The GDense Algorithm for Clustering Data Streams with High Quality

Ye-In Chang, Chia-En Li, and Shu-Yi Lin

***Abstract*—A data streams is a sequence of dynamic, continuous, unbounded and real time data items with a very high data rate that can only be read once. In data mining, clustering is one of useful techniques for discovering interesting data in the underlying data objects. The problem of clustering can be defined formally as follows: given *n* data points in the *d*-dimensional metric space, partition the data points into *k* clusters such that the data points within a cluster are more similar to each other than data points in different clusters. In the data streams environment, the difficulties of data streams clustering contain storage overhead, low clustering quality and a low updating efficiency. Therefore, in this paper, we present a new clustering algorithm with high quality, GDense, for data streams. The GDense algorithm has high quality due to two kinds of partition: cells and quadcells, and two kinds of threshold: δ and (1/4)δ. From our simulation results, no matter what condition (including the number of data points, the number of cells, the size of the sliding window, and the threshold of dense cell) is, the clustering purity of our GDense algorithm is always higher than that of the CDS-Tree algorithm.**

***Index Terms*—Clustering, data mining, data stream, density-based, grid-based.**

I. Introduction

Data mining involves the use of sophisticated data analysis tools to discover previously unknown, valid patters and relationships in large data sets. Data mining can be used in many areas in our life. Data mining algorithms can be classified into the following categories: classification, clustering, association rules, sequential patterns, time series patterns, link analysis and text mining. A data stream is an ordered sequence of items that arrives in timely order [1].

A data stream is a sequence of dynamic, continuous, unbounded and real time data items with a very high data rate that can only read once. High speed refers to the phenomenon that the data rate is high relative to the computational power. Mining data streams is concerned with extracting knowledge structures represented in models and patterns in non-stopping streams of information. It raises new problems for the data mining community in terms of how to mine continuous high-speed data items that you can only have one look at [2], [3] Mining data streams is a real time process of extracting interesting patterns from high-speed data streams. Therefore, the research in data stream mining has gained a high attraction due to the importance of its applications and the increasing generation of streaming information. Applications of data stream analysis can vary from critical scientific and astronomical applications to important business and financial ones [4], [5].

Clustering is an important task in data mining. The problem of clustering can be defined formally as follows: given *n* data points in a *d*-dimensional metric space, partition the data points into *k* clusters such that the data points within a cluster are more similar to each other than data points in different clusters [6]. A common form of data analysis in many applications involves clustering. It is useful for discovering groups and identifying interesting distributions in the underlying data. Clustering algorithms are attractive for the task identification in spatial databases.

The density-based approach applies a local cluster criterion. Clusters are regarded as dense regions in the data space, and separated by regions of low object density (noise or outlier). These regions may have an arbitrary shape and the points inside a region may be arbitrarily distributed [7]. The advantages of the density-based approach are that it can discover clusters with arbitrary shapes and it does not need to preset the number of clusters.

Although this method has a lot of advantages, it still has some disadvantages. If users do not give the appropriate values of these parameters, the result would be very bad. In the data streams environment, Cao *et al*. proposed DenStream [8]. The method can discover clusters of arbitrary shape in data stream, but it is insensitive to noises. The grid-based approach is the feature space quantized into cells using a grid structure. The cells can be merged together to form clusters [9].

The ubiquitous presence of data streams in a number of practical domains has generated a lot of research in this area [10]-[17]. Discovery of the patterns hidden in streaming data imposes a great challenge for cluster analysis. The goal of clustering is to group the streaming data into meaningful classes [9]. Furthermore, because of the dynamic nature of evolving data streams, the role of outliers and clusters are often exchanged, and consequently new clusters often emerge, while old clusters fade out. The difficulties of data streams clustering contain storage overhead, low clustering quality and a low updating efficiency. Therefore, the clustering algorithm should adopt an incremental manner. It becomes more complex when data streams insert and delete.

The CDS-Tree algorithm combines the advantages of the grid-based approach and the density-based approach. It can handle large databases. However, the clustering quality is restricted to the grid partition and the threshold of a dense cell. In the CDS-Tree algorithm, when the threshold is set too small, it cannot discard noises and it needs many merging

Manuscript received January 24, 2014; revised April 2, 2014. The research was supported in part by the National Science Council of Republic of China under Grant No. NSC-101-2221-E-110-091-MY2.

Ye-In Chang, Chia-En Li, and Shu-Yi Lin are with the Computer Science and Engineering Department, University of National Sun Yat-Sen University, 70 Lienhai Rd., Kaohsiung 80424, Taiwan, R.O.C. (e-mail: changyi@cse.nsysu.edu.tw, lice@db.cse.nsysu.edu.tw, suyiinformation@gmail.com).

times at the merging stage. On the other hand, when the threshold is set too large, it has some missing cases which will affect clustering quality. Therefore, to improve the disadvantages of the CDS-Tree algorithm, in this paper, we present a new clustering algorithm, GDense, for data streams. It combines the grid-based approach with the density-based approach. We use the clustering algorithm of the grid-based approach, because it is efficient for large special databases. So, it adapts to the data streams environment. We also apply the idea of the density-based approach, because it can discover clusters with arbitrary shapes, and it does not need to preset the number of cluster.

The rest of the paper is organized as follows. Section II gives a survey of several well-known clustering algorithms for data streams. Section III, presents the proposed new clustering algorithm based on the grid and dense approaches for data streams. In Section IV, we give a comparison of the performance of the CDS-Tree algorithm and our new clustering algorithm. Finally, we give a conclusion.

II. Related Work

The CDS-Tree (Cell Dimension Tree for Streams) algorithm is a cell-based approach. The CDS-Tree [18] stores only non-empty cells and keeps the position relationship among cells. The CDS-Tree of dataset X under a partition is a balanced tree <*Root-Node, MidNodes, Leaf-Nodes, Edges*>. Suppose database X has the dimensions $A_1, A_2, \ldots, A_k$, we have

1) *Root-Node* is the root-node of the tree and corresponding to the first dimension A_1.
2) *Mid-Nodes* is the set of intermediate nodes that are between the root nodes and the leaf nodes. The *Mid-Nodes* at the *i*-th level corresponds to the dimension A_i, uniquely, and contain all the distinct interval numbers of the dimension A_i corresponding to the non-empty cells based on intervals of the anterior the *i*-th dimension.
3) *Leaf-Nodes* is the set of the leaf-nodes, which is at the level (*k*+1). A leaf-node is represented as <*Coor, Total-Num, Num-Point-List*>. *Coor* is the coordinate of a cell represented as ($cNO_1, cNO_2, \ldots, cNO_k$). *Total-Num* is the total number of points falling in this cell. *Num-Point-List* is a list of the number of points, whose length is equal to the number of buckets in the sliding window, representing the number of points in the corresponding bucket that fall in the cell. Fig. 1 illustrates a sliding window model of the data stream. There are u buckets in window, whcih are numbered with $B_1, B_2, \ldots, B_n$. We have the following properties:
4) The *Edges* is a set of the pointers in non-leaf nodes (*Root−Node* and *Mid−Nodes*).

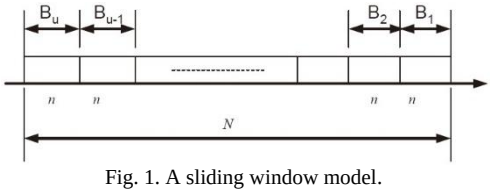
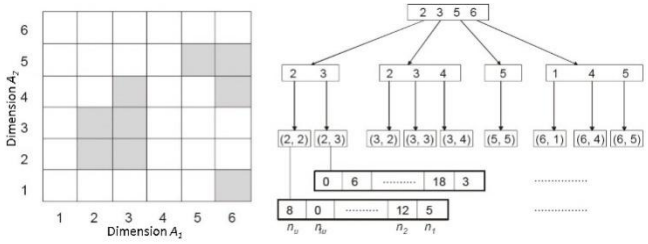


Fig. 1. A sliding window model.

The non-leaf nodes have the form <*cNO, pointers*>, where *cNO* is a keyword. The keyword of the *i*-th level is a distinct interval number of the *i*-th dimension corresponding to a cell, and *pointer* is a set of pointers; each of which points to the next level node. The *pointer* of the *k*-th level internal-nodes points a leaf node.

5) A path from the root node to a leaf node corresponds to a cell.

Fig. 2 a) shows the cell structure of a 2-dimension (A_1 and A_2) dataset under a partition, so the interval number of each dimension is from 1 to 6. Suppose that the gray cells represent the non-empty cells, and they show the distribution of data stream in a window. Fig. 2 b) is the CDS-Tree structure corresponding to the cell structure, which only stores non-empty cells.



a) b)

Fig. 2. An example of a CDS-Tree: a) the cell structure of a 2-dimension; b) the CDS-Tree structure corresponding to the cell structure.

The CDS-Tree building algorithm receives the current data object and computes coordinates (i.e. the interval number of each dimension) of them. If the cell corresponding to this object exists, we add the number of points in this cell. Otherwise, we create a new cell. The algorithm does not deal with the discard of buckets. The Clustering algorithm based on the CDS-Tree executes a width-first search on the CDS-Tree. Fig. 3 gives the clustering procedure on data streams, which marks adjacent dense cells with the same cluster number.

Input: CDS-Tree, dnesity threshold τ
Output: ClusteringResult
1. **For** each cell C of CDS-Tree **do**
2. **If** C is unmarked and $C.Total - Num >= \tau$ **then**
3. Mark C with a new cluster number $ClusterNo$;
4. **For** each neighbor C' of C in CDS-Tree **do**
5. **If** C' meets: (1) adjacent with C, (2) are unmarked, and (3) $C'.Total - Num >= \tau$ **then**
6. Mark C' with $ClusterNo$ of C and push C' into stack S;
7. **If** stack S is not empty **then**
8. Pop stack S as C, go to step 4;

Fig. 3. The Clustering algorithm based on the CDS-Tree.

III. The GDense Algorithm

In this section, we present our *GDense* algorithm for clustering data streams. The *GDense* algorithm is a clustering algorithm based on the grid and dense approaches. Moreover, it is based on the sliding window model. In the data stream environment, as the data increases, the requirement of the storage space will also increase. Since the *GDense* algorithm only stores information about the non-empty grids, it needs only small memory space. On the other hand, the clustering result based on the grid approach is limited by the grid size. Therefore, we propose a refinement algorithm to improve those quality of the grid-based algorithms. We focuses on the data stream environment. Several assumptions should be restricted in order to make our work feasible. These assumptions are as follows:

1) The input data must be number data.
2) The input data is two-dimension data.
3) Cluster similarity is based on the distance.

The *GDense* algorithm maps each incoming data point into the one non-overlapping rectangular cells (grids), and a cluster is composed of a set of dense cells and dense quadcells. A dense cell is defined as a cell whose number of data points is larger than or equal to δ. Table I shows the parameters used in our *GDense* algorithm. A cell as shown in Fig. 4 a) is divided into four quadcells according to different directions: *UL* (upper-left), *UR* (upper-right), *DL* (down-left) and *DR* (down-right) as shown in Fig. 4 b).

TABLE I: DESCRIPTION OF PARAMETERS

Str	The stream data
δ	The threshold of a cell
count	The count of data points in the cell
CNum	The cluster number of the cell
cell x	The x coordinate of a cell
cell y	The y coordinate of a cell
UL	The count of data points in an upper-left quadcell
UR	The count of data points in an upper-right quadcell
DL	The count of data points in a down-left quadcell
DR	The count of data points in a down-right quadcell
UL_CNum	The cluster number of a *UL*quadcell
UR_CNum	The cluster number of a *UR*quadcell
DL_CNum	The cluster number of a *DL*quadcell
DR_CNum	The cluster number of a *DR*quadcell
Old_Object	The oldest object
ClusterResult	The cluster result

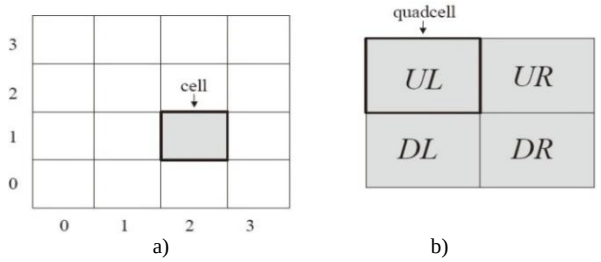


Fig. 4. Cells and quadcells: a) a cell; b) the corresponding quadcells.

In addition to four quadcells, we also consider the threshold of a quadcell. Take cell (4, 3) which is a dense cell in Fig. 5 as an example. If we only take the threshold of a cell into account, some data points may become outliers. The cluster only contains cell (4, 3). If we consider quadcells but not their thresholds, all data points will be in a cluster. But the distance is far between cell (0, 3) and cell (4, 3). On the contrary, if we take the threshold of the quadcell into account, the cluster will be the shadow region, which is more precise than previous results. To improve the clustering quality, we take the threshold of the quadcell into consideration. Therefore, a dense quadcell is defined that the number of data points is larger than or equal to (1/4)δ, where$\delta$is the threshold of a cell.

Fig. 6 shows our algorithm. The *GDense* algorithm has two procedures: Procedure *ClusterDS* and Procedure *User_Request*. Basically, the main algorithm is in Procedure *ClusterDS*. When users request for the clustering result, the *GDense* algorithm will call Procedure *User_Request* and response in real time. Procedure *ClusterDS* contains two steps: data insertion and data deletion. Fig. 7 shows Procedure *ClusterDS*. Because of the sliding window model, we must record the oldest object. If the sliding window is not full, we will insert data and cluster data by calling Procedure *InsertionD*. Otherwise, when the sliding window is full, we will delete the oldest data and cluster data by calling Procedure*DeletionD*.

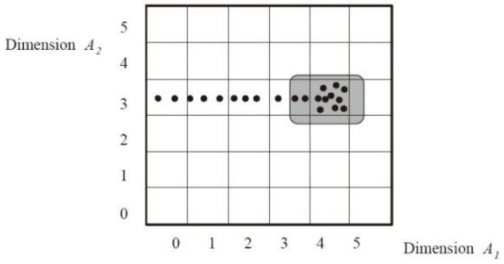


Fig. 5. An example of the threshold of the quadcell of cluster data streams.

```
1:  Procedure GDense(Str, δ);
2:  begin
3:      ClusterDS(Str, δ);
4:          InsertionD(Str, δ, Old_Object);
5:          DeletionD(Old_Object, δ);
6:      User_Request(ClusterResult);
7:  end;
```
Fig. 6. Procedure GDense.

```
1: Procedure ClusterDS(Str, δ, CellP);
2: begin
3:     for each object X in Str
4:         Old_Object := Object X0;
5:     initialize CellP, QuadcellP;
6:     if(FullSW = false) then /*Step 1*/
7:         InsertionD(Str, δ, Old_Object);
8:     else /*Step 2*/
9:         DeletionD(Old_Object, δ, CellP);
10: end;
```
Fig. 7. Procedure ClsuterDS.

Step 1: Data Insertion

In this step, first, when the stream data is inserted, we will map the original data to the space database and update the count of the cell structure. Procedure *InsertionD* is shown in Fig. 8.The original cell structure is shown in Fig. 9 a), and data point *X* is mapped to cell (1, 2) as shown in Fig. 9 b). After mapping the data point to cell structure, it will update the value of count to 1, *cell_x*to 1, *cell_y* to 2 and *UL* to 1. If the count of data points in the quadcell is more than (1/4)δ, these data points may be merged. Finally, it will return *CellP* = (1, 2) and *QuadcellP* = *UL*.Procedure*InsertionD* contains 7 cases as shown in Fig. 10 and Fig. 11. We take 3 factors about the cell and the quadcell into consideration: whether there is *CNum*, whether it is dense, and whether it has outer neighbors.

Step 2: Data Deletion

When the sliding window is full, Step 2 will be applied. First, we apply Procedure *Del_Update*(*Old_Object*,δ, *CellP*) as shown in Fig. 12 to delete data*Old_Object* and update the cell structure of the corresponding cell. If the count of the corresponding cell is less than the density threshold after deleting data *Old_Object*, *CNum* of the cell and its quadcells will be updated to 0. *CNum* = 0 means that those data points in the cell are outliers. Procedure *DeletionD* is shown in Fig. 12. In this step, it contains 10 cases as shown in Fig. 13. Let *NumX* be the *CNum* of *CellP*, where *CellP* is the cell of deleted point. We take account of the following 5 factors

about the cell as shown in Fig. 14: whether *NumX* is updated, whether the *CNum* of the right diagonal quadcell is *NumX*, whether the *CNum* of left diagonal quadcell is *NumX*, whether the*CNum* of the up and down quadcells are *NumX* and whether the*CNum* of the left and down quadcells are *NumX*. If *NumX* is updated, it may affect other clusters.

```
1: Procedure InsertionD(Str, δ, QuadcellP);
2: begin
3:   Ins_Map_Update(Str, CellP, QuadcellP);
4:   /*Map data into cell position and update the count of the cell structure*/
5:     if CellP has CNum then
6:         Stop                                                    /*Case 1*/
7:     else /* The cell has no CNum*/
8:         if (CellP.count ≥ δ) then
9:            Mark CellP with a new CNum;
10:           for each inner neighboring cell X
11:              Mark X with CNum of CellP and mark neighboring cells and outer
12:              neighboring quadcells wiht CNum of CellP;
13:           if (CellP has outer neighboring quadcell) then        /*Case 2*/
14:              Connect the outer neighboring quadcell
15:           else Stop                                             /*Case 3*/
16:       else /*the cell is not dense*/
17:           if (QuadcellP.count ≥ (1/4)*δ) then
18:              if (QuadcellP has CNum) then
19:                 Stop                                           /*Case 4*/
20:              else
21:                 if (QuadcellP has outer neighboring cell) then
22:                    Connect the neighboring cell                /*Case 5*/
23:                 else Stop                                      /*Case 6*/
24:           else Stop;                                           /*Case 7*/
25: end;
```

Fig. 8. Procedure InsertionD.

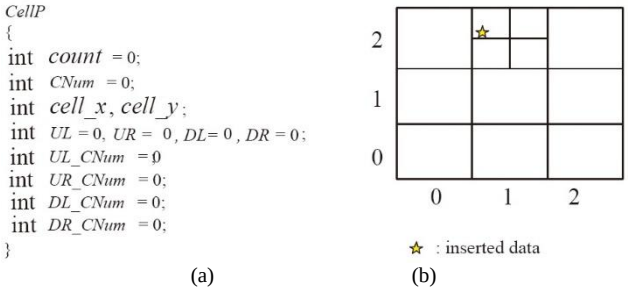


Fig. 9. An example: data structure of Procedure Ins_Map_Update: a) the original cell structure of cell (1, 2); b) data point X (1.2, 2.6) is mapped to cell (1, 2).

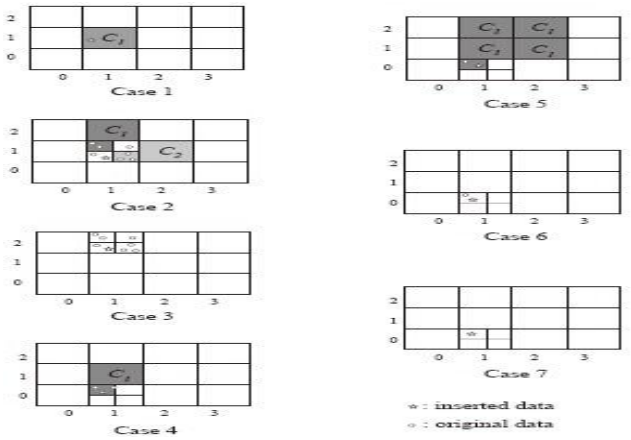


Fig. 10. The overview of 7 cases.

Now, let us use a simple example to show our algorithm with high quality. In the example of Table II, data point (1.1, 1.8) in transaction Tid_1 will be mapped to cell (1, 1) by calling Procedure *Ins_Map_Udpate*. Let δbe 8 and $(1/4)\delta$ be 2. Because both of cell (1, 1) and the quadcell in cell (1, 1) have no *CNum* and are not dense, it is Case 7 of Procedure*InsertionD*. In the same way, the insertions from transaction Tid_1 to transaction Tid_6 are also Case 7 of Procedure *InsertionD*. After inserting the 7th data point, the quadcell in cell (0, 1) becomes dense, but it has no outer neighboring cell. So, it cannot form a cluster. Therefore, it is Case 6 of Procedure*InsertionD*.

Case 1	*CNum*	dense	outer neighbor
Cell	Y	X	X
Quadcell	X	X	X

Case 2	*CNum*	dense	outer neighbor
Cell	N	Y	Y
Quadcell	X	X	X

Case 3	*CNum*	dense	outer neighbor
Cell	N	Y	N
Quadcell	X	X	X

Case 4	*CNum*	dense	outer neighbor
Cell	N	N	X
Quadcell	Y	Y	Y

Case 5	*CNum*	dense	outer neighbor
Cell	N	N	X
Quadcell	N	Y	N

Case 6	*CNum*	dense	outer neighbor
Cell	N	N	X
Quadcell	N	Y	X

Case 7	*CNum*	dense	outer neighbor
Cell	N	N	X
Quadcell	X	N	X

N: No
Y: Yes
X: Don't care

Fig. 11. The 7 cases of data insertion.

```
1: Procedure DeletionD(Old_Object, δ, CellP);
2: begin
3:   Del_Update(Old_Object, δ, CellP);
4:   /*delete Old_Object and update the count of the cell structure*/
5:   /*Let NumX be the CNum of CellP*/
6:   if (NumX is updated) then
7:     if (there is any right diagonal quadcell with NumX) then
8:       if (three is any left diagonal quadcell with NumX) then
9:         if(there are cells with NumX in up and down directions) then
10:          if (there are cells with NumX in left and right directions)then
11:            Stop                                              /*Case 1*/
12:          else Split the cluster into up and down clusters    /*Case 2*/
13:        else Split the cluster into left and right clusters   /*Case 3*/
14:       else Split the cluster into two clusters               /*Case 4*/
15:     else
16:       if (there is any left diagonal quadcell with NumX) then
17:         Split the cluster into two clusters                  /*Case 5*/
18:       else
19:         if (there are cells with NumX in up and down directions) then
20:           if (there are cells with NumX in left and right directions) then
21:             Split the cluster into four clusters             /*Case 6*/
22:           else  Split the cluster into up and down clusters  /*Case 7*/
23:         else
24:           if (there are cells with NumX in right and left directions) then
25:             Split the cluster into left and right cluster    /*Case 8*/
26:           else Stop                                          /*Case 9*/
27:   else Stop;                                                 /*Case 10*/
28: end;
```

Fig. 12. Procedure DeletionD.

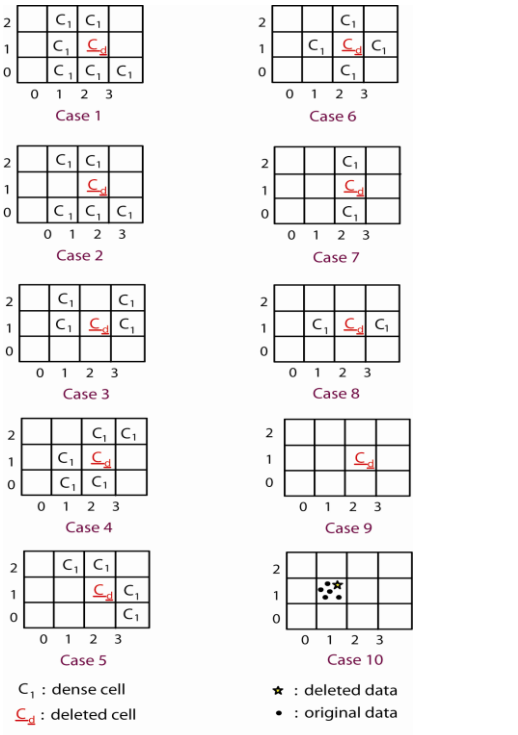


Fig. 13. The overview of 10 cases.

After inserting the 10th data point into cell (1, 1), the count of the cell is greater than δ. So, cell (1, 1) becomes dense. Because cell (1, 1) has no *CNum*, the cell can form a cluster. Moreover, the cell has outer neighboring quadcell and the count of the quadcell is greater than (1/4) δ, so it connects the outer neighboring quadcell. Therefore, this case is Case 2 of Procedure *InsertionD*.

When the 11th and 12th data points are inserted, the two data points will become outliers. Although the count of the quadcells in cell (1, 2) is greater than (1/4)δ, it has no outer neighboring cell. Therefore, due to a cluster has to be dense, so the two data points can not form a cluster. In the same way, from order 13 to order 14 are also Case 7 of Procedure *InsertionD*. After inserting the 15th data point, the quadcell in cell (0, 1) will be dense. It is Case 4 of Procedure *InsertionD*. The data points in the quadcell will be connected into cluster C_1.

In the same way as that of the above-mentioned cases, after inserting 20 data points, the result is shown in Fig. 15. There are 5 outliers in the 20 data points.

	CNum updated	right diagonal quadcell with the same *CNum*	left diagonal quadcell with the same *CNum*	up and down quadcell with the same *CNum*	left and right quadcell with the same *CNum*
Case 1	Y	Y	Y Y Y	Y	Y
Case 2	Y	Y	Y Y Y	N	N
Case 3	Y	Y	Y Y	N	XX
Case 4	Y	N	N N X	X	X
Case 5	Y	N	Y Y X	X X	X
Case 6	Y	N	N N Y	Y Y	Y
Case 7	Y	N	N N Y	Y N	N
Case 8	Y	N	N N N	N Y	Y
Case 9	Y	N	N N N	N	N
Case 10	N	X	X	X	X

N: No
Y: Yes
X: Don't care

Fig. 14. The 10 cases of data deletion.

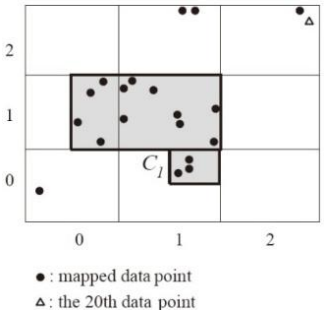


Fig. 15. The result of the insertion from Tid1 to Tid20, where δ= 8, (1/4)δ= 2.

TABLE II: GIVEN STREAM DATA POINTS, WHERE WINDOW SIZE = 30, Δ = 8.

Tid	data point	Tid	data point
1	(1.1, 1.8)	11	(1.7, 2.9)
2	(1.2, 1.9)	12	(1.8, 2.9)
3	(0.6, 1.4)	13	(1.7, 0.8)
4	(1.6, 1.3)	14	(0.9, 1.9)
5	(1.9, 1.6)	15	(0.7, 1.6)
6	(1.4, 1.8)	16	(1.8, 0.7)
7	(0.8, 1.1)	17	(2.7, 2.9)
8	(1.5, 1.4)	18	(0.2, 0.4)
9	(1.9, 1.1)	19	(1.6, 1.6)
10	(1.1, 1.4)	20	(2.8, 2.6)

IV. PERFORMANCE

In this section, we study the performance of the GDense algorithm by simulation, and make a comparison with the CDS-Tree algorithm [18]. Our experiments were performed on an Intel Pentium IV 2.66G Hz, 1GB RAM, running Windows XP Professional, and coded in Java.

The generation of data sets is controlled by a set of parameters that are summarized in Table III. We performed evaluation with these data sets which contain data points in the two-dimensional space. Clustering validation [1] refers to procedures that evaluate the results of cluster analysis in a quantitative and object fashion. The clustering validation usually uses the *execution time* and *cluster quality* to evaluate the cluster. One of the ways of measuring the quality of a clustering solution is the cluster purity. Let there be *k* clusters of the dataset *D* and the size of cluster C_j be $|C_j|$. Let $|C_j|_{class=i}$ denote number of items of class*i* assigned to cluster *j*. Given the true set of clusters (referred to as class henceforth to avoid confusion), $C_T = \{c_1, c_2,...,c_L\}$. The cluster obtained from an algorithm is $C_s = \{C_1, C_2,...,C_k\}$. The purity of this cluster is given by [19]

$$purity(Cj) = \frac{1}{|Cj|}\max_i(|Cj\,|class = i) \qquad (1)$$

The overall purity of a clustering solution could be expressed as a weighted sum of individual cluster purities

$$purity = \sum_{j=1}^{k}\frac{|Cj|}{D}purity(Cj) \qquad (2)$$

In general, the larger the value of the purity is, the better the solution is. Purity lies in the range [0, 1], with a perfect clustering corresponding to the purity value of 1 [20].

TABLE III: PARAMETERS OF THE DATA GENERATOR

Parameter	Meaning
N	The total number of data points
m	The number of cells
w	The size of the sliding window
δ	The threshold of a dense cell

We make a comparison with the CDS-Tree algorithm [18]. In our simulation, four parameters and their default settings are listed in Table IV. The synthetic datasets used in our simulation were generated by randomly creating 5 clusters on 1000 × 1000 space. We set 5 clusters which contain 2 circles and 3 ellipses. Given a circle, suppose the center of the circle locates at (a, b), so the equation of the circle is

$$(x - a)^2 + (y - b)^2 = r^2 \qquad (3)$$

Fig. 16 shows an ellipse. Moreover given an ellipse, suppose the center of ellipse locates at (*c*, *d*), so the equation of the ellipse is

$$(x - c)^2/\,h^2 + (y - d)^2/\,v^2 = 1 \qquad (4)$$

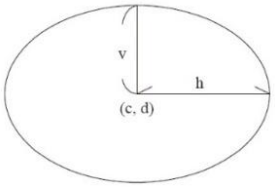


Fig. 16. The ellipse.

TABLE IV: PARAMETERS OF THE GENERATOR

Parameter	Value
N	20000...100000
m	100 * 100...200 * 200
w	200...1000
δ	8...40

Table V and Table VI show the parameters of the 5 graphs. The total number of points is changed from 20000 to 100000 in the 5 graphs, and the threshold of a dense cell is equal to 0.4 percentage of the total number of data points. We map the data points to 100 × 100 cell space. Fig. 17 shows a comparison of the purity between our GDense algorithm and the CDS-Tree algorithm in 10 times averages. The purity of the GDense algorithm is above 95%, based on the distribution of 5 clusters. The data points in the 5 clusters are created randomly. Even when the density of a cell $A_1\delta$is less than, but the cell is close to a cluster C_1, our algorithm will add such a cell A_1 to the cluster C_1. Because the points the quadcell in the cell A_1 is more than the threshold of a dense quadcell. While in the CDS-Tree algorithm, the points in such a cell A_1 will be considered as outliers. If such points are considered as outliers, the purity will be affected.

TABLE V: PARAMMETERS OF 2 CIRCLES

Circle	Center	Radius (r)
Circle 1	(150, 200)	130
Circle 2	(600, 300)	100
Circle 3	(400, 550)	60

TABLE VI: THE PARAMETERS OF 2 CIRCLES

Ellipse	Center	Horizon (h)	Vertical (v)
Ellipse 1	(100, 650)	200	150
Ellipse 2	(800, 800)	120	100

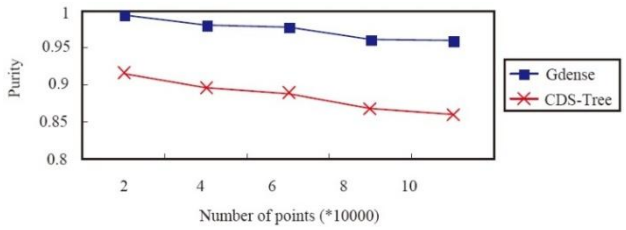


Fig. 17. A comparison of the purity under 100 × 100 grids partition.

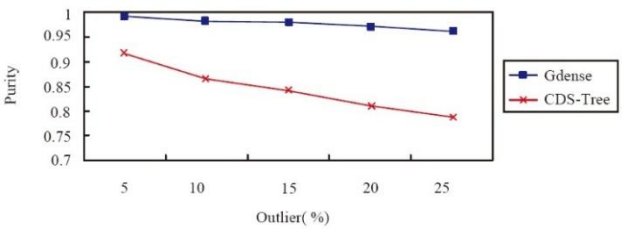


Fig. 18. A comparison of the purity with outliers.

Next, we simulate the data sets with outliers. Fig. 18 shows the comparison of the purity between the GDense algorithm and the CDS-Tree algorithm with outliers. The finer the partition is, the higher the purity is. Our GDense algorithm is under 100 × 100 grid partitions, and the CDS-Tree algorithm is under 200 × 200 grid partitions. Therefore, the threshold of the GDense algorithm is four times than the CDS-Tree. The number of points is 100000, and the window sizes is 1000. The percentage of outliers that we set is from 5% to 25%. In the CDS-Tree algorithm, as the number of outliers increases, the purity will decrease. This is because the capability of detecting outliers with finer partition is poor in the CDS-Tree algorithm. However, no matter whether the number of the outlier becomes larger or not, the GDense algorithm still can discard outliers. Therefore, the purity of the GDense algorithm is more stable than that of the CDS-Tree algorithm.

As mentioned above, we observe that there are two main conditions which affect the purity. One condition occurs, when the density of a cell is less than δ and the points in the *cellP* is close to a cluster. So, the *cellP* should be in the cluster. But when the threshold of a dense cell is set too large, the *cellP* will become another cluster. The other condition occurs, when the threshold of a dense cell is set too small. The CDS-Tree algorithm cannot detect outliers accurately. According to these simulation results, our GDense algorithm can improve the above two cases, resulting in the high purity of the results of clusters.

V. CONCLUSION

In this paper, we have proposed a new algorithm called GDense to cluster large databases in data streams. The basic idea is to use the hybrid partitions, i.e., cells and quadcells, to raise the clustering quality. Our result not only confirms that the quality of cluster produced by GDense is much better than CDS-Tree algorithm based on the grid-based approaches, but also discards noises during the mining process.

REFERENCES

[1] C. Jin, W. Qian, C. Sha, J. X. Yu, and A. Zhou, "Dynamically maintaining frequent items over a data stream," in *Proc. the 12th ACM Conf. on Information and Knowledge Management*, 2003, pp. 287–294.

[2] M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky, "Cost-Efficient mining techniques for data streams," in *Proc. the Australasian Workshop on Data Mining and Web Intelligence*, 2004, pp. 109–114.

[3] Y. Yogita and D. Toshniwa, "Clustering techniques for streaming Data-Asurvey," in *Proc. the IEEE Int. Conf. on Advance Computing Conference*, 2013, pp. 951-956.

[4] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy, "Mining data streams: A review," in *Proc. ACM SIGMOD Conf.*, vol. 34, no. 2, pp. 18–26, June 2005.

[5] M. S. B. PhridviRaj and C. V. GuruRao, "Data mining – Past, present and future – A typical survey on data streams," *Journal of Procedia Technology*, vol.12, pp. 255-263, 2014.

[6] J. Han, J. Pei, and Y. Yin, "CURE: An efficient clustering algorithm for large databases," *Information Systems*, vol. 26, no. 1, pp. 35–58, March 2001.

[7] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noises," in *Proc. the 2nd Int. Conf. on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[8] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-Based clustering over an evolving data stream with noise," in *Proc. the SIAM Conf. on Data Mining*, 2006, pp. 326–337.

[9] I. Kunttu, L. Lepisto, J. Rauhamaa, and A. Visa, "Grid-Based clustering in the content-based organization of large image databases," in *Proc. 5th Int. Workshop on Image Analysis for Multimedia Interactive Services*, 2004, pp. 21–23.

[10] A. Banerjee and S. Basu, "Topic models over text streams: A study of batch and online unsupervised learning," in *Proc. the SIAM Conference on Data Mining, Minneapolis*, April 2007, vol. 15, no. 3, pp. 437–442.

[11] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and Issues in Data Stream Systems," in *Proc. the 21th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, 2002, pp. 1–16.

[12] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for projected clustering of high dimensional data streams," in *Proc. the 30th VLDB Conf.*, 2004, pp. 852–863.

[13] J. Feigenbaum, S. Kannan, M. Strauss, and M. Viswanathan, "Testing and spot-checking of data streams," in *Proc. ACM-SIAM Symposium Conf. on Discrete Algorithms*, 2000, pp. 165-174.

[14] L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, and R. Motwani, "Streaming-Data algorithms for high-quality clustering," in *Proc. the IEEE Int. Conf. on Data Engineering*, 2002, pp. 685-694.
[15] P. Chaovalit and A. Gangopadhyay, "A method for clustering transient data streams," in *Proc. the ACM symposium on Applied Computing*, 2009, pp. 1518–15193.
[16] P. Domingos and G. Hulten, "Models and issues in data stream systems," in *Proc. ACM SIGMOD Conf.*, 2000, pp. 71–80.
[17] S. Guha, N. Mishra, and R. Motwani, "Clustering Data Streams," in *Proc. the IEEE Conf. on Foundations of Computer Science*, 2000, pp. 359–366.
[18] H. Sun, G. Yu, Y. Bao, F. Zhao, and D. Wang, "CDS-Tree: An effective index for clustering arbitrary shapes," in *Proc. the 15th Int. Workshop on Research Issues in Data Engineering: Stream Data Mining and Applications*, 2005, pp. 81–88.
[19] Q. He, K. Chang, E. Lim, and J. Zhang, "Bursty feature representation for clustering text streams," in *Proc. the SIAM Conf. on Data Mining*, 2007, pp. 26–28.
[20] V. Chaoji, M. Hasan, S. Salem, Mohammed, and J. Zaki, "SPARCL: efficient and effective shape-based clustering," in *Proc. the IEEE Int. Conf. on Data Mining*, 2008, pp. 93-102.

Ye-In Chang was born in Taipei, Taiwan, R.O.C. in 1964. She received her B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1986. She received her M.S. and Ph.D. degrees in computer and information science from Ohio State University, Columbus, Ohio, in 1987 and 1991, respectively. From August1991 to July 1999, she joined the faculty of the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. From August 1997, she has been a professor in the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. Since August 1999, she has been a professor in the Department of Computer Science and Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan. Her research interests include database systems, distributed systems, multimedia information systems, mobile information systems and data mining.

Chia-En Li received his B.S. degree in information management from I-Shou University in 2007 and his M.S. degree in Computer Science from National Pingtung University of Education in 2009. He is currently a Ph.D. student in Department of Computer Science and Engineering at National Sun Yat-Sen University. His research interests include database systems and data mining.

Shu-Yi Lin received her B.S. degree in computer science from Feng Chia University in 2007, and her M.S. degree in computer science and engineering from National Sun Yat-Sen University in2009. She is currently a system design engineer in Taiwan.