Grid-based Data Stream Clustering for Intrusion Detection

Qian Quan, Chao-Jie Xiao, and Rui Zhang
(Corresponding author: Qian Quan)

School of Computer Engineering & Science, Shanghai University, Shanghai, 200072 China
State Key Laboratory of Information Security (Institute of Software, Chinese Academy of Sciences)
(Email: qqian@shu.edu.cn)

(Received Mar. 11, 2011; revised and accepted July 13, 2011)

Abstract

As a kind of stream data mining method, stream clustering has great potentiality in areas such as network traffic analysis, intrusion detection, etc. This paper proposes a novel grid-based clustering algorithm for stream data, which has both advantages of grid mapping and DBSCAN algorithm. The algorithm adopts the two-phase model and in the online phase, it maps stream data into a grid and the geometric center of all the data in the grid is used to represent the characteristic of entire data in the grid approximately. In the offline phase, grid-based DBSCAN clustering algorithm is used to cluster all grids in the space based on density. Meanwhile, extension of the algorithm to an incremental one is also presented in detail in the paper. The algorithm proposed in the paper can solve the problem that it is difficult to find neighbor grids in DStream algorithm and also solve the incompetency of DBSCAN in data compression, which makes it capable for DBSCAN to be used for stream data. Experimental results on KDDCUP99 intrusion detection dataset show that the algorithm can achieve a good clustering quality and efficiency. The average accuracy is above 92% and the highest order of magnitude of SSQ is 104 and the average processing time of 10,000 sessions is about 3 seconds.

Keywords: Grid-based Clustering, Stream data mining, DBSCAN

1 Introduction

Stream data mining is currently a hot research, which has great potential demands on network traffic analysis, telecommunication, planetary remote sensing, website analysis, etc. Clustering stream data is a very difficult task because we have a large volume of stream data and these data should be processed in real-time. Also the data can be processed only once, and once data flow away they cannot be processed any more. In 2003, Barbará summarized requirements for clustering stream data and made a summary for some algorithms which may be applied to clustering stream data [2]. He proposed that clustering stream data should satisfy three requirements: (1) data compression and expression of the compressed data; (2) processing new data point in a fast and incremental way; (3) distinguishing outliers quickly and clearly.

Currently, influential clustering algorithms for stream data are: Clustream [1], Dstream [5], DenStream [3], P-Stream [6] etc. P-Stream is a probability-based clustering algorithm for stream data. Denstream is a density-based clustering algorithm for stream data. An excellent framework for clustering stream data is proposed in Clustream [1], which uses a two-phase scheme that consists of online phase and offline phase. Many recent clustering algorithms in different extent refer to the two-phase framework proposed by Clustream. The online phase collect, preprocess and compress stream data. The offline phase includes clustering stream data that collected in the online phase. Assigning clustering job to the offline phase can raise real time processing capability of the online phase. At the same time, Clustream proposed Characteristic Feature (CF) vector which is used to compress data. CF vector consists of first-order geometric center and second-order geometric center of a cluster. The method is very useful when the clustering algorithm is based on k-means algorithm, but the method cannot be used in clustering data of arbitrary shapes because the geometric center of a cluster with irregular shape cannot be determined.

Dstream algorithm retains the two-phase framework of Clustream and proposes a framework for clustering stream data using a density-based approach which is directed at the deficiencies in the use of k-means algorithm in the Clustream. Dstream proposes a clustering algorithm that is based on grid mapping and neighbor searching. But during the experiment we find that the number of grid is too large after grid mapping, which leads to a huge volume of calculation. There exists the problem that the distribution of grids is too sporadic to find a neighboring grid for a given grid, which leads to the failure of clustering. A further explanation using the method of probability analysis about this problem will be given in the latter paper.

DBSCAN is an excellent density-based clustering algorithm [7, 8]. It can be used to find clusters of arbitrary shapes, which solves the problem of using k-means algorithm in the Clustream. But the limitation of DBSCAN lies in that all the data point should be saved for global clustering. It is impossible to save all the data points of the stream data due to the memory constraint. Since DBSCAN algorithm cannot compress data, it means that DBSCAN algorithm cannot be used in clustering stream data. However,

Dstream proposes the method bases on grid mapping that can convert stream data to the density of grid, which solves the problem of data compression.

This paper proposes a grid-based DBSCAN clustering algorithm for stream data. The algorithm combines the advantages of DBSCAN and grid mapping. Using grid mapping can solve the problem of data compression of DBSCAN; meanwhile using DBSCAN can overcome the difficulty in finding neighbor grids in the high dimensional space when using Dstream. The method will continue to use two-phase model proposed by Clustream, dividing process into online phase and offline-phase. During online phase, we adopt grid mapping and each grid will store density and CF vector. And during the offline phase, we use grid-based DBSCAN to cluster all the grids in the space.

The rest of the paper is organized as follows: Section 2describes grid mapping and CF vector; Section 3 presents a brief introduction of DBSCAN algorithm; Section 4 extends DBSCAN to grid-based DBSCAN for clustering stream data; the experiment and result analysis is in Section 5; Section 6 summarizes the whole paper and presents some directions of the future work.

2 Grid Mapping and CF Vector

The paper adopts grid mapping technology during online phase [3]. At first we divide *d* dimensional space, and each dimension of d dimensional space is partitioned into *p* segments and the length of each segment is *len*. So each dimension is composed as follow:

$$S = S_1 \cup S_2 \cup .... \cup S_p \qquad (1)$$

Where $p = \left\lceil \frac{1}{len} \right\rceil$. So we can know that the number of total grids is p^d . If $len = 0.04$ then each dimension is partitioned into $\frac{1}{0.04} = 25$ segments and if *d*=37 then the number of total grids will reach $25^{37} = 5.3e + 51$. So from the above calculation we can know that the number of grids will rise exponentially with the increase of *p* and *d*. In the grid space, the probability of two grids being neighbor is $\frac{2*d}{N*N}$, where *d* is dimension of space and *N* is number of total grids in the space. When $N = 25^{37}$, the probability is almost zero, which is the main reason why it is hard to find neighbor grid for a given grid in the DStream algorithm.

The method of mapping a *d* dimensional data $\mathrm{x} = (\mathrm{x}_1, x_2, x_3, ..., x_d)$ into a grid is: for each x_i computing coordinate in the corresponding dimension and by calculating coordinate for each x_i , a unique grid $\mathrm{g} = (\mathrm{g}_1, g_2, g_3, ..., g_d)$ can be determined. If a *d* dimensional data is mapped into a grid *g*, we say that the d dimension data belongs to the grid *g*.

For each grid, it will store density and a CF vector. The density of the grid can be calculated by adding up density factor of all data in the grid. For a data point x, its density factor can be defined as:

$$DF(x) = \lambda^{t_c - t} \qquad (2)$$

Where λ is decay factor(less than 1), t_c is the current time, *t* is the arriving time of data point *x*. As decay factor commonly is less than 1, we can know from the formula that density factor of each data point will be decreased with change of time and finally it will be near zero. Density of a grid can be calculated by the following equation:

$$\text{Density(g)} = \sum^{m} DF(x) = \sum_{i=1}^{m} \lambda^{t_c - t_i} \qquad (3)$$

Where t_c is current time, t_i is the arriving time of data point *x*. we can know from the Equation (3) that density of a grid is changing constantly with the change of time.

If a new data point is mapped into a grid, it is not necessary to add up density factor of all data points in the grid again. The density can be calculated incrementally by the following formula (4), and the details of the proof can be referred to paper [5].

$$Density^{i+1} = Density^{i} * \lambda^{\Delta t} + 1 \qquad (4)$$

In Equation (4), $\Delta t = t_{i+1} - t_i$ is the time difference between current time and the last updating time, where $t_{\mathrm{i+1}}$ is the current time and $\mathrm{t_i}$ the last updating time.

According to the size of density of grid, grids can be classified into 3 categories: dense grid, transitional grid and sparse grid. Detailed classification method can be referred to paper [5].

To save data points mapped into grids as completely as possible, at the same time to compress data as much as possible, each grid will store a CF vector [5,12]. A CF vector consists of the first-order geometric center $CF1^x$ and the second-order geometric center $CF2^x$ of all the data points in the grids, which are two d dimensional vectors. $CF1^x$ and $CF2^x$ can be calculated using Equation (5) and (6):

$$CF1^x = \frac{\sum_{i=1}^{n} x_i \lambda^{(t_c - t_i)}}{n} \qquad (5)$$

$$CF2^x = \frac{\sum_{i=1}^{n} x_i^2 \lambda^{(t_c - t_i)}}{n} \qquad (6)$$

In Equations (5) and (6), t_c is the current time and t_i is the arriving time of data point *x*. If length of each segment of each dimension in the space is 0.04 and *d* = 37, the volume of a grid is 0.04^{37}=1.9e-52. So we believe that the volume of a grid is very small and we can use the first-order geometric center of all the data in the grid to approximately represent the characteristic of entire data in the grid. The paper uses first-order geometric center $CF1^x$ to represent a grid. The benefit of introducing $CF1^x$ is that it can not only solve the problem of stream data compression, but also it can comparatively completely save information of all the data in the gird. First-order geometric center $CF1^x$ is used in

the grid-based DBSCAN algorithm to calculate distance between two grids. Similar to calculation of density, if a new data point is mapped into a grid it is not necessary to add up all data points in the grid to calculate geometric center again, because it can be calculated incrementally using the Equations (7) and (8):

$$CF1^x = \frac{CF1^x * n * \lambda^{\Delta t} + X_i}{n+1} \qquad (7)$$

$$CF2^x = \frac{CF2^x * n * \lambda^{\Delta t} + X_i^2}{n+1} \qquad (8)$$

Where $\Delta t = t_{i+1} - t_i$ is the time difference between current time and the last updating time, where t_{i+1} is current time and t_i is last updating time.

Proof. Suppose last updating time is t_1 and at time t_1 $CF1^x$ is

$$\mathrm{CF1^x}(t_1) = \frac{\sum_{j=1}^{n} x_j \lambda^{(t_l - t_j)}}{n}$$

At time t_c $CF1^x$ is $\mathrm{CF1^x}(t_c)$ and,

$$CF1^x(t_c) = \frac{\sum_{j=1}^{n} x_j \lambda^{(t_c - t_j)} + X_i}{n+1} = \frac{\sum_{j=1}^{n} x_j \lambda^{(t_l - t_j)} * \lambda^{(t_c - t_l)} + X_i}{n+1}$$

$$= \frac{CF1^x(t_l) * n * \lambda^{(t_c - t_l)} + X_i}{n+1} = \frac{CF1^x(t_l) * n * \lambda^{\Delta t} + X_i}{n+1}$$

The proof method of calculation formula of $CF2^x$ is similar to that of $CF1^x$. Storing second-order geometric center CF2x is for calculation of *SSQ* (sum of square distance) in the experiment.

3 DBSCAN Algorithm

DBSCAN is an excellent and very effective density-based clustering algorithm [7,8]. Its core idea is to classify all the data points into two categories: core points and border points. If the number of points distributed within the radius of a point is not less than a threshold (*minPts*), the point can be considered as core point. All other points which are not core points are border points. The working principle of DBSCAN is firstly to search for a core point *P* which does not belong to any cluster and then construct a new cluster where point *P* is treated as the center. All the points distributed within the radius of point *P* will be added to the cluster of point *P*, and then these points will be checked one by one if it is a core point. If it is a core point (we call it *P'*), all the points which is not clustered distributed within the radius of point *P'* will be added to the cluster of *P*. Expansion operation will be repeated until no new core point can be found.

The idea of DBSCAN clustering algorithm is: if two data points are density-reachable then they belong to the same cluster. Since DBSCAN algorithm finds clusters through expanding core points and there is no need to determine geometric center, DBSCAN can be used in clustering data of arbitrary shape.

In this paper we choose DBSCAN as clustering algorithm because DBSCAN has advantage that it can cluster all the points distributed within the radius of the core point and the expansion scope is a smooth circle, while clustering using neighbor searching will form a jagged graphic. It is because neighbor searching will neglect the grids located in the diagonal, which stops clustering. For example, it can be known from Figure 1 that grid *B* and grid *C* are all neighbors of grid *A*. In DStream two d dimensional grids are neighbor only if *d-1* dimensions of the two grids are same and the remaining one dimension of the two grids has a difference of 1 or a difference of -1. If we search for neighbors according to the method of Dstream, grid *C* will be missed because there are two dimensions out of *d* dimensions that are different. If we search for neighbors using DBSCAN grid *B* and *C* will be added to the cluster of grid *A* because DBSCAN will seek all the grids distributed within the radius of grid *A*, whose expansion scope is a circle.

C	B	C
B	A	B
C	B	C

Figure 1: Clustering diagram of 9 grids

The core idea of grid-based DBSCAN algorithm is to consider grid as a data point in the DBSCAN algorithm, because the *CF* vector stored in the grid can represent the information of all the data of the grid. If a grid *G* is a dense grid and the number of grids distributed within the radius of grid *G* is not less than a threshold (*minPts*), *G* can be considered as a core point and be expanded outwards for clustering.

4 Grid based DBSCAN Clustering Algorithm for Stream Data

Grid-based DBSCAN clustering algorithm for stream data consists of two parts: online processing and offline processing [2]. The online part is responsible for data collection, data preprocess and grid mapping. The offline part does clustering using grid-based DBSCAN algorithm. The offline part can be called by user or by program. For example, offline operation will be called at regular intervals to adjust clustering. The offline part is divided into initial phase and incremental clustering phase. The initial phase clusters all the data collected initially to build initial cluster model. The incremental phase incrementally clusters grids whose status is changed. Clustering all the grids is not necessary, while only those grids distributed within the radius of the grid whose status is changed should be adjusted, which can speed up processing. Meanwhile, in the offline phase sparse grids could be deleted for reducing the number of grids to accelerate neighbor searching.

Framework for grid-based DBSCAN algorithm for stream data is as follows:

Algorithm 1: Framework of the grid-based DBSCAN

```
1: Begin
2: tc=0;
3: While (new stream data flows in)
4:    Collect a window size d dimensional data;
5:    Preprocess each record within the time window;
6:    Mapping each data into grid;
7:    If grid does not exist then
8:          Create a new grid;
      Else
9:          Map data into the grid and update density and CF
            vector of the grid;
10:   end if
11:   If （tc == gap） then //initial clustering
12:      Call grid-based DBSCAN algorithm in the initial
         phase for initial clustering;
13:   Else if (tc % gap == 0) //reach predefined time
                              //interval
14:      Call grid-based incremental DBSCAN algorithm
         in the offline phase for incremental clustering;
15:   Endif
16:   tc = tc + 1;
17:   end while
18: End
```

4.1 Online Processing Phase

(1) ***Data Collection***. Divide stream data by time window. Each time a block of d dimensional data within a time window will be collected. The form of each *d* dimensional data is $x=(x_1, x_2, x_3,...., x_d)$.

(2) ***Data Preprocessing***. All the attributes of each data in the window need to be preprocessed. Preprocessing can be divided into two steps: standardization and normalization. Standardization is to prevent that the value of some dimension is so large that it could affect calculation. The method of standardization is:

$$x_A' = \frac{x_A - \overline{A}}{\sigma_A} \qquad (9)$$

In Equation (9), x_A is the original data, $\overline{A}$ is the mean value of attribute A, σ_A is the standard deviation of attribute *A*. The method of normalization is described as Equation (10), which max-min normalizes attributes of all data sets to [0, 1]. In Equation (10), *X* is the value of attribute after normalized.

$$X = \frac{x_A' - \min(A)}{\max(A) - \min(A)} \qquad (10)$$

For discrete attribute computation requirement, encoding is adopted. Different value of the discrete attribute will be mapped into integer values ranging from 1 to *S*, where *S* is the number of different values of the discrete attribute. For example, there are 3 different values: *TCP*, *UDP* and *ICMP*, for network protocol attribute, so after encoding we use *TCP*=1, *UDP* =2, *ICMP* =3.

(3) ***Grid Mapping.*** Each attribute of *d* dimensional data $x=(x_1, x_2, x_3,...., x_d)$ is normalized to [0,1], and after this grid mapping goes.

If each dimension of d dimensional space is divided into *p* segments, whose length is *len*, and *len*p* =1. For each x_i, we can get $g_i = \left\lfloor \frac{x_i}{len} \right\rfloor$. For example, when *len* = 0.04, $x_i = 0.6$, we get $g_i = \left\lfloor \frac{0.6}{0.04} \right\rfloor = 15$, which means that x_i is mapped into the 15th segment of dimension *i*. Through this method, a *d*-dimensional data can be mapped into a unique *d*-dimensional grid $g=(g_1, g_2, g_3,...., g_d)$.

After determining the coordinate of grid, we can try to read the grid information. If the grid does not exist, then a new grid should be created and data will be added into the grid; if the grid does exist then data will be added into the grid directly.

(4) ***Updating Grid Information.*** Update density and CF vector of the grid which has new data mapped into with the calculation Equations (4), (7) and (8).

4.2 Offline Processing Phase

The offline phase consists of two parts: initial phase and incremental adjusting phase. Grid-based DBSCAN algorithm is used in the initial phase and grid-based incremental DBSCAN algorithm is used in the incremental adjusting phase. Incremental algorithm will be introduced in the latter paper. A grid in the space is considered as a point in the DBSCAN algorithm and distance between two grids can be calculated using Equation (11).

$$\text{Distance} = \sqrt{\sum_{i=1}^{d} (CF1^x_{1i} - CF1^x_{2i})^2} \qquad (11)$$

The Equation (11) shows that the value of Distance is the square root of the sum of difference between the squares of each dimension of $CF1^x$.

4.2.1 Initial Phase

First of all, update density of all grids. The reason of updating density of grids in the offline phase is that there are many grids that do not have new data mapped into in the online phase, which may cause these grids not being updated in the online phase.

Then, treat all the dense grids which do not belong to any cluster as core points in the DBSCAN algorithm and begin initial clustering. The clustering method can be referred to DBSCAN algorithm; however the following adjustment will be made: dense grids are treated as core points and all the grids distributed within the radius of the dense grid will be found. The processing on these grids found is as follows:

1) If the grid is a dense grid then treat the grid as a core point and continue to expand it outwards. Expansion rule is to add all the dense and transitional grids within the radius of the expanded grid to the current cluster.

2) If the grid is a transitional grid, then just add it to the current cluster and do not go on expanding outwards.

3) If the grid is a sparse grid, then do nothing. Sparse grids will be deleted in the latter processing.

4.2.2 Incremental Adjustment Phase

As stream data is continuously increasing, a fast incremental clustering mechanism should be built to satisfy the requirement of clustering stream data in real time. Grid-based incremental *DBSCAN* algorithm is proposed here. Algorithm detail is as follows:

Algorithm 2: Incremental grid-based DBSCAN

1: Begin
2: ***Step1:*** Delete the sparse grids without new data points mapped into in the last phase.
3: ***Step2:*** Update density and CF vector of all the grids according to the Equations (4) (7) (8).
4: ***Step3:*** For each grid G whose status is changed:
5: ***Step3.1:*** **If** G is dense grid and G is clustered **then**:
6: ***Step3.1.1:*** **If** there are grids within the radius of G **then**:
7: **If** there exists dense grid G' within the radius of G and G' is clustered **then** merge the cluster of G and G'. The principle of merging is that the small cluster will be merged into the large one.
8: **If** there does not exist dense grid within the radius of G but there exists transitional grid G' and G' is not clustered, **then** add G' to the cluster of G.
9: ***Step3.2:*** **If** G is a dense grid and G is not clustered **then**:
10: **If** there exists dense grid G' within the radius of G and G' belongs to the cluster C, and **then** add G to cluster C.
11: **If** the number of grids within the radius of G is less than ***minPts***, **then** treat G as NOISE.
12: **If** the number of grids within the radius of G is not less than ***minPts***, **then** create a new cluster using G as a core point.
13: ***Step3.3:*** **If** G is a transitional grid and G is not clustered **then**:
14: **If** the number of grids within the radius of G is less than ***minPts***, **then** treat G as NOISE.
15: If there exists dense grid G' within the radius of G and G is clustered then add G to the cluster of G'.
16: ***Step 3.4:*** If G is a transitional grid and G is clustered **then**,
17: **if** there exists dense grid G' and the cluster of G' is different from the cluster of G, **then** add G to the cluster of G'.
18: ***Step 3.5:*** If G is a sparse grid, then delete G.
19: End

In ***Step1***, the sparse grids are not useful during the clustering and if there are too many sparse grids they will slow down the speed of grid searching, so deleting the sparse grid which is not updated in a certain period of time can speed up processing and improve the efficiency of the algorithm.

In ***Step3.5***, the impact of deleting G is that if G belongs to Cluster C then we should check whether the cluster C will be split if G is deleted. The method is, for all the dense grids within the radius of G, if there exist a dense grid which is not density-connected to all the other dense grids within the radius of G, then it means that cluster C is split in the G' and a new cluster should be created from the G'.

5 Experiment Results

The experiments evaluate the quality and efficiency of the Grid-based DBSCAN clustering algorithm for stream data proposed in this section. All the experiments are conducted on a PC with 2.8GHz CPU and 2GB memory running red hat Linux 5. We have implemented the grid-based DBSCAN clustering algorithm in C++. In experiment, the parameters are: *cm* = 3.0, *cl* = 0.8, *lemda*= 0.998, *beta* = 0.3, *eps* = 1.0 *minPts* = 3. All the parameters can be referred to [5, 7, 8].

The testing dataset used in experiments is KDD CUP-99. It contains network intrusion detection stream data collected by the MIT Lincoln laboratory [10]. KDD CUP-99 10% dataset is used in the experiments. The dataset contains 494021 records which can be classified into 5 categories: NORMAL, DOS, R2L, U2R and PROBING. Among them, DOS contains *smurf*, *neptune*, etc. R2L contains *warezclient* , *guess_passwd*, etc. U2R contains *buffer_overflow* , *perl*, etc. PROBING contains *satan*, *ipsweep*, etc. [10]. Each record in the KDD CUP-99 dataset contains 41 attributes which have 3 different forms: discrete, continuous, and symbolic. All 34 continuous attributes and 3 symbolic attributes (PROTOCOL, SERVICE, and FLAG) will be used in the experiment. All the attributes in the experiments below will be normalized into [0, 1]. Each dimension is partitioned into several segments, each with length *len*.

In the experiment through simulating stream data we send KDD CUP-99 dataset continuously to the receiving end where the program is running and test the algorithm proposed in the paper. In the experiments, the speed of sending data is 10 thousand records per time unit and the speed can be adjusted to simulate different network environment. In all the figures shown in the following paper, time (unit) in the *x* axis means the time it takes to send 10 thousand records.

In the experiments we test the clustering quality and accuracy rate of the algorithm proposed in the paper. At the same time, we will compare the clustering quality and accuracy in the circumstances that *len* and *gap* has different values.

Quality of clustering can be measured using SSQ and purity. SSQ is the sum square of the distance between each point in the cluster and the center of the cluster. SSQ is used to measure concentration of a cluster and the lower the SSQ, the higher the concentration of the cluster. SSQ calculation is:

$$SSQ = \frac{\sum_{j=1}^{K}\sum_{i=1}^{N}\left|(x_{ji} - \overline{x_j})\right|^2}{K} \qquad (12)$$

In equation (12), x_{ji} is the i^{th} data point of the j^{th} cluster. $\overline{x_j}$ is the center of the j^{th} cluster. The average SSQ can be calculated by sum the SSQ of each cluster and divided by the number of clusters.

Purity is an indicator used to measure the accuracy of a cluster. Each record of KDD CUP-99 10% dataset has been labeled correctly. We can compare the clustering result with the corrected label to calculate the purity of the cluster. The purity calculation formula is:

$$purity = \frac{\sum_{i=1}^{K}\left|\frac{c_i^d}{c_i}\right|}{K} * 100\% \qquad (13)$$

Where k is the number of clusters, $\left|c_i^d\right|$ denotes the number of points with the corrected label in cluster i. $\left|c_i\right|$ denotes the number of points in cluster ***i***.

5.1 Clustering Quality Analysis

We evaluate the quality of the grid-based DBSCAN algorithm proposed in the paper. We can know from Figures 2, 3, and 4 that the algorithm can reach a satisfactory result. Average accuracy is more than 92% and average SSQ can be maintained at a relatively low value, but the number of clusters is comparatively large, which needs to be improved. Meanwhile, we compare our result with the results published in the Dstream [5]. Dstream algorithm is also tested on the KDD CUP99 dataset and from Figures 5 and 6, we can see that the accuracy can reach more than 92.5% and SSQ is between 10e+02 and 10e+10.

So we know that the algorithm proposed in the paper produces similar performance in accuracy and a better average SSQ. As a result, the clustering quality produced by the algorithm is similar to that of Dstream.

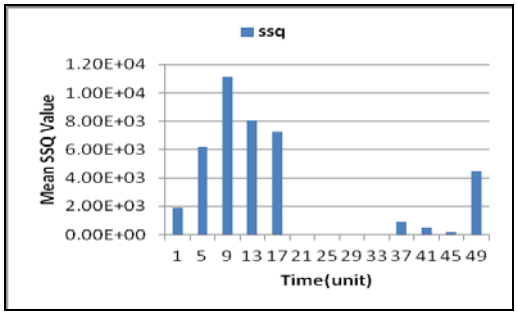

Figure 2：Clustering quality

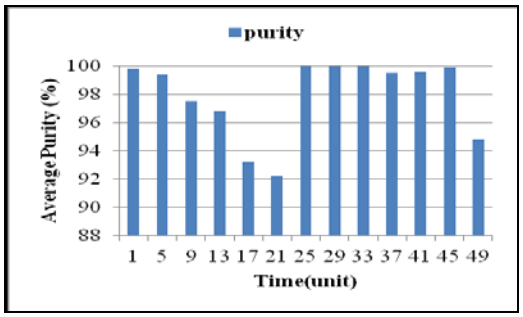

Figure 3：Clustering purity

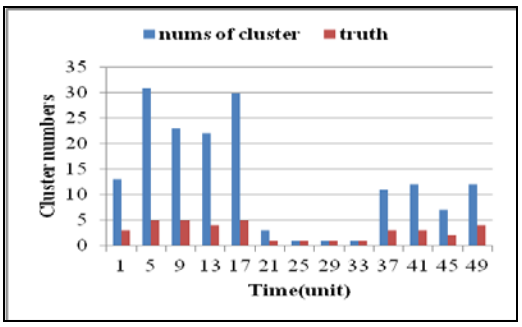

Figure 4: Number of clusters

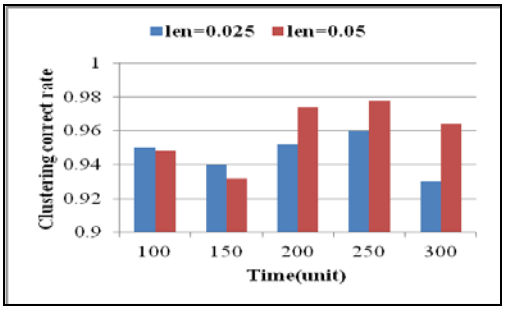

Figure 5：DStream clustering accuracy

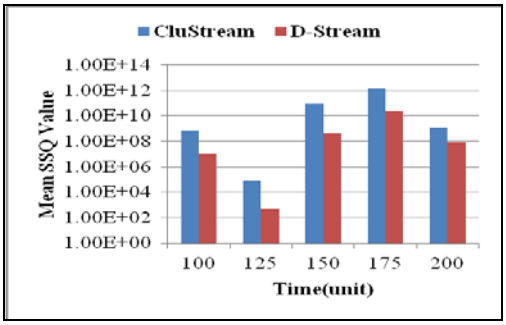

Figure 6：Dstream average SSQ

5.2 Clustering Quality and Efficiency of different Len

We test different *len* to evaluate clustering quality and efficiency. The different *len* is 0.02, 0.04 and 0.05 respectively. We can conclude from Figures 7 and 8 that when *len*=0.02 average SSQ and purity are relatively good at various times.

The less the *len* is, the more precise the grid partition is. Figure 9 is a curve of grid number with different *len*. When *len*=0.02, grid number is at various times greater than that when *len*=0.04 and *len*=0.05. So it comes that the number of grids depends on the *len*.

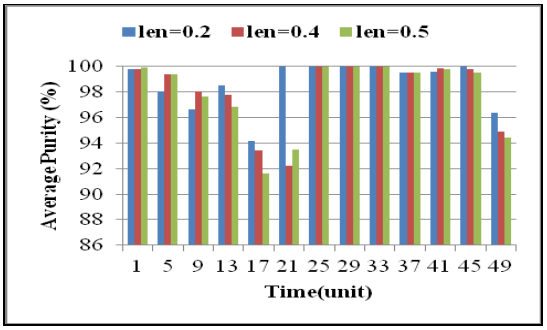

Figure 7: Average purity

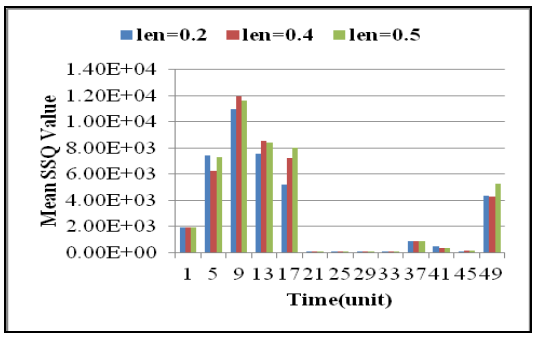

Figure 8：Average SSQ

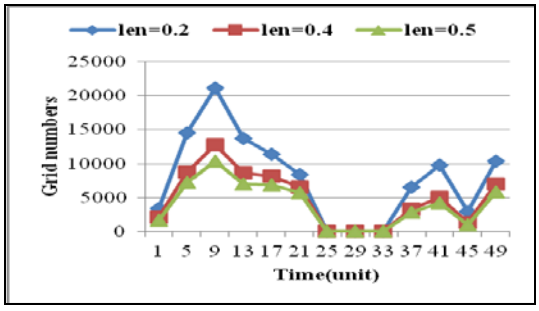

Figure 9：Number of grids with different *len*

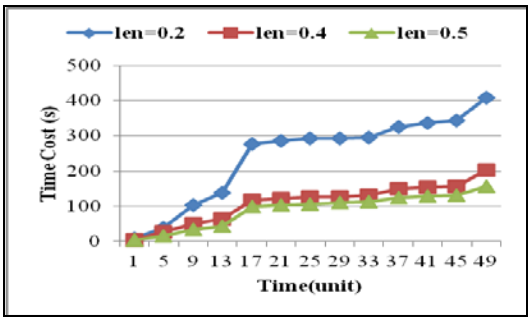

Figure 10：Efficiency with different *len*

We will further test different *len* to evaluate clustering efficiency. From Figure 10, we know that average processing time of 10,000 sessions is 3 seconds when *len*=0.05. The less the *len* is, the more time the clustering consumes. The result is consistent with the one we get on the grid number. As the less the *len* is, the more precise the grid partition is, the number of grids will increase, which results in more time consumed by the clustering.

5.3 Clustering Quality and Efficiency of different Gap

We will test different *gap* to evaluate clustering quality

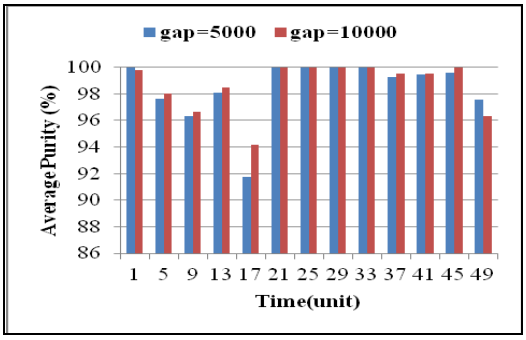

Figure 11：Purity with different gap

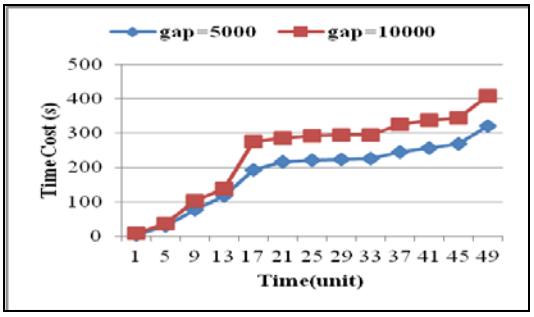

Figure 12：Efficiency with different gap

and efficiency. From Figure 11 we can know that the purity of *gap*=5000 is similar to that of *gap*=10000, but from Figure 12 it shows that the clustering efficiency has more than 20% promotion when reducing *gap* from 10000 to 5000.

We can conclude from the above experiments that the grid-based DBSCAN clustering algorithm for stream data can produce comparative good performance in clustering quality and efficiency.

6 Conclusion and Future Work

The paper proposes a grid-based DBSCAN clustering algorithm for stream data. Stream data can be rapidly compressed with grid mapping in the online phase, and the geometric center of all the data in the grid is used to approximately represent the characteristic of entire data in the grid. A grid is treated as a data point in the space and we use grid-based DBSCAN clustering algorithm for stream data to do clustering. Experiment is tested on the KDD CUP-99 dataset and we can conclude that the algorithm proposed in the paper can produce a satisfactory clustering quality. Average SSQ can be maintained in a relatively low value and the highest order of magnitude is

10^4 and average purity of clustering is above 92% and the average processing time of 10,000 sessions is 3 seconds.

As for future work, firstly, we will compare continually the advantage and disadvantage of the algorithm proposed in the paper with those of other novel stream clustering algorithm to get move objective evaluations. Secondly, although the algorithm proposed in the paper can achieve a very high purity and accuracy with relatively low mean SSQ, but the disadvantage is the number of clusters is relatively large. So the next work is to solve the problem, for example, to optimize the parameters in algorithm. Finally, facing the high bandwidth network, we should accelerate the speed of clustering further to adapt to the high speed stream data. And we will conduct research on multi-grid based parallel clustering algorithm and evaluate the performance of the algorithm in the real network environment.

Acknowledgments

This work is supported by Shanghai Leading Academic Discipline Project(J50103), the Innovation Project of Shanghai Municipal Education Committee (09YZ05), Doctoral Fund of Ministry of Education of China for Youth Teachers (20093108120016), National Natural Science Foundation of China(61003248).

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. "A framework for clustering evolving data streams, VLDB Conference Proceedings, vol.29, pp. 81-92, 2003.

[2] D. Barbará. "Requirements for clustering data streams," ACM SIGKDD Explorations Newsletter, vol. 3, no. 2, pp. 23-27, 2003.

[3] F. Cao, M. Ester, W. Qian, and A. Zhou. "Density-based clustering over an evolving data stream with noise," Proceedings of the 6th SIAM Conference on Data Mining, Bethesda, MD, USA, pp. 328-339, 2006.

[4] F. Cao and A. Y. Zhou. "Fast clustering of data streams using graphics processors," Journal of Software, vol.18, no.2, pp.291-302, 2007.

[5] Y. X. Chen and L. Tu. "Density-based clustering for real-time stream data," Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.133-142, California, USA, 2007.

[6] D. B. Dai, G. Zhao, and S. L. Sun. "Effective clustering algorithm for probabilistic data stream," Journal of Software, vol. 20, no. 5, pp. 1313-1328, 2009.

[7] M. Ester, H. P. Kriegel, J. Sander, and X. Xu. "A density-based algorithm for discovering clusters in large spatial databases with noise," Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, pp. 226-231, Portland, Oregon, AAAI Press,1996.

[8] M. Ester, H.P. Kriegel, J. Sander and X. Xu. "Incremental clustering for mining in a data warehousing environment," Proceedings of the 24th Internatioal Conference on Very Large Databases, pp. 323-333, ACM Press, 1998.

[9] M. Sheikhan and Z. Jadidi. "Misuse detection using hybrid of association rule Mining and connectionist modeling," World Applied Sciences Journal, vol.7, pp. 31-37, 2009.

[10] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. "A detailed analysis of the KDD CUP 99 data set," Proceedings of 2009 IEEE Symposium on Computational Intelligence in Security and Defense Applications, IEEE Press, pp. 53-58, 2009.

[11] T. Zhang, R. Ramakrishnan and M. Livny. "BIRCH: An efficient data clustering method for very large databases," ACM SIGMOD, vol. 25, no. 2, pp. 103-114, 1996.

[12] W. H. Zhu, J. Yin and Y. H. Xie. "Arbitrary shape cluster algorithm for clustering data stream," Journal of Software, vol.17, no.3, pp.379-387, 2006.

Qian Quan is an associate professor in Shanghai University, China. His main research interests concerns computer network and network security, especially in cloud computing, IoT and wide scale distributed network environments. He received his computer science Ph.D. degree from University of Science and Technology of China (USTC) in 2003 and conducted postdoc research in USTC from 2003 to 2005. After that, he joined Shanghai University and now he is the lab director of network and multimedia.

Chao-Jie Xiao received BS degree in computer science from Shanghai University in 2009. He is currently working toward a master degree in the school of computer science, Shanghai University. His research interests include computer and network security, data mining.

Rui Zhang received her B.E. and Ph.D. degree from Department of Electronic Engineering & Information Science, University of Science and Technology of China, in 2003 and 2008, respectively. After graduation, she works in School of Computer Engineering and Science, Shanghai University. Her main research interests include computer networks, network coding for wireless networks and wireless communication, etc.