An Effective Clustering Algorithm in Large Transaction Databases*

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

[1](Economics and Mathematics Institute, The Chinese Academy of Sciences, Beijing 100080, China);
[2](Management School, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)
E-mail: anchen1@yahoo.com
http://www.math.ac.cn
Received July 28, 2000; accepted December 19, 2000

Abstract: Clustering of transactions can find potential useful patterns to improve the product profit. In this paper, a two-step clustering algorithm——CATD is proposed, applicable in large transaction databases. First, the database is divided into partitions in which transactions are partially clustered into a number of subclusters. A hierarchical clustering algorithm is used to control the distance between these subclusters. In the global clustering, a k-medoids clustering algorithm is performed on the subclusters to get a set of k global clusters and identify noise. The algorithm is feasible for large databases because it only scans the original databases once and the clustering process can be performed in main memory due to the partitioning scheme and the support vector representative of subclusters.

Key words: data mining; clustering; singie linkage

In a given transaction database, each transaction contains a set of items purchased by the customer. Clusters of customers with similar purchase patterns can be used to characterize the different customer groups, and these characterizations are useful in targeted marketing and advertising such that specific products are directed towards specific customers based on their profiles. Similarly, Web-based organizations often generate and collect large volumes of data in their daily operations. Such information is generally gathered automatically by Web servers and collected in access logs. The access log is transformed into a transaction database. Clustering of client transactions can be used to dynamically present users with customized information or with targeted advertising. Clustering can also be applied to group web documents with similar term vector to improve the resource management and search efficiency.

Recently, clustering has been recognized as a primary data mining method for knowledge discovery in spatial databases. Various methods have been studied in considerable detail. Most previous work in clustering focused on numeric data whose inherent geometric properties can be exploited to naturally define distance functions between points. But seldom was it focused on transaction databases in which the attributes are binary type. A simple approach of transforming the transaction database into a high-dimensional data space is not feasible because a

---

* **CHEN Ning** was born in 1974. She received her Ph. D. degree in computer software from Mathematics Institute, The Chinese Academy of Sciences in 2001. Her research interests are data mining and knowledge discovery. **CHEN An** was born in 1970. He received his Ph. D. degree from Economics and Management School, Beijing University of Aeronautics & Astronautic in 2001. His research interests are supply chain management, operational research and data mining. **ZHOU Long-xiang** was born in 1938. He is a professor and doctoral supervisor of Mathematics Institute, The Chinese Academy of Sciences. His current research interests are distributed database, multimedia database, data mining and data warehouse.

transaction database always contains hundreds even thousands of items so that the dimensionality of the transformed database is very large. As we know, most clustering algorithms for low-dimensional databases do not work well for high-dimensional database due to the curse of dimensionality[1]. In reality, transaction databases usually contain gigabytes, terabytes, or petabytes of data. The huge size of databases requires efficient and effective data access. It also obligates methods to scale up clustering. Clustering algorithms should be fast, scale well with the database growth. Partitioning is an effective method for large databases. The partitioning scheme is employed to ensure that the input set to clustering algorithm is always in main memory though the database cannot fit in memory. If the partition size is chosen to be smaller than the main memory size, then the input data for clustering are always main memory resident.

Reference [2] discusses the similarity measures for large transaction databases. Reference [3] maps the relationship among items into a weighed hypergraph. Although the item clustering result can be used to cluster transactions, it is based on the results of the former. It makes an assumption that the item clusters have no overlap among them. This may not be true in practice since transactions in different clusters may have a few common items. Reference [4] first proposes the idea of partially clustering with each partition achieving a set of preclustering schemes. The preclustering algorithm is incremental and scans the entire data set. CURE[5] first partitions the sample into p partitions, then partially clusters each partition until the final number of clusters in each partition reduces to a given value. A clustering pass is run on the partial clusters for all the partitions resulted from the first pass. The problem is the suitable number of clusters in each partition may be much diverse because of the different distributions of different partitions. If a partition contains points from all clusters, the suitable value of partial clusters should be large. On the contrary, if a partition contains points from limited clusters, the suitable value of partial clusters should be small. We must ensure that the number of partial clusters for each partition is sufficiently large compared with the number of desired clusters so that even though each partition contains fewer points from each cluster, the closest points merged in each partition generally belong to the same cluster and do not span across clusters. Thus, it can ensure that partitioning does not adversely impact clustering quality. To avoid storing all the points in clusters for the second pass, CURE stores only the representative points for each cluster input to the second pass because it relies on the representative points for each cluster. Thus it reduces the input size for the second clustering pass and ensures that it fits in the main memory. However, the generation of representative points relies on vector operations and therefore cannot be used in transaction databases.

In this paper, we focus on transaction clustering for transaction databases. We present a two-step algorithm for clustering transactions of similar purchase patterns. We first divide the original database into a number of partitions each of which can be stored in memory. One partition is read once a time and the transactions in the partition are partially clustered into a variety of subclusters using a hierarchical agglomerative clustering algorithm. The distance between transactions is defined as the Jaccard coefficient and the distance between clusters is defined as single linkage. A maximum distance threshold is used as the termination condition and different numbers of subclusters can be obtained from different partitions. In the procedure, the statistical information of subclusters is accumulated simultaneously. A support vector can be generated from every subcluster at the end of the partial clustering. It contains the support values of all items and represents the distribution of items in the subset. In the multidimensional space of the support vectors, a global clustering pass is performed on the subclusters and a set of k clusters is generated. As the support vector space is non-coordinate, some operations such as addition, subtraction of vectors, and getting mean of a cluster to form condensed representations of clusters and to reduce the time and space requirements of clustering problem lose their meaning. We adopt a k-medoids algorithm on the subclusters in which only distance operation is used. Experiments demonstrate that our algorithm is feasible for large databases.

The rest of the paper is organized as follows: In Section 1, we present a two-step algorithm for clustering

transactions. Some experimental results are shown in Section 2 and Section 3 concludes the paper.

1 Algorithm Description

In this section, we propose a two-step clustering on transactions. We first introduce the overview of our algorithm, after that a detailed description of each step and the time and space complexity analysis are given. The overview of our algorithm is seen in Fig. 1.

Problem Statement:

Input: A transaction database $D=\{T_1,T_2,\ldots,T_N\}$ and a set of items $I=\{i_1,i_2,\ldots,i_m\}$ present in D.

Output: A clustering of $C=\{C_1,C_2,\ldots,C_k,\text{Noise}\}$, where C_i is a set of transactions, satisfying $C_1\cup C_2\cup\ldots\cup C_k\cup\text{Noise}=D$ and $\forall C_i,C_j\in C,C_i\cap C_j=\varnothing$.

1.1 Overview: clustering algorithm for large transaction databases (CATD)

Step 1. Partitioning: Divide the transaction database D into p partitions: $\{D_1,D_2,\ldots,D_p\}$. A large number of partitions need more time on data reading and too small number will make some partitions not fit in memory. In general, the more transactions are contained in the database and average items in transactions, the more partitions are necessary. We can determine a suitable threshold for partitions according to the size of databases and main memory.

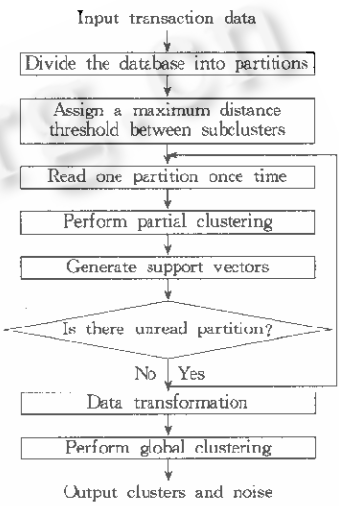

Fig. 1 Overview of CATD

Step 2. Partial clustering: Read each partition $D_i(i=1,\ldots,n)$ in turn and perform partial clustering for the partition. The support vector of each subcluster is also calculated as its representative. Assuming the number of subclusters in D_i is n_i, the result of local clustering in D_i is a set of subclusters: $\{I_1,I_2,\ldots,I_{n_i}\}$.

Step 3. Data transformation: After every partition is processed, the whole database D is primarily partitioned into a set of subclusters $\{I_1,I_2,\ldots,I_n\}$, where $n=\sum_{i=1}^{p}n_i$. Each subcluster is mapped into a vector in the transformed database with the items as the set of attributes.

Step 4. Global clustering: The distance between two subclusters is defined as the metric distance of their support vectors. We use a modified CLARANS algorithm on the subclusters and get a set of k clusters. Each cluster can be characterized as the support vector of the medoid.

1.2 Partial clustering

Definition 1. Given two transactions T_i and T_j, the distance between them is defined as the Jaccard coefficient: $\text{dist}(T_i,T_j)=1-\dfrac{|T_i\cap T_j|}{|T_i\cup T_j|}$.

The definition has the following properties:

(1) Value region: $\text{dist}(T_i,T_j)\in[0,1]$

(2) Reflexive: $\text{dist}(T_i,T_j)=0$ iff $T_i=T_j$

(3) Symmetric: $\text{dist}(T_i,T_j)=\text{dist}(T_j,T_i)$

(4) Maximum value: $\text{dist}(T_i,T_j)=1$ iff $T_i\cap T_j=\varnothing$

(5) Triangle property: $\text{dist}(T_i,T_j)\leqslant\text{dist}(T_i,T_k)+\text{dist}(T_j,T_k)$ *

* A similar inequality is seen in Ref. [6].

Definition 2. In a set of n transactions, T_j is called a nearest neighbor of T_i denoted as $T_j=NN(T_i)$, if $\forall k\in[1,n]$, $\mathrm{dist}(T_i,T_j)\leqslant\mathrm{dist}(T_i,T_k)$.

Definition 3. In a set of transactions, T_i and T_j are called a nearest neighbor pair (NNP) if $T_i=NN(T_j)$, and $T_j=NN(T_i)$, that is, $\forall k\in[1,n]$, $\mathrm{dist}(T_i,T_j)\leqslant\mathrm{dist}(T_i,T_k)$ and $\mathrm{dist}(T_i,T_j)\leqslant\mathrm{dist}(T_j,T_k)$.

Definition 4. Let C_i and C_j be two clusters of transactions, the distance between them is defined as the single linkage distance, that is the minimum distance of the pair of individuals, where all pairs consisting of one individual from each cluster are considered, i.e., $\mathrm{dist}(C_i,C_j)=\min\{\mathrm{dist}(x,y)\mid x\in C_i,y\in C_j\}$. Then Definition 2 and Definition 3 can be extended to clusters.

Definition 5. In a set of n clusters, C_j is called a nearest neighbor of C_i denoted as $C_j=NN(C_i)$, if $\forall k\in[1,n]$, $\mathrm{dist}(C_i,C_j)\leqslant\mathrm{dist}(C_i,C_k)$.

Definition 6. In a set of clusters, C_i and C_j are called a nearest neighbor pair (NNP) if $C_i=NN(C_j)$, and $C_j=NN(C_i)$, that is, $\forall k\in[1,n]$, $\mathrm{dist}(C_i,C_j)\leqslant\mathrm{dist}(C_i,C_k)$ and $\mathrm{dist}(C_i,C_j)\leqslant\mathrm{dist}(C_j,C_k)$.

In this step, we use a hierarchical agglomerative clustering algorithm on transactions in a partition. Considering the different distributions of transactions in different partitions, it is more reasonable to set the maximum distance as the common threshold than the number of subclusters for all partitions. The maximum distance of the merged clusters is used as the termination condition and different numbers of subclusters can be obtained from different partitions. Thus, we obtain a small number of subclusters if the transactions are close to each other in a partition, otherwise we get a large number of subclusters. We start with a disjoined set of clusters, placing each transaction in an individual cluster. A heap Q is used to keep each cluster, its nearest neighbor, the minimum distance between them, the number of transactions, and the support vector in one entry (shown in Fig. 2). The entries are arranged in the heap in the increasing order of the minimum distances. Pairs of clusters are then successively merged until the distance of all clusters is more than a given maximum distance threshold: max_distance. According to the definition of distance between clusters, the nearest neighbor search is efficient. Assuming two clusters u and v are merged, for a non-merged cluster x, if its nearest neighbor is not u or v, it does not change. Otherwise, its nearest neighbor changes to w while other information does not change. The nearest neighbor of the new merged cluster can be got by comparing the distances between all other clusters. We stop merging subclusters in a partition if the distance between the nearest pairs of clusters to be merged next increases above max_distance. In partial clustering, the statistical information such as the number of transactions and the support count of items, is accumulated in the process of merge. The support of an item is its support count divided by the number of transactions. Thus, the support vector of each subcluster is generated at the end of local clustering procedure. The choice of a suitable value for max_distance is important because a big value probably causes objects from different clusters to belong to the same subcluster. On the contrary, a small value results in too many subclusters to be stored in the memory. At the beginning, a small value for max_distance threshold is used. If too many subclusters are generated at the end, partial clustering will be repeated with a bigger max-distance until the number of subclusters is small enough that the global clustering can be processed in memory.

```
Struct ClusterInfo{
ClusterId: cluster number
Nearest: the nearest neighbor cluster
Mindist: the distance between the cluster and its nearest neighbor
Number: the number of transactions in the cluster
Vector: the support vector
}
```

Fig. 2 Data structure of entry in heap

In traditional agglomerative clustering, the pair of clusters merged at each step is the ones between which the

distance is the minimum. Since only two clusters are merged in each iteration, the number of clusters decreases by one after each iteration. In our algorithm, the clusters merged at each step are the nearest neighbor pairs if the distance between them is no more than max_distance. Since each partition can be stored in memory, partial clustering is processed very efficiently. The detail of the algorithm is given in Fig. 3.

```
Procedure Partial-Clustering ()
Input: D_i={T_1,T_2,...,T_n}: a partition of D;
    max_distance: the maximum distance between two merged subclusters;
Output: a set of subclusters: {I_1,I_2,...,I_n};
{Initialize: C={C_i|1≤i≤n}, C_i={t_i}; // place each transaction in a cluster
For each C_i∈C do
{FindNearest(C_i); // find the nearest neighbor for each cluster in C
Assign C_i.vector: {v_1,v_2,...,v_m}={0,0,...,0};
For each item contained in C_i do
Add 1 to the corresponding bit of C_i.vector;
Insert Q(C_i, C_i.nearest, C_i.mindist, 1, C_i.vector);
}
u:=extract_first(Q); // extract the pair of clusters with the minimum distance
mergedist:=u.mindist;
While mergesim≤max_distance do {
// If the minimum distance is above the given threshold, the algorithm terminates.
For each u∈C do
v:=nearest(u);
If nearest(v)=u and u.mindist≤max_distance then // extract the nearest neighbor pairs
{w:=merge(u,v);
 FindNearest(w); // calculate the nearest neighbor of the new cluster
 w.number=C_i.number+C_j.number;
 w.vector=C_i.vector+C_j.vector;
 Insert Q(w, w.nearest, w.mindist, w.number, w.vector);
 For all x∈C-{u,v} do
 If x.nearest is either u or v then
 Update Q(x).nearest to w;
}
u:=extract_first(Q); // extract the pair of clusters with the minimum distance
mergedist:=u.mindist;
}
For each C_i∈C do C_i.vector=C_i.vector/C_i.number;
}
Procedure FindNearest(C,u) // Find the nearest neighbor of u in a set of clusters C
{mindist:=+Infinity;
For all v∈C-u do {
dist:=+Infinity;
For all t∈u, t'∈v do
If dist(t,t')≤dist then dist=dist(t,t');
 }
If dist<mindist then {mindist:=dist; nearest:=v; }
Return nearest, mindist;
}
```

Fig. 3 Algorithm of partial clustering

Lemma 1. Assuming C_i and C_j become NNP clusters at step k of partial clustering and $dist(C_i,C_j)\leq min\_distance$, they will be merged at some step before the partial clustering terminates.

Proof. First, the pair of clusters in the first entry of the heap Q is always NNP. If C_i becomes the first entry of the heap Q at step k, that is the distance between C_i and its nearest neighbor C_j is minimum, then they must

be NNP. Otherwise, we assume $NN(C_j)\neq C_i$, there exists a cluster C_k, and $k\neq i$, satisfying dist$(C_k,C_j)<$dist(C_i,C_j). Thus, C_i should not be the first entry of Q at step k because dist(C_i,C_j) is not the minimum.

Next, we prove NNP is still NNP after some merge of other clusters. Assume C_i and C_j become NNP at step k, that is C_i is the nearest neighbor of C_j, and C_j is the nearest neighbor of C_i. Let B be a cluster merged by two clusters B_1 and B_2 at step j, $j\geqslant k$. Because we use single linkage as the distance measure between two clusters, dist$(C_i,B)=$min (dist(C_i,B_1), dist(C_i,B_2)). Since dist$(C_i,C_j)\leqslant$dist(C_i,B_1) and dist$(Ci,C_j)\leqslant$dist(C_i,B_2), then dist$(C_i,C_j)\leqslant$dist(C_i,B). That is, $C_j=NN(C_i)$. we also have $C_i=NN(C_j)$. That is, C_i and C_j are still NNP clusters in later steps. Because dist$(C_i,C_j)\leqslant$min_distance, C_i and C_j will be merged at a step. □

Lemma 1 implies that we can merge NNP clusters at the step in which they appear no matter whether they are the first entry of Q.

1.3 Global clustering

Definition 7. Let the support vector of subcluster I_i be $V_i=\{v_{i1},v_{i2},\ldots,v_{im}\}$, the distance between two clusters I_i and I_j is defined as the metric distance between their support vectors: $d(I_i,I_j)=d(V_i,V_j)$, where the metric distance satisfies:

(1) Value region: $d(I_i,I_j)\geqslant 0$

(2) Reflexive: $d(I_i,I_j)=0$ iff for $1\leqslant i\leqslant m$, $v_i=v_j$

(3) Symmetric: $d(I_i,I_j)=$dist(I_j,I_i)

Definition 8. Given two clusters C_p and $C_q(p\neq q)$, the intra-distance within C_p is defined as the maximum distance between objects and the medoid of C_p, and inter-distance between C_p and C_q is defined as the distance between two medoids. That is,

$$\text{Intra-dist}(C_p)=\max_{i=1}^{|C_p|}d(x_i,m_p)$$

$$\text{Inter dist}(C_p,C_q)=d(m_p,m_q)$$

A clustering C is perfectly clusterable if $\forall\ C_p, C_q(p\neq q)$, Inter-dist $(C_p,C_q)>$Intra-dist(C_p) and Inter-dist$(C_p,C_q)>$Intra-dist (C_q). It means the distance between different clusters is always larger than maximum distance between objects within a cluster.

At the end of partial clustering, we calculate the support of items for subclusters. Thus, each subcluster is represented by its support vector. The distance between subclusters is defined as the metric distance of the support vectors. The most popular metric distance is Minkowski metrics, defined as $d_p(v_i,v_j)=\left(\sum_{k=1}^{m}|v_{ik}-v_{jk}|^p\right)^{1/p}$, where v_i, v_j are two m-dimensional vectors. Its special forms include Euclidean distance $(p=2)$ and Manhattan distance $(p=1)$. We perform a modified CLARANS algorithm on the subclusters and get k clusters. Initially, an arbitrary set of k objects is set to be the current medoid-set and a fixed number of iterations are performed. In each iteration, a random neighbor of the current medoid-set is got by changing one medoid to a non-medoid. The new medoid-set is set to be the current node if it results in better clustering and the process is started again. If the current medoid-set has already been compared with the maximum number of the neighbors and is still of the lowest cost, the current clustering produces a local optimum. If the local optimum is found, it starts with the new randomly selected medoid-set in search for a new local optimum. Our algorithm is based on perfectly clusterable criterion: if the inter-distance between two clusters is no more than the intra-distance within a cluster, the two clusters should be merged. That is, if the distance between an object and its nearest medoid is no less than the minimum inter-distance between the medoid and any other medoid, the object is identified as noise. The most commonly used cost function is $\sum_{i=1}^{k}\sum_{x_j\in m_i}d(x_j,m_i)$. The detail of the algorithm is given in Fig. 4.

```
Procedure Global-Clustering ()
Input: a set of subclusters: {I1,I2,...,In} represented by their support vectors
Output: k clusters and noise
{
Initial numlocal and maxneighbor;
i=1; j=1;
mincost=+Infinity;
Select a random set of k medoids: {m1,m2,...,mk} from I to current;
For i=1 to numlocal do {
While j≤maxneighbor do {
Get a new set of medoids by changing one medoid of current to a non-medoid;
If Cost (new)<Cost (current) then {set current to new; j=1;}
}
If j>maxneighbor then
If Cost (current)<mincost then {mincost=cost; best=current;}
}
}
Procedure Cost(s)
Input: s: a set of medoids
Output: the cost of s
{For p:=1 to k do{
max_dist(mp):=+Infinity;
For q:=1 to k do
If d(mp,mq)<max_diet(mp) then max_dist(mp):=d(mp,mq);
 }
Cost:=0;
For all objects x do
 {Find the nearest medoid mi of x;
 If d(x,mi)<max_dist(mi) then
 Assign x to mi;
 Else assign x to Noise;
 Cost:=Cost+d(x,mi); //calculate the cost of clusters;
 }
Return Cost;
}
```

Fig. 4 Algorithm of global clustering

1.4 Complexity analysis

Let the number of transactions in database D be N, the number of items be m and the partition number of d be p. Partial clustering algorithm is based on the inter-object distances and on finding the nearest neighbors of objects. The time complexity is $O(N^2)$ at worst, and $O(\log N\times N^2)$ on the average if the number of data is N[7]. So the average time complexity and the worst time complexity of one partition are $O\left(\log\frac{N}{p}\times\left(\frac{N}{p}\right)^2\right)$ and $O\left(\frac{N^2}{p^2}\right)$ respectively. Thus, the average time complexity and the worst time complexity of partial clustering are $O\left(\log\frac{N}{p}\times\frac{N^2}{p}\right)$ and $O\left(\frac{N^2}{p^2}\right)$ respectively. The space complexity of partial clustering is $O\left(\frac{N}{p}\times m\right)$ since we store the support vector for each cluster.

In global clustering, the time complexity is $O(n^2)$ where n is the number of subcluters resulted from the partial clustering. After partial clustering, the number of subclusters taking part in global clustering is much less than that of the original databases. So the efficiency of global clustering on subclusters is much higher than direct clustering on the original database. The space complexity of global clustering is determined by the number of subclusters generated in partial clustering. By storing the support vector as representative for each cluster, we can ensure the input data of global clustering fit in the main memory. Table 1 shows the time and space complexities of three clustering algorithms. Obviously, CATD has a clear advantage over others on disk I/O.

Table 1 Complexity comparison

Algorithms	Time complexity (average)	Space complexity
RNN-CLINK	$O(\log N\times N^2)$	$O(N)$
CLARANS	$O(N^2)$	$O(N)$
CATD	$O\left(\log\frac{N}{p}\times\frac{N^2}{p}+n^2\right)$	$O\left(\frac{N}{p}\times m\right)$

2 Performance

In this section, we carry out some experiments to evaluate the performance of CATD. We compare its performance with hierarchical and partitional clustering because of the characteristics of transaction data. All the experiments are performed on a Pentium II PC with 300MHZ, 2.0G disk and 64M main memory.

2.1 Synthetic data

We first determine the numbers of transactions $|D|$ and items $|I|$. The length of a transaction is determined by Poisson distribution with mean u equal to $|T|$. The transaction is repeatedly assigned items from the item set until the length of the transaction satisfies the generated length.

Table 2 Parameters

Parameters	Description
$D\|$	Number of transactions
$\|I\|$	Number of items
$\|T$	Average size of the transactions

2.2 Performance comparison

CATD is motivated by the need to reduce disk I/O. Obviously, it is superior to hierarchical clustering and partitional clustering in this aspect. The latter two types of clustering need multiple iterations over the database, and the exact number depends on the convergence speed of the criterion function (partitional clustering) or the termination condition (hierarchical clustering). In CATD, the partitions are chosen such that all data structures can be accommodated in the main memory. We process the entire database as a single partition for small database and more partitions are chosen to guarantee each partition can be stored in memory as the size of database increases.

We study the performance of three clustering algorithms——CATD, CLARANS, and RNN CLINK——by varying the number of transactions from 10,000 to 100,000. The latter two algorithms are performed directly on original databases using Jaccard coefficient as similarity measures. The number of partitions of CATD increases from 1 to 10 with the size of database. The experimental results are shown in Fig. 5. For small database (<50,000) CATD is superior to CLINK, but inferior to CLARANS. As the number of transactions increases, the other two algorithms spend more and more time on disk I/O, while CATD keeps one scan of database and stores the data in memory by increasing data partitions. Obviously, CATD is very suitable for large databases.

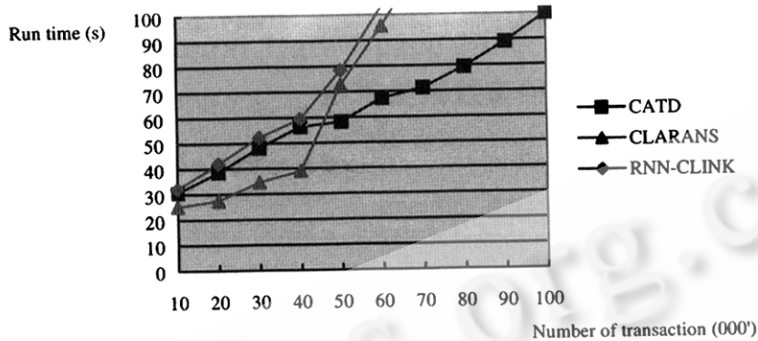

Fig. 5 Execution time comparison

3 Conclusion

Due to the very large number of transactions and high-dimensionality, traditional clustering algorithms are not appropriate for transaction databases. In this paper, we study the problem of clustering in large transaction databases. We present a two-step algorithm of clustering transactions. It first divides the database into partitions in which transactions are partially clustered into subclusters in partial clustering. We do not predetermine a constant number of partial clusters for each partition. Instead, we obtain a variety number of subclusters for a partition depending on the distribution of objects in it. We generate a support vector from every subcluster at the end of the partial clustering. A support vector contains the support values of all items and represents the distribution of items in the subset. In global clustering, a k-medoid clustering algorithm is performed on the subclusters. The partitioning scheme ensures a primary clustering of transactions in main memory using only one scan of databases, and the global clustering partitions the partial clusters into k clusters and identifies noise. In fact, the two-step mechanism is not confined to transaction databases. Future work includes adopting the approach to clustering other type of databases. We also plan to extend the algorithm to parallel and incremental environment.

References:

[1] Weber, R., Schek, H.-J., Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Ashish, G., Oded, S., Jennifer, W., eds. Proceedings of the 24th International Conference on Very Large Data Bases. New York, USA: Morgan Kaufmann, 1998. 194~205.

[2] Aggarwal, C. C., Wolf J. L., Yu, P. S. A new methods for similarity indexing of market basket data. In: Alex, D., Christos, F., Shahram, G., eds. Proceedings of the ACM SIGMOD International Conference on Management of Data. Philadephia, Pennsylvania, USA: ACM Press, 1999. 407~418.

[3] Han, E., Karypis, G., Kumar, V. Hypergraph based clustering in high-dimensional data sets: a summary of results. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998,21(1):15~22.

[4] Zhang, T., Ramakrishnan, R., Linvy, M. BIRCH: an efficient data clustering method for very large databases. In: Jagadish, H. V., Mumick, I. S., eds. Proceedings of the ACM SIGMOD International Conference on Management of Data. Montreal, Canada: ACM Press, 1996. 103~114.

[5] Guha, S., Rastogi, R., Shim, K. CURE: an efficient clustering algorithm for large database. In: Laura, M. H., Ashutosh, T., eds. Proceedings of the ACM SIGMOD International Conference on Management of Data. Seattle, Washington, USA: ACM Press, 1998. 73~84.

[6] Cheung, D. W., Hu, K., Xia, S. An adaptive algorithm for mining association rules on shared-memory multi-processors parallel machine. Distributed and Parallel Databases, Kluwer Academic Publishers, (to appear).

[7] Lang, S. D., Mao, L.-J., Hsu, W.-L. Probabilistic analysis of the RNN-CLINK clustering algorithm. In: Proceedings of the SPIE on Data Mining and Knowledge Discovery: Theory, Tools, and Technology. Orlando, Florida, 1999. 31~38.

大规模交易数据库的一种有效聚类算法

陈 宁[1], 陈 安[2], 周龙骧[1]

[1](中国科学院 数学与系统科学研究院,北京 100080);

[2](北京航空航天大学 管理学院,北京 100083)

摘要: 研究大规模交易数据库的聚类问题,提出了一种二次聚类算法——CATD. 该算法首先将数据库划分成若干分区,在每个分区内利用层次聚类算法进行局部聚类,把交易初步划分成若干亚聚类,亚聚类的个数白聚类间的距离参数控制. 然后对所有的亚聚类进行全局聚类,同时识别出噪声. 由于采用了分区方法和聚类的支持向量表示法,该算法只需扫描一次数据库,聚类过程在内存中进行,因此能处理大规模的数据库.

关键词: 数据挖掘;聚类分析;层次聚类;单连距离

中图法分类号: TP391 **文献标识码**: A