Improved DBSCAN for spatial databases with noise and different densities

Fahim A.M.[1] Saake G. [2] Salem A. M. [3] Torkey F. A. [4] Ramadan M. A. [5]

[1]Faculty of Information, Otto-von-Guericke-University, Magdeburg, Germany, ahmed.fahim@iti.cs.uni-magdeburg.de
[2]Faculty of information, Otto-von-Guericke-University, Magdeburg, Germany, saake@iti.cs.uni-magdeburg.de
[3]Faculty of Computers & Information, Ain Shams University, Cairo, Egypt, absalem@asunet.shams.edu.eg
[4]Kaferel shiekh University, Kaferel sheikh, Egypt, fatorkey@Yahoo.com
[5]Faculty of Science, Minufiya University, Shbien el koum, Egypt, mramadan@mailer.eun.eg

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. many useful spatial data clustering algorithms have been proposed in the past decade. DBSCAN is the most popular density clustering algorithm, which can discover clusters of any arbitrary shape and can handle the noise points effectively. However, DBSCAN can not discover clusters with large variance in density because it depends on globular value for its parameters Eps and MinPts which depend on each other. This paper presents an improved DBSCAN which can cluster spatial databases that contain clusters of different densities, shapes, and sizes effectively. Experimental results included to establish that the proposed improved DBSCAN is superior to DBSCAN in discovering clustering with large variance in density.

Key words*: DBSCAN, Data clustering, spatial databases.*

1. 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 high similarity in comparison to one another, but are dissimilar to objects in the other clusters. Based on a general definition, data clustering algorithms can be classified into four categories; (1) partitioning, (2) hierarchical, (3) density-based and (4) grid-based. However, some algorithms may fall into more than one category. By clustering one can identify dense and sparse regions and, therefore, discover overall distribution patterns. Finding clusters in data is challenging when the clusters are of widely differing sizes, shapes and densities and when the data contains noise and outliers. Although many algorithms exist for finding clusters with different sizes and shapes, there are a few algorithms that can detect clusters with different densities. Basic density based clustering techniques such as DBSCAN [3] and DENCLUE [4] treat clusters as regions of high densities separated by regions of no or low densities. So they are able to suitably handle clusters of different sizes and shapes besides effectively separating noise (outliers). But they fail to identify clusters with differing densities unless the clusters are separated by sparse regions. For example, in the dataset shown in Fig.1, DBSCAN can not find the four clusters, because this data set contains four level of density, the clusters are not totally separated by sparse regions and the value of *Eps* is globular for the data set. So we need an approach able to stop clustering expanding at different levels of densities.

We propose an improved version of the DBSCAN algorithm to discover clusters with different densities. A cluster is a spatial region containing objects of similar (homogeneous) density. Two adjacent spatial regions are separated into two clusters when the density difference violates a threshold. The proposed algorithm requires one input parameter only. This algorithm alleviates the dependency between the two input parameters of the DBSCAN, and it does not require the *Eps* at all.

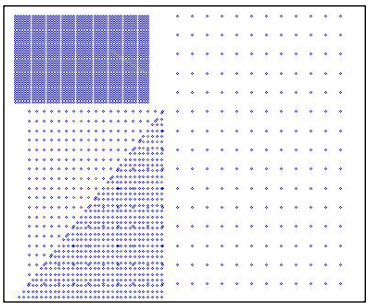


Fig.1. clusters with varying densities

The paper is organized as follows. In section 2 we review the DBSCAN algorithm. Section 3 briefly surveys some of recent existing clustering methods related to our approach. In section 4, we present the proposed algorithm. Section 5 presents some experimental results to evaluate the algorithm and we conclude with section 6.

2. Density based clustering (DBSCAN)

DBSCAN is based on the concept of dense areas to form data clustering. The distribution of points in the cluster should be denser than that outside of the cluster. DBSCAN is the first algorithm that clusters using density. Density based clustering methods allow the identification of arbitrary, not necessarily convex regions of data points that are densely populated. Density based clustering does not need the number of clusters beforehand but relies on a density-based notion of clusters such that for each point of a cluster the neighborhood of a given radius *Eps* has to contain at least a minimum number of points *MinPts*. However, finding the correct parameters for standard density based clustering [3] is more of an art than science. Here we review the basic ideas and definitions which involved in the DBSCAN algorithm.

Let D is a database of N points in d-dimensional space R^d. The distance between two data points p and q is given by the Euclidean distance and denoted by $d(p,q)$. The DBSCAN depends on the following definitions:

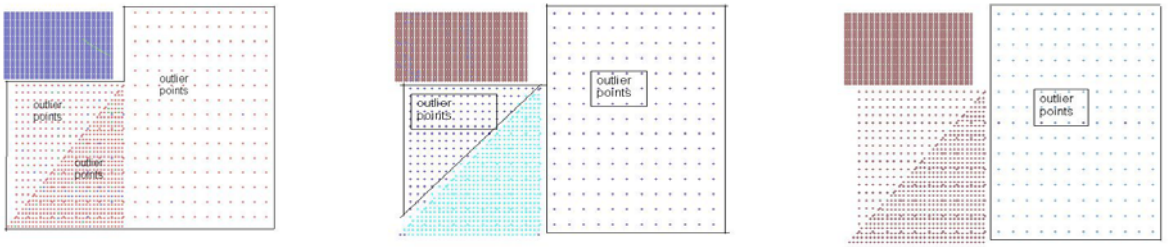
- The *Eps-neighborhood* of a point p, denoted by $NEps(p)$, is defined by $NEps(p) = \{q \in D \mid d(p,q) \le Eps\}$.
- A point p is *directly density-reachable* from a point q wrt. *Eps* and *MinPts* if $p \in NEps(q)$ and $|NEps(q)| \ge MinPts$ (core point condition).
- A point p is *densityreachable* from a point q wrt. *Eps* and *MinPts* if there is a chain of points $p_1, ..., p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density-reachable from p_i.
- A point p is *densityconnected* to a point q wrt. *Eps* and *MinPts* if there is a point o such that both, p and q are density-reachable from o wrt. *Eps* and *MinPts*.
- A *cluster* C wrt. *Eps* and *MinPts* is a non-empty subset of D satisfying the following conditions:
  1) $\forall\ p, q$: if $p \in C$ and q is density-reachable from p wrt. *Eps* and *MinPts*, then $q \in C$. (Maximality)
  2) $\forall\ p, q \in C$: p is density-connected to q wrt. *Eps* and *MinPts*. (Connectivity)
- If $C_1,..., C_k$ be the clusters of the database D wrt. parameters Eps_i and $MinPts_i$, $i = 1, ..., k$. Then the noise is defined as the set of points in the database D not belonging to any cluster C_i, i.e. noise = $\{p \in D \mid \forall\ i: p \notin C_i\}$.
- If $d(p,q) \le Eps$, q is core point and $|NEps(p)| < MinPts$ then p is a border point.

DBSCAN searches for clusters by checking the *Eps neighborhood* of each point in the database. If the *Eps neighborhood* of a point P contains more than *MinPts,* a new cluster with P as

core point is created. DBSCAN then iteratively collects directly density-reachable points from these core points, which may involve the merge of a few *density-reachable clusters*. The process terminates when no new points can be added to any cluster. In DBSCAN the *MinPts* is fixed to 4 points, and the number of clusters discovered depends on the *Eps* value which is fixed during the execution process, and this value is not suitable for discovering clusters having different densities except when the clusters are totally separated. But when the clusters are not totally separated the DBSCAN algorithm faces the problem of varying densities and produces inaccurate clusters.

3. Related works

The DBSCAN (Density Based Spatial Clustering of Applications with Noise) [3] is a basic density based clustering algorithm. The density associated with a point p is obtained by counting the number of points in a region of specified radius, *Eps*, around p. A point with density greater than or equal to a specified threshold, *MinPts*, is treated as core point (dense), otherwise non-core (sparse). Non-core points that do not have a core point within the specified radius are discarded as noise. Clusters are formed around core objects by finding sets of density connected points that are maximal with respect to density-reachability. DBSCAN can find clusters having variable sizes and shapes, but there may be wide variation in local densities within a cluster since it uses global density parameters *MinPts* and *Eps*, which specify only the lowest possible density of any cluster. Figure 2 shows the results from applying the DBSCAN on the dataset in Figure 1.



(a) (b)

Fig. 2 in (a) DBSCAN discovers only the most density cluster (*Eps*=0.25). In (b) it discovers two clusters satisfying the lowest possible density (*Eps*=0.4). In (c) it merges the two clusters in (b) with the intermediate points to produce only one cluster (*Eps*=0.5) and discards the other points as noise

From Figure 2, we note that for some datasets we need many different values for the *Eps* parameter of DBSCAN algorithm to discover clusters with different densities, and we can not depend on a single value for this sensitive parameter. Also the value of *Eps* depends on the *MinPts* which determine the minimum level of density for core points. This problem motivates us to propose an improvement for DBSCAN to make it able to discover clusters having significant difference in their densities.

To find clusters that are naturally present in a dataset very different local densities needed to be identified and separated into clusters. There are some improvement have been done on the DBSCAN algorithm. But these improvements focus on the scalability of the algorithm by reducing the number of query region [2]. This has been done by sampling the points in the *Eps* neighborhood of core point p and marked these points as Marked Boundary Objects (*MBO*), for each of these *MBOs*, the closest point in the *Eps*-neighborhood -as in Figure 3- is identified and selected as a seed. If the same point is identified as the nearest point for more than one *MBO*, then this point must be regarded only once as a seed. Therefore total number of seeds selected may be less than or equal to eight in 2-dimensional case. In general, if the points are of dimension d then there will be $(3^d\text{-}1)$ *MBOs,* 2^d quadrants and number of seeds selected is at most $3^d\text{-}1$. This number is lower than the corresponding number in DBSCAN. IDBSCAN [2] yields the same quality of DBSCAN but is more efficient.

Another algorithm is introduced to improve the efficiency of DBSCAN by merging between the k-means and IDBSCAN, this algorithm is known as KIDBSCAN [5]. This algorithm combines

K-means and IDBSCAN. K-means yield the core points of clusters. Then, clusters are expanded from these core points by executing IDBSCAN. IDBSCAN has the advantages of density-based clustering and employs sampling techniques, to reduce the execution time below that of DBSCAN. The synergy provided by merging K-means and IDBSCAN is an effective way to reduce the I/O cost. The purpose of the k-means here is to determine the *k* highest density center points and the points that are closest to the center points. These points are moved to the front of the data. And the IDBSCAN start the expansion of clusters from high density center points, this process reduces the number of redundant steps and increases efficiency.

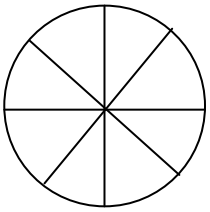


Fig.3 Circle with eight distinct points (*MBO*s)

Recently DDSC (Density Differentiated Spatial Clustering) [1] is proposed. It is an extension of the DBSCAN algorithm to detect clusters with differing densities. Adjacent regions are separated into different clusters if there is significant change in densities. DDSC starts a cluster with a homogeneous core object and goes on expanding it by including other directly density reachable homogeneous core objects until non homogeneous core objects are detected. A homogeneous core object *p* is a core object whose density (the count of points in its *Eps*-neighborhood) is neither more nor less than α time the density of any of its neighbors, where $\alpha > 1$ is a constant. DDSC requires three input parameters, they are *Eps*, *MinPts* and α. As we know *Eps* and *MinPts* depend on each other. However, finding the correct parameters for standard density based clustering [3] is more of an art than science. In addition to the third parameter α, which represent the allowed variance densities inside the cluster, and proper tuning of the parameter values is very important for getting good quality results.

4. The Proposed Algorithm

In this section, we describe the details of our proposed algorithm. As DBSCAN is sensitive to *Eps* we will not use this parameter in our proposed algorithm. The alternative choice is to find the *k*-nearest neighbors for each point in given dataset. Based on the *k*-nearest neighbors we use a local density function that is an approximation of the overall density function [4]. The overall density of the data space can be calculated as the sum of the influence function of all data points. We compute local density of point according to the following functions:

Influence function represents the impact of point *x* on point *y* as the Euclidean distance between them.

$$INF(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

As the distance between the two points decrease the impact of *x* on *y* increase, and vice versa.

The local density function for point *x* is defined as the summation of (influence functions within the *k*-nearest neighbors) distances among the point *x* and the *k*-nearest neighbors. The local density function is defined as

$$DEN(x, y_1, ..., y_k) = \sum_{i=1}^{k} INF(x, y_i)$$

The definition of density based on the summation of distances of the *k*-nearest neighbors is better than counting the points in the neighborhood radius (*Eps*). Consider the following example as in Figure 4.

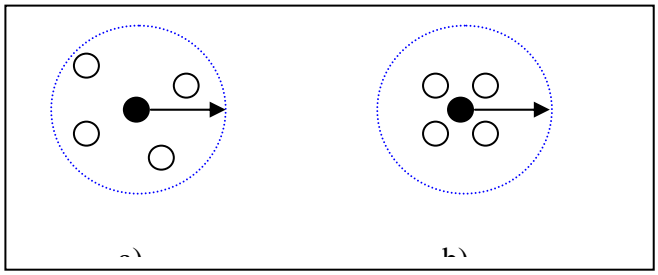


Fig.4 Density based on distance to *k* nearest neighbors

As we see in Figure 4.a and 4.b both black points have four points in *Eps*-neighbor (the *Eps* is the same represented by black arrow) but Figure 4.b is more densely than Figure 4.a. The summation of distances reflects this fact accurately. But based on the *Eps* neighborhood radius as in DBSCAN algorithm there is no difference, because each point has four points in its *Eps*. And we can note that the value of *Eps* vary according to the density using the *k*-nearest neighbors. Also the point inside the cluster has high density, on the other hand the point at the edge (border) of cluster has low density, since the neighbors for this point lie on one side of it, but the point at the core of cluster has its neighbors surrounding it from all sides. So density based on distance is better than based on point counting in neighborhood radius.

The algorithm finds the *k*-nearest neighbors for each point *p*, and keeps them in ascending order according to their distances to *p*. (i.e. $N_k(p)=\{q \in D,\ d(p,q_{i-1}) \le d(p,q_i),\ i=1,\ldots,k\}$), for each neighbor q_i of *p* its distance to *p* and input order of q_i in the input dataset are kept in one vector. Figure 5 explains this idea. The algorithm computes the density on each point *p* according Equation 2. We arrange the points in dataset in ascending order according to the local density of each point using the quick sort. We put the data points in ordered linked list. So the first point in the list will be the highest densely point and the last one will be the lowest densely. And we create the high density clusters first, i.e. we move from high density to low density.

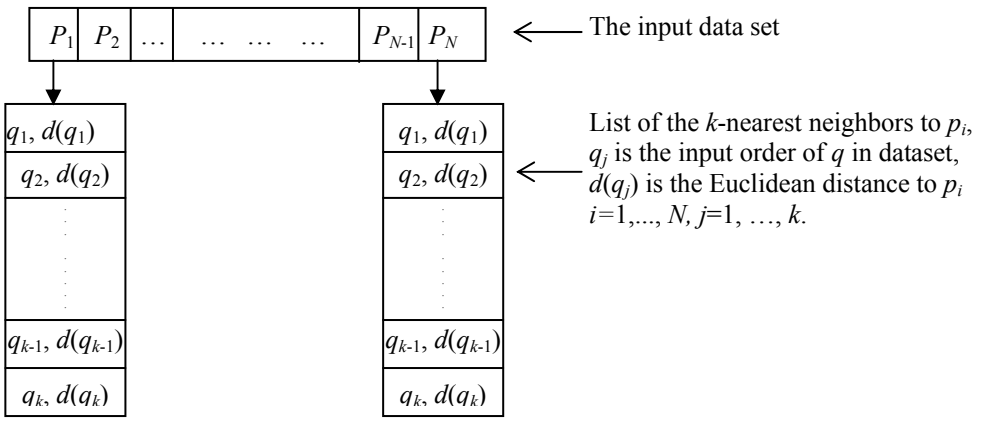


Fig.5 The arranged *k*-nearest neighbors for each point

We initialize all point as unclassified (*ClusId* = -1). Starting from the first(highest densely) point *p* as the first point in the first cluster, we expand the cluster around *p* starting from the nearest point q_i from its neighbors, all unclassified neighbors q_j that are in the *k*-nearest neighbors list are appended to the seed list such that the density difference between q_i and the first (starting) point of the cluster is less than or equal a threshold, this process continues until no point can be added to the current cluster, and new cluster will be created in the same manner as the first cluster until all points

are classified. The very small clusters are discarded as noise. By this way every created cluster will be start with highest densely point *p*, all points added to the cluster will have less density than *p*, but must satisfy level of difference in density in the cluster, this level of difference in density controlled by the input parameter density ratio (*Dr*), which is the only parameter of the algorithm. More formally, let *p* is the highest density point in the current cluster, then all points directly-reachable from *p* satisfying inequality 3 and are not classified, will be assigned to the current cluster and appended to the seed list to test its *k*-nearest neighbors, the cluster continue grow until the seed list is empty and a new cluster will be start until all points are classified.

$$Dr.\frac{DEN(q,y_1,...y_k)}{k} \le \frac{DEN(p,x_1,...,x_k)}{k}$$

Where *k* is the number of nearest neighbors, since $DEN(q,y_1,\ldots,y_k)$ is the summation of distance among the point *q* and its *k* neighbors, as *k* increase the $DEN(q,y_1,\ldots,y_k)$ increase, so it is preferable to use the ratio between them as in equality 3. This ratio represents the mean variance of the *k* points from a central point *q*. to help the user determining a suitable value for *Dr*, the algorithm divides the data set into 20 intervals based on the value of $DEN(x,y_1,\ldots,y_k)$, and numerates the points in each interval, this is look like the histogram of DBSCAN. The following are the main steps of our proposed algorithm.

1. Find the *k*-nearest neighbors for each point *p* ($N_k(p)$) and keep them in ascending order from *p*.
2. Set local density value for each point p as $DEN(p,y_1,\ldots,y_k)$.
3. Rearrange the data in ascending order based on the local density
4. Starting from the first point *p* in sorted data do the following
   a. Assign the point *p* to the current cluster (*ClusId*)
   b. Append its unclassified neighbors q_i, $i=1,\ldots,k$ to the seed list SL_p in ascending order of their distance to *p* such that inequality 3 is satisfied, where *p* is the first point added to the cluster, i.e. *p* is the highest density point in the cluster, and assign the points appended to SL_p to the current cluster *ClusId*.

Step 1 imposes an ordering on the *k*-nearest neighbors, and these neighbors will be appended to the seed list SL_p in the same order as in step 4 b. Step 3 makes the creation of clusters start from high densely clusters and ended with low densely clusters, at the same time allows density variance inside a cluster according the value of *Dr*, which is the only input parameter to our algorithm. Appending the points in ascending order to the seed list impose growing of cluster in contiguous regions. Since the density of every point in the cluster compared with the density of the starting point of the cluster, the algorithm may discover small high density clusters.

4.1 Time complexity

The most time consuming part of the algorithm is to find the k-nearest neighbors. The neighborhood size (k=10) is very small compared to the size of the dataset. So, the different tests performed on the neighborhood to arrange them will not consume much time. While expanding a cluster the list of newly contributed seeds by each object of the cluster are already sorted before. For all objects only a small fraction of the neighbors become new seeds, whereas some points contribute no new seeds at all. The time required for a neighborhood query is O(log n) by using a spatial access method such as R*-tree. Neighborhood query is performed for each of the *n* points in the dataset. Also we arrange the points according their local density using quick sort, which require O(*n log n*). If we don't arrange the points, we must search for the highest densely point in each new cluster creation, this process requires O(*nm*); where *m* is the number of obtained clusters and *n* is the size of the input dataset. So the run time complexity is O(*n log n*) which is the same as that of DBSCAN.

5. Experimental Results

In this section we evaluate the performance of our proposed algorithm. We implemented this algorithm in C++. We have used many synthetic datasets to test our proposed algorithm. We experimented with four different data sets containing points in two dimensions whose geometric shape are shown in Figure 6.

Dataset 1 (8000 points) Dataset 2 (8000 points) Dataset 3 (24000 points) Dataset 4 (3147 points)

Fig.6 Datasets used in our algorithm and their sizes

The first dataset has six clusters of different size, shape, and orientation, as well as random noise points and special artifacts such as streaks running across clusters. The second dataset has six clusters of different shape. Moreover, it also contains random noise and special artifacts, such as a collection of points forming horizontal streak. The third dataset has three nested circular clusters of different size, and density. A particularly challenging feature of this data set is that clusters are very close to each other and they have different densities and there is no separation between clusters. Finally, the fourth dataset has four clusters of different shapes, sizes, and densities. The size of these data sets ranges from 3147 to 24000 points, and their exact size is indicated in Figure 6.

The clustering results for the four datasets in Figure 6 are shown in Figure 7. Different colors are used to indicate the clusters. The very small clusters are discarded as noise. It can be seen from Figure 7 that triangular and rectangular clusters are discovered and extracted based on differences in densities although they are not separated by sparse regions. This is will be more clearly when we compare the result of dataset 4 in Figure 7 with Figure 2 that result from the DBSCAN algorithm.

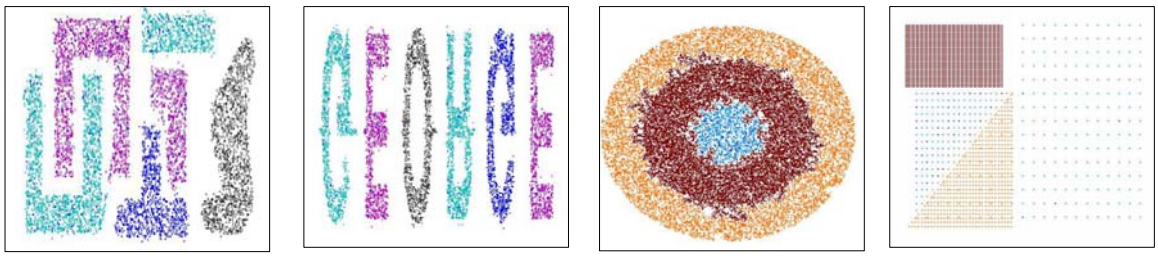


Fig.7 The results from our proposed algorithm, the very small clusters are discarded as noise

6. Conclusion

In this paper, we have introduced a simple idea to improve the results of DBSCAN algorithm by detecting clusters with large variance in density without requiring the separation between clusters. Our proposed algorithm requires only one input parameter that specifies the allowed level of density difference inside the cluster. The time complexity of the algorithm remains O(*n log n*). It also alleviates another important disadvantage of DBSCAN; sensitivity of input parameter *Eps* and *MinPts*. Our experimental results are the evidence on the efficiency of the proposed algorithm.

7. References

[1] Borah B., Bhattacharyya D.K.: " DDSC: A Density Differentiated Spatial Clustering Technique." JOURNAL OF COMPUTERS, FEBRUARY 2008, VOL. 3, NO. 2, pp. 72-79.

[2] Borah B. and Bhattacharyya D.K.: "An Improved Sampling-Based DBSCAN for Large Spatial Databases." proceedings of International Conference on Intelligent Sensing and Information, 2004, pp. 92-96.

[3] Ester M., Kriegel H. P., Sander J., and Xu X., "A density based algorithm for discovering clusters in large spatial data sets with noise." in *2nd International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[4] Hinneburg A. and Keim D.: "An efficient approach to clustering in large multimedia data sets with noise." in *4th International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 58–65.

[5] Tsai C-F. and Liu C-W. : "KIDBSCAN: A New Efficient Data Clustering Algorithm.", ICAISC, 2006, PP. 702-711.

This article contains 7 figures.

Paper received: 2008-08-25