IT 11 011
Examensarbete 30 hp
Mars 2011

An Algorithm for Streaming Clustering

Jiaowei Tang

Institutionen för informationsteknologi
Department of Information Technology

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

An Algorithm for Streaming Clustering

Jiaowei Tang

Abstract
A simple existing data stream clustering algorithm DenStream based on DBScan is studied. Based on DenStream a modified algorithm called DenStream2 is proposed. It follows most of the framework and theory of DenStream. Denstream2 is implemented as a foreign function in an extensible data stream management system (DSMS), where queries over streams are allowed. The generated clusters inferred from each window of an input a data stream are emitted as new stream clusters. The output stream can be stored in database for later queries, or be queried directly.
Keywords: DBScan, DenStream

Handledare: Kjell Orsborn
Ämnesgranskare: Tore Risch
Examinator: Anders Jansson
IT 11 011
Tryckt av: Reprocentralen ITC

Contents

1. Introduction

A data stream can be seen as a sequence of time-stamped data records. The difference between data streams and static data stored in regular databases is that data records in streams are always time stamp ordered. Another difference is that a data stream may be very large or continuously evolve and therefore streaming data has to be processed in one pass. The data records in a stream could be any kind of data from applications. Examples of data streams are time series data produced by applications in different areas such as calling and communication records in the telecommunication and network area, stock exchange and credit card transaction records in the business area, accident records in city traffic systems, weather records in climate research, and so on.

Data stream mining is a way to extract knowledge from streams and find the change or evolution of a stream over time. A number of summarization methods, like sampling, sketching, load shedding, synopsis data structures, histograms, aggregation, damped window, and sliding window, can be used to transform streaming data into specific forms for further data analysis. Further analysis techniques to mine streaming data can be classification, clustering, frequent pattern counting, and so on [9], which are variations of similar regular data mining techniques. However, most of the traditional data mining methods, like K-means [23], hierarchical clustering [17], DBSCAN [7], and Affinity Propagation [18] developed for static data can't be used directly to streaming data in view of its big or indefinite data size [8] and requirement for single pass evolutionary processing [1]. However, new stream mining methods are developed based on the traditional data mining methods to address the new requirements by data stream mining.

This project investigates cluster detection from data streams, i.e. *stream clustering*. Because of the huge size and evolving property of data streams, a good stream clustering algorithm should meet the following requirements [1,15,4,13,3,10,2]:

I. Single scan of data. This is a natural constraint of streaming data because of its very large or infinite size. There is no time or may even be impossible to reread the stream for the computation [1].

II. Low memory and CPU usage. The system should be able to process very large or infinite streams in main memory of limited size.

III. No prior knowledge about clusters, including numbers and shapes. In clustering, the user often knows very little about how data is clustered before processing. Additionally, with streaming data the clusters changes as the processing progresses over time.

IV. Ability to discover arbitrary shaped clusters without prior knowledge. Not all clustering methods have this ability. In static data clustering, the DBSCAN algorithm [7] and the soft-constraint affinity propagation method [11] can determine clusters with arbitrary shape. These algorithms have been generalized for stream clustering [4].

V. Ability to filter out noise in continuously evolving streams. Random noise can occur anywhere in the data stream and filtering the noise can help getting a good clustering result.

VI. Discover and explore clusters in data *windows*. The data stream is always evolving over time and the clustering algorithm should maintain a limited number of clusters over a recent relevant section (window) of the stream in order to constrain memory usage.

VII. Compactness of representation of clusters. It is not possible to store the whole clustered data stream in the database as with traditional data clustering methods [2], since the volume of data is huge or infinite. A compact memory-limited representation of clusters is needed, which does not only have the ability to show the current state of the data stream over time but also uses limited space despite the huge or unlimited data stream being mined.

Above are some basic requirements to a scalable stream clustering algorithm. Actually, there are factors to take into account that are valid for data mining in general. One is the *objective function*, whose value is used as a measure of when clusters can be determined sufficiently precise. For example, the K-means algorithm [23], minimizes the sum of squares of the distances between objects within each cluster.

Other factors that could affect the clustering include the performance of the clustering algorithm, parameters of the algorithm, and the distribution of data over time. High-throughput streams will force the algorithm to use a lot of CPU resources to check and process the stream, and it may be a challenge for the algorithm to have sufficient performance. Optimizing the parameters in the algorithm is a way to get both good clustering and process the stream efficiently. The reason is that the data distribution in a stream, e.g. the data density, cluster shape, or other properties can affect the chosen parameter values. However, some parameters may be application dependent. An algorithm that needs not use application dependent parameters will be easier to use. The idea of parameter free stream clustering algorithms was proposed in [22]. However, the work in this thesis is based on using application parameters.

In this thesis, a stream mining algorithm called DenStream [4] is used as basic skeleton. To get better clustering, the original DenStream algorithm is modified in its online component [4], in particular the method to merge new arriving data points, the way to merge data points priority between potential core-microcluster and outlier core-microcluster, and the strategy to merge data points to microclusters between which there is an overlapping. The new algorithm is called *DenStream2*.

A prototype implementation of Denstream2 is made in the extended database management system Amos II [23]. Amos II can be used as a *Data Stream Management System (DSMS)* [28, 29], which is a system to make queries directly to streaming data. The callout interface of Amos II [21] provides a way to implement streamed foreign functions in some external programming language, such as Java or C. In the project foreign functions in Java were used to implement Denstream2. Foreign functions can return an indefinite stream since elements in the result are returned one-by-one iteratively using an *emit* method until the processing is stopped. The implementation is tested over synthetic or some publically available data streams.

The approach enables analyzes about streams by either stating queries directly on the stream of clusters returned by the foreign stream clustering function or by computing stream summaries and store them in the database.

2. Background and related work

This section goes through the technologies used in the thesis, namely stream mining and its relationship to databases.

2.1 Database and data stream management systems

A database is a collection of persistent data managed by a DataBase Management System (DBMS). Compared to a file system a DBMS provides several advantages, like low redundancy, high data consistency, high integrity, data security, transaction support, and so on [26]. To manage the data the database administrator must design a database *schema*, which is a description of how the data is stored in the DBMS. The most well known DBMSs are Oracle, Microsoft SQL Server, DB2, and MySQL [27].

In this project the prototype DBMS Amos II is the database management system used. Amos II is an extensible functional database system. It uses a functional query language AmosQL to express object-oriented and functional queries [24, 25]. Amos II can be used as a stand-alone main-memory DBMS. Foreign functions are defined by external subroutines in some regular programming language and is allowed to be called from Amos II queries. The foreign function implementing the DenStream2 algorithm for Amos II was written in Java. Foreign functions in Amos II allow implementing stream interfaces since the result of a foreign function is continuously returned as a stream through an *emit* primitive. The foreign function can be used in queries about the clustering result of a stream per window.

A data stream management system (DSMS) is a system to directly query streaming data. It is similar to a traditional DBMS, with a few differences. The first difference is that a DBMS handles explicitly stored data, while a DSMS handles streaming data. A second difference is that queries in a DBMS return a set of search result on demand for a given query, while a DSMS also may return streaming query results as a continuous data flow [28, 29].

In the implementation, the generated clusters from DenStream2 are emitted as a stream. Summaries of the streaming data can also be stored in the database if so desired. If the stream is of limited size it can be entirely stored in the database.

2.2 Stream clustering

A stream clustering program reads a stream of data where each record is a data point in format <time, data points>. The records are read in the order of time. After clustering, a stream of time stamped clusters will be emitted by the stream clustering algorithm also as a stream.

Most clustering methods use static data clustering methods. Recently much research has been done on stream clustering, and quite a few excellent methods were

suggested in [10]. Here I will simply discuss some of them and most are related to density-related clustering.

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) [12,16] and STREAM [13] are two predecessors of stream clustering. They were designed to cluster static time series data in a large database in four phases, but it can be used also for non-evolving data streams of limited size because of their ability to deal with large size data sets.

Clustream [1] is the first algorithm that supports the evolutionary property of data stream mining. It discovers clusters in stream windows. The algorithm is composed of an *online* and an *offline* part, and proposes idea of micro clustering and macro clustering combined into the two parts, respectively.

The known density-based clustering method for static data is DBSCAN [7], C-DBSCAN [14], GDD [20], and incremental DBSCAN [6]. They all can be extended to algorithms that mine streaming data. For example, DenStream [7] is a good model in density-based clustering, being an extension of DBSCAN. It inherits the framework of Clustream [1], including its online and offline components, and micro/macro clustering. rDenStream [19] , C-DenStream [18], D-Stream [5], and MR-Stream [15] are extensions of DenStream. Other methods in static data clustering are AP (affinity propagation) [18] and SCAP [11]. The stream mining version of AP is StrAP [17]. They are similar to density-based clustering methods and overcome the disadvantage of density-based algorithm that clustering is bad when data density changes sharply in different clusters.

In this project, a modified version of the stream clustering algorithm DenStream [4], *DenStream2*, is developed as the basic data stream clustering algorithm. DenStream2 closely follows the theories and concepts in DenStream (all details will be described in the next section). There are a number of window models in stream mining, like landmark window, sliding window, damped window, tilted-time window[30], and so on. Denstream (and Denstream2) is a *damped window model* based algorithm with an *online* and an *offline* component. In the online part, Denstream creates a number of *microclusters,* which have small orbicular shapes and represent groups of close data points, by merging data points. The microclusters are represented by its center coordinators, weight, and radius. As time goes on,

microclusters will evolve as the stream progresses. A *window* is an evolving time interval of data points and defined according to the need of the application. In the evolving, the center coordinators will be adjusted by merging new data points using the damped window model; the weight will also be changed at the same time, which is increased by merging new data points and decreased by the damped window model; the function of damped window model in this process is to address the importance of new data and lower the importance of old data. It is easy to see that it is impossible to search for all historical data points from the existing micro-clusters and the damped window model successively decreases the importance of old data points. What is more, the microclusters are classified into potential *core-microclusters* and *outlier microclusters* by a threshold for weight and the type of microclusters is exchangeable in the evolving. At the end of each window, the *offline* macro clustering component is started. In the offline component an algorithm similar to DBSCAN [7] further clusters the online potential core-microclusters to produce a group of *macroclusters* containing microclusters from the online component. Then a window representing a snapshot of clusters for a time interval in the stream is created.

After offline clustering, the macroclusters will be sent to a DSMS. However, the idea here is not to send the clusters directly but send the microclusters belonging to the clusters one by one to the DSMS along with associated features of each microcluster: the center and average radius of microclusters, the label of its offline macrocluster, and the time stamp of the window stream. Thus, in the DSMS, each window is regarded as a specific snapshot of their microclusters. The windows of generated clusters from the stream clustering algorithm are emitted at the end of each time period to the DSMS. This allows executing streaming queries over the windows without storing them in the database. If desired, if the stream is of limited size all clusters can be stored in the database, which may be useful for later analyses and further summarization of the data stream. However, it may be impossible to produce clusters for every time point of a continuously evolving stream. If so, the solution in this work is setting up a time interval, taking cluster snapshots in the end of every time interval, storing the clustered data for a time interval in a window, and finally sending the window to the DSMS. So, the size of the window controls the frequency of snapshots and how much we want to retrieve from the stream.

3. DenStream2

After investigating algorithms such as Clustream [1], DenStream [4], C-Denstream [18], rDenStream [19], D-Stream [5], IncrementalDBSCAN [6], MR-Stream [15] and strAP [17], it was found that DenStream is relatively simple in understanding, easy in implementation, and an excellent example of stream clustering. So, DenStream was chosen as the basis for the algorithm implemented in this work. Some changes to the original algorithm were made and define a new version called *DenStream2*. DenStream2 is implemented in Java as a foreign function of Amos II.

3. 1 System overview

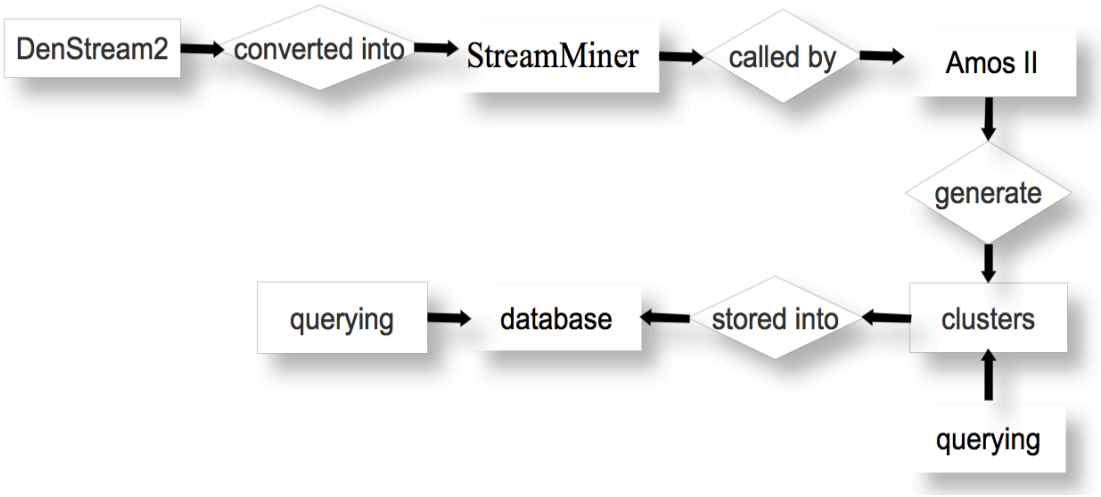


Fig. 1 The architecture of the implemented system

The architecture of the system is shown in **Fig. 1.** A foreign function in Java called *StreamMiner* implements Denstream2.

3.1.1 The StreamMiner function

create function StreamMiner(data stream, other parameters)-> Stream of <integer windowID, real time, integer clusterID, vector microCluster> as stream of DenMiner(data stream, other parameters).

Table. 1 Signature of foreign function

The StreamMiner function (Signature in **Table. 1**) returns representations of a stream of microclusters. The representation of a microcluster includes the ID of a window in the stream, the time stamp of the end of the time period for the window, the ID of the cluster to which the emitted microcluster belongs, and coordinators of center and

radius of the microcluster. It is defined a derived function calling another foreign function, *DenMiner* implementing Denstream2 in Java. The macroclusters in each window can be computed by the feature 'clusterID' of each microcluster.

3.1.2 Parameters of StreamMiner

Actually in the code, StreamMiner could be represented as StreamMiner($ds, \mu, \beta, \lambda, \varepsilon, d, o, h$). The parameters are passed to DenStream2 implemented in Java when StreamMiner is called. The parameter *ds* is the stream that will be processed, and four of the parameters ($\mu, \beta, \lambda, \varepsilon$) are used in the StreamMiner algorithm for Denstream [4]. Parameter ε is used to control the size of microclusters and each microcluster can't have radius bigger than ε; another two parameters μ and β are used to distinguish the core-microcluster (weight $\geq \mu$), potential core-microcluster (weight $\geq \mu\beta$), outlier microcluster (weight $< \mu\beta$) and deleted outlier microclusters (weight $< \xi$); the last parameter is λ which determines how fast the weight of microcluster should decrease.

The extended algorithm Denstream2 has three more parameters to control the execution. One is d, the dimension of the data in the stream. The other two are o and h and are used to tune clustering. To test their effects some specific experiments were done for different cases in the evaluation section. Their details follow.

The parameter o specifies the maximum allowed number of microclusters for a common data point when a new data point arrives for clustering. The value for o can be 1, 2, or 3. Here the overlapping between microclusters at the same time point are controlled. If the value is 1, a new arriving data point will be merged into nearest microcluster. The result is that some microclusters are deleted and some important information may be lost. These microclusters may be preserved if a big value for o is used. If o is 2 or 3, the new data points can be possibly assigned to 2 or 3 nearest microclusters and the weight of each microcluster will grow by one. In this case, more microclusters are produced and the memory usage will increase. In the DenStream suggested in [4], a different solution is proposed about how to deal with the overlapping of potential core-microclusters, but no test about such overlapping was done. In my opinion, this overlapping can exist not only between potential core-microclusters, but also between potential core-microcluster and outlier microcluster,

or between outlier microclusters. Experiments have been done to test the overlapping to determine the accuracy of clustering. This is elaborated in the discussion section.

The parameter h is time interval and determines the timed window size. It isn't possible for the program itself to decide when to request for offline macro clustering without such a time interval. The choice of h is critical. If a too small h is chosen, a lot of offline data will be produced and consume a lot of computing resources; if h is too large, some important changes might be missed. In [15], a memory sampling method is used to detect the cluster evolution automatically.

3.2 Transferring Clusters to the DSMS and processing

The microclusters generated by Denstream2 are emitted one by one by the foreign function StreamMiner calling *emit* [21]. *emit* in **Table. 2** is a predefined method in the Amos II foreign function interface, which is called from the Java code of DenStream2. This *emit* method will send one window of features of a microcluster to the DSMS.

emit <windowID, time, clusterID, microCluster >

Table. 2 "emit" method in Java

3.2.1 Storing clusters in the database

The emitted clusters can be explicitly stored in the database. The entity relationship of stored data is shown in **Fig. 2**. The macroclusters are generated by offline clustering at the end of each time periods. In each window represented by entity type *Horizon*, there are a number of macroclusters and each of them is represented by a group of associated microclusters. The three stored tables *Offline Macro-Clustering, Contains* and *Represented-by* are created and the database schema is shown **Table. 3**, **Table. 4** and **Table. 5**. In the *Offline Macro-Clustering* table**,** the information of each window is stored; attributes "horizonID" is the primary key and it has N:1 relationship with attribute "name" and 1:1 relationship with "time". In the *Contains* table, the information of each macrocluster is stored; the attribute "clusterID" determines one macrocluster and has N:1 relationship with attribute "horizonID". In the *Represented-by* table, composite key for this table consists of attributes "clusterID" and "center".

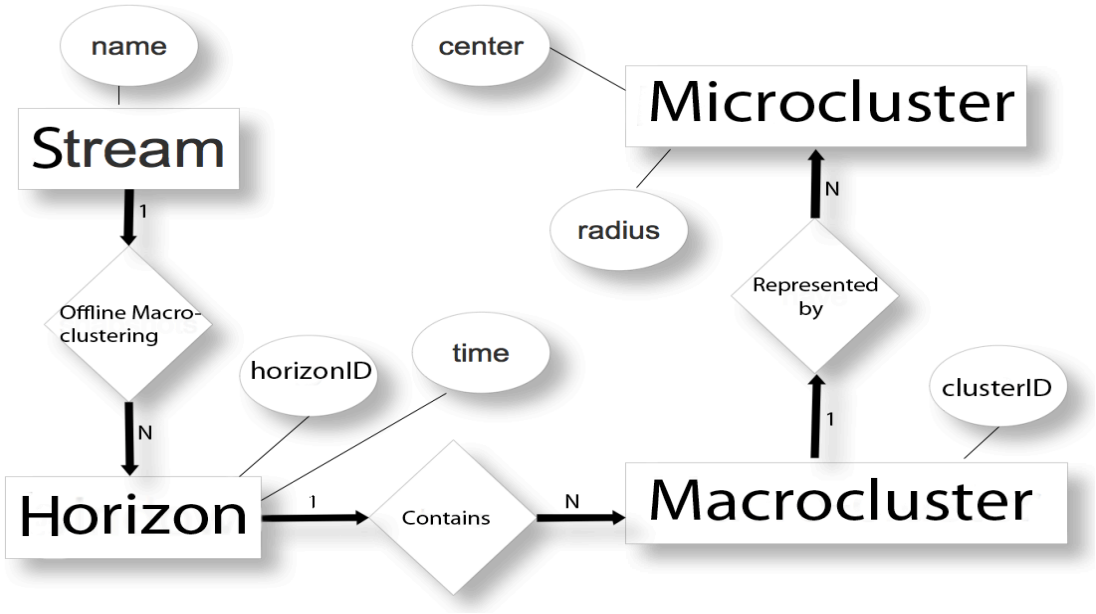


Fig. 2 The ER diagram for storing the macroclusters in the database

Attribute	Attribute description
horizonID	The id of window where snapshot taken in online micro-clustering
name	The name of stream processed.
time	The stream time at which snapshot is taken

Table. 3 *Offline Macro-Clustering* table (the name of attributes in bold is the primary key)

Attribute	Attribute description
clusterID	The ID of a macrocluster which the microclusters belong to
horizonID	The id of window where representing a cluster

Table. 4 *Contains* table (the name of attributes in bold is the primary key)

Attribute	Attribute description
clusterID	The ID of a macrocluster which the microclusters belong to
center	The coordinates of the microcluster center
Radius	The radius of the microcluster

Table. 5 *Represented-by* table (the name of attributes in bold is the composite key)

3.2.1.1 Creating the Database

Following the schema shown in **Table. 3**, **Table. 4** and **Table. 5**, three tables *Offline Macro-Clustering, Contains* and *Represented-by* were defined to keep the produced microclusters in database. The stored procedures *loadMicroClusters* and *addMicroClusters* are used to process the microclusters one by one and add them into database. Indexes were created on the key in all tables to speed up later queries to the database. Notice that storing the stream in tables like this has the restriction that the stream is of limited size.

3.2.1.2 Queries to the database

If the clusters are stored in the database using the above schema, queries can be done once the database is populated with the clusters computed from the stream. For example, finding out the number of macroclusters and microclusters in specific window in one stream, searching the windows to which a data point belongs, finding the windows which have maximum number of macrocluster, and so on.

3.2.2 Queries directly on stream of clusters

In this case, queries will work on the generated stream directly without storing all clusters in the database. Queries on the stream are similar to the queries searching stored data.

3.3 The Denstream2 algorithm

First some basic concepts of the original Denstream will be explained for better understanding the algorithm. The detail can be found in the original paper [4].

- Damped window model

  Window model in which the weight of data points or microclusters is decreased exponentially over time via a *fad* function. The damped window model helps the algorithm represent the current state of the stream while limiting the historical state. There is a question about how long history and how much historical data should be included. This is adjusted by the value of a parameter in the fad function [4], and depends on the application.

- Potential core-microclusters

  This definition and the next one are useful in online micro clustering. "Potential" here means it might be a core-microcluster or an outlier at some time, which depends on the evolution of data stream.

  A potential core-microcluster is a microcluster at time t and contains a number of close data points ($x_1, x_2, x_3 \cdots x_n$) with time stamp ($t_1, t_2, t_3 \cdots t_n$). t is no smaller than data time stamps. The potential core-microcluster is defined by a feature vector $\{\overline{CF^1}, \overline{CF^2}, w\}$ and w is the weight of the microcluster. It also has some features like real core-microcluster as following:

  Weight: $w = \sum_{i=1}^{n} f(t - t_i)$ where $w \geq \mu\beta$ (Equation 7)

  Center: $c = \dfrac{\overline{CF^1}}{w}$ (Equation 8)

  Radius: $r = \sqrt{\dfrac{|\overline{CF^2}|}{w} - (\dfrac{|\overline{CF^1}|}{w})^2}$ where $r \leq \varepsilon$ (Equation 9) or $r = \sqrt{(\sum_{i=1}^{n} \dfrac{\overline{CF^2(i)}}{w} - (\dfrac{\overline{CF^1(i)}}{w})^2)/n}$ (Equation 10 suggested in [1]) .

- Potential outlier microcluster

  A potential core-microcluster is a microcluster at time t and contains a number of close data points ($x_1, x_2, x_3 \cdots x_n$) with data time stamps ($t_1, t_2, t_3 \cdots t_n$). t is no smaller than data time stamps. The outlier microcluster is defined as a feature vector $\{\overline{CF^1}, \overline{CF^2}, w, t_o\}$. w is the weight and t_o ($t_o = t_1$)

is the creation time of the microcluster. It has the same features as the potential core-microcluster and is calculated in the same way. The only difference is that $w < \mu\beta$.

μ and β are additional parameters in the DenStream2.

DenStream2 is a modification of Denstream that inherits most concepts and theories from the original DenStream [4]. The pseudo code of DenStream2 is described in **Table. 6.** There are some important changes to the original Denstream algorithm,shown in bold in **Table. 6**. The changes determines if a new point should be incorporated into a microcluster, as further discussed later. The code in **Table. 6** implements DenStream2. Actually, there are further minor changes compared to the original DenStream and these changes will be discussed more in the experiment and discussion parts of this report.

DenStream2 can be simply divided into three steps, initialization, online micro clustering, and offline macro clustering, analogous to DenStream [4].

DenStream2($ds, \mu, \beta, \varepsilon, \lambda, h, o$ **)**

```
/**initialization part**/
1: initial a few potential core-microclusters
/**online part**/
2: for each data point q arrives at t from stream ds
3:      if size(potential core-microcluster buffer) > 0 Then
4:          find the nearest potential core-microcluster c_p;
5:          if d_p(distance from q to c_p) ≤ ε then
6:              merge q into c_p;
7:      else if size(outlier microcluster buffer) > 0
8:          find the nearest potential core-microcluster c_o;
9:          if d_o(distance from q to c_o) ≤ ε then
10:             merge q into c_o;
11:         else
12:           create a new outlier microcluster and put it into outlier microcluster buffer;
13:         end if
14:     else
```

15: create a new outlier microcluster and put it into outlier microcluster buffer;

16: end if

17: if ($nT_p \le t < (n+1)T_p$) then

18: for each potential core-microcluster c_p

19: update the feature vector of c_p

20: if w (weight of c_p) $< \mu\beta$ then

21: Remove c_p from potential core-microcluster buffer and add it into outlier microcluster buffer;

22: end if

23: end for

24: for each potential core-microcluster c_o

25: update the feature vector of c_o

26: if w (weight of c_o) $\ge \mu\beta$ then

27: delete c_o from outlier microcluster buffer

28: else if w (weight of c_o) $< \xi(t,t_o) = \dfrac{2^{-\lambda(t-t_o+T_p)}-1}{2^{-\lambda T_p}-1}$ then

29: remove c_o from outlier microcluster buffer and add it into potential core-microcluster buffer;

30: end if

31: end for

/**offline part**/

32: if $mT_{interval} \le t < (m+1)T_{interval}$ then

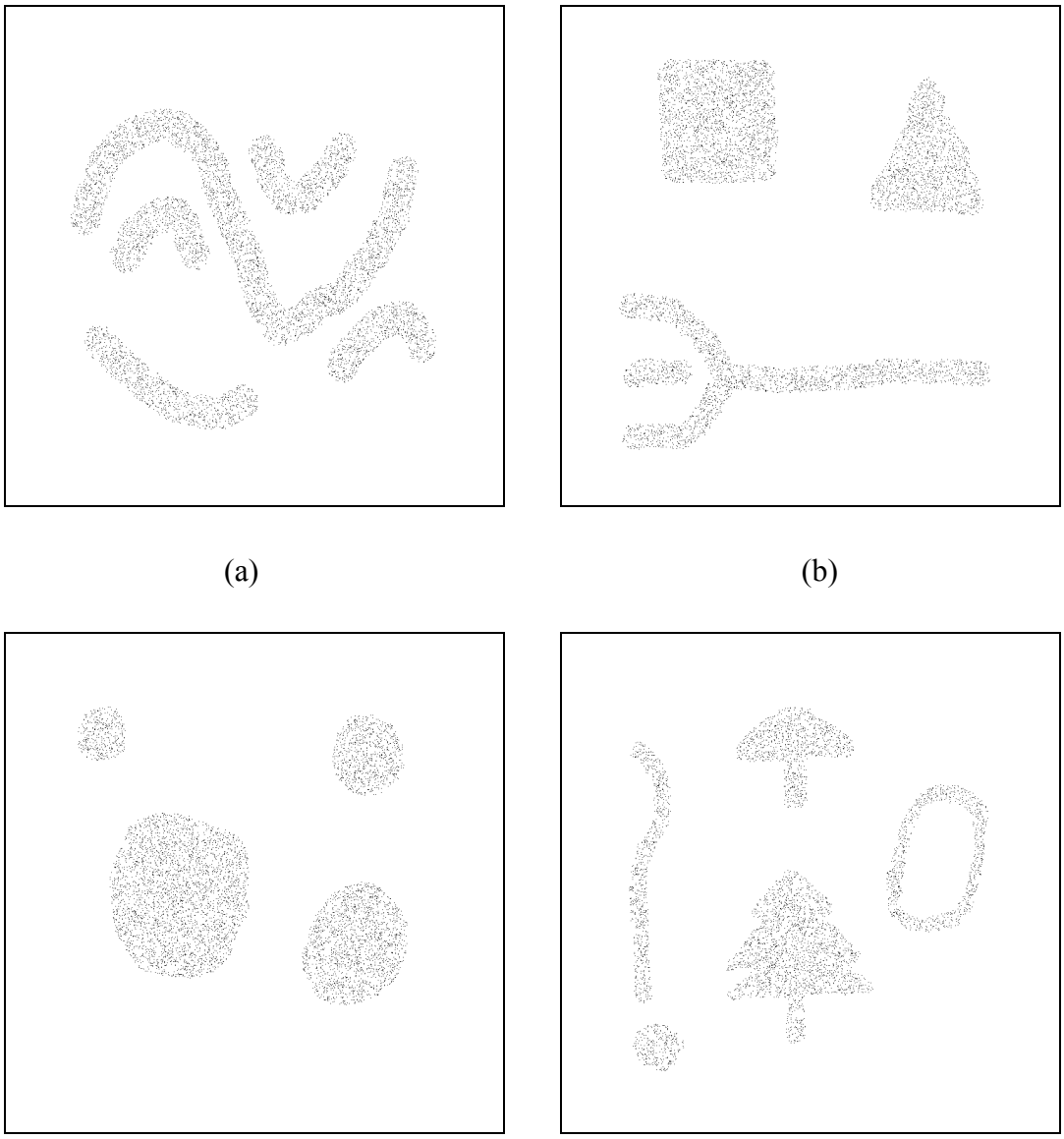
33: offline macro clustering on microclusters in potential core-microcluster buffer ;

34: send data (snapshot number, time, microcluster center, microcluster radius, macro cluster label) to Amos II;

35: end if

36: end for

Table. 6 Pseudo code for DenStream2. Variable *n* is number of checks in buffers and *m* is the number of snapshot taken in online micro clustering.



(a) (b)

(c) (d)

Fig. 3 Original synthetic data set. (a) Data set ST1 (b) Data set ST2 (c) Data set ST3 (d) Data set ST4

4. Experiments and Evaluation

In this section, a few experiments have been done to test the performance of DenStream2 by a group of synthetic data set and two real data sets from Internet. DenStream2 is implemented in Java and executed it on a MacBook pro with Intel Core 2 Duo 2.2GHZ, 4G memory and Snow Leopard operation system.

In the tests, the values of parameters used in [4] are adopted. So the value choice for parameters are: $\lambda = 0.25$, $\varepsilon = 16$, $\mu = 10$ and $\beta = 0.2$. Other parameters will be varied in the different tests.

4.1 Data sets and parameters in DenStream2

4.1.1 Synthetic data set

To create a data stream for the testing, four two dimensional test data sets were created. They are ST1, ST2, ST3, ST4 and each has 20000 data points. The distribution of data points at each data set in the data space are shown in **Fig. 3**, and the order of data points in each data set is random. After clustering by DBSCAN [7] on each data set, the results are shown in **Fig. 4** and the data points in the same cluster are represented in the same color. An evolving data stream was generated by combining data sets in order like ST1, ST2, ST3, ST4 and each data point was assigned a time stamp that grows as the stream evovles.

4.1.2 Real data set

The first real data set is a subset from a network intrusion detection data set used in KDD-CUP'99. It is also used in [1,4]. This network intrusion detection data set is about a series of TCP connection records from two weeks of LAN network traffic managed by MIT Lincoln Labs, around five million connection records. Each record can either be a normal connection, or an attack. Attacks fall into four big categories: DOS (denial-of-service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to local super user 'root' privileges), and probing (surveillance and other probing). There are around 21 attack subtypes. As we can see, this data set stands for one kind of automatic and real time cyber attack detection problem. So, it is a good example of a stream clustering test. In the original network intrusion detection data set, there are total 42 available attributes in each record. To generate a stream, only 34 numerical attributes of were extracted and each record in the stream is in similar format as data points in the synthetic stream. The mining of this real data stream is based on the 22 attack subtypes, not four main attack types. For $v = 1000$, the distribution of attack subtypes is displayed in **Fig. 5.**

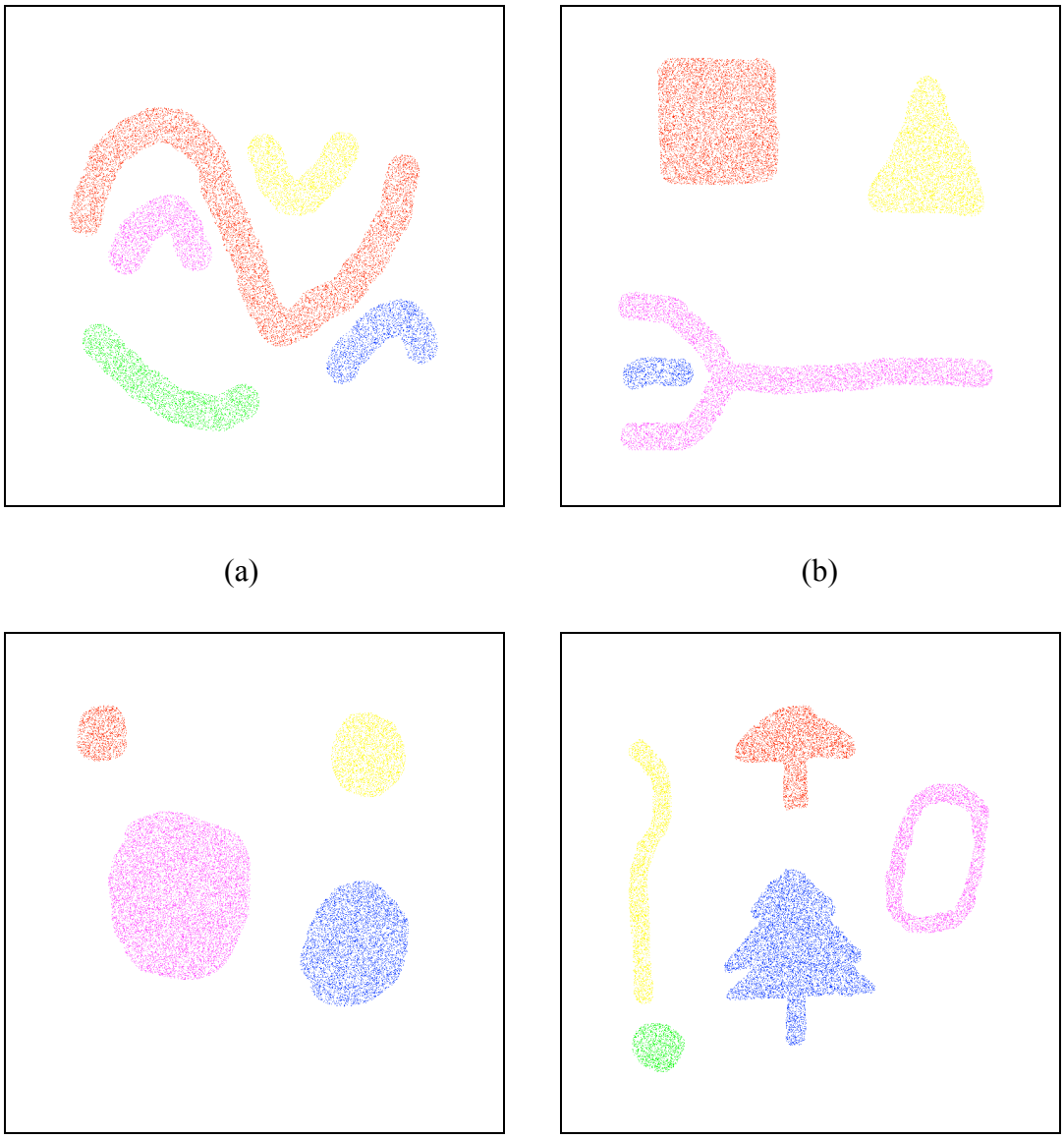


Fig. 4 Synthetic data set after clustering. In each data set, each color represents one cluster. (a) five clusters, (b) five clusters, (c) 4 clusters (d) 5 clusters.

4.1.3 Other

There is one important thing to consider in the experiments, namely. It is the new data point merging priority. In the DenStream suggested in [4], potential core-microclusters have higher priority to merge the new arriving data points than the outlier microclusters. In this way of merging, computing time will become shorter than with equal priority for merging because the conversion from outlier

microclusters to potential core-microclusters is minimized, and this is a bias in online micro-clustering. I will do some tests on its effect to the clustering quality, compared with equal priority to merge new data points.

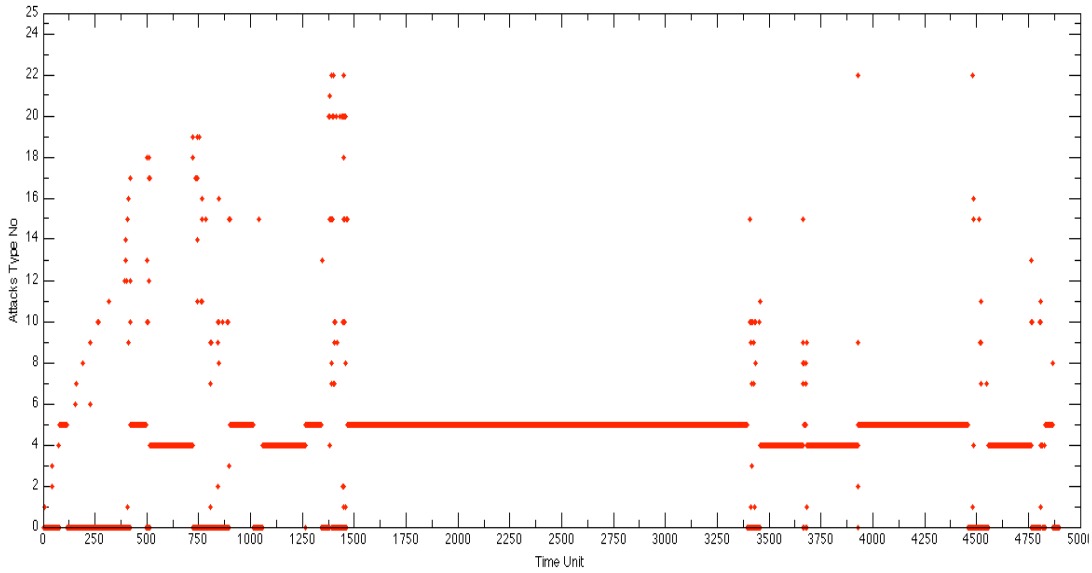
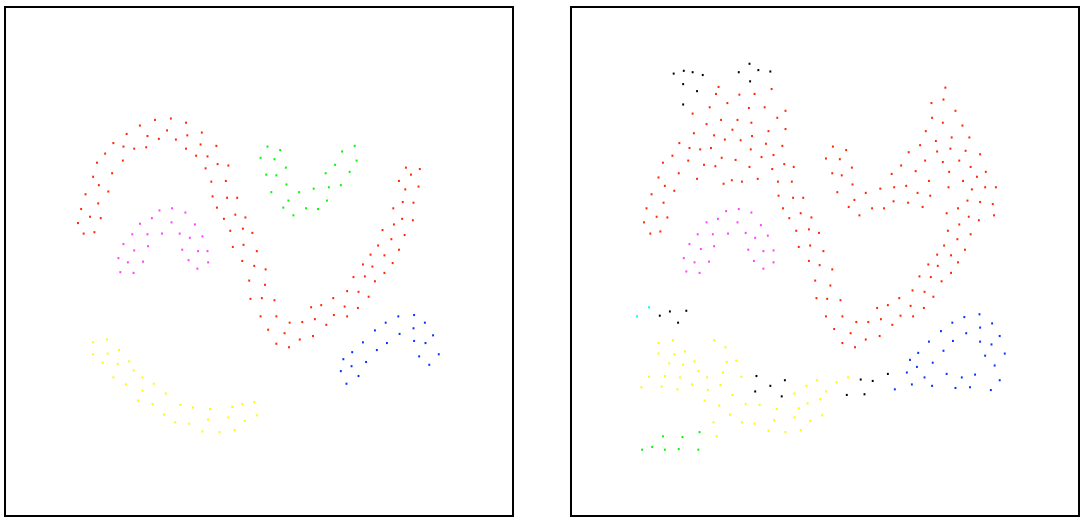
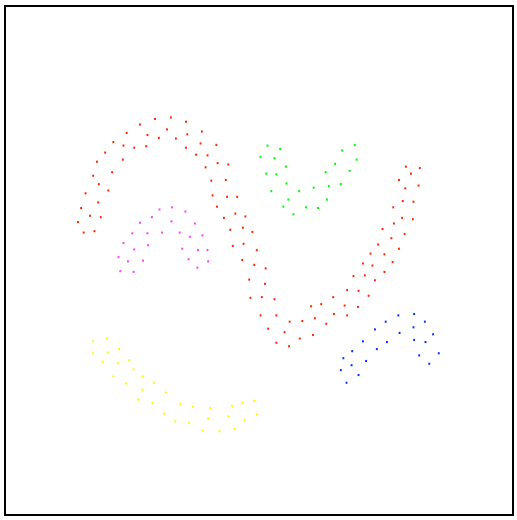


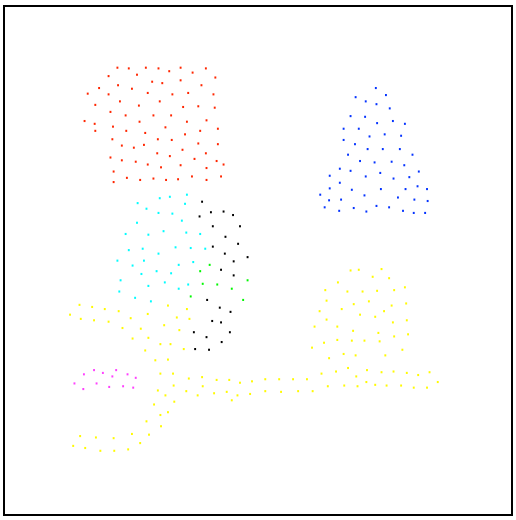
Fig. 5 Attacks distribution in the network intrusion detection data set. Different attack subtypes are represented by numbers, and the representation list is 0-normal, 1-buffer_overflow, 2-loadmodule, 3-perl, 4-neptune, 5-smurf, 6-guess_passwd, 7-pod, 8-teardrop, 9-portsweep, 10-ipsweep, 11-land, 12-ftp_write, 13-back, 14-imap, 15-satan, 16-phf, 17-nmap, 18-multihop, 19-warezmaster, 20-warezclient, 21-spy, 22-rootkit.



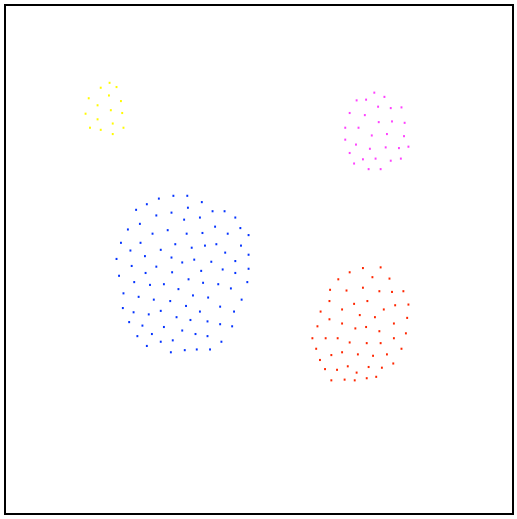
(a) (b)



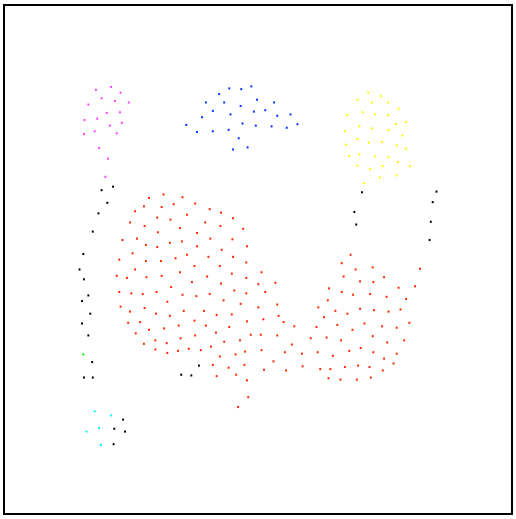
(c)



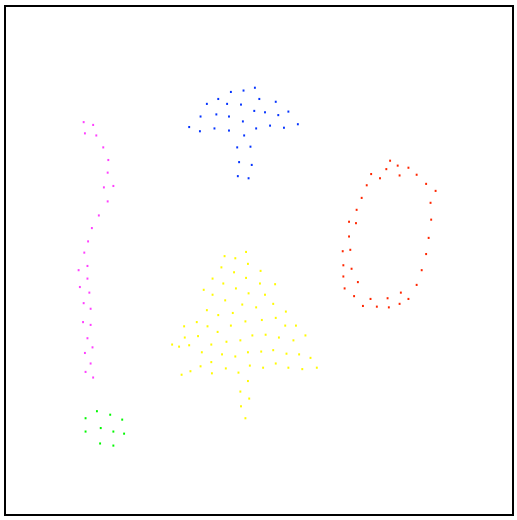
(d)



(e)



(f)



(g)

Fig. 6 Snapshots of stream after macro clustering with $h = 1$, and $o = 1$, and potential core-microclusters having higher merging priority. Each data point here represent one microcluster and microclusters in the same macrocluster are in the same color. (a) t=18, (b) t=22, (c) t=38, (d) t=42, (e) t=58, (f) t=62, (g) t=78.

4.2 Clustering quality evaluation

The clustering quality can be evaluated by the average purity of clusters resulted from the offline clustering. It is defined as follows:

$$purity = \frac{\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|}}{K}$$

where K is number of total macro clusters in one snapshots. $|C_i^d|$ is the number of data points of the dominant class in the cluster C_i and $|C_i|$ is the total number of data points in the cluster C_i. Since a damped window model is used in DenStream2, the purity will be calculated in each window and only the data points that arrives to the window will be included in the calculation.

In the first test, DenStream2 processed a data stream generated by the synthetic data sets ST1, ST2, ST3, ST4, having 80000 data points with speed 1000 data point per time unit ($v = 1000$). In the processing, potential core-microclusters have higher merging priority, $h = 1$, and $o = 1$. Snapshots are taken in the processing and examples are shown in **Fig. 6. Fig. 6a, c, e, g,** respectively, display one snapshot example of different stream parts produced by data set ST1, ST2, ST3, ST4 and macro-clustering results is the same as static data clustering in **Fig. 4**. Moreover, snapshots in **Fig. 6b, d, f** show the transition state between different stream parts produced by data set ST1, ST2, ST3, ST4, which clearly show a big evolution in the stream where it is impossible to define the clusters in these states before the stream processing is started. In fact, there is relatively smaller evolution during the specific stream parts produced by the other data sets. The movements of microclusters and the change in the number of microclusters in the buffers express this evolution. Therefore, DenStream2 works very well for the synthetic stream.

Further experiments have been done below to test the performance of DenStream2 with change in some factors, such as noise, window, stream speed, overlapping between microclusters, and priority in new data points merging.

4.2.1 Synthetic data stream with noise

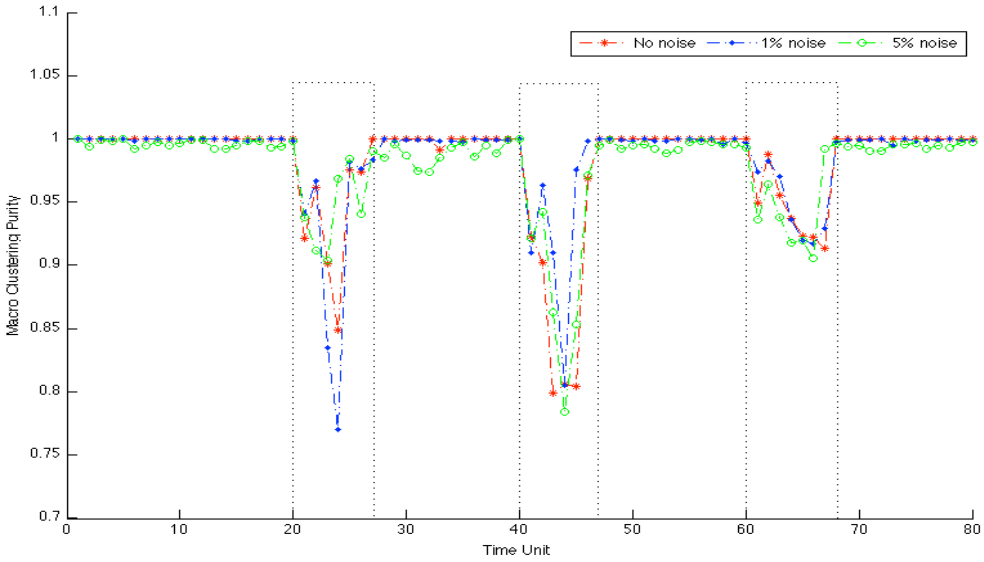


Fig. 7 Clustering quality on stream with different percentage of noise

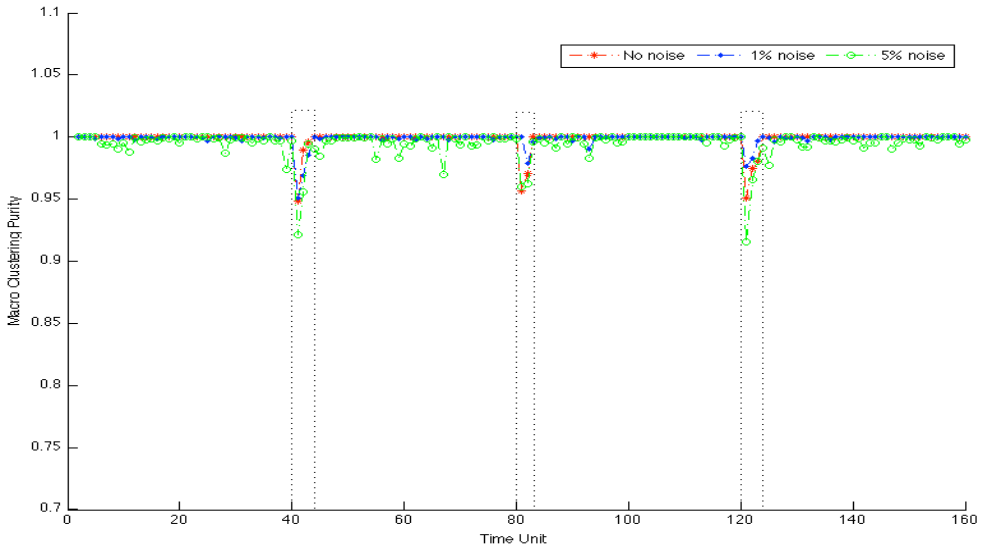


Fig. 8 Clustering quality when stream speed changes. Other setting in this test: data streams=synthetic data stream with different noise percentage (0%, 1%, 5%), $o = 1$, $v = 500$, and potential mciroclusters have high priority to merge new data points.

In this test, some random noises are uniformly mixed into the synthetic stream above in two different percentage levels, 1%, and 5%. These two streams were processed by DenStream2 and the clustering quality was compared with the synthetic stream without noise. **Fig. 7** shows the results. In this figure, rectangles in the figure whose edges are represented by suspension points mark the transition states between

stream parts generated by data sets ST1, ST2, ST3, ST4, and they are also present in the following figures for the synthetic data stream. Because the ID of clusters in transition state is unknown, the purity in transition states is lower than other stream parts and it means nothing. Hence, the test results on these states aren't useful. In periods outside the rectangles, the macro clustering purity is 100% when there is no noise in the stream, nearly 100% when there is 1% noise and above 95% with 5% noise. In all, Denstream2 works very well in filtering the noise data points.

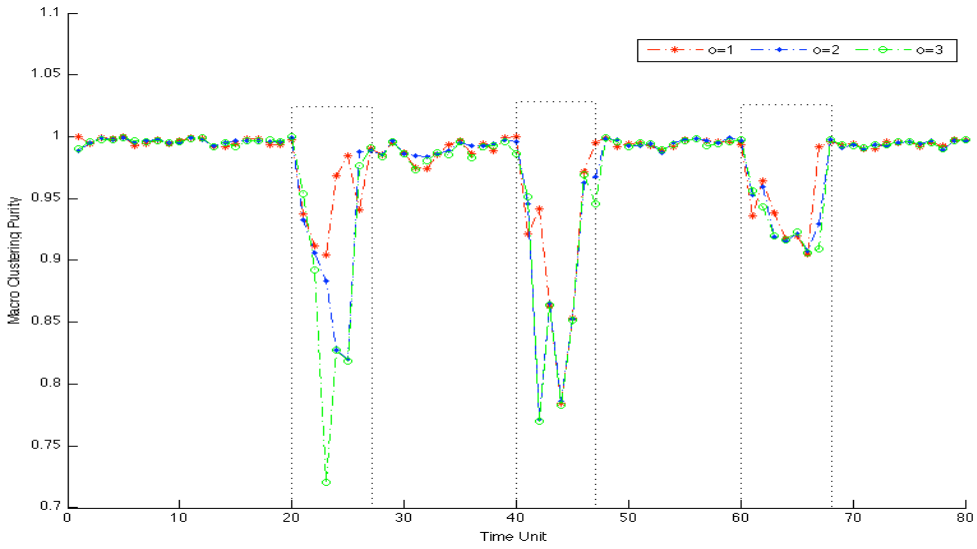


Fig. 9 Clustering quality with different overlapping factors. Other settings in this test: data streams=synthetic data stream with 5% noise, $h = 1$, $v = 1000$, and potential mciroclusters have high priority to merge new data points.

4.2.2 Stream speed

Stream speed is another factor that can change the behavior of stream. It affects the performance in a similar way as parameter λ, through the fade function. When stream speed decreases from high level to low level, the connection between historical data and current data will become weak, that is, historical information diminishes very fast. This effect is very obvious in that the transition state length become shorter in **Fig. 8** than in **Fig. 7,** but the clustering quality in other parts doesn't change too much. This means that each synthetic stream part has weaker relationship with its succeeding stream.

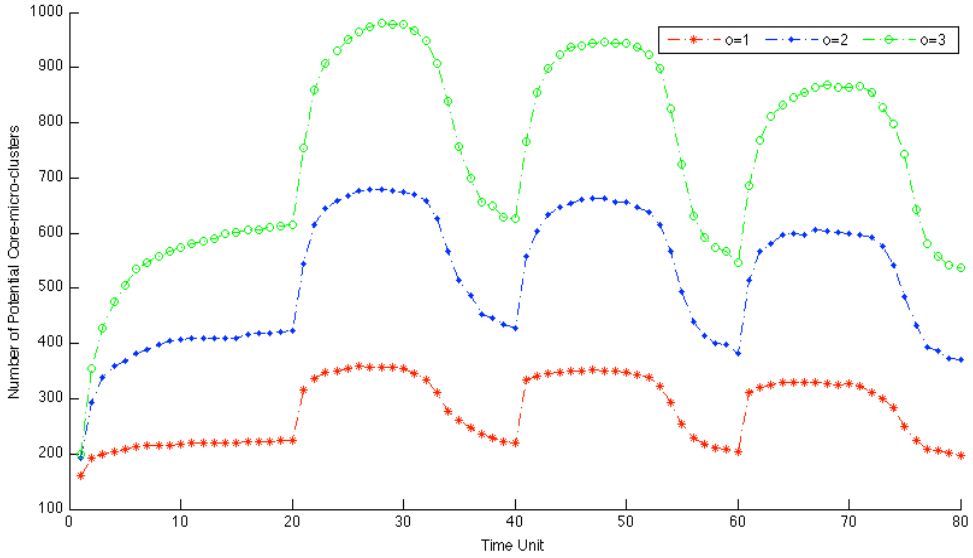


Fig. 10 Number of potential core-microclusters with different overlapping factor. Other setting are the same as **Fig. 9.**

4.2.3 Overlapping

In the online micro clustering, it is very likely that two microclusters have overlapping area. The data points in this overlapping can be merged in different ways. One way is to merge it only to the nearest microcluster, and another way is to merge it to all the microclusters which contain the overlapping area. In the first way, a microcluster that doesn't receive a data point in the overlapping area will be possibly deleted. Sometimes, this is unacceptable because some information of the stream will be lost. By contrast, the second way saves the life of some microclusters, but the computing complexity of the algorithm will increase very sharply when the density of data points in a cluster is high. In DenStream2, this overlapping effect is tuned through the overlapping factor o. From the experiments of the synthetic stream in **Fig. 9, 10** and real data stream in **Fig. 11, 12,** including the overlapping factor into clustering doesn't change cluster quality much, but the higher the overlapping factor is, the larger the number of produced potential core-microclusters becomes.

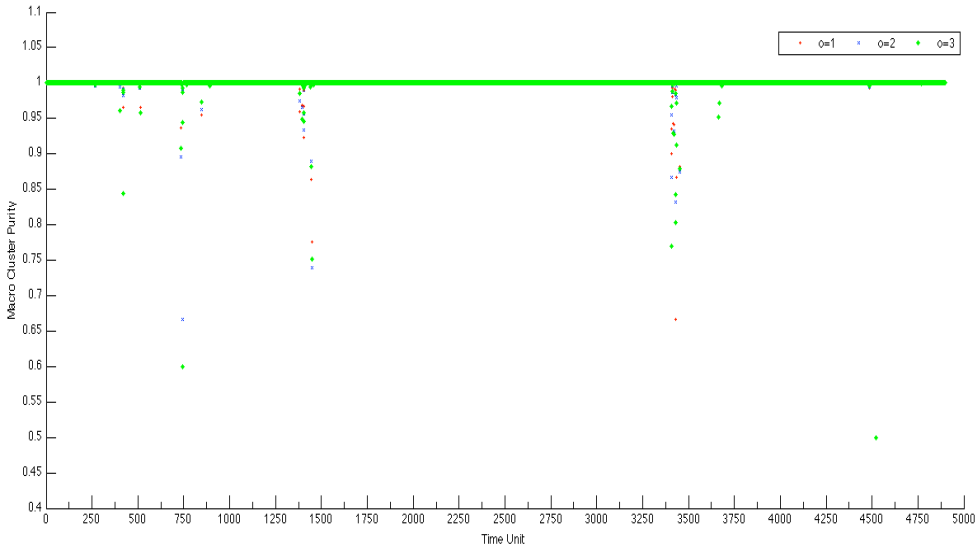


Fig. 11 Clustering Quality with different overlapping factor. Other settings in this test: data streams=network intrusion detection data stream, $h = 1$, $v = 1000$, and potential microclusters have high priority to merge new data points.

Fig. 12 Number of potential microclusters with different overlapping factor. Other settings are the same as **Fig. 11.**

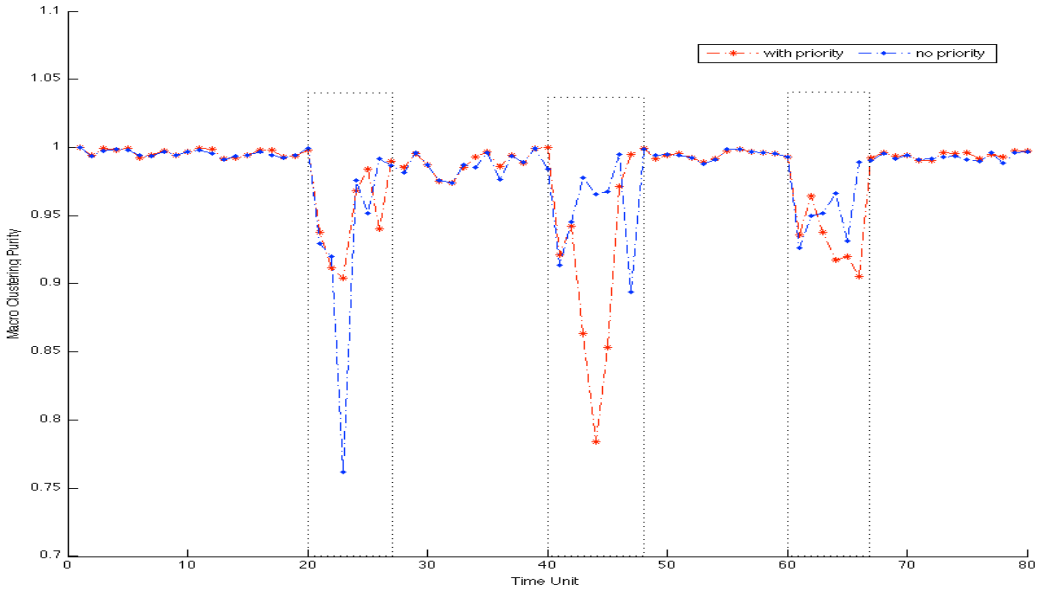


Fig. 13 Macro clustering quality with or with merging point priority. Other settings in this test: data streams=synthetic data stream with 5% noise, $h = 1$, $v = 1000$, and $o = 1$.

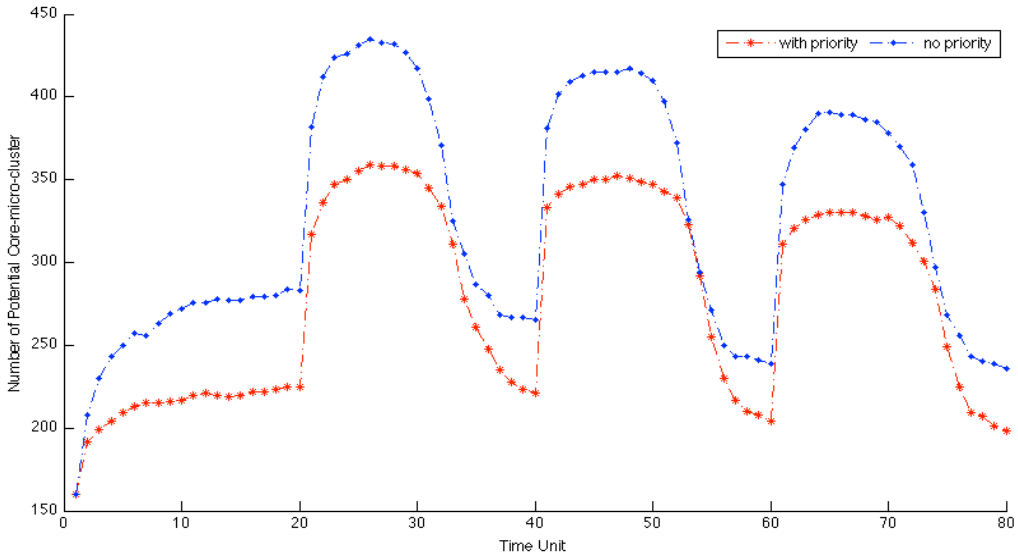


Fig. 14 Number of potential core-microcluster with or with merging point priority. Other settings are the same as **Fig. 13.**

4.2.4 Merging priority

In the DenStream [4], the new arriving data points are merged to potential microclusters first and then to the outlier microcluster. In this case, the outlier microcluster can easily be deleted, and the computing complexity will decrease somewhat. If there is no priority, the new arriving data point can be assigned to the nearest microcluster, which may be a potential core-microcluster or outlier microcluster. To test the effect of priority in clustering, experiments have been done with DenStream2 for the synthetic data stream and the real data stream, with and without merging priority. As the results shown in **Fig. 13** and **Fig. 15**, the clustering quality is nearly the same in the cases with or without merging priority. But without merging priority, the number of potential core-microclusters (in **Fig. 14** and **Fig. 16**) will become larger than the one with merging priority. Therefore, equal priority in merging data points can save some microclusters that will be deleted in the case of merging priority. So, merging priority has the same situation as the overlapping factor.

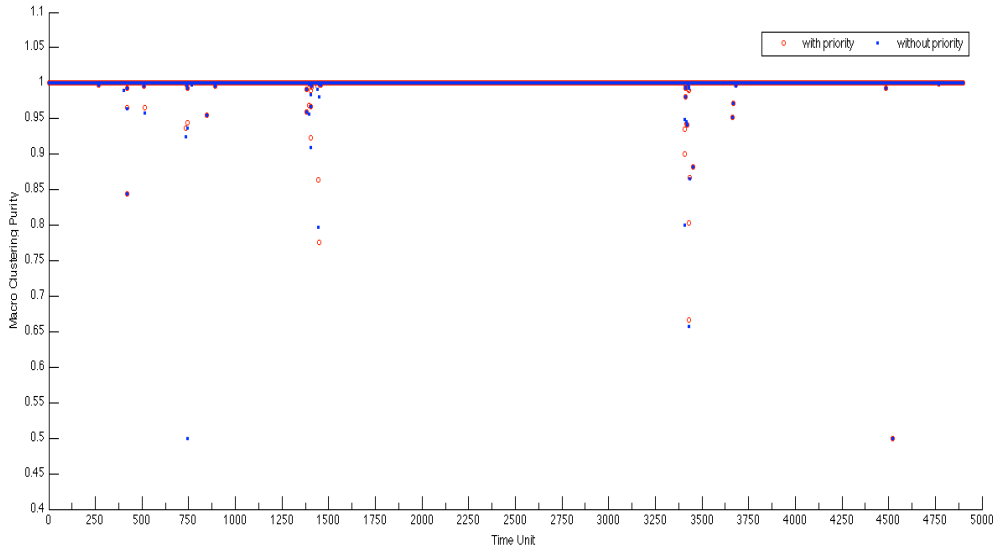


Fig. 15 Clustering quality with or without merging priority. Other settings in this test: data streams=network intrusion detection data stream, $h = 1$, $v = 1000$, and $o = 1$.

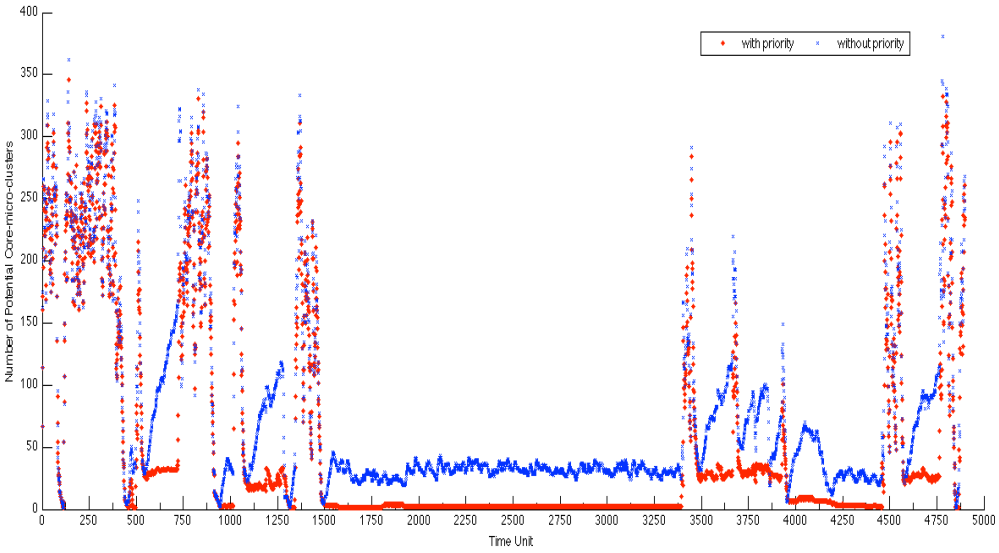
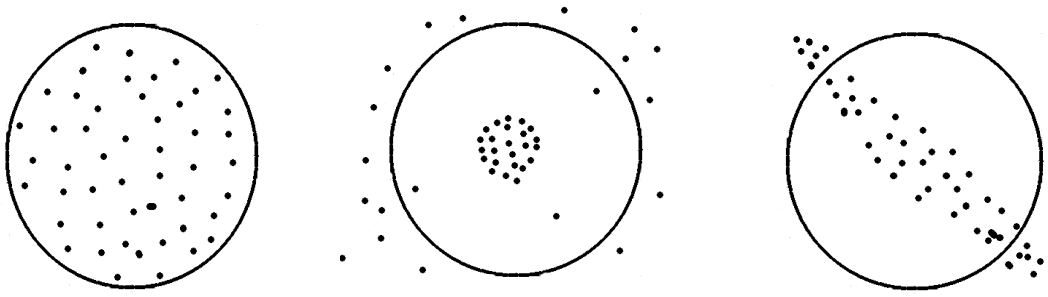


Fig. 16 Number of potential core-microclusters with or without priority. Other settings are the same as **Fig. 15**.

5. Discussion

The major difference between the DenStream2 algorithm and DenStream is the method to incorporate new data points into the existing microclusters (marked by bold words in **Table 8**) in the online micro-clustering component. The difference is that in DenStream, the average radius is

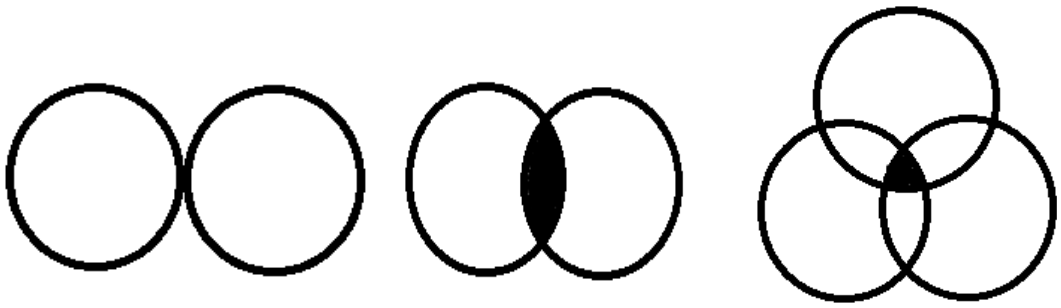


(a) (b) (c)

Fig. 17 Three possible cases of online microclusters with maximum radius ε. (a) all data points are in a circle with radius ε, (b) some data points concentrates around the center of microclusters and others outside the circle with radius ε, (c) the microcluster is in a very strange shape, not a circle.

used to measure if a new data point should be merged into a microcluster and in DenStream2, the measure is the distance between a new data point and center of a microcluster. This small change in measurement can make big difference in microcluster creation. In DenStream, three possible cases can happen because average radius is used to determine the merging of new data point which case is going to happen to a microcluster depends on the order of data points in the stream. The cases for two dimensions are shown in **Fig. 17**, perfect cyclic microcluster (**Fig. 17a),** data point outside of circle with radius ε (**Fig. 17b),** and microclusters in arbitrary shape (**Fig. 17c)**. Actually, the microcluster in **Fig. 17a** is the expected microcluster and DenStream can't guarantee this. Hence, it will be very hard to determine the region of each microcluster. Therefore, in DenStream2, the data points in one microcluster are always in a circle with radius ε and this will be very useful in a database query if the microcluster is stored in a database, compared to using a a random data point.

Overlapping between microclusters is another factor which differentiates the stream clustering. In Denstream2 a data point can be assigned to 1, 2 or 3 existing microclusters if the distance between the data point and the microclusters is smaller than the radius threshold ε; the three cases are shown in **Fig. 18**. Through the experiments, we can see that clustering quality doesn't change much in different overlapping levels, but a number of microclusters is possibly deleted as $o = 1$ when data points can be assigned to more than one microcluster. This means that allowing overlaps between microclusters can save more information of the stream. At the same



(a) (b) (c)

Fig. 18 Three cases of overlapping between microclusters. (a) $o = 1$, (b) $o = 2$, (c) $o = 3$. The filled black areas are the overlapping area between microclusters

time, some redundancy is produced and computing complexity will increase. How much overlapping is needed can be tuned based on the application.

In DenStream, the data points may be merged into a nearest potential core-microcluster first, and then the nearest to the outlier-microcluster. This is a discrimination in clustering that can save computing time if there is a nearest outlier-microcluster which meets the merging requirement. Without priority in merging, new data point can cost more in computing time, but new microclusters are detected as shown in the experiment of DenStream2. Therefore, removing priority is a good way to detect more useful information of a data stream.

DenStream2 has been used to do some tests on real data sets, and the clustering quality is satisfactory. As in [1,4], ahead of the stream clustering there is no normalization done to the data, and this means that the contribution of each dimension in the distance measure is different. It may be impossible to do such preprocessing of the data stream before further clustering because lack of global information of the data stream.

In many data stream mining algorithms, there is the assumption that the speed of the stream is constant. However, because there is an extra requirement in DenStream2 that each data record has its own time stamp, DenStream2 could possibly process data stream with continuously varying speed.

As the clustering results shown in **Fig. 6**, the stream data are compressed very much after the processing. So not much secondary memory is needed to save the summarization of data stream into a database. Actually, DenStream2 can also be used for static data compression before we save the data into database. What we need to do is to assume that all data in stream has same time stamp. This is more efficient than DBSCAN in data clustering because much less data are stored into database after clustering.

6. Conclusion

The experiments show that DenStream2 works very well in evolving data stream clustering and can detect the change in the stream. It can be used in many stream-clustering applications.

However, a big effort is needed for a good clustering effect because there are many parameters to tune in DenStream2. The parameters in the algorithm need to be adjusted to get the expected clustering result efficiently. Future work could be to remove some parameters and improve the algorithm; an ultimate goal is an algorithm free of parameters. In fact, the overlapping between microclusters is avoidable if the shape of microclusters is square (in 2D) but not round, which could be another important improvement to the algorithm.

Both DenStream [4] and DenStream2 are density-based algorithms. As we all know, densitybased algorithms have a common advantage: to identify clusters in arbitrary shape, and they also have a common problem that the algorithm works badly if the densities among clusters in same data are very different. Some algorithms are suggested to solve the disadvantage by decreasing density thresholds [20] in static data processing or using a hierarchical structure in the stream mining algorithm [15] in stream mining. Perhaps DenStream2 can be modified to processing data streams with changing density using these strategies. Actually, a new algorithm, affinity propagation [18] can be adapted to process streams in an efficient way by releasing the hard constraints as has been done in [11]. The density problem in DenStream2 could be overcome using this method.

7. References

1. C. C. Aggarwal, J. Han, J. Wang, P. S. Yu. A framework for clustering evolving data streams. In *Proc. of VLDB*, 2003.
2. C. C. Aggarwal, J. Han, J. Wang, P. S. Yu. A framework for projected clustering of high dimensional data streams. In *Proc. of VLDB*, 2004.
3. D. Barbará. Requirements for Clustering Data Streams. *SIGKDD Explorations, pages* 23-27, 2002.
4. F. CAO, M. ESTER, W. QIAN, A. ZHOU. Density-based clustering over an evolving data stream with noise. In *Proc. of SIAM*, 2006.

5. Y. Chen, L. Tu. Density-based clustering for real-time stream data *In Proc. of the ACM SIGKDD,* 2007.

6. M. Ester, H. Kriegal, J. Sander, M. Wimmer, X. Xu. Incremental clustering and mining in a data warehousing environment. *In Proc. of VLDB,* 1998.

7. M. Ester, H. Kriegel, J. Sander, X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD,* 1996.

8. S. Guha, A. Meyerson, N. Mishra, R. Motwani. Clustering data streams: Theory and practice. In *IEEE Transactions on Knowledge and Data Engineering,* 2003

9. E. Ikonomovska, S. Loskovska, D. Gjorgjevik. A survey of stream data mining. In *Proc. of ETAI*, 2007.

10. M. Khalilian, N. Mustapha. Data Stream Clustering: Challenges and Issues. In *Proc. of IMECS,* 2010.

11. M. Leone, Sumedh, M. Weigt. Clustering by soft-constraint affinity propagation: applications to gene-expression data. *Bioinformatics,* 2007.

12. L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, S. Guha. Streaming-Data Algorithms for High-Quality Clustering. In *Proc. of ICDE,* 2002.

13. C. Ruiz, E. M. Ruiz, M. Spiliopoulou. C-DenStream: Using Domain Knowledge on a Data Stream. *Discovery Science.* 2009.

14. C. Ruiz, M. Spiliopoulou, E. M. Ruiz. C-DBSCAN: Density-Based Clustering with Constraints. *RSFDGrC,* 2007, 216-223

15. L. Wan, W. K. Ng, X. H. Dang, P. S. Yu, K. Zhang. Density-based clustering of data streams at multiple resolutions. In *Proc. of TKDD,* 2009

16. T. Zhang, R. Ramakrishnan, M. Livny. BIRCH: A New Data Clustering Algorithm and Its Applications. *Data Min. Knowl. Discov.,* 1997.

17. X. Zhang, C. Furtlehner, M. Sebag. Data Streaming with Affinity Propagation. *ECML/PKDD,* 2008.

18. J. F. Frey, D. Dueck. Clustering by passing messages between data points. *Science*, 2007.
19. L. Liu, H. Huang, Y. Guo, F. Chen. rDenStream, A Clustering Algorithm over an Evolving Data Stream. *IEEE*, 2009.
20. B. Qiu, X. Zhang, J. Shen. Grid-based clustering algorithm for multi-density. In *Proc. of MLCG*, 2005.
21. D. Elin, T. Risch. Amos II Java Interfaces, http://user.it.uu.se/~torer/publ/javaapi.pdf.
22. P. Kranen, I. Assent, C. Baldauf, T. Seidl. Self-adaptive anytime stream clustering. In *Proc. of ICDM*, 2009.
23. J. B. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. In *Proc. of BSMSP*, 1967.
24. T.Risch, V.Josifovski, and T.Katchaounov: Functional Data Integration in a Distributed Mediator System, in P.Gray, L.Kerschberg, P.King, and A.Poulovassilis **(eds.):** *Functional Approach to Data Management - Modeling, Analyzing and Integrating Heterogeneous Data*, Springer, ISBN 3-*540-00375-4, 2004.*
25. S. Flodin, M. Hansson, V. Josifovski, T. Katchaounov, T. Risch, M. Sköld, and E. Zeitler. Amos II Release 12 User's Manual, http://www.it.uu.se/research/group/udbl/amos/doc/amos_users_guide.html.
26. C. J. Date. An introduction to database systems. 8th Edition.
27. http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems
28. A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, J. Widom. STREAM:the stanford data stream management system. Technical report. 2004, Stanford InfoLab.
29. http://en.wikipedia.org/wiki/Data_Stream_Management_System
30. W. Ng, M. Dash. Discovery of Frequent Patterns in Transactional Data Streams. T. Large-Scale Data- and Knowledge-Centered Systems, Vol. 2 (2010), p. 1-30