An Incremental Grid Density-Based Clustering Algorithm*

CHEN Ning[1], CHEN An[2,3], ZHOU Long-xiang[1]

[1](*Academy of Mathematics and System Sciences, The Chinese Academy of Sciences, Beijing* 100080, *China*);

[2](*Institute of Policy and Management, The Chinese Academy of Sciences, Beijing* 100080, *China*);

[3](*Software Engineering Technology Center, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*)

E-mail: anchen1@yahoo.com

http://www.amss.ac.cn

Received July 24, 2000; accepted August 16, 2001

Abstract: Although many clustering algorithms have been proposed so far, seldom was focused on high-dimensional and incremental databases. This paper introduces a grid density-based clustering algorithm——GDCA, which discovers clusters with arbitrary shape in spatial databases. It first partitions the data space into a number of units, and then deals with units instead of points. Only those units with the density no less than a given minimum density threshold are useful in extending clusters. An incremental clustering algorithm——IGDCA is also presented, applicable in periodically incremental environment.

Key words: clustering; grid; incremental algorithm

Clustering is a descriptive task which groups a set of data without a predefined class attribute to maximize the intra-class similarity and minimize the interclass similarity. Many databases are composed of millions of data with several tens even hundreds of dimensions. Unfortunately, there are few clustering algorithms applicable to high-dimensional database. Density-Based clustering[1,2] and grid clustering[3,4] can discover clusters with arbitrary shape and separate noise. DBSCAN[1] finds dense regions that are separated by low density regions and clusters together the points in the same dense region at two parameters: Eps-radius of the neighborhood of a point and Minpts-minimum number of points in the neighborhood. Clusters are then found by starting from an arbitrary point and including the points in its neighborhood into the cluster if its neighborhood satisfies the minimum density (core points). Those points (boundary points) whose Eps-neighborhood contains less than Minpts of points are marked noise. The noise points can be clustered if they belong to the Eps-neighborhood of a core point of a cluster in later process. The process is then repeated until there is no point which has not been clustered or marked noise. Since the region queries can be supported efficiently by spatial access method such as R*-tree, the average rum time complexity of DBSCAN is $O(N*\log N)$, where N is the number of points. But overlap in directory of R*-tree is increasing very rapidly with growing dimensionality, which leads to go through many paths for region queries. So, R*-tree can only adequately support an effective indexing of moderate values of dimensions so that DBSCAN has

* Supported by the National Natural Science Foundation of China under Grant No.69983011 (国家自然科学基金); the National Grand Fundamental Research 973 Program of China under Grant No.G1999035807 (国家重点基础研究发展规划 973 资助项目)

CHEN Ning was born in 1974. She received her Ph.D. degree in computer software from the Academy of Mathematics and System Sciences, The Chinese Academy of Sciences in 2001. Her research interests are data mining, information system. **CHEN An** was born in 1970. He received his Ph.D. degree in management science and engineering from the Beijing University of Aeronautics and Astronautics in 2001. His research interests are supply chain management, E-Commerce. **ZHOU Long-xiang** was born in 1938. He is a professor and doctoral supervisor of the Academy of Mathematics and System Sciences, The Chinese Academy of Sciences. His current research interests are distributed database and E-Commerce.

an almost quadratic time complexity for high-dimensional data. In this paper, we present a grid density-based clustering algorithm——GDCA by first partitioning the data space into a number of units, and then dealing with units instead of points. Only those units with the density no less than a given minimum density threshold are useful in extending clusters. Due to the sparsity property of high-dimensional data space, noise becomes more ubiquitous than in low-dimensional space. Our algorithm improves the efficiency of DBSCAN by only searching the neighbors of dense units. Besides, it has the following properties: discovery of clusters with arbitrary shape, the ability of handling noise and good efficiency on large databases.

Clustering can discover potentially useful patterns from databases. But these patterns may become old after a series of updates to the databases, which may lead to mistakes in decision support. So, it is important to keep the patterns up to date. Due to the large size of the databases and the high time complexity of clustering algorithms, it is highly desirable to perform these updates incrementally. Adopting the density-based nature of DBSCAN, an incremental algorithm[5] making the insertion and deletion of a point affects the current clustering only in the neighborhood of this point. Thus, efficient algorithms can be given for incremental insertions and deletions to an existed clustering. But this algorithm is sensitive to the updates of database. Even one addition or deletion may bring mergence or separation of clusters. And it only processes one update at a time without considering the relationship between the single updates. In fact, a merged cluster resulting from some additions may be split again due to the following deletions. Similarly, two split clusters resulting from some deletions may be merged due to the following additions. So, the strategy of dealing with single update is unnecessary and time-consuming. In this paper, we present an incremental clustering algorithm——IGDCA based on GDCA, dealing with a bulk of updates.

The rest of the paper is organized as follows: In section 1, we present a grid density-based clustering algorithm ——GDCA followed by some definitions. We introduce an incremental clustering algorithm——IGDCA based on GDCA in section 2. Lastly, we conclude the paper.

1 GDCA ——a Grid Density-Based Clustering Algorithm

Let $A=\{A_1,A_2,\ldots,A_d\}$ be a set of numeric attributes (dimensions), each dimension having a bounded, totally ordered domain. $S=A_1\times A_2\times\ldots\times A_d$ is the minimum bounding hyper-rectangle of the database, constructing a d-dimensional numeric space. $D=\{p_1,p_2,\ldots,p_N\}$ is a set of d-dimensional points, where $p_i=\{v_{i1},v_{i2},\ldots,v_{id}\}$. We partition the data space S into non-overlapping rectangle units, with the j^{th} dimension being divided into m_j intervals of equal length. Given an order to the intervals of each dimension, $\{r_1,r_2,\ldots,r_d\}$ is called the position of a unit. Assuming m is equal to m_i for all dimensions, S is divided into m^d units. Let s_j be the length of the intervals of the i^{th} dimension. A d-dimensional point $p_i=\{v_{i1},v_{i2},\ldots,v_{id}\}$ is contained in c having the position $\{r_1,r_2,\ldots,r_d\}$, if and only if $(r_j-1)\times s_j\le v_{ij}<r_j\times s_j$, $1\le j\le d$. The density of c is defined as the number of points contained in it, denoted as $N(c)$. A unit c is called non-empty if $N(c)>0$, or dense if $N(c)\ge\delta$, where δ is a density threshold. We denote the set of non-empty units and the set of dense units as $C_{ne}=\{c \mid N(c)\ge 0\}$ and $C_d=\{c \mid N(c)\ge\delta\}$ respectively. $|C_{ne}|$ can range from 1 to min (m^d,N) depending on m, N and the distribution of points. In low dimensional space, the number of non-empty units is exponential to d. In high-dimensional data space, the overwhelming majority of units are empty so that the number of non-empty units is linear to N.

Definition 1. Let c_1, c_2 be two units. The distance between c_1 and c_2 is defined as dist(c_1,c_2)=Euclidean-distance(mean(c_1),mean(c_2)), where mean(c) is the geometric center of c. Given a distance threshold τ, the neighborhood of c is defined as near(c)=$\{c' \mid c'\in C_{ne}, dist(c',c)\le\tau\}$.

Definition 2. Given τ and δ, we can define some relation between two units c and c' as follows:

(1) c' is directly density-reachable from c if $c'\in near(c)$ and $c\in C_d$;

(2) c' is density-reachable from c if there is a chain of units $c_1,c_2,\ldots,c_k$, $c_1=c$, $c_k=c'$, such that c_{i+1} is directly

density-reachable from c_i;

(3) c' is density-connected to c if there is another unit c'' such that both c and c' are density-reachable from c''.

Definition 3. A cluster C is a non-empty subset of C_{ne} with all points contained in it being assigned to C, if

(1) $\forall c,c'$: if $c\in C$ and c' is density-reachable from c, then $c'\in C$; (2) $\forall c,c'\in C$: c' is density-connected to c.

Definition 4. Let $\{C_1,C_2,\ldots,C_k\}$ be a set of clusters. C_n is defined as the set of noise units not belonging to any cluster, i.e. $C_n=\{c|c\notin C_i,1\le i\le k\}$. All points contained in C_n is assigned to noise. A clustering of D is to partition D into a set of clusters $C_1,C_2,\ldots,C_k$ and noise C_n, satisfying $C_1\cup C_2\cup\ldots\cup C_k\cup C_n=D$ and $\forall i,j\in\{1,2,\ldots,k,n\}$, $C_i\cap C_j=\varnothing$.

If c is a dense unit, then $C=\{c'|\ c'\in C_{ne}$ and c' is density-reachable from $c\}$ is a cluster. On the other hand, if c is an arbitrary dense unit of C, then $C=\{c'|\ c'\in C_{ne}$ and c' is density-reachable from $c\}$. It implies that a cluster can be determined by any dense unit it contains. Starting from an arbitrary dense unit, a cluster is created by extending to its neighbor units. This procedure is applied to the new retrieved dense units until all density-reachable units from the start unit are retrieved. Then another unclustered dense unit is selected to discover a new cluster.

1.1 Algorithm description

In general, GDCA can be divided into three steps:

Step 1. Preprocess: Map each point into the corresponding unit and stores position, density, sum of the non-empty units as well as pointers to the points using a k-d tree.

Step 2. Clustering C_{ne}: Find the cluster of units based on density-reachable and density-connected. Initially, all units are identified as "unclustered". To find a cluster, GDCA starts from startc using a breadth-first search. If a neighbor unit of startc is unclustered, it is identified as the current cluster. Moreover, if it belongs to C_d, it is added to the end of seeds. Then currentc is deleted from the seeds. Next, the first unit of seeds is extracted to perform the same procedure. When all density-reachable units in the cluster had been visited, a cluster is discovered. Consequently, the procedure repeats by starting with an unclustered dense unit until there is no unclustered dense unit.

Step 3. Map each point in D to the cluster: If $p\in c$, and $c\in C_i$, we identify p as C_i.

1.2 Time complexity

DBSCAN computes the neighbor points for every point, since it does not know whether a point is a core point or a boundary point without searching its neighborhood. In fact, the number of core points is much less than that of all points and it is unnecessary to search the neighborhood of the boundary points because no point can be density-reachable from them. In GDCA, we partition the data into units and deal with units instead of points. If the unit is small enough, all points in a unit are close enough to each other so that a unit can be considered as a subcluster. Then we get the dense units (similar to core points) in the pre-process. Only those dense units are extended during the clustering step. So, the time complexity of GDCA is $O(N+|C_d|\ \log\ |C_{ne}|)$. GDCA has the following properties: (1) GDCA only stores the non-empty units, the number of which is linear to N as the dimensionality increases so that GDCA is applicable for high dimensionality; (2) GDCA only searches the neighbor units of dense units, the number of which is much smaller than that of the non-empty units; (3) If the distance of a unit to the nearest dense unit is more than τ, then it will not be density-reachable from any dense units. After pruning such units, k-d tree is much smaller than the original tree, that will improves the efficiency of queries.

2 IGDCA——an Incremental Algorithm of GDCA

Assuming we have got a set of clusters using GDCA. When new data are inserted or original data are deleted, we must modify the existed clusters to reflect the changes. Let Δdb be the inserted data to D, and ∇db be the deleted data from D. The new database $D'=D\cup\Delta db-\nabla db$. Since updates can be seen as a series of insertions and deletions,

we only consider insertions and deletions. A unit is called updated if at least one point is inserted to or deleted from it. Since an inserted or deleted point only affects the density of unit in which it is contained, we consider units instead of points. The new density of an updated unit is denoted as $N'(c)$.

After some insertions, non-dense units may become dense implying that new density connection may be established between two units c and c' which were not density-reachable from each other, i.e., a chain $c_1,c_2, \ldots,c_n$, $c_1=c$, $c_n= c'$ with c_{i+1} directly density-reachable from c_i. In the chain, at least one unit c_j, $1\leq j\leq n$ is non-dense in D, but becomes dense in D'. After some deletions, dense units may become non-dense implying that old density connection may be removed between c and c' which were density-reachable from each other, i.e., a chain c_1, $c_2,\ldots,c_n$, $c_1=c$, $c_n=c'$ with c_{i+1} directly density-reachable from c_i. In the chain, at least one unit c_j ($1\leq j\leq n$) is dense in D, but becomes non-dense in D'. Note that if insertions and deletions occur in the same unit, their affection may counteract to each other.

If an updated unit keeps its density property, i.e., it is dense or non-dense in both D and D', it won't result in any change on the cluster state of other units. Thus, we only consider those units which change the dense property, denoted as $UC_d = \Delta C_d \cup \nabla C_d$, where $\Delta C_d = \{c \mid N'(c)\geq \delta$, and $c\notin C_d\}$ and $\nabla C_d = \{c \mid N'(c)<\delta$, and $c\in C_d\}$. The affected units of $c\in \Delta C_d$ are those units density-reachable from c in D', i.e., $C\_eff(c)=\{c' \mid c'$ is density-reachable from c in $D'\}$. Similarly, $C\_eff(c) = \{c' \mid c'$ is density-reachable from c in $D\}$ represents the affected units that may change their cluster identifier because of $c\in \nabla C_d$.

2.1 Algorithm description

Step 1. Discover updated units: A data structure C_upd is used to keep the updated units. Each unit is stored as one element using position as the key. After $\Delta db \cup \nabla db$ have been scanned, we find the units whose density property change. For each $c\in C\_upd$, if $c\in C_{ne}$, it is marked as "old", and identified as its original cluster identifier. Otherwise it is marked as "new" and identified as "unclustered". We then calculate the new density of c. If $c\notin C_d$ and becomes dense (new dense unit), then c is put into ΔC_d; If $c\in C_d$ and becomes non-dense (lost dense unit), then c is put into ∇C_d. Furthermore, we update the information of c. Specially, if $c\notin C_{ne}$, a new entry is created; If the new density of c decreases to 0, the corresponding entry is deleted. ΔC_d and ∇C_d store all new dense units and lost dense units respectively, arranged by the increasing order of Clid. At last, $C_d = C_d + \Delta C_d - \nabla C_d$, $C_{ne} = C_{ne} \cup C\_upd$.

```
Struct C_upd
{ position // position of units
  insert_number // the number of inserted points to the unit
  delete_number // the number of deleted points from the unit
  insert_sum // the sum of all inserted points
  delete_sum // the sum of all deleted points
  mark // "new" if it is a new unit, "old" otherwise
  Clid // the ClusterId of the unit
  Pointers [insert_number + delete_number]
    {mark // "insertion" or "deletion"
     point // the inserted or deleted points }
}
```

Step 2. Modifying affected clusters: Cluster new points and modify the existed clusters.

Case 1. $\Delta C_d = \nabla C_d = \varnothing$: There is neither new dense unit nor lost dense unit so that none of existing units will change the cluster. For each new non-empty unit c, if the distance between c and its nearest dense unit c' is no more than τ, c is identified as the same cluster as c', else c is identified as "noise". If c is an existing unit, every new point

in c is added to the cluster of c and every deleted point is removed from the corresponding cluster. The details are described as follows.

```
Procedure Modify-Cluster()
{For i:=1 to C_upd.size do
   {c:= C_upd.get (i);
    If c. mark = "old" then AssignClid (c, c.Clid);
    Else {c':= C_d.NearestQuery (c);
          If d (c, c') ≤ τ, then AssignClid (c, c'.Clid);
          Else AssignClid (c, "noise");}
   Delete all p∈ ∇db from the original cluster;}
}
```

Case 2. $\Delta C_d \neq \varnothing$, $\nabla C_d = \varnothing$: New density-reachable chain may be established but none is removed. Three cases are considered:

- Creation: A new cluster is created by a new dense unit which is not density-reachable from any old dense unit;
- Absorption: Noise units and new non-empty units are absorbed into an existed cluster if they are density-reachable from an dense unit of the cluster;
- Mergence: Several clusters may be merged if dense units of different cluster are density-reachable from each other. A principle of merge is that all merged clusters should change their identifiers to the minimum identifier. For example, if one merge relates to a set of clusters $C_1,C_2,\ldots,C_k$, and $C_1.Clid<C_2.Clid<\ldots<C_k.Clid$, then all units of C_i, $2\leq i\leq k$, change their *Clid* to $C_1.Clid$. We rearrange the units in ΔC_d by the increasing order of their Clid. Those units with "unclustered" or "noise" are identified as the minimum *Clid* of their neighbor dense units, otherwise they are added to the last of ΔC_d.

Theorem 1. Let c be the first unit of ΔC_d at the end of the k^{th} call of ExpandCluster (), c will not be reassigned as other *Clid* during the r^{th} ($r>k$) *ExpandCluster* ().

Proof. The principle of merge is to change the *Clid* of the merged clusters to the minimum one and keep ΔC_d as the increasing order of *Clid*. At any time, the first element has the minimum *Clid* in ΔC_d. If c is reassigned to other *Clid* during the r^{th} ($r>k$) call, then c is density-reachable from c' and $c'.Clid < c.Clid$. That is, there exists chain $c_1,c_2,\ldots,c_n$, $c_1=c'$, $c_n=c$ with c_{i+1} directly density-reachable from c_i. In the chain, there must exist $c_j\in\Delta C_d$, $1\leq j\leq n$. Otherwise c is density-reachable from c' and then c has been reassigned to c'. *Clid* before the k^{th} call. From the chain, we can select $c_j\in\Delta C_d$, and $c_j.Clid<c.Clid$. It is paradox with the assuming that c is the first element of ΔC_d. □

The process of modifying clusters is similar to that of finding clusters described in section 3. Clustering () is used to create new clusters and modify the existing clusters so that all units of the merged clusters can change their identifiers to the minimum as described in Theorem 1. Firstly, all units density-reachable from a starting c are retrieved, in which those units with *Clid* more than *c.Clid* are identified as *c.Clid* (mergence). We also assign c.Clid to the "unclustered" and "noise" units (absorption). If c is an "unclustered" or "noise" dense unit, a new cluster is created (creation).

```
Procedure Clustering (C_ne, C_d, τ)
{   For i:=1 to | ΔC_d | do
     { startc = C_d.get(i);
  If c.Clid in {"unclustered","noise"} then
  {ClusterId := NextId (ClusterId); AssignClid (c, ClusterId); } // create a new cluster
  ExpandCluster (ΔC_d, startc, startc.Clid, τ); // modify all units density-reachable from startc
   }}
```

```
Procedure ExpandCluster (ΔC_d, startc, ClusterId, τ)
{Seeds.append (startc);
 While Seeds<> Empty do
 { currentc := Seeds.first ();
   result := C_ne.RegionQuery (currentc, τ);// return the neighbor units of currectc
  For i from 1 to results.size do
  { resultc := result.get (i);
   If resultc.Clid in {" unclustered"," noise" } then
     AssignClid (resultc, ClusterId);
   Else If resultc.Clid > currentc.Clid then
      {AssignClid (resultc, ClusterId);
       MarkEquivalent (resultc.Clid, currentc.Clid);} // make the merged clusters
   If resultc ∈ C_d then Seeds.append (resultc);
   If resultc ∈ ΔC_d  then ΔC_d.delete (resultc);
  } Seeds.delete (currentc);
} }
```

Case 3. $\nabla C_d \neq \varnothing$: There are some lost dense units which may result into the split of Clusters. For each unit $c \in \nabla C_d$, we have to check whether or not the neighbor units of c are density-connected by other dense units in the cluster. It is an expensive procedure, equivalent to test all dense units of the cluster. For this case, we call the original GDCA to get the modified cluster. Fortunately, insertion is much more frequent than deletion in many applications such as transaction database, Web access log database, and data warehouse. So, IGDCA is quite useful for such applications.

2.2 Efficiency analysis

The larger the incremental data is, the longer it would take reading the incremental data and updating the information of related units so that the speed gain would slow down. Moreover, the second case would possibly lead to creation or mergence of clusters, which is more time-consuming than the first case. The more updated units resulted from incremental data, the more effected units would change their cluster identifiers so that IGDCA would obtain less gain compared to running GDCA on the whole database. Experiments are performed to support this analysis.

3 Conclusions

Recently, clustering has been recognized as a primary data mining method for knowledge discovery in spatial databases. In this paper, we introduce a grid density-based clustering algorithm——GDCA. In order to make GDCA applicable in periodically incremental environment, we also present an incremental clustering algorithm——IGDCA, dealing with a bulk of updates instead of single update. Future work includes automatic determination of thresholds and improvement on the efficiency of deletion.

Acknowledgement The authors gratefully acknowledge the support of K.C. Wong Education Foundation, Hong Kong.

References:

[1] Ester, M., Kriegel, H.-P, Sander, J. *et al*. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Simoudis, E., Han, J., Fayyad, U.M., eds. Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining. Portland, Oregon: AAAI Press, 1996. 226~231.

[2] Zhou, B., Cheung, D., Kao, B. A fast algorithm for density-based clustering. In: Zhong, N., Zhou, L., eds. Methodologies for Knowledge Discovery and Data Mining, the 3rd Pacific-Asia Conference. Berlin: Springer, 1999. 338~349.

[3] Agrawal, R., Gehrke, J., Gunopolos, D., Raghavan, P. Automatic subspace clustering of high dimensional data for data mining application. In: Haas, L.M., Tiwary, A., eds. Proceedings of the ACM SIGMOD International Conference on Management of Data. Seattle, Washington, USA: ACM Press, 1998. 94~105.

[4] Schikuta, E. Grid clustering: an efficient hierarchical clustering method for very large data sets. In: Proceedings of the 13th International Conference on Pattern Recognition. IEEE Computer Society Press, 1996. 101~105.

[5] Ester, M., Kriegel, H.-P, Sander, J. *et. al*. Incremental clustering for mining in a data warehousing environment. In: Gupta, A., Shmueli, O., Widom, J., eds. Proceedings of the 24th International Conference on Very Large Data Bases. New York: Morgan Kaufmann Publishers Inc., 1998. 323~333.

基于密度的增量式网格聚类算法

陈 宁[1], 陈 安[2,3], 周龙骧[1]

[1](中国科学院 数学与系统科学研究院,北京 100080);

[2](中国科学院 科技政策与管理科学研究所,北京 100080);

[3](中国科学院 软件研究所 软件工程技术研究开发中心,北京 100080)

摘要: 提出基于密度的网格聚类算法 GDCA,发现大规模空间数据库中任意形状的聚类.该算法首先将数据空间划分成若干体积相同的单元,然后对单元进行聚类.只有密度不小于给定阈值的单元才得到扩展,从而大大降低了时间复杂性.在 GDCA 的基础上,给出增量式聚类算法 IGDCA,适用于数据的批量更新.

关键词: 聚类;网格;增量算法

中图法分类号: TP311 文献标识码: A