An Improved Sampling-Based DBSCAN for Large Spatial Databases

B Borah
Department of IT, Tezpur University
Tezpur, Assam, India
email: bgb@tezu.ernet.in

D K Bhattacharyya
Department of IT, Tezpur University
Tezpur, Assam, India
email: dkb@tezu.ernet.in

Abstract

Spatial data clustering is one of the important data mining techniques for extracting knowledge from large amount of spatial data collected in various applications, such as remote sensing, GIS, computer cartography, environmental assessment and planning, etc. Several useful and popular spatial data clustering algorithms have been proposed in the past decade [1]. DBSCAN [2] is one of them, which can discover clusters of any arbitrary shape and can handle the noise points effectively. However, DBSCAN requires large volume of memory support because it operates on the entire database. This paper presents an improved sampling-based DBSCAN which can cluster large-scale spatial databases effectively. Experimental results included to establish that the proposed sampling-based DBSCAN outperforms DBSCAN as well as its other counterparts [3], in terms of execution time, without losing the quality of clustering.

Keywords :
Clustering, density, sampling, DBSCAN, spatial data

INTRODUCTION

Spatial data clustering is one of the promising techniques of data mining, which groups a set of objects into classes or clusters so that objects within a cluster have higher similarity in comparison to one another, but are dissimilar to objects in the other clusters. Several useful clustering techniques have been proposed in the last few years for spatial data [1]. DBSCAN [2] is one of them. It tries to recognize the clusters by taking advantage of the fact that within each cluster the typical density of points is considerably higher than outside of the cluster. Furthermore, the density within areas of noise is lower than the density in any of the clusters. DBSCAN is a high-performance clustering algorithm that can discover clusters of arbitrary shape and handle the noise points effectively. However, for large-scale spatial databases, DBSCAN can be found to be expensive as it requires large volume of memory support due to its operations over the entire database. To overcome it, this paper presents an improved version of DBSCAN, which can handle large spatial databases with minimum I/O cost. In the proposed algorithm, the DBSCAN is extended by incorporating a better sampling technique, owing to which, the I/O cost and the memory requirement for clustering are reduced dramatically and hence a considerable amount of run-time is reduced. The rest of the paper is organized as follows : *Section 2* describes the existing DBSCAN. In *Section 3*, the proposed extended DBSCAN is presented. Detailed experimental results are included in the *Section 4*. Finally, in *Section 5*, the concluding remarks are given.

DBSCAN [2]

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. The basic ideas of density-based clustering involve a number of definitions, which are presented below.

- The neighbourhood within a radius ε of a given object is called the ε-*neighbourhood* of the object.
- If the ε-*neighbourhood* of an object contains at least a minimum number, *MinPts*, of objects, then the object is called a *core object.*
- Given a set of objects, *D*, we say that an object *P* is *directly-density-reachable* from object *Q* if *P* is within ε-*neighbourhood* of *Q*, and *Q* is a *core object.*
- An object *P* is *density-reachable* from object *Q* with respect to ε and *MinPts* in a set of objects, *D*, if there is a chain of objects P_1, ... P_n, $P_1=Q$ and $P_n=P$ such that P_{i+1} is *directly density-reachable* from P_i, with respect to ε and *MinPts*, for $1\leq i\leq n$, $P_i \in D$.
- An object *P* is *density-connected* to object *Q* with respect to ε and *MinPts* in a set of objects, *D*, if there is an object $O\in D$ such that both *P* and *Q* are *density-reachable* from *O* with respect to ε and *MinPts.*
- *density-based cluster* is a set of *density-connected* objects that is maximal with respect to *density-reachability*. Every object not contained in any cluster is considered to be a *noise*.

DBSCAN searches for clusters by checking the ε-*neighbourhood* of each point in the database. If the ε-*neighbourhood* of a point *P* contains more than *MinPts*, a new cluster with *P* as *core object* is created. DBSCAN then iteratively collects directly density-reachable objects from these core objects, which may involve the merge of a few *density-reachable clusters*. The process terminates when no new points can be added to any cluster.

0-7803-8243-9/04/$17.00 © 2004 IEEE

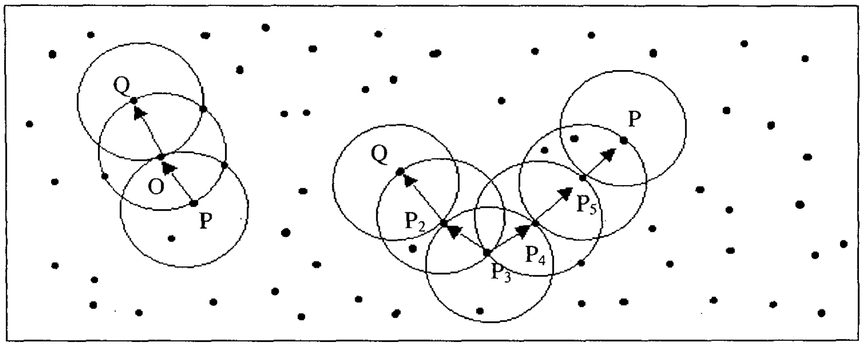

Fig 1 : Density reachability and density connectivity in density-based clustering

Analysis of DBSCAN

DBSCAN computes ε-*neighbourhood* of each object in the database. If a spatial index is not used the complexity of a neighbourhood query in DBSCAN is O(n) and using a spatial index such as a R*-tree it is O($\log_m n$), where n is the size of the dataset and m is the number of entries in a page of R*-tree. Correspondingly, the complexity of the DBSCAN algorithm becomes O(n^2) or O($n\log_m n$). For very large database the neighbourhood query becomes time consuming even if spatial index is used. Because the entire index tree cannot be accomodated in the main memory. A single page of the index tree is brought to the main memory at a time and search is performed in each of the m entries there. To complete the query $\log_m n$ such pages has to be examined. How to make the DBSCAN algorithm scalable? Suppose, the maximum size of the database that can be clustered by DBSCAN within a reasonable time is n, then within the same time we want to cluster a database of size kn, where k is a constant. To do this, one can speed up the algorithm by using two approaches : (*i*) by reducing the query *time* or (*ii*) by reducing the *number* of queries. This paper has attempted to exploit the *second* approach. The concept is presented below.

SAMPLING-BASED DBSCAN ALGORITHM : IDBSCAN

DBSCAN expands a cluster beginning with a core object P. It expands the cluster by carrying out neighbourhood queries for every object contained in P' s *neighbourhood.* But the neighbourhoods of the objects contained in P will intersect with each other. Suppose Q is an object in P' s neighbourhood, if its neighbourhood is covered by the neighbourhoods of other objects in P, then the neighbourhood query operation for Q can be omitted, which means that Q is not necessary to be selected as a *seed* for cluster expansion. Therefore, both time consumed on neighbourhood query operation for Q and memory requirement for storing Q as core object can be cut down. In fact, for the dense clusters, quite a lot of objects in a core object's neighbourhood can be ignored, being selected as *seeds.* So, we should *sample* some representatives rather than take all of the objects in a core object's neighbourhood as new seeds for the sake of reducing memory usage and I/O costs to speed up DBSCAN algorithm . Intuitively, the outer objects in the neighbourhood of a core object are favourable candidates to be selected as seeds, because the neighbourhood of inner objects tend to being covered by the neighbourhoods of outer objects. Hence, sampling the seeds is in fact a problem of selecting representative objects that can accurately outline the neighbourhood shape of a core object.

Selection of seeds

Consider a core object P with given ε and *Minpts*. Here we will consider two-dimensional objects only. The method can be generalized for any possible dimensions greater than two. As shown in the *Figure 2*, a circle is drawn with radius ε and center at object P. Let, the coordinate axes pass through the center object. Also, draw two diameters of the circle making an angle of 45^{O} and 135^{O} to the x-axis. Now the neighbourhood region of object P is divided into four quadrants. We have eight distinct points marked on the circle as $A(0, \varepsilon)$, $B(\varepsilon/\surd 2, \varepsilon/\surd 2)$, $C(\varepsilon,0)$, $D(\varepsilon/\surd 2, -\varepsilon/\surd 2)$, $E(0, -\varepsilon)$, $F(-\varepsilon/\surd 2, -\varepsilon/\surd 2)$, $G(-\varepsilon,0)$ and $H(-\varepsilon/\surd 2, \varepsilon/\surd 2)$. Let us call these objects as *Marked Boundary Objects* (MBO). In each quadrant we can identify three *MBO*s, for example in upper right quadrant A, B and C, in lower right quadrant C, D and E etc. For each of the MBOs identify the nearest object in

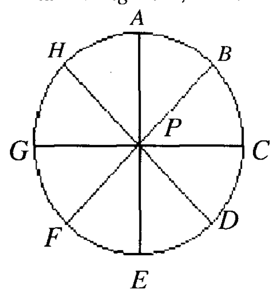

Fig 2 : Circle with eight distinct points

the neighbourhood and select it to be a seed. Do not select the same object as seed more than once even if it is nearest to some other *MBO* in the quadrant. Therefore total number of seeds selected may be less than or equal to eight in this 2-dimensional case. In general, if the objects are of dimension *d* then there will be (3^d-1) *MBO*s, 2^d quadrants and number of seeds selected is at most 3^d-1.

The Algorithm : IDBSCAN

```
function $IDBSCAN(D, \varepsilon, MinPts)$
   do for all objects $O$ in dataset $D$
         if $O$ is unclassified
            call function expand_cluster($O, D, \varepsilon, MinPts$);
   enddo
return


function $expand\_cluster(O, D, \varepsilon, MinPts)$
call function find_neighbours($O, D, \varepsilon, MinPts,$
$NEIGHBOURS, SIZE$);
if ( $SIZE < MinPts$ )
         mark $O$ as noise ;
         return;
else
  select a new $CLUSTER\_ID$;
  call function find_seeds($O, NEIGHBOURS, \varepsilon, SEEDS$) ;
  include all objects in $SEEDS$ into $SEED\_LIST$ ;
  mark all objects in $NEIGHBOURS$ with $CLUSTER\_ID$;
     do while $SEED\_LIST$ is not empty
         select an object from $SEED\_LIST$ as $CUR\_OBJ$ ;
         call function $find\_neighbours(CUR\_OBJ, D, \varepsilon,$
         $MinPts, NEIGHBOURS, SIZE$);
         if ( SIZE $\geq MinPts$ )
           call function $find\_seeds(CUR\_OBJ,$
           $NEIGHBOURS, \varepsilon, SEEDS$) ;
           include all objects in $SEEDS$ into $SEED\_LIST$;
           mark all unclassified and noise objects in
           $NEIGHBOURS$ with $CLUSTER\_ID$;
         endif
     enddo
   endif
return ;

function $find\_seeds(OBJ, NEIGHBOURS, \varepsilon, SEEDS)$
initialize each $MIN\_DIST\_TO\_MBO$ to max_value ;
initialize $SEEDS$ to null ;
identify the quadrant in which the object $OBJ$ falls ;
identify the $MBO$s for the quadrant ;
do for each unclassified objects in $NEIGHBOURS$
   do for each $MBO$ in the quadrant
       compute $DISTANCE$ of $OBJ$ to the $MBO$ ;
         if $DISTANCE$ is less then corresponding
         $MIN\_DIST\_TO\_MBO$
           set $MIN\_DIST\_TO\_MBO$ eual to $DISTANCE$;
           assign $OBJ$ to the $SEED$ corresponding to the
           $MBO$;
         endif
   endo
endo
eliminate duplicate $SEED$s if present in each quadrant ;
return ;
```

Complexity Analysis

IDBSCAN uses one more function than DBSCAN named *find_seeds*. Use of this function does not increase the overall complexity. IDBSCAN has complexity $O(nlog_m n)$ as that of DBSCAN but it speeds up DBSCAN by many folds. The extra function used here has complexity $O(sd)$, where s is the neighbourhood size and d is the dimensionality of the object. The neighbourhood size and dimensionality of an object are very very small compared to the size of the database.

EXPERIMENTAL RESULTS

We have evaluated the performance of our IDBSCAN algorithm in comparision to that of DBSCAN. For doing this, we have created several 2-dimensional synthetic datasets. Each dataset contains 13 clusters of different shapes -*circular, semicircular, rectangular, triangular* and *S-shaped* with varying sizes as shown in the *Figure 3*.

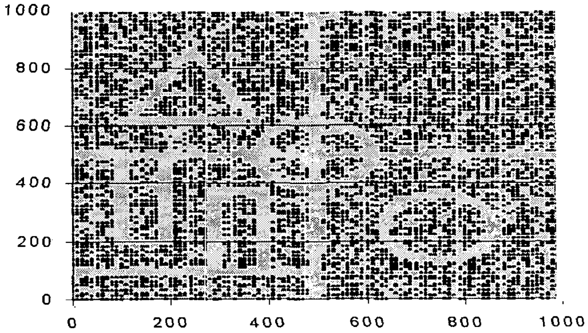

Fig 3 : Clusters of various shapes generated by IDBSCAN are shown. There are all total 13 clusters in 4 quadrants. Quadrant-1 contains 2, Quadrant 2 contains 3, Quadrant 3 contains 5 and Quadrant 4 contains 3 clusters. Black dots represent clusters on white background

Retaining the clusters to be the same we have created several datasets with increased number of objects present in the datasets. Objects are created in 1000×1000 screen avoiding duplicates. There is some random variation of density at different regions. The proposed IDBSCAN was implemented in C++ in a 1.1 GHz, HCL Infinity-2000 machine with 128 MB RAM. We have performed a series of experiments. In all experiments IDBSCAN finds. the same clusters as that of DBSCAN. Occasionally, IDBSCAN may treat some non-core boundary objects as noise producing a few more noise objects than DBSCAN. R*-tree indexing was also used with both DBSCAN & IDBSCAN. Next, we present the results of three experiments.

Experiment 1

The performance of DBSCAN and IDBSCAN was compared for data sets of various *sizes* and *densities*, keeping the value of ε constant. *Table 1* presents the results. It is clear from the graphical presentation (as shown in *Figure 4*) of the same results as depicted in *Table 1* that IDBSCAN is faster than DBSCAN. As the dataset size increases the speed difference between DBSCAN and IDBSCAN also increases

Table 1 : Comparison of IDBSCAN & DBSCAN for increasing size of datasets

Data size	ε	MinPts	Time (in sec.) IDBSCAN	Time (in sec.) DBSCAN
100000	8	7	32	62
200000	8	20	63	284
400000	8	40	125	1160
800000	8	75	423	5834

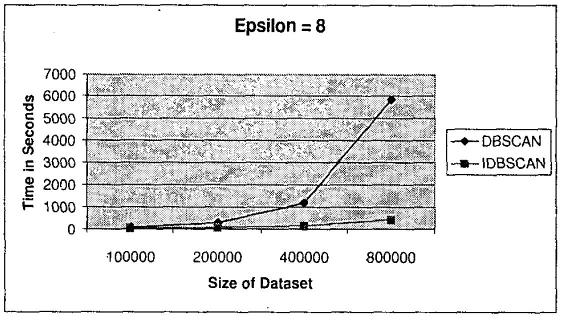

Figure 4 : DBSCAN vs IDBSCAN for varying dataset size & density

Experiment 2

In this experiment, with an increasing dataset size, the value of ε is allowed to decrease so that the *MinPts* remains constant. *Table 2* represents the results for two different *MinPts* values. From the graph as shown in *Figure 5*, it can be seen that IDBSCAN is less affected with the variation in *MinPts*. As the MinPts increases, the execution performance of DBSCAN deteriorates.

Table 2: IDBSCAN vs DBSCAN for the various sizes of datasets with two different MinPts

Data size	ε	MinPts	Time (in sec.) IDBSCAN	Time (in sec.) DBSCAN
100000	7	3	33	56
200000	5	3	78	124
400000	5	3	158	624
800000	4	3	498	1410
100000	10	12	27	73
200000	8	12	65	211
400000	6	12	140	749
800000	5	12	455	2366

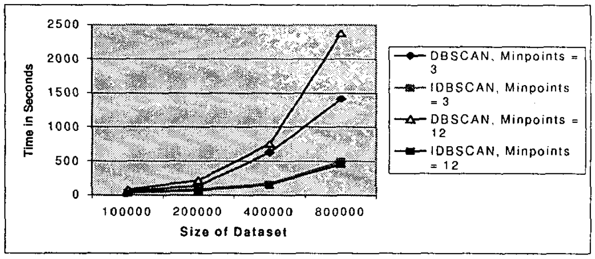

Fig 5: DBSCAN vs IDBSCAN for two different MinPts

Experiment 3

In this experiment the behaviour of both the algorithms have been studied by varying *MinPts* in the same dataset. The results are presented in *Table 3* and *Figure 6*. As *MinPts* increases, DBSCAN becomes slower, whereas the performance of the IDBSCAN improves. Even for smaller values of *MinPts*, a distinct time gap between IDBSCAN and DBSCAN can be observed.

Table 3: Comparison of IDBSCAN & DBSCAN for increasing MinPts

Data size	ε	MinPts	Time (in sec.) IDBSCAN	Time (in sec.) DBSCAN
400000	5	3	158	624
400000	6	20	141	878
400000	8	40	125	1160
400000	10	60	119	1454

Thus based on the results, it can be concluded that the proposed IDBSCAN (*i*) outperforms DBSCAN in terms of execution time and (*ii*) is capable to handle large volume of data.

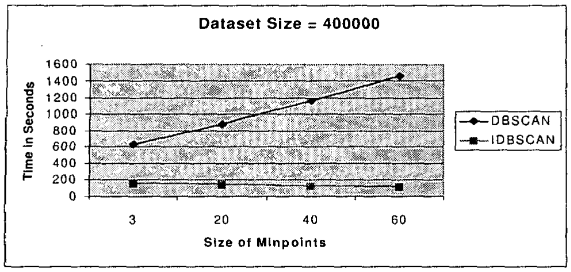

Fig 6: IDBSCAN vs DBSCAN as MinPts increases

CONCLUSIONS

This paper has presented an improved version of the DBSCAN algorithm for handling large volume of Spatial databases. The proposed algorithm extends the existing DBSCAN algorithm by incorporating a better sampling technique. Experimental results have established that due to the sampling technique, the I/O cost and the memory requirement for clustering are reduced dramatically and hence a considerable amount of run-time is reduced. It outperforms DBSCAN [2] and other sampling-based algorithm [3] in terms of execution time, without losing the quality of clustering.

REFERENCES

[1] E Kolatch, *Spatial Data Clustering Algorithms*, http://citseer.nj.nec.com/436843.html

[2] M Ester, H P Kriegel, J Sander and X Xu, *A Density Based Algorithm for Discovering Clusters in Large Spatial Databases*, in the Proc of 2nd Int'nl Conf. on Data Mining, 1996

[3] S Zhou, A Zhou, J Cao, J en, Y Fen and Y Hu, *Combining Sampling Technique with DBSCAN Algorithm for Clustering Large Spatial Databases*, Knowledge Discovery and Data Mining: Current Issues and New Applications, PAKDD2000, Japan, April 2000.

[4] J Han, M Kamber, *Data Mining Concepts and Techniques*, Morgan Kaufmann Publishers, 2001

[5] A K Pujari, *Data Mining Techniques*, University Press, 2001

[6] N Beckmann, H P Kriegel, R Schneider, *The R*-tree an Efficient and Robust Access Method for Points and Rectangles*, Proc. ACM SIGMOD, Utlantic City, USA, pp 322-331, May 1990