Wuhan University Journal of Natural Sciences Vol. 6 No. 1-2 2001,467~473

Article ID: 1007-1202(2001)01-0467-07

Scaling up the DBSCAN Algorithm for Clustering Large Spatial Databases Based on Sampling Technique

Guan Ji-hong[1], Zhou Shui-geng[2], Bian Fu-ling[3], He Yan-xiang[1]

1. School of Computer, Wuhan University, Wuhan 430072, China;
2. State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China;
3. College of Remote Sensing and Information Engineering, Wuhan University, Wuhan 430072, China

Abstract: Clustering, in data mining, is a useful technique for discovering interesting data distributions and patterns in the underlying data, and has many application fields, such as statistical data analysis, pattern recognition, image processing, and etc. We combine sampling technique with DBSCAN algorithm to cluster large spatial databases, and two sampling-based DBSCAN (SDBSCAN) algorithms are developed. One algorithm introduces sampling technique inside DBSCAN, and the other uses sampling procedure outside DBSCAN. Experimental results demonstrate that our algorithms are effective and efficient in clustering large-scale spatial databases.

Key words: spatial databases; data mining; clustering; sampling; DBSCAN algorithm

CLC number: TP 311.13

0 Introduction

Clustering, which is the task of grouping the data of a database or data warehouse into meaningful subclasses in such a way that minimizes the intra-differences and maximizes the inter-differences of these subclasses, is one of the most widely studied problems in data mining field[1]. There are a lot of application areas for clustering technique, such as statistical data analysis, pattern recognition, image processing, and other business applications, to name a few. Up to now, a lot of clustering algorithms have been proposed, in which famous algorithms contributed from the database community include CLARANS[2], BIRCH[3], DBSCAN[4], CURE[5], and recently the STING[6], CLIGUE[7] and WaveCluster[8]. All these algorithms try to challenge the clustering problems of handling huge amount of data in large-scale databases or data warehouses.

As an outstanding representative of clustering algorithm, DBSCAN algorithm shows good performance in spatial data clustering. It can discover clusters of arbitrary shape and handle the noise points (outliers) effectively. However, for large-scale spatial databases, DBSCAN requires large volume of memory support and could incur substantial I/O costs because it operates directly on the entire database.

The aim of this paper is to extend the DBSCAN algorithm to cluster large-scale spatial databases by data sampling technique. Two novel sampling-based clustering algorithms are proposed and implemented by combining the sampling technique with DBSCAN algorithm, which are called sampling-based DBSCAN algorithms and abbreviated to SDBSCAN. One SDBSCAN algorithm introduces sampling technique inside DBSCAN, i.e. inside sampling approach, and the other SDBSCAN algorithm applies sampling procedure out-

Received date: 2000-12-20
Foundation item: Supported by the Open Researches Fund Program of LIESMARS(WKL(00)0302)
Biography: Guan Ji-hong(1969-), female, Associate professor, research direction: distributed GIS, spatial database. E-mail: jhguan@wtusm.edu.cn

side DBSCAN, i. e. outside sampling approach. Owing to data sampling, the I/O cost and memory requirement for clustering large-scale spatial databases are reduced dramatically, and consequently the run-time of clustering is cut down considerably. Experimental results demonstrate that our approach is effective and efficient in clustering large-scale spatial databases.

1 SDBSCAN Algorithms

In order to alleviate the bottleneck problem of I/O cost and memory requirement while clustering large spatial databases or data warehouses with DBSCAN algorithm, here we develop two sampling-based DBSCAN (SDBSCAN) algorithms by combining sampling technique with DBSCAN algorithm. One SDBSCAN algorithm adopts sampling technique inside DBSCAN, i. e. inside sampling approach, and the other SDBSCAN algorithm uses sampling procedure outside DBSCAN, i. e. outside sampling approach. Comparing with other traditional clustering algorithms using sampling technique, our approaches have two outstanding features:

1) Sampling technique and clustering algorithm are bound together.

2) Clustering is carried out not only over the sampled data set, but also over the whole data set simultaneously.

Following are the details of these two SDBSCAN algorithms.

1.1 Inside Sampling: SDBSCAN-1

1.1.1 The idea of sampling inside DBSCAN

Inside sampling is a technique that sampling data inside the DBSCAN algorithm. We know the process of clustering by DBSCAN is an iterative procedure of executing region query. DBSCAN selects a global k-dist value for clustering. For the thinnest clusters, the number of objects contained in their core objects' neighborhoods with radius Eps equal to k-dist is k (The default value of k in DBSCAN is 4). However, for the other clusters, the number of objects contained in most of their core objects' neighborhoods of the same radius is more than k. DBSCAN carries out region query operation for every object contained in the core object's neighborhood. For a given core object p in cluster C, it's conceivable that the neighborhoods of the objects contained in p will intersect with each other. To suppose q is an object in p's neighborhood. If its neighborhood is covered by the neighborhoods of other objects in p, then the region query operation for q can be omitted because all objects in q's neighborhood can be fetched by the region queries of the other objects in p, which means that q is not necessary to be selected as a seed for cluster expansion. Therefore, both time consuming on region query operation for q and memory requirement for storing q as a core object can be cut down. In fact, for the dense clusters, quite a lot of objects in a core object's neighborhood can be ignored being chosen as seeds. So for the sake of reducing memory usage and I/O costs to speed up the DBSCAN algorithm, we should sample some representatives rather than take all of the objects in a core object's neighborhood as new seeds. We call these sampled seeds representative object of the neighborhood where these objects are held.

Intuitively, the outer objects in the neighborhood of a core object are favorable candidates of representative object because the neighborhoods of inner objects tend to being covered by the neighborhoods of outer objects. Hence, sampling the representative seeds is in fact a problem of selecting representative objects, which can accurately outline the neighborhood shape of a core object. Fig. 1 illustrates such an example in 2-dimensional space in which p is a core object in cluster C, $q_i (i = 1, 2, 3, 4)$ are sampled representative objects which are used as seeds for further cluster expansion.

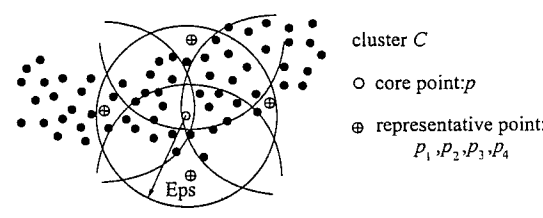

Fig. 1 Neighborhood and sampled representative points

1.1.2 Sampling representative objects

A key problem is how many representatives should be sampled for every core object's neighborhood. This parameter should not be too small and too large. If it's too small, a lot of lost objects

may be yielded (we will address this problem in next section). Otherwise, the advantage of sampling technique cannot be fully exploited. In our SDBSCAN-1 algorithm, while 2-dimension spatial data is considered, this parameter is set to 4, which is equal to the default value of MinPts. The reason is, intuitively, that the neighborhood of a core point can be covered approximately by 4 well scattered representative points' neighborhoods of the same radius Eps. Our experiments also show that by using 4 as the number of representatives, there are very few lost objects generated after the ordinary DBSCAN clustering process is finished. Generally, for the case of N-dimensional data space, $2 * N$ representative objects may be sampled from neighborhood of each core object for cluster expansion, i. e. two representatives are sampled over each dimension.

Without special declaration, we use the terms point and object equally in this paper. In what follows, an algorithm for sampling representative seeds from a core point's neighborhood is given. This algorithm iteratively samples the number of Rep_Minpts well-scattered points from a core point's neighborhood. In the first iteration, the point farthest from the core point is chosen as the first representative point. In the subsequent iteration, a point from the core point's neighborhood is chosen that be farthest from the previously chosen representative points.

```
//Algorithm for sampling representative points
Rep_Seeds_Sample(candidate_seeds, rep_seeds,
Rep_Minpts, Point)
  rep_seeds: =0; // initialize the representative
seeds set
  FOR i: =1 to Rep_Minpts DO
    maxDist: =0;
    FOR each point p in candidate_seeds DO
      IF i=1 THEN minDist: =dist(p,Point);
      ELSE minDist: =min {dist(p,q)|q∈rep_
seeds};
      IF (minDist≥maxDist) THEN
        maxDist: =minDist; maxPoint: =p
      END IF; // (minDist≥maxDist)
    END FOR; // each point p in candidate_
seeds
    rep_seeds: =rep_seeds∪{maxPoint};
  END FOR; // i: =1 to Rep_Minpts
END; //Rep_Seeds_Sample
```

1.1.3 About the lost objects

Theoretically, because we sample only a limited and fixed number of representative objects in a core object p's neighborhood as seeds for cluster expansion, it's likely that some core objects in p's neighborhood are ignored. In such a case, objects that are uniquely density-reachable from these ignored core objects will not be included in the cluster when the expansion process is completed. We call these objects lost objects.

Fig. 2 demonstrates such a case in 2-dimensional space in which lost objects exist. In Fig. 2, p_1 and p_2 are uniquely density-reachable from p_3 and p_4 respectively. However, in the clustering process, p_3 and p_4 are unfortunately not sampled as representative points, so p_1 and p_2 are lost when C_1 is clustered over. Because p_2 is core point, but p_1 is not, p_1 is labeled as NOISE and p_2 is assigned to the cluster C_2 that has a different cluster Id from C_1.

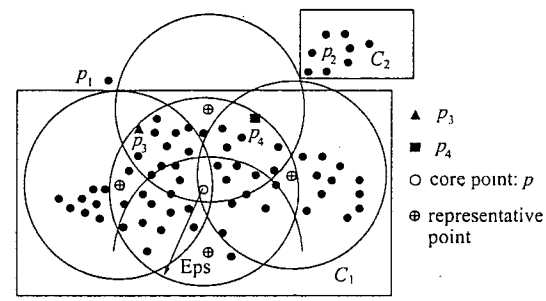

Fig. 2 Lost objects in a cluster

The lost objects are resulted from sampling. When the ordinary clustering phase is over, the lost border objects are labeled as noise, and the lost core objects forms new clusters with other objects. The task of handling the lost objects is, on the one hand, to reassign the lost border objects to the corresponding clusters to which these lost border objects should belong. And on the other hand, to merge the clusters where the lost core objects locate with other clusters in which these lost core objects should have been.

There is no obvious difference between the lost border objects and real noises, so we have to re-examine all noise objects to find the lost border objects. The process is as follows. For a noise object, firstly, we get its neighborhood. If all objects in its neighborhood are marked as noise, then it is

a real noise. Otherwise, if some objects are classified, we should further check whether they are core objects. If the answer is yes, then we assign the noise object to the cluster to which its nearest classified core object belongs. Otherwise, it is still a real noise object. As to handling the lost core objects, it is in fact an issue of clusters merging problem. Generally, the divided clusters must locate close to each other. We can directly use the representative objects of one cluster to carry out region query and get the neighborhood of these representative objects. If one of the representative objects is core object and in its neighborhood there are some objects labeled as another cluster and at least one of them is core object, then we treat the two clusters as one similar cluster, i. e. the two clusters are merged. Clearly, lost objects handling will trade off efficiency of SDBSCAN-1. Practically, an extra lost objects handling procedure is not indispensable. Our strategy is to accept the reality of lost objects' existence. The underlying reasons are as follows:

1) That some border points are assigned to noise will not greatly degrade the whole clustering quality. The border points are actually in a status between noise and genuine cluster members, so classifying some of them to noise is not unacceptable.

2) The possibility of splitting a cluster due to loss of core objects is very low. Usually, two adjacent parts of a cluster are density-reachable from each other side through multiple core objects. So it is very rare, if not impossible, for all core objects density-connecting the two parts to be lost altogether.

3) The number of the sampled representative objects and the sampling algorithm are crucial to the occurrence of lost objects. Through selecting an appropriate number of representative objects in core object's neighborhood and adopting a proper sampling algorithm, the lost objects can be controlled at a very low level.

Our experiments also show that in 2-dimensional space, by choosing 4 as the number of sampled representative points and using the sampling algorithm in Fig. 2, the ratio of lost points over genuine noise points is less than one percent. Practically and empirically, the lost points handling procedure can be ignored.

1.1.4 Algorithm description

In SDBSCAN-1, when the first core point is found in a new cluster, the first batch of representative points is sampled as seed points for the first iteration of cluster expansion. And in the subsequent iterations, more representative seeds are added up for cluster expansion till rep_seeds turns empty, which means the expansion of the current cluster is finished. SDBSCAN-1 differs from DBSCAN mainly in two aspects: 1) In the main procedure, there is an additional lost objects handling procedure; 2) In procedure responsible for cluster expansion, a procedure is added up to sample representative objects for cluster expansion.

1.2 Outside Sampling: SDBSCAN-2

Generally speaking, outside sampling is in fact a traditional sampling technique used in clustering[5,9,10]. A naive outside sampling DBSCAN algorithm may consist of three major steps as follows:

1) Sample database DB to produce the sampled data set sdb;

2) Carry out clustering over the sampled data set sdb with DBSCAN;

3) Label the un-sampled data from disk file or database.

In this naive outside sampling DBSCAN algorithm, we need only create R^*-tree for the sampled data set and do clustering over it with DBSCAN. The un-sampled data is stored in file or database. It can be labeled directly from disk file or database. Unfortunately, whatever traditional labeling method may be used, the labeling efficiency is still un-tolerable. So in our SDBSCAN-2 algorithm, a novel and efficient labeling mechanism is adopted to implementing the labeling process of the unsampled data on the basis of R^*-tree. The scheme of SDBSCAN-2 is like this:

1) Sample database DB to produce sampled dataset sdb;

2) Create R^*-trees for DB and sampled data set sdb;

3) Cluster sampled data set sdb with DBSCAN;

4) FOR each core point p in sampled data set sdb DO:

(A) resultP := DB.regionquery (p, Eps),

(B) DB. changeClIds (resultP, p. ClId).

Here, Step 1 is the sampling procedure. A sampling algorithm proposed in Ref. [11] is used for drawing a sample randomly from data in file in one pass and using constant space. In order to guarantee the clustering quality, the same analytical limitation on minimum sampled data amount as in Ref. [5] is used. Step 2 is responsible for building R^*-trees for DB and the sampled data set sdb. Step 3 and 4 are the key steps that are used for clustering and labeling respectively. We cluster the sampled data set sdb with DBSCAN algorithm. Once a core point is found in sdb, all points in its neighborhood of the same radius in DB, UNCLASSIFIED or CLASSIFIED as NOISE, sampled or un-sampled, are labeled as members of the current cluster. Hence, the clustering process (over the sampled data set sdb) and the labeling process (over un-sampled points in DB) are in fact carried out concurrently. When clustering is over, labeling is also finished. Fig. 3 demonstrates the processes of clustering and labeling in 2-dimensional space. Fig. 3(a) illustrates the sampled data set sdb and the clustering process in sdb, where p_1 is a core point in sdb and the radius of its neighborhood is Eps. Fig. 3(b) shows the labeling process in DB. Points in the neighborhood of core point p_1 are all labeled as members of cluster C_1.

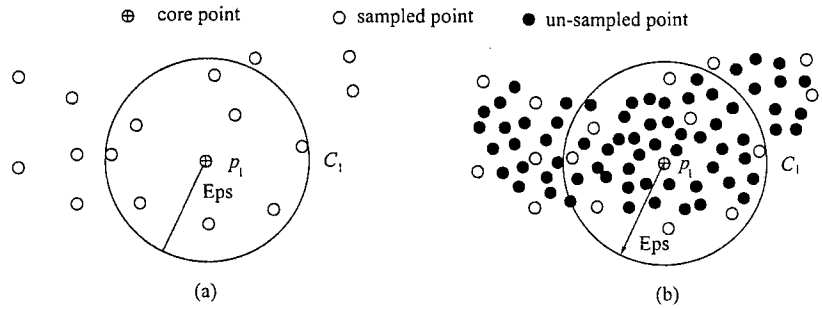

Fig. 3 Illustration of SDBSCAN-2
(a) Before labeling, (b) After labeling

However, just as in SDBSCAN-1, also due to similar reason, some un-sampled points may be lost or can not be covered by the neighborhoods of the core points in the sampled data set during the clustering process of sdb, which consequently leads to some un-sampled points in DB not being labeled in the labeling process. So we should have an extra procedure to cope with these lost points, which is similar to the lost points handling process in SDBSCAN-1 methodologically. However, if the data set is sampled in a sufficiently even way, the number of lost points can be controlled at a very low or acceptable level.

Further improvement on SDBSCAN-2 algorithm can be done as follows. While building R^*-tree, we do not create a separate R^*-tree for the sampled data set sdb. Instead, we build only one R^*-tree for DB. In other words, we merge the R^*-trees of DB and sdb in the former version of SDBSCAN-2 into one single R^*-tree, in which we mark out which point is sampled and which is not. Therefore, the operations of clustering and labeling are carried out over the same R^*-tree. And for each core point in sdb only one time of region query operation is executed, which is apparently unlike in the former version of SDBSCAN-2 where two times is needed: the first time for clustering over the R^*-tree of sdb, and the second time for labeling over the R^*-tree of DB. Obviously, this kind of improvement will result in almost half of the region queries being cut down.

2 Discussions and Performance

Like the DBSCAN algorithm, the average run time complexity of SDBSCAN-1 and SDBSCAN-2 is also $O(n\log n)$ (n is the number of objects in the database). However, the frequency of region query execution is cut down dramatically both in SDBSCAN-1 and SDBSCAN-2. The number of region query operation in DBSCAN is n, while in SDBSCAN-2, this value is at most $2n * s$ (s is sampling ratio), and for the improved version of SDBSCAN-2, it's about $n * s$. As for SDBSCAN-1, it is difficult to estimate the exact frequency of region query execution, which depends on the selection of

Eps (supposes MinPts is fixed) and n_r (n_r is the number of sampled representative objects in the neighborhood of a core object) values. Only in these neighborhoods that hold more than n_r objects could the number of region query execution be cut down. Generally, when MinPts is settled, the larger Eps is, the more the number of region query execution can be cut down, and consequently the more the run-time of clustering can be reduced. Clustering quality of SDBSCAN-2 relies on the sampling ratio and sampling algorithm. If the random sampling is sufficiently even and the sampling size satisfies the chernoff bounds, clustering quality can be guaranteed. As for SDBSCAN-1, clustering quality depends on n_r value and the sampling algorithm. In fact, there is a problem of trade-off between efficiency and quality. While clustering very large-scale databases or data warehouses, it is a realistic strategy to achieve better efficiency at the cost of losing certain clustering quality.

We implement the two SDBSCAN algorithms in Borland C++5.0 under the software framework of the original DBSCAN algorithm[4]. All experiments have been completed on a PC with a P2 CPU (350 MHz), 128 M memory and 9.6 G secondary storage volume. Both synthetic sample databases and real world databases are used for algorithms test. Fig. 4 illustrates the typical results of scale-up experiments with DBSCAN, SDBSCAN-1 and SDBSCAN-2. The curves in Fig. 4 show that SDBSCAN algorithms have better scalability over data set size than DBSCAN, and SDBSCAN-2 shows advantage over SDBSCAN-1.

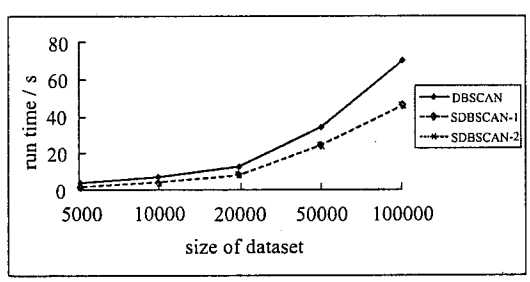

Fig. 4 Performance comparison: DBSCAN, SDBSCAN-1 and SDBSCAN-2
(sampling ratio is 20%)

3 Related Work

Generally, there are two types of clustering algorithm[9]: partitioning and hierarchical algorithms. The first type constructs a partition of a database D of n objects into a set of k clusters, and the second type creates a hierarchical decomposition of database D. While handling large-scale databases, one common used technique in clustering analyses is data sampling[5,9,10], which selects a relatively small number of representatives from databases or data warehouses and apply the clustering algorithms only to these representatives. CLARA (Clustering LARge Applications)[10] is a k-medoid clustering algorithm that relies on sampling. It draws a sample of the data set, and applies PAM algorithm[10] on the sample, then finds the medoids of the sample. A sampling technique for the CLARANS algorithm is presented in Ref. [9] based on R^*-tree. This approach consists of two steps. First, extract one representative for each page of the R^*-tree. From each data page of the R^*-tree, the most central object is taken as a representative. Second, Cluster the representatives using CLARANS and return the k medoids. The clustering strategy of the R^*-tree, which minimizes the overlap between directory rectangles, yields a well-distributed set of representatives. Another case of using data sampling technique in clustering large databases is CURE[5], which is a hierarchical clustering algorithm that can find clusters of arbitrary shapes. CURE uses chernoff bounds to analytically derive values for sample sizes for which the probability of missing clusters is low. However, to the best of our knowledge, up to now there is no report on research that combines sampling technique with DBSCAN algorithm. Although Ref. [9] also used the sampling technique on the basis of R^*-tree, it is entirely different from our algorithms proposed in this paper in terms of how and what for R^*-trees are used. On the one hand, Ref. [9] introduced data sampling to improve the efficiency of CLARANS algorithm, and we use sampling technique to extend DBSCAN algorithm to cluster large-scale databases more efficiently. On the other hand, Ref. [9] used R^*-tree only as an approach of data sampling, while we use R^*-tree in the process of clustering.

4 Conclusion

Based on the DBSCAN algorithm, in this pa-

per, we developed two new sampling-based clustering algorithms, i. e. sampling-based DBSCAN (SDBSCAN) algorithms by combining the sampling technique with DBSCAN algorithm. SDBSCAN-1 introduces sampling technique inside DBSCAN, and SDBSCAN-2 applies sampling procedure outside DBSCAN. Both algorithms can cut down I/O cost and memory requirement for clustering large-scale spatial databases dramatically, and consequently reduce the run-time of clustering considerably. Experimental results show that these algorithms are effective and efficient in clustering large spatial databases. In the future, we prepare to integrate SDBSCAN-1 and SDBSCAN-2 into one single algorithm applies the SDBSCAN algorithms to high dimension data clustering.

References:

[1] Chen M S, Han J H, Yu P S. Data Mining: an Overview from a Database Perspective. *IEEE Trans KDE*, 1996,**8**(6): 866-883.

[2] Ng R T, Han J. Efficient and Effective Clustering Methods for Spatial Data Mining. *Proceedings of the 20th VLDB Conference*. San Francisco:Morgan Kaufmann Publishers,1994. 144-155.

[3] Zhang T, Ramakrishnan R, Livny M. BIRCH: an Efficient Data Clustering Method for very Large Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. USA: ACM Press, 1996. 103-114.

[4] Ester M, Kriegel H P, Sander J, *et al*. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of 2nd International Conference on Knowledge Discovering in Databases and Data Mining* (*KDD*-96). USA: ACM Press, 1996.

[5] Guha S, Rastogi R, Shim K. CURE: an Efficient Clustering Algorithm for Large Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. USA: ACM Press, 1998. 73-84.

[6] Zhang W, Yang J, Muntz R. STING: a Statistical Information Grid Approach to Spatial Data Mining. *Proceedings of the 23rd VLDB Conference*. San Francisco:Morgan Kaufmann Publishers, 1997. 186-195.

[7] Agrawal R, Gehrke J, Gunopulos D, *et al*. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. USA: ACM Press, 1998. 73-84.

[8] Sheikholeslami G, Chatterjee S, Zhang A. WaveCluster: a Multi-Resolution Clustering Approach for very Large Spatial Databases. *Proceedings of the 24th VLDB Conference*. San Francisco:Morgan Kaufmann Publishers, 1998. 428-439.

[9] Kaufman L, Rousseeuw P J. *Finding Groups in Data: an Introduction to Cluster Analysis*. New York: John Wiley & Sons, 1990.

[10] Ester M, Kriegel H P, Xu X. Knowledge Discovery in Large Spatial Database: Focusing Techniques for Efficient Class Identification. *LNCS: Proceedings of 4th International Symposium on Large Spatial Databases*. Berlin:Springer-Verlag. 1995,**951**:67-82.

[11] Vitter J. Random Sampling with Reservoir. *ACM Transactions on Mathematical Software*, 1985,**11**(1): 37-57.