Faster Clustering with DBSCAN

Marzena Kryszkiewicz and Łukasz Skonieczny

Institute of Computer Science, Warsaw University of Technology, Nowowiejska 15/19, 00-665 Warsaw, Poland

Abstract. Grouping data into meaningful clusters belongs to important tasks in the area of artificial intelligence and data mining. DBSCAN is recognized as a high quality scalable algorithm for clustering data. It enables determination of clusters of any shape and identification of noise data. In this paper, we propose a method improving the performance of DBSCAN. The usefulness of the method is verified experimentally both for indexed and non-indexed data.

1 Introduction

Grouping data into meaningful clusters belongs to important tasks in the area of artificial intelligence and data mining. Clustering can be perceived as unsupervised classification of data. A number of clustering algorithms were offered in the literature. Usually, different clustering algorithms group data differently. Some of the algorithms are capable to discover proper clustering of data only when the number of the clusters is known. Other algorithms are capable to discover clusters only of particular shapes. There are algorithms that are unable to identify noise data. The DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise) [5] is recognized as a high quality scalable algorithm for clustering, which is free of these limitations. It belongs to the class of density-based algorithms. Other distinct representative algorithms of this class are: Denclue [8], DBCLASD [10], Optics [1] and O-Cluster [9]. The characteristic feature of such algorithms is that they perceive the elements of the data set not as particular objects, but a substance that fills data space. We say that the area is of high density if it contains a large number of elements; otherwise, the area is of low density. Under this understanding of space, a cluster is an area of density exceeding the required threshold value or greater than the density of the surrounding space. The areas that do not constitute clusters are treated as noise.

A distinct feature of DBSCAN is its simplicity. Although simple, DBSCAN is very efficient and finds clusters of high quality, as well as determines noise correctly. In this paper, we propose a method improving the performance of this algorithm. Our solution is based on the observation that the fewer points are in a data set, the shorter is a clustering process. In accordance with this observation, we propose a method that deletes some points from the data set as soon as possible without changing the clustering results. The usefulness of the method is verified experimentally both for indexed and non-indexed data.

The paper has the following layout. Section 2 recalls the notion of a cluster, which was introduced in [5]. Section 3 presents the DBSCAN algorithm. In Section 4, we offer an optimized version of DBSCAN. Section 5 reports the performance of the DBSCAN algorithm and its optimized version on different indexed and non-indexed data sets. The results are concluded in Section 6.

2 Basic Notions

The authors of the DBSCAN algorithm [5] understand a cluster as an area of high density. Low density areas constitute noise. A point in space is considered a member of a cluster if there are a sufficient number of points within a given distance. The distance between two points p and q will be denoted by dist(p,q). The authors of DBSCAN do not impose the usage of any specific distance metric[1]. Depending on an application, one metric may be more suitable than the other. In particular, if Euclidean distance is used, a neighborhood of a point has a spherical shape; when Manhattan distance is used the shape is rectangular. For simplicity of the presentation, the examples we provide in this paper refer to Euclidean distance. Below we recall formal definitions of a density based cluster and related notions.

Definition 1. (Eps-neighborhood of a point)
Eps-neighborhood of a point p ($\mathrm{N_{Eps}(p)}$) is a set of points q in data set D that are distant from p by no more than Eps; that is, $\mathrm{N_{Eps}(p) = \{q \in D \mid dist(p, q) \leq Eps\}}$.

Definition 2. (a core point)
p is a *core point* if its *Eps-neighborhood* contains at least MinPts points; that is, if $\mathrm{\mid N_{Eps}(p) \mid \geq MinPts}$.

Definition 3. (directly density-reachable points)
A point p is *directly density-reachable* from a point q w.r.t. Eps and MinPts if the following two conditions are satisfied:
1) $\mathrm{p \in N_{Eps}(q)}$,
2) q is a core point.

Suppose MinPts = 6. Point q in Figure 1a has 6 neighbors (including itself) in its neighborhood $\mathrm{N_{Eps}(q)}$, so it is a core point. Point p has 2 neighbors in $\mathrm{N_{Eps}(p)}$, hence it is not a core point. Point p, however, is directly density-reachable from q, while q is not directly density-reachable from p.

Definition 4. (density-reachable points)
A point p is *density-reachable* from a point q w.r.t. Eps and MinPts if there is a sequence of points $\mathrm{p_1, \ldots, p_n}$ such that $\mathrm{p_1 = q}$, $\mathrm{p_n = p}$ and $\mathrm{p_{i+1}}$ is directly density-reachable from $\mathrm{p_i}$, i = 1..n-1.

[1] Overview of metrics used for clustering purposes can be found e.g. in [7].

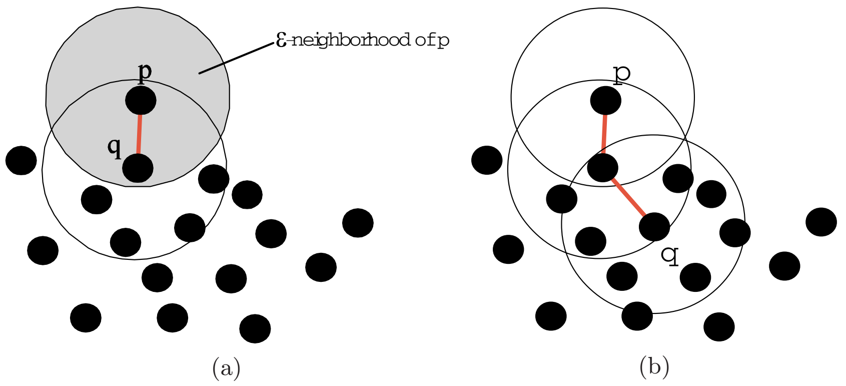


Fig. 1. (**a**) p is directly density-reachable from q (MinPts = 6). (**b**) p is density-reachable from q (MinPts = 6)

Point p in Figure 1b is density-reachable from point q, since there is a point such that p is directly density-reachable from it and it is directly density-reachable from q. Please note that p, which is density-reachable from core point q, is not a core point.

Definition 5. (a border point)
p is a *border point* if it is not a core point, but is density-reachable from some core point.

Hence, a point is a border one if it is not a core point, but belongs to the Eps-neighborhood of some core point.

Let C(p) determine all points in D that are density-reachable from point p. Clearly, if p is not a core point, then C(p) is empty.

Definition 6. (cluster)^2
A cluster is a non-empty set of all points in D that are density-reachable from a core point.

Hence, each C(p) is a cluster provided p is a core point. Interestingly, if p and q are core points belonging to the same cluster, then C(p) = C(q); that is, both points determine the same cluster [5]. Thus, a core point p belongs to exactly one cluster, namely to C(p). However, a border point may belong to more than one cluster.

Definition 7. (noise)
The *noise* is the set of all points in D that do not belong to any cluster; that is, the set of all points in D that are not density-reachable from any core point.

2 This definition differs from the original one provided in [5]. However, the definition of a cluster presented here is equivalent to the original one by Lemma 1 in [5], and is more suitable for our presentation.

Hence, points that are neither core points, nor border ones, are noise.

3 Clustering with DBSCAN

In this section, we recall the DBSCAN algorithm. Its input parameters are: the set of points D, the maximal radius of the neighborhood Eps and the required minimal number of points MinPts within Eps-neighborhood (please, see [1,5] for the method of determining appropriate values of parameters Eps and MinPts). There is associated ClusterId field with each point in D. Initially, each point in D is assumed to be unclassified (ClusterId = UNCLASSIFIED).

The algorithm starts with generating a label for first cluster to be found. Then, it analyzes point after point in D. Clearly, the value of ClusterId field of a first checked point equals UNCLASSIFIED. As will follow from the logic of the program, the labels of some points may be changed before they are analyzed. Such a situation occurs when a point is density-reachable from a core point analyzed earlier. In this case, the point will be assigned ahead to the cluster of the core point. Such pre-classified points will be skipped from analysis. If, however, a currently analyzed point p is still unclassified, then the ExpandCluster function (described later) is called for it. If p is a core point, then the function assigns the current cluster's label to all points in C(p), and DBSCAN generates a label for a new cluster to be built. Otherwise, the function assigns label NOISE to point p.

After all points in D are analyzed, ClusterId of each point is assigned either a label of a respective cluster or label NOISE.

Algorithm DBSCAN(set of points D, Eps, MinPts);

```
ClusterId = label of first cluster;
for each point p in set D
    if (p.ClusterId = UNCLASSIFIED) then
        if ExpandCluster(D, p, ClusterId, Eps, MinPts) then
            ClusterId = NextId(ClusterId)
        endif
    endif
endfor
```

The ExpandCluster function starts with calculating Eps-neighborhood of point p passed as a parameter. If its Eps-neighborhood does not contain sufficient number of points, then it is not a core point. In such a case, it is temporary labeled as a noise point and the function reports failure of expanding a cluster. Otherwise, the examined point is a core point and all points that are density-reachable from it will constitute a cluster. First, all points in the neighborhood of the examined point are assigned a label (ClId) of the currently created cluster. All points in $N_{Eps}(p)$, except for p, are stored in the seeds collection and are subject to determination of their Eps-neighborhood. Each seed point, which is a core point, further extends the seeds collection with the points in its Eps-neighborhood that are still unclassified. Clearly, the points in the seeds collection that were classified earlier as noise are border points of the current cluster, and hence are assigned the current cluster label[3]. After processing a seed point, it is deleted from the seeds collection. The function ends when all points found as cluster seeds are processed.

function ExpandCluster(D, point p, cluster label ClId, Eps, MinPts)

```
seeds = Neighborhood(D, p, Eps);
if |seeds| < MinPts then
    p.ClusterId = NOISE;
    return FALSE
else
    for each point q in seeds do                    // including point p
        q.ClusterId = ClId;
    endfor
    delete p from seeds;
    while |seeds| > 0 do
        curPoint = first point in seeds;
        curSeeds = Neighborhood(D, curPoint, Eps);
        if |curSeeds| >= MinPts then
            for each point q in curSeeds do
                if q.ClusterId = UNCLASSIFIED then
                /* $N_{Eps}(q)$ has not been evaluated yet, so q is added to seeds */
                    q.ClusterId = ClId;
                    append q to seeds;
                elseif q.ClusterId = NOISE then
                /* $N_{Eps}(q)$ has been evaluated already, so q is not added to seeds */
                    q.ClusterId = ClId;
                endif
            endfor
        endif
        delete curPoint from seeds;
    endwhile
    return TRUE
endif
```

Please note that Eps-neighborhood of point p passed as parameter to the

[3] Although a border point may belong to many clusters, DBSCAN assigns it only to one of them. It is straightforward to modify the algorithm so that the border points are assigned to all including clusters.

ExpandCluster function may contain points classified earlier as NOISE. For such points, the function redundantly calculates their neighborhoods.

Figure 2 illustrates the initial part of sample execution of the DBSCAN algorithm.

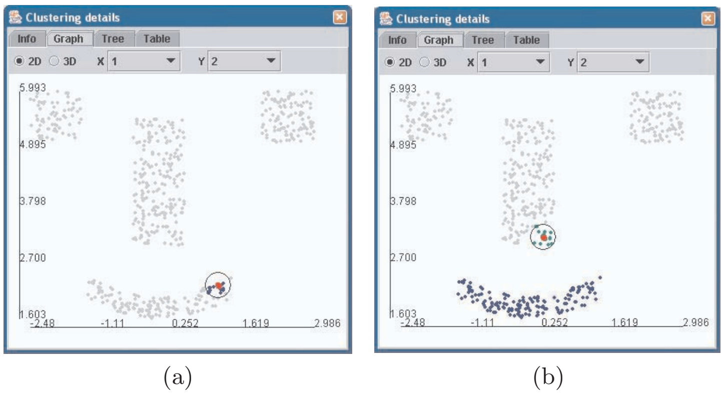


Fig. 2. (**a**) Neighborhood of the first (core) point assigned to a cluster. (**b**) Subsequent assignment of density-reachable points forms first cluster; the initial seeds are determined for the second cluster

4 Optimizing DBSCAN by Early Removal of Core Points

The solution we propose consists in removing a point from set D as soon as it is found to be a core point by the ExpandCluster function.

Clearly, if a point p is deleted, then for each point q in its Eps-neigborhood, $|\ N_{Eps}(q)\ |$ determined in the reduced set D is smaller than $|\ N_{Eps}(q)\ |$ determined in the original set D. In order to calculate the sizes of Eps-neighborhoods correctly, there is associated the NeighborsNo field with each point in D. The NeighborsNo field of point q stores the number of the q's neighboring core points that were earlier deleted from D. More specifically, whenever a core point p is deleted, the NeighborsNo field is incremented for all points in $N_{Eps}(p)$. Now, the real size of neighborhood of each point r remaining in D is counted as the sum of the cardinality of $N_{Eps}(r)$ found in the reduced set D and the value of the NeighborsNo field of r.

The ExpandCluster function can be optimized further by skipping redundant determination of neighborhood of points already classified as NOISE.

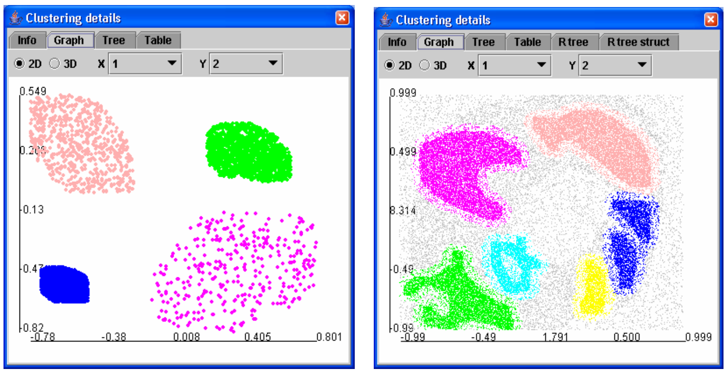
Our optimized version of the ExpandCluster function is called ERCP-ExpandCluster (ExpandCluster using Early Removal of Core Points). The code of the ERCP-ExpandCluster function is provided below.

function ERCP-ExpandCluster(D, point p, ClId, Eps, MinPts)

```
seeds = Neighborhood(D, p, Eps);
p.NeighborsNo = p.NeighborsNo + |seeds|;
if p.NeighborsNo < MinPts then
    p.ClusterId = NOISE;
    return FALSE
else
    p.ClusterId = ClId;
    move p from D to D';          // clustered core points will be stored in D'
    delete p from seeds;
    for each point q in seeds do
        if q.ClusterId = NOISE then
            q.ClusterId = ClId;
            /* $N_{Eps}(q)$ of noise q shall not be determined */
            delete q from seeds;
        else
            q.ClusterId = ClId;
            q.NeighborsNo = q.NeighborsNo + 1;
        endif;
    endfor
    while |seeds|> 0 do
        curPoint = first point in seeds;
        curSeeds = Neighborhood(D, curPoint ,Eps);
        curPoint.NeighborsNo = curPoint.NeighborsNo + |curSeeds|;
            if curPoint.NeighborsNo >= MinPts then
            for each point q in curSeeds do
                if q.ClusterId = UNCLASSIFIED then
                    q.ClusterId = ClId;
                    q.NeighborsNo = q.NeighborsNo + 1;
                    append q to seeds;
                elseif q.ClusterId = NOISE then
                    q.ClusterId = ClId;          // q is a border point
                endif
            endfor
            move curPoint from D to D';
        endif
        delete curPoint from seeds;
    endwhile
    return TRUE
endif
```

5 Experimental Results

To examine the practical relevance of our proposal, we performed an experimental evaluation of the optimized algorithm and compared it to that of original DBSCAN. We tested data of dimensionality from 2 up to 8. We have also examined how the usage of spatial indexes for evaluating Eps-neighborhoods influences the algorithms' performance. Namely, we investigated the usefulness of applying R tree index [6], R*-tree index [4], and UB-tree index [2,3] for clustering with DBSCAN. Table 1 presents the experimental results. The table shows that the optimized version performs faster then the original DBSCAN by up to 50%. The best speed-up can be observed for data sets



(a) (b)

Fig. 3. (**a**) Data set d1-2D. (**b**) Data set ds1-2D

containing little noise. In particular, such a speed-up can be observed for little noise data set d1-2D, which is presented in Figure 3a. On the other hand, for large noise data set d1s-2D, which is presented in Figure 3b, speed-up does not exceed 40%. This phenomenon can be justified as follows: Most of the points in data sets with little noise are core points and are removed from the set immediately after their neighborhood is determined, which makes the evaluation of neighborhood of the remaining points faster. A simplified theoretical cost model of neighborhood evaluation can also lead us to this result. Consider f(n) to be the cost of evaluating Eps-neighborhood of a given point in the set of n points[4]. Original DBSCAN has to perform such evaluation for every point in the set giving the total cost $F_1(n) = n \times f(n)$. Optimized version also evaluates neighborhood for every point, but the cost of single evaluation decreases while subsequent points are removed from the data set.

[4] Notice that for simplicity we assumed that the cost of determining Eps-neighborhood of a given point is the same for every point and depends only on the size of data set.

Table 1. Results of experiments

Clustering time [s]				
DataSet	Index	Original DBSCAN	Optimized DBSCAN	Optimized/ Original
d1-2D	R-tree	3.23	1.65	51.08%
size: 5550	R*-tree	2.87	1.44	50.17%
MinPts: 5	UB-tree	2.66	1.44	54.14%
Eps: 0.1	no index	4.77	2.48	51.99%
ds1-2D	R-tree	10.5	7.49	71.33%
size: 50000	R*-tree	8.8	6.2	70.45%
MinPts: 5	UB-tree	22.2	13.4	60.36%
Eps: 0.1	no index	313	253	80.83%
ds2-2D	R-tree	18.7	11.5	61.50%
size: 50000	R*-tree	13.9	8.68	62.45%
MinPts: 5	UB-tree	30.8	18.2	59.09%
Eps: 0.1	no index	318	266	83.65%
ds3-2D	R-tree	5.17	3.94	76.21%
size: 40000	R*-tree	3.97	2.86	72.04%
MinPts: 5	UB-tree	9.8	6.7	68.37%
Eps: 0.1	no index	201	168	83.58%
d1-4D	R-tree	5.76	3	52.08%
size: 5550	R*-tree	5.46	2.78	50.92%
MinPts: 5	UB-tree	6.89	5.31	77.07%
Eps: 0.1	no index	7.91	4.05	51.20%
ds1-4D	R-tree	21.7	13.1	60.37%
size: 5550	R*-tree	16.8	10	59.52%
MinPts: 5	UB-tree	103	77	74.76%
Eps: 0.1	no index	409	303	74.08%
d1-8D	Rtree	9.32	5.09	54.61%
size: 5550	R*-tree	9.5	5.33	56.11%
MinPts: 5	UB-tree	58.8	46	78.23%
Eps: 0.1	no index	7.94	4.39	55.29%

In an ideal case, every point in the data set is a core point and can be removed from it as soon as its neighborhood has been found. Therefore, the total cost of neighborhoods' evaluation by the optimized version of DBSCAN $\mathrm{F}_2(\mathrm{n}) = \mathrm{f}(\mathrm{n}) + \mathrm{f}(\mathrm{n1}) + \ldots + \mathrm{f}(2) + \mathrm{f}(1)$. In the case when index is not used, we have $\mathrm{f}(\mathrm{n}) = \mathrm{n}$. Hence, $\mathrm{F}_1(\mathrm{n}) = \mathrm{n} \times \mathrm{n} = \mathrm{n}^2$ and $\mathrm{F}_2(\mathrm{n}) = \mathrm{n} + (\mathrm{n} - 1) + \ldots + 2 + 1 = \mathrm{n} \times (1 + \mathrm{n})/2 \approx 1/2 \times \mathrm{F}_1(\mathrm{n})$.

Our optimization is particularly useful for data of high dimensionality. Indexes on such data are known to be of little use in finding Eps-neighborhood. Our experiments confirm this fact. Table 1 shows that using an index for 8 dimensional data set d1-8D results in slower clustering of data than clustering without index. This observation holds both for clustering data by means of original DBSCAN and its optimized version. The achieved experimental results did not point out which index is the best in terms of DBSCAN efficiency. Usefulness of an index depends on data and input parameters.

6 Conclusions

A method for improving the performance of DBSCAN has been offered. It is based on early removal of core points. The experiments show that using the proposed method speeds up DBSCAN's performance by up to 50%. As a future work, we plan to enhance this method by enabling early removal of border and noise points as well.

References

1. Ankerst, M., Breunig, M.M. et al. (1999) OPTICS: Ordering Points To Identify the Clustering Structure. SIGMOD, 49-60
2. Bayer, R. (1997) The Universal B-Tree for Multidimensional Indexing. WWCA, 198-209
3. Bayer, R. (1997) UB-Trees and UB-Cache, A new Processing Paradigm for Database Systems. Technical Report TUM-I9722
4. Beckmann, N., Kriegel, H.P. (1990) The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. SIGMOD, 322-331
5. Ester, M., Kriegel, H.P. et al. (1996) A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise. KDD, 226-231
6. Guttman, A.(1984) R-Trees: A Dynamic Index Structure For Spatial Searching. SIGMOD, 47-57
7. Han, J., Kamber, M.(2000) Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers
8. Hinneburg, A., Keim, D.A. (1998) An Efficient Approach to Clustering in Large Multimedia Databases with Noise. KDD, 58-65
9. Milenova, B.L., Campus, M.M. (2002) O-Cluster: Scalable Clustering of Large High Dimensional Data Set. ICDM, 290-297
10. Xu, X., Ester, M. et al. (1998) A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases. ICDE, 324-331