadic and cannot be a transitional or dense grid since $D_a(g,t) < D_l$.

PROPOSITION 4.5. Suppose the density of a grid g at time t is $D(g,t)$, and g receives no data from $t+1$ to $t+gap$, then there exist $t_0 > 0$ and $t_1 > 0$ such that:

(a) If $D(g,t) < D_l$, then $D_a(g,t+gap) < D_l$, for $t \geq t_0$.

(b) If $D(g,t) < D_m$, then $D_a(g,t+gap) < D_m$, for $t \geq t_1$.

Proof. We prove (a). (b) can be proved similarly. Suppose the grid g has been previously deleted for the periods of $(0, t_1)$, (t_1+1, t_2), $\cdots$, $(t_{m-1}+1, t_m)$, then:

$$D_a(g, t+gap) = \sum_{i=1}^{m} D(g, t_i)\lambda^{t-t_i+gap} + D(g, t+gap) \quad (31)$$

Since we assume that g receives no data from $t+1$ to $t+gap$,

$$\begin{aligned} D_a(g, t+gap) &= \sum_{i=1}^{m} D(g, t_i)\lambda^{t-t_i+gap} + D(g, t)\lambda^{gap} \\ &< \sum_{i=1}^{m} \pi(t_{i-1}+1, t_i)\lambda^{t-t_i+gap} + D_l\lambda^{gap} \\ &= \pi(0, t_m)\lambda^{t-t_m}\lambda^{gap} + D_l\lambda^{gap} \\ \text{(according to (S2))} \quad &< \pi(0, t_m)\lambda^{\beta t/(1+\beta)}\lambda^{gap} + D_l\lambda^{gap} \end{aligned}$$

In order to ensure $D_a(g, t+gap) < D_l$, we require:

$$\pi(0, t_m)\lambda^{\beta t/(1+\beta)}\lambda^{gap} + D_l\lambda^{gap} < D_l$$

$$\Rightarrow \lambda^{\beta t/(1+\beta)} < \frac{(1-\lambda^{gap})D_l}{\lambda^{gap}\pi(0, t_m)} = \frac{1-\lambda^{gap}}{\lambda^{gap}(1-\lambda^{t_m+1})}$$

Thus, (a) is true for t_0 satisfying:

$$t_0 > \left(\frac{1+\beta}{\beta}\right)\log_\lambda\left(\frac{1-\lambda^{gap}}{\lambda^{gap}(1-\lambda^{t_m+1})}\right)$$

■

Proposition 4.5 is a key result showing that (S1), (S2), (D1) and (D2) work together correctly. It implies that, as time extends for long enough, we will never delete a potential transitional or dense grid due to the previous removals of data. If a grid is sparse (*resp.* not dense), then when it is deleted, it must be sparse (*resp.* not dense) even considering those deleted data. Note that $D_a(g, t+gap)$ is the density of the grid upon deletion assuming no previous deletion has ever occurred. The result shows that, after an initial phase, deleting sporadic grids does not affect the clustering results.

4.3 Clustering algorithms

We describe the algorithms for generating the initial cluster and for adjusting the clusters every *gap* steps. The procedure initial_clustering (used in Line 10 of Figure 1) is illustrated in Figure 3. The procedure adjust_clustering (used in Line 14 of Figure 1) is illustrated in Figure 4. They first update the density of all active grids to the current time. Once the density of grids are determined at the given time, the clustering procedure is similar to the standard method used by density-based clustering.

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 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 (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 (h is transitional) label h as in c;
13.     until no change in the cluster labels can be made
14. end_procedure
```

Figure 3: The procedure for initial clustering.

```
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, 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 clusters of g, find the largest one
                c' satisfying that g is an outside grid if added to it;
21.             label g as in c';
22.     end for
23. end_procedure
```

Figure 4: The procedure for dynamically adjusting clusters.

5. EXPERIMENTAL RESULTS

We evaluate the quality and efficiency of D-Stream and compare it with CluStream [1]. 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*.

5.1 Evolving data streams with many outliers

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

First, we randomly generate 30K 2-dimensional data set in 4 clusters, including 5K outlier data that are scattered in the space. The distribution of the original data set is shown in Figure 5. 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 6. we set $len = 0.05$. From Figure 6, 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.

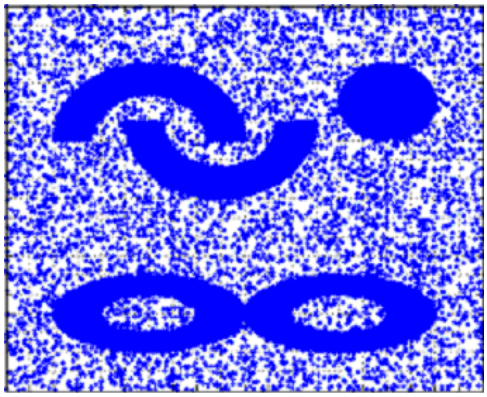

Figure 5: Original distribution of the 30K data.

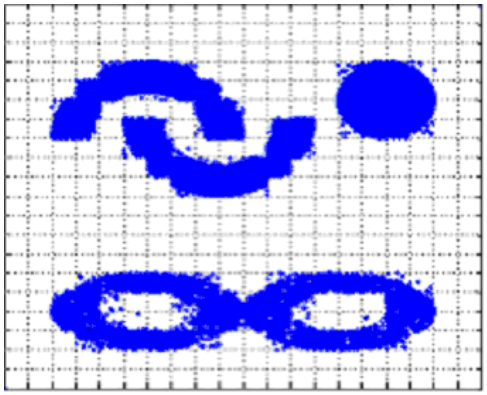

Figure 6: Final clustering results on the 30K data.

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 7. 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 8 to 10. It clearly illustrates that D-Stream can adapt timely to the dynamic evolution of stream data and is immune to the outliers.

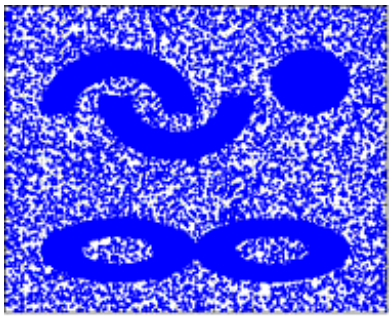

Figure 7: Original distribution of the 85K data.

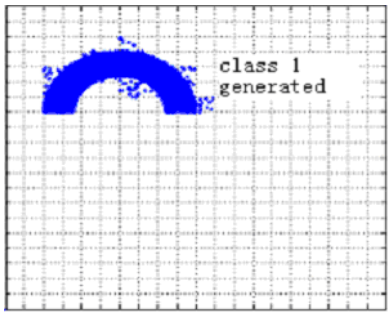

Figure 8: Clustering results at $t_1 = 25$.

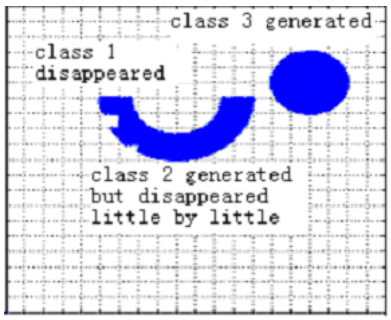

Figure 9: Clustering results at $t_2 = 55$.

5.2 Clustering quality comparison

We test D-Stream on the synthetic data set and KDD CUP-99 data set described above under different grid granularity. The correct rates of clustering results at different times are shown in Figure 11 and 12. In the figures, *len*

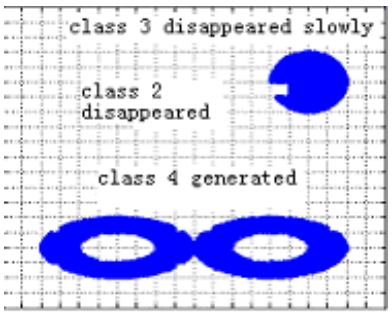

Figure 10: Clustering results at $t_3 = 85$.

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 11, the average correct rates on the synthetic data set by D-Stream is above 96.5%. From Figure 12, the average correct rate on KDD CUP-99 is above 92.5%.

We also compare the qualities of the clustering results by D-Stream and those by CluStream. Due to the non-convexity of the synthetic data sets, CluStream can not get a correct result. Thus, its quality can not be compared to that of D-Stream. Therefore, we only compare the sum of squared distance (SSQ) of the two algorithms on the network intrusion data from KDD CUP-99. Figure 13 shows the results. We can see that the average SSQ values of D-Stream at various times are always less than those of CluStream, which implies that data in each of the cluster obtained by D-Stream are much more similar than that obtained by CluStream.

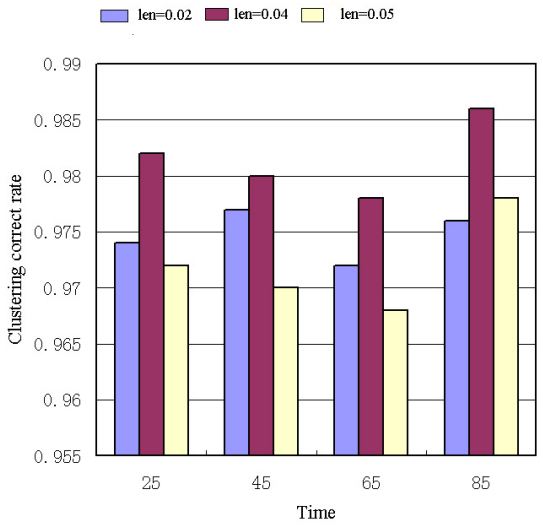

Figure 11: Correct rates of D-Stream on synthetic data.

5.3 Time performance comparison

We test and compare the clustering speed of D-Stream and CluStream. First, both algorithms are tested on the KDD

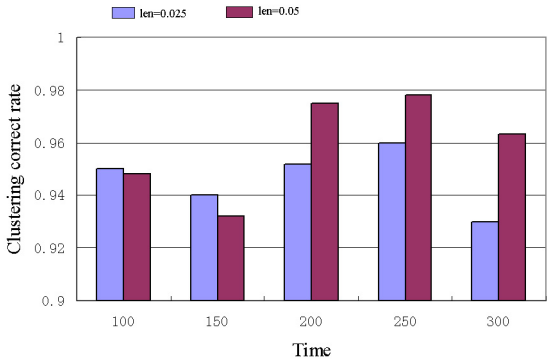

Figure 12: Correct rates of D-Stream on KDD CUP-99 data.

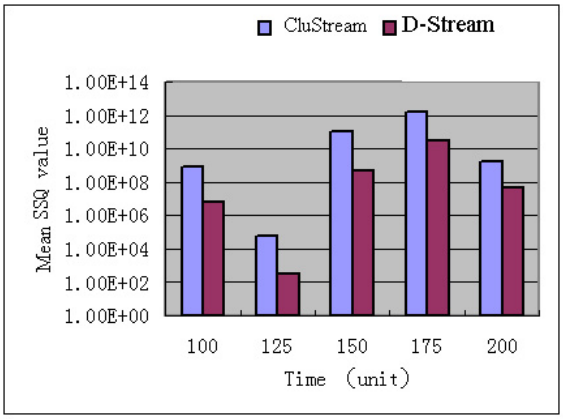

Figure 13: Comparison of D-Stream and CluStream on KDD CUP-99 data.

CUP-99 data with different sizes. The results are shown in Figure 14. We can see that CluStream requires four to six times more clustering time than D-Stream. 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 tremendous time. It can also be seen that D-Stream has better scalability since its clustering time grows slower with an increasing data size.

Next, both 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 the two algorithms in Figure 15. D-Stream is 3.5 to 11 times faster than CluStream and scales better. For example, when the dimensionality is increased from 2 to 40, the time of D-Stream only increases by 15 seconds while the time of CluStream increases by 40 seconds.

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

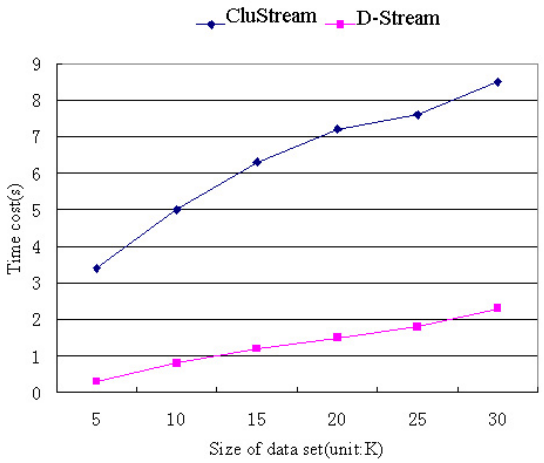

Figure 14: Efficiency comparison with varying sizes of data sets.

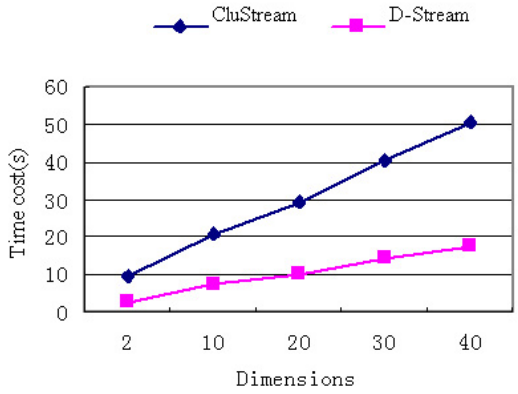

Figure 15: Efficiency comparison with varying dimensionality.

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. 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.

7. ACKNOWLEDGEMENT

This work is supported by Microsoft Research New Faculty Fellowship and National Natural Science Foundation of China Grant 60673060.

8. REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

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

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.