DPQR: an improved parallel DBSCAN algorithm based on data partition and QR*-tree

Hongbo Xu*, Nianmin Yao, Qilong Han, Haiwei Pan

College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, Heilongjiang, China

Received 1 June 2014, www.cmnt.lv

Abstract

As a typical clustering algorithm based on the density, DBSCAN shows good performance in spatial data clustering. When clustering large-scale database, DBSCAN requires the overhead of memory and I/O. With the development of high-performance computers and the appearance of cluster computers in particular, this gives us a way to solve the defects of DBSCAN. The paper presents an improved parallel DBSCAN algorithm DPQR based on data partition and *QR**-tree. According to the distribution of data on one or more dimensions, the entire data space is divided into a number of local regions. Each local region is transmitted to a processing unit. The processing unit calculates local *k*-dist graph for local region to get the local value *Eps*, and builds *QR**-tree. DPQR executes the partial clustering on *QR**-tree. Finally, the clustering results are merged in accordance with the merging rules. Experimental results show that DPQR is better than DBSCAN.

Keywords: large-scale database, clustering, data partition, DBSCAN, parallel computing

1 Introduction

Spatial database manages spatial objects such as points and polygons. In recent years, a large number of data which modern technical tools (satellite remote sensing and X-ray imaging and so on) generate is stored in the spatial data. The knowledge discovery on the large-scale spatial database has become particularly important. Data clustering is a major issue in the field of data mining. The data in the database would be divided into sub-classes with a certain sense. Data from different sub-classes is as dissimilar as possible, and data from the same sub-class is as similar as possible. So far, the researchers have presented many data clustering algorithms, such as CLARANS, BIRCH, DBSCAN [1], CURE, STING, CLIQUE and Wave cluster and so on. All of these algorithms are trying to solve the problem about data clustering on large-scale databases [2].

DBSCAN is a spatial clustering algorithm based on the density. The algorithm uses the concept of density-based clustering. The number of the objects which a certain region in the space contains is not less than a given threshold. The significant advantage of DBSCAN is fast clustering. DBSCAN can effectively deal with the noise points (outliers) and discover the clusters of arbitrary shape. Because DBSCAN operates directly on the whole database, and uses a global parameter which represents the density, DBSCAN has two obvious weaknesses. The first weakness is requiring larger memory support. The second weakness is larger I/O consumption. When the density of spatial data is uneven, and the distance between the clusters varies greatly, the quality of clustering is relatively poor.

To solve these problems, this paper proposes an improved parallel DBSCAN algorithm DPQR based on data partition and *QR**-tree. According to the spatial distribution of data, divide the entire data space into multiple smaller partitions. Then cluster these partial partitions. Eventually merge each local cluster. The experimental results show that this method is effective.

This paper is organized as follows. Section 2 introduces related works; Section 3 briefs the basic idea of DBSCAN, and analyzes its limitations; Section 4 introduces the structure of *QR**-tree; Section 5 presents an improved parallel DBSCAN algorithm DPQR based on data partition and *QR**-tree; Section 6 describes the experimental results; Section 7 summarizes the paper, and points out the focus of future work.

2 The related works

For the advantages and disadvantages of DBSCAN, many scholars have conducted many studies. The paper [3] has proposed the algorithm SDBSCAN based on data sampling. The algorithm uses data sampling to extend DBSCAN. So that it can effectively carry out clustering analysis on large-scale database. It uses a fast clustering tagging method. Thus this increases speed and efficiency of the whole process of clustering. The paper [4] has proposed the algorithm PDBSCAN based on data partition. This method determines the breakpoints according to the diagram of data distribution. According to

* ***Corresponding author's*** e-mail: xhb0451@qq.com

the data distribution, determine which dimension to be divided, how many regions to be divided. As to the sensitive issue for the parameters, the paper [5] has proposed an improved algorithm DBSCANCC (DBSCAN with Cluster Connection). It solves the problem of the poor clustering results due to the improper parameter *Eps*. Thereby this shields the sensitivity of input parameters of the algorithm, and maintains higher efficiency. The paper [6] has proposed a parallel DBSCAN algorithm based on data overlap. After the data partition, each partition is sent to a processor. The processor processes parallel clustering, and finally merges local clustering. The paper [7] has proposed a fast clustering algorithm FDBSCAN based on the density. The algorithm selects the representative objects of all the objects in the neighborhood of the core object as the seed objects to extend the cluster. Thereby the algorithm reduces the number of region query and I/O overhead to achieve rapid clustering.

These algorithms significantly improve the speed of clustering. But the time complexity of them is still relatively high, and I/O consumption of them is not very satisfactory.

3 DBSCAN

DBSCAN which Ester Martin proposed is a clustering method of spatial data based on the density. The central idea is that for each object in a cluster, at a given radius *Eps* of the neighborhood, the number of data objects must be greater than a given value. In other words, the density of the neighborhood must exceed a certain threshold *MinPts*. To find a cluster, DBSCAN finds any object *p* from *D*, and finds all the objects which are density-reachable from *p* according to the parameters *Eps* and *MinPts*.

If *p* is a core object, that is the number of objects which *Eps*-neighborhood of *p* contains is not less than *MinPts*, the algorithm finds a cluster according to the parameters *Eps* and *MinPts*. If *p* is a boundary point, that is the number of objects which the *Eps*-neighborhood of *p* contains is less than *MinPts*, no object is density-reachable from *p*. The point *p* is temporarily marked as noise point. Then, DBSCAN handles the next object in database *D*.

Get all the data objects which is density-reachable from a core object by repeated region query to achieve. A regional query returns all objects in a given query region. R*-tree implements this query. Therefore, before performing clustering, R*-tree must be built.

DBSCAN requires the user to specify a global value *Eps*. To reduce the amount of the computation, *MinPts* is set to 4. To determine *Eps*, DBSCAN need to calculate the distance from all the data objects to its *k* nearest-neighbor objects, and the results are sorted by the distance to get *k*-dist graph.

The horizontal coordinate of *k*-dist graph shows the distances from the data objects to its *k* nearest-neighbor objects. The vertical coordinate of *k*-dist graph shows the number of the objects corresponding to a distance value. To establish R*-tree and draw *k*-dist graph is very time-consuming task, especially in large-scale database. In addition, users have to repeat the test, select the appropriate *k*-dist value in order to achieve better clustering results.

In the absence of any pre-processing, DBSCAN directly operates the entire database. On the one hand, DBSCAN requires a lot of memory and I/O overhead; On the other hand, since the global value of *Eps* is used, the size of the neighborhood in the data space is the same.

The distance between the clusters and the distribution of the density are uneven. If a smaller value of *Eps* is selected based on those dense clusters, then the number of data objects in their neighborhood will be less than *MinPts* for those dilute clusters. These objects will be considered to be boundary objects which are not used for further expansion. So the dilute cluster is divided into multiple similar clusters. On the contrary, if a larger value of *Eps* is selected according to those dilute clusters, then those clusters which are closer and denser will likely be merged into the same cluster. The differences between them will be ignored because of choosing the larger value *Eps*. Obviously it is difficult to select an appropriate value *Eps* to obtain more accurate clustering results in both cases.

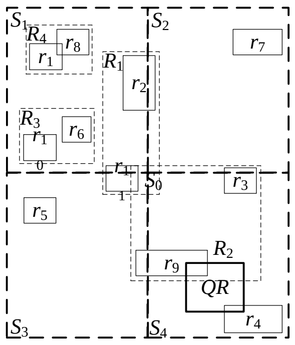
4 The structure of *QR**-tree

QR*-tree is a spatial index structure, which is a combination of quad-tree [8] and R*-tree [9]. Suppose $d>0$ and $n=\sum_{l=0}^{d-1}(2^k)^l$ (*k* is the dimension of the space, *d* is the depth of quad-tree), QR^* is composed of *d*-depth quad-tree *Qt* and *n* R*-trees. *Qt* has a total of *n* nodes, which are expressed as $Qt_0, Qt_1, \ldots, Qt_{n-1}$. *Qt* divides data space *S* into *n* *d*-level sub-spaces which are expressed as $S_0, S_1, \ldots, S_{n-1}$ ($S_0=S$). All subspaces of each level are disjoint, and constitute the entire index space *S* together. [10] The *n* R*-trees are expressed as $Rt_0, Rt_1, \ldots, Rt_{n-1}$, respectively are associated with *n* nodes and *n* sub-spaces of *Qt*. S_i is associated with Rt_i. Rt_i is used to index space objects which belong to S_i. That the space object *P* belongs to S_i refers to

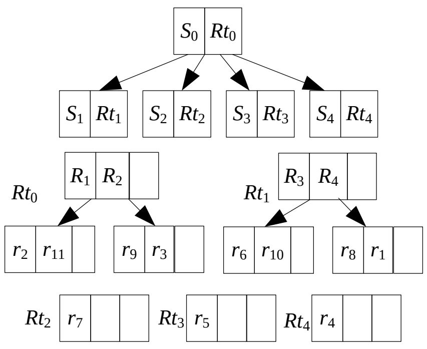
1) *P* completely falls on S_i or S_i completely surrounds *P*,

2) S_i is the smallest sub-space which completely surrounds *P*. Figure 1 is an example of two-dimensional *QR**-tree. In this example, *QR**-tree is composed of two-depth quad-tree and five *R**-trees. Entire space is divided into five two-level sub-spaces which are expressed as S_0, S_1, S_2, S_3, S_4 ($S_0=S_1\cup S_2\cup S_3\cup S_4$). Rt_0, Rt_1, Rt_2, Rt_3, Rt_4 are associated with them. S_1 is the minimal sub-space which encloses r_1. Therefore, r_1 is assigned to Rt_1. S_0 is the minimal subspace which encloses r_2. Therefore, r_2 is assigned to Rt_0. *QR**-tree is composed of quad-tree and R*-tree. The node structure of R*-tree is shown below. Leaf node: (*COUNT*, *LEVEL*, <OI_1,MBR_1>, <OI_2,MBR_2>,…,<OI_m,MBR_m>); Non-leaf node: (*COUNT*, *LEVEL*, <CP_1,MBR_1>, <CP_2,MBR_2>,…,<CP_m,MBR_m>).

OI_i of leaf node is the identifier of space object, and MBR_i is the minimal constraint rectangle of space object in the k-dimensional space. CP_i of non-leaf node is the pointer pointing to the root node of the sub-tree, and MBR_i represents the index space of the sub-tree. See Figure 1.



a) The diagram of space division



b) The structure diagram of QR^*

FIGURE 1 QR^*-tree in two-dimensional space

5 DPQR

When clustering large-scale database, data partitioning is an effective method. For DBSCAN, the data partition has following benefits.

1) In the clustering process, once DBSCAN finds a core object, the object is as the center to expanse outward. In this process, the number of core objects will continue to increase. Unprocessed core objects are retained in memory. If there is a very large cluster in the database, the memory demand used for storing information of core objects will be large and unpredictable. Then dividing the data into partitions can avoid this situation.

2) As analyzed in Section 3, due to the use of global value *Eps*, when the distribution of the density and the distance between the clusters are uneven, DBSCAN will be difficult to get a clustering result of higher quality. Partition the data and select local value *Eps*, these should be able to reduce or avoid the above problem of the deterioration of the clustering quality. This is the case shown in Figure 3(a). Obviously, there are five categories in Figure 4. Three left clusters are dense and close. Two right clusters are sparse. If the density of right clusters is referenced to select the value of *Eps*, DBSCAN may merge three left clusters into one cluster. Conversely, if the density of left clusters is referenced to select the value of *Eps*, the right cluster will be decomposed into many sub-clusters. Even generate many noises. In this case, DBSCAN cannot select the appropriate value for the clustering of data.

Based on the above considerations, the paper presents an improved parallel DBSCAN algorithm DPQR based on data partition and QR^*-tree. The basic idea of DPQR is: according to the distribution of data on one or more dimensions, the entire database space is divided into a number of local regions. The purpose is to make the data of each local region evenly distributed. Each local region is transmitted to a processing unit. The processing unit calculates local k-dist graph for local region to get the local value *Eps*, and builds QR^*-tree. DPQR executes partial clustering on QR^*-tree. Finally, the clustering results obtained are merged in accordance with the merging rules. Since each local region uses its local value *Eps* to cluster, the original problems of clustering quality caused by the use of global value *Eps* can be alleviated or even eliminated.

5.1 THE PARTITION OF DATA SETS

Data partitioning solves the problems about much memory occupying and I/O excessive consumption in the process of using DBSCAN. At the same time, the idea of parallel processing is introduced. But the idea of parallel processing is build on the theory of data partition.

Data partitioning algorithm Partition is described below:

```
Input: one-dimensional data file of samples $F$, the number of processors $N$.
Output: dividing points $S$ on each dimension.
Begin
Step 1: according to the number of samples, determine the number of the groups $m$, the maximum max and minimum min in the sample data;
Step 2: calculate the step $d = \frac{max - min}{m}$;
Step 3: for $i$=1 to $m$ do
    calculate the density of the step $\rho_i$;
    store $m$ vectors (starting point of step, ending point of step, $\rho_i$);
Step 4: for $i$=1 to $m$ do
    if ($\rho_i$ has no change or $\rho_i$ increases monotonically or $\rho_i$ decreases monotonically) then
        allocate on average $N/K_i$;
        determine $S$ ($S$=$N/K_{i-1}$) demarcation points;
    else
        determine the absolute value of the difference between adjacent wave peaks is greater than the threshold $\lambda$. If the condition is satisfied, write down the coordinate $S_i$ which the wave valley $S_o$ corresponds to
        S=S∪Si;
Step 5: output all the division points $S$;
```

End.

Use this method to determine the division points respectively on the x-axis and y-axis. Define a partition vector $V_i(i=X$ or $Y)=(P_{i0}, P_{i1},\dots, P_{in})$. Define the kth interval on the ith dimension as $I_{ik}=[l_{ik}, h_{ik}]$. So each rectangle is Cartesian produce $[l_{1k1}, h_{1k1}]*[l_{2k2}, h_{2k2}]$ of two dimensional different intervals. This rectangle is called the grid. Each grid can be represented by the expression $(l_{1k1}\le x\le h_{1k1}\wedge l_{2k2}\le y\le h_{2k2})$. This grid can be defined as (K_1, K_2) using the coordinates. Divide the uniform distribution data into the rectangular grids. Thus assign each grid to multiple processors to separately cluster. After this processing, data distribution is more uniform. The selection of value *Eps* will not affect the various regions. This improves the quality of clustering. On the other hand, multiple processors cluster the data. Thereby the efficiency of clustering is increased.

5.2 THE SELECTION OF THE SEED POINT

Select the point in the neighborhood of hollow spherical of core point as the seed point. With the number of seed points reducing, the number of region queries reduces, I/O overhead decreases, clusters expanse rapidly.

Note that the neighborhoods of the points in the same neighborhood will overwrite each other. When the neighborhood of a point is completely covered by the neighborhood of another point, its neighborhood can be got through the region query of this point. This means that the point is not necessary as a seed point. This point is basically in the inner neighborhood. Therefore, the points in the outer ring of the neighborhood are as seed points. Figure 2 shows the process of selecting the seed point. See Figure 2.

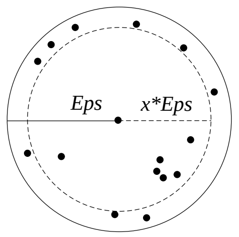


FIGURE 2 The diagram of selecting the seed point

The larger the value x is, the smaller the selection range of seed points is. The more inadequate cluster extension is, the more serious error division is. Thus error rate of clustering is the higher. The smaller the value x is, the larger the selection range of seed points is. The more the number of region query is, the slower the clustering speed is. The experiments show that when x is 60%, the quality and speed of clustering can achieve the best balance.

5.3 CLUSTERING AND MERGING OF PARTITION DATA

After each data partition is clustered, there are some cases need to be considered. A point z respectively belongs to the clusters C_1 and C_2, and is the core point in C_1 and C_2; A point z respectively belongs to the clusters C_1 and C_2, is the core point of C_1 and is the boundary point of C_2; A point z belongs to the cluster C_1, is noise point to another cluster; A point z is noise point both times. These cases require separate treatment. The following rules are obtained.

Rule 1: for point z, cluster C_1 and C_2, if $z\in C_1$ and $z\in C_2$, z is the core point of C_1 and C_2, C_1 and C_2 can be merged into one cluster;

Rule 2: for point z, cluster C_1 and C_2, if $z\in C_1$ and $z\in C_2$, z is the core point of C_1 and z is the boundary point of C_2, C_1 and C_2 can be merged into one cluster;

Rule 3: for point z, cluster C_1 and C_2, if $z\in C_1$ and $z\in C_2$ and z is the boundary point of C_1 and C_2, z belongs to C_1 or C_2 and C_1 and C_2 cannot be merged into one cluster;

Rule 4: if z belongs to C_1 and z is a noise point to C_2, C_1 and C_2 cannot be merged into one cluster;

Rule 5: if z is a noise point to C_1 and C_2, z is a noise point in the global clustering.

5.4 THE BASIC FRAMEWORK OF THE ALGORITHM DPQR

The algorithm DPQR is described as follows: For a given database D and *MinPts*, parameter *Eps* which k-dist graph obtains, any element E in D has m dimensions E $(E_1, E_2,\dots, E_m)$, D is divided into n partitions each of which has the same amount of data approximately. Steps are as follows:

Step 1: for any integer i $(1\le i\le m)$, database D is projected on the ith dimension. That is $\forall E\in D$, $E\rightarrow E_i$. Database D is mapped to a one-dimensional data set D_i, $D_i=\{E_i \mid E\ (E_1, E_2,\dots, E_m)\in D\}$. During projection, algorithm Partition divides data sets into partitions. Declare a variable *tmp* which is assigned an initial value of m.

Step 2: A is the lower limit of dataset D_i, and B is the upper limit of dataset D_i. Data set D_i distributes in the interval $[A, B]$;

Step 3: find the sequence a_0, $a_1,\dots$, a_n to satisfy $A=a_0<a_1<a_2<\dots<a_n=B$ $(0\le i\le n-1)$;

Step 4: If the partition results meet the Step 3, then go to Step 6;

Step 5: If the partition results do not satisfy Step 3 and *tmp*>1, then *tmp*=*tmp*-1 and go to Step 1; If the partition results do not satisfy Step 3 and *tmp*≤1, then go to Step 10;

Step 6: Partition results are processed redundantly. Modify the partition intervals to $[a_0, a_1+Eps]$, $[a_1-Eps, a_2+Eps],\dots, [a_i-Eps, a_{i+1}+Eps],\dots, [a_{n-1}-Eps, a_n](i=2, 3,\dots, n-2)$. For convenience, modified partitions are named as $C_1, C_2,\dots, C_{i+1},\dots, C_n$;

Step 7: C_i $(i=1, 2,\dots, n)$ is sent to the ith processor memory;

Step 8: Each processor node uses DPQR to cluster local data. The ith partition is a case.

Step 8.1: For a given partition C_i build QR^*-tree;

Step 8.2: For a given partition C_i, calculate k-dist diagram to get Epsi;

Step 8.3: Initialize each point to the initial state. Initialize the queue seeds of seed points to NULL. Initialize the result set results to NULL;

Step 8.4: define tag ID of the first cluster as 1;

Step 8.5: obtain the first point in partition Ci to cluster;

Step 8.6: execute region query of points on QR*-tree, and return the query results;

Step 8.7: According to *MinPts*, determine whether this point is a core point. If it is not a core point, it is recorded as a noise point; If it is a core point, all points are marked as current ID. Then find next point from point set *Fseeds*. The distance from the point in *Fseeds* to the core point is in the range of 0.6Eps_i to Eps_i. The processed points have been removed from the *Fseeds* until *Fseeds* is empty.

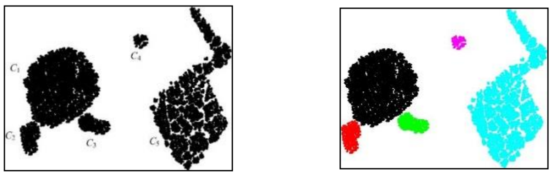
Step 8.8: get the next point in partition C_i to cluster, increase label ID by 1, until all points in partition C_i have been processed;

Step 9: Cluster merging. The i^{th} processor ($1 \le i \le n$-1) transmits the cluster data to $(i+1)^{th}$ processor. The $(i+1)^{th}$ processor compares the cluster data. If point z belongs to cluster C_i and C_{i+1}, is the core point in C_i and C_{i+1}, C_i and C_{i+1} are merged into one cluster; If point z belongs to C_i and C_{i+1}, z is the core point of C_i and z is the boundary point of C_{i+1}, C_i and C_{i+1} can be merged into one cluster; if $z \in C_i$ and $z \in C_{i+1}$ and z is the boundary point of C_i and C_{i+1}, z belongs to C_i or C_{i+1} and C_i and C_{i+1} cannot be merged into one cluster; if z is a noise point to C_1 and C_2, z is a noise point in the global clustering.

Step 10: the algorithm ends.

6 Experimental results

The experimental data set is composed of 5 clusters. The distance of three clusters on the left is relatively close, and the density of them is relatively large. The distance between two clusters on the right is relatively far, and the density of them is relatively thin. The implementation environment of DPQR algorithm uses high-performance cluster computer. Its nodes are interconnected through a high-speed 100Mbps network. The number of available nodes is ten. The memory of each node is 512MB. The hard disk is 15GB. CPU is PIII 500MHz. The operating system is Redhat Linux. On the basis of DBSCAN packages Martin Easter developed, implement DPQR with MPI and C++. See Figure 3.



a) b)

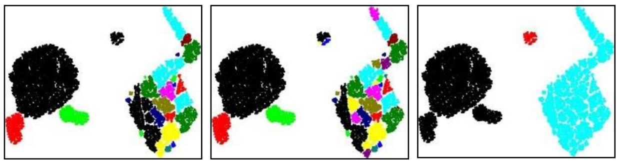
FIGURE 3 The experimental data sets

The experimental steps of DBSCAN algorithm are as follows:

Step 1: the value *MinPts* is 4. This is the value which Martin Easter determines by a large number of experiments;

Step 2: determine the value of *Eps*.

When the value of *Eps* is 2.24, the result is shown in Figure 4a. The number of clusters on the left is 3. However the number of clusters on the right is 36. Distinguish between different clusters with different colors. When the value of *Eps* is 3.63, the result is shown in Figure 4b. The number of clusters on the left is 3. However the number of clusters on the right is 24. When the value of *Eps* is 4.5, the result is shown in Figure 4c. The number of clusters on the left is 1. However the number of clusters on the right is 2. See Figure 4.



a) *Eps*=2.24 b) *Eps*=3.63 c) *Eps*=4.5

FIGURE 4 The result when *Eps*=2.24, 3.63, 4.5

In order to discuss the efficiency of running time, execute DBSCAN on two-dimensional large-scale data. The statistical results are shown in Table 1.

TABLE 1 The average running time of DBSCAN and DPQR

The number of data	Running time of DBSCAN (ms)	Running time of DPQR (ms)
256	269	268
512	371	347
1024	618	598
5000	3156	1765
10000	5048	2648
20000	11560	3754
30000	14890	4987
40000	20044	7231
50000	32078	9684
60000	75641	12450

First DPQR determines the value of *MinPts*. The value of *MinPts* is 4. DPQR divides the experimental data set into two partitions, left partition and right partition. The value of *Eps* of left partition is 2.3. The value of *Eps* of right partition is 4.49. The obtained results are shown in Figure 3b.

As shown in Figure 3b, left partition is divided into three clusters; Right partition is divided into two clusters. Then discuss the efficiency of running time, DPQR processes large-scale data set. The results are shown in Table 1.

The clustering result which the algorithm DBSCAN obtains is not ideal. When dealing with data sets of uneven density distribution, the clustering quality of the algorithm DBSCAN is not high. Mainly due to the uneven distribution of the density, there is unable to find the value of *Eps* which is suitable for the regions of large and small density. When the value of *Eps* is too small, it is suitable for large density regions to cluster, and the small density regions are clustered too much. When the value of *Eps* is too large, it is suitable for small density regions to cluster, and the large density regions are clustered too much.

The clustering result of DPQR is ideal. Because select different *Eps* for different density regions. The problem which improper selection of *Eps* causes does not occur. When processing the data of less than 10000, the difference of the performance is not particularly large. When processing the data of larger than 20000, the time performance of DPQR is better. The running time of DPQR increases moderately. This is more suitable for handling large data sets.

7 Conclusions

The paper presents a parallel algorithm DPQR based on data partition and QR^*-tree. According to the spatial distribution of data, the entire data space is divided into multiple smaller partitions. So that local density of the partitions is relatively more uniform. Then each local partition is respectively sent to a processing unit. Establish QR^*-tree on the basis of each processing unit to improve the efficiency of region query. Finally, the clustering results are merged in accordance with the merging rules. Experimental results show that the algorithm DPQR can reduce the consumption of memory and I/O. For data sets of uneven density, clustering works well. This has been greatly improved relative to the algorithm DBSCAN.

Acknowledgments

This work was financially supported by National Natural Science Foundation of China (61073047, 61272184), Fundamental Research Funds for the Central Universities (HEUCFT1202, HEUCFT100609), Harbin Special Funds for Technological Innovation Talents (2012RFLXG023).

References

[1] Ester M, Kriegel H-P, Sander J, Xu X 1996 A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise *In Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96) Portland ACM Press* 226-31
[2] Han J, Kamber M, Pei J 2012 *Data mining Concepts and Techniques Third Edition China Machine Press*
[3] Zhou S-g, Fan Y, Zhou A-y 2012 SDBSCAN: A sampling-based DBSCAN algorithm for large-scale spatial database [J] *Mini-Micro System* **21**(12) 1270-4
[4] Zhou S-G, Zhou A-Y, Cao J 2000 A data-partitioning-based DBSCAN algorithm *Journal of Computer Research and Development* **37**(10) 1153-9
[5] Cai Y, Xie K, Ma X 2004 An Improved DBSCAN Algorithm which is Insensitive to Input Parameters *Acta Scientiarum Naturalium Universitatis Pekinensis* **40**(3) 480-6
[6] Song M, Liu Z-t 2004 A Data-overlap-partitioning-based Parallel DBSCAN Algorithm *Computer Application Research* (7) 17-20
[7] Zhou S-G, Zhou A-Y, Cao J, Hu Y-F 2000 A fast density-based clustering algorithm *Journal of Computer Research and Development* **37**(10) 1287-92
[8] Hao Z 2011 New spatio-temporal database theory *Science Press*
[9] Beckmann N, Kriegel H-P, Schneider R, Seeger B 1998 The R*-tree: An Efficient and Robust Access Method for Points and Rectangles *Proc Of ACM SIGMOD International Conference on Management of Data Atlantic ACM Press* 73-84
[10] Qiu J-h, Tang X-b, Huang H-g 2003 An index structure based Quad-tree and R*-tree-QR*-tree *Computer Applications* **23**(8) 124-7

Authors

Hongbo Xu, 29.02.1980, China.

Current position, grades: postdoctoral at Harbin Engineering University, China.
University studies: PhD degree in computer science and technology from Harbin University of Science and Technology, China in 2010.
Scientific interest: parallel algorithm, index structure and massive information processing.

Nianmin Yao, 26.09.1974, China.

Current position, grades: a professor at Harbin Engineering University, China.
University studies: PhD degree in computer science and technology from Jilin University, China in 2003.
Scientific interest: network storage, system performance analysis and theory of computation.

Qilong Han, 25.07.1974, China.

Current position, grades: associate professor at Harbin Engineering University, China.
University studies: PhD degree in computer science and technology from Harbin Institute of Technology, China in 2006.
Scientific interest: spatio-temporal data mining, graph mining, sensitive data protection, massive information processing and real-time database.

Haiwei Pan, 15.07.1974, China.

Current position, grades: associate professor at Harbin Engineering University, China.
University studies: PhD degree in computer science and technology from Harbin Institute of Technology, China in 2006.
Scientific interest: data mining, multimedia mining, massive data processing and information retrieval.