TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality

Marzena Kryszkiewicz and Piotr Lasek

Institute of Computer Science, Warsaw University of Technology
Nowowiejska 15/19, 00-665 Warsaw, Poland
{mkr,p.lasek}@ii.pw.edu.pl

Abstract. Grouping data into meaningful clusters is an important data mining task. DBSCAN is recognized as a high quality density-based algorithm for clustering data. It enables both the determination of clusters of any shape and the identification of noise in data. The most time-consuming operation in DBSCAN is the calculation of a neighborhood for each data point. In order to speed up this operation in DBSCAN, the neighborhood calculation is expected to be supported by spatial access methods. DBSCAN, nevertheless, is not efficient in the case of high dimensional data. In this paper, we propose a new efficient TI-DBSCAN algorithm and its variant TI-DBSCAN-REF that apply the same clustering methodology as DBSCAN. Unlike DBSCAN, TI-DBSCAN and TI-DBSCAN-REF do not use spatial indices; instead they use the triangle inequality property to quickly reduce the neighborhood search space. The experimental results prove that the new algorithms are up to three orders of magnitude faster than DBSCAN, and efficiently cluster both low and high dimensional data.

1 Introduction

Grouping data into meaningful clusters is an important data mining task. The quality of clustering depends on a used algorithm. The DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise) [3] is recognized as a high quality scalable algorithm for clustering low dimensional data. The most time-consuming operation in DBSCAN is the calculation of a neighborhood for each data point. In order to speed up this operation in DBSCAN, it is expected to be supported by spatial access methods such as R*-tree [1] (R-tree [4]). DBSCAN, nevertheless, is not able to cluster high dimensional data efficiently. A method for improving the performance of DBSCAN based on early removal of core points has been offered in [6]. There, the carried out experiments showed that using the proposed method speeded up DBSCAN's performance by 50%.

In this paper, we propose a new efficient TI-DBSCAN algorithm and its variant TI-DBSCAN-REF that apply the same clustering methodology as DBSCAN. Unlike DBSCAN, TI-DBSCAN and TI-DBSCAN-REF do not use spatial indices; instead they use the triangle inequality property to quickly reduce the neighborhood search space. To the best of our knowledge, our proposal is the first one that relates to a

M. Szczuka et al. (Eds.): RSCTC 2010, LNAI 6086, pp. 60–69, 2010.

density-based clustering; the other trials of using the triangle inequality in clustering were related to *k*-means algorithm [2][7] and hierarchical algorithms following the results presented in [7].

The paper has the following layout. Section 2 recalls the notion of a cluster and noise according to [3]. In Section 3, we a offer theoretical basis of our approach to optimizing DBSCAN-like clustering. In Section 4, we propose the TI-DBSCAN algorithm and its modification TI-DBSCAN-REF. Section 5 reports the performance of TI-DBSCAN, TI-DBSCAN-REF as well as the performance of DBSCAN. Section 6 concludes the obtained results.

2 Basic Notions

In the context of the DBSCAN algorithm [3], a cluster is an area of high density. Data points in a low density area constitute noise. A point in space is considered a member of a cluster if there is a sufficient number of points within a given distance. In the sequel, the distance between two points p and q will be denoted by distance(p,q). Please, note that one may use a variety of distance metrics. 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, in our examples we will refer to Euclidean distance, although our approach is suitable for any distance metric. Below, we recall definitions of a density based cluster and related notions after [3].

Eps-neighborhood of a point p (denoted by $N_{Eps}(p)$) is defined as the set of points q in dataset D that are distant from p by no more than Eps; that is, $N_{Eps}(p) = \{q \in D \mid distance(p,q) \leq Eps\}$.

A point p is defined as a *core point* if its *Eps-neighborhood* contains at least MinPts points; that is, if $|N_{Eps}(p)| \geq MinPts$.

A point p is defined as *directly density-reachable* from a point q with respect to Eps and MinPts if $p \in N_{Eps}(q)$ and q is a core point.

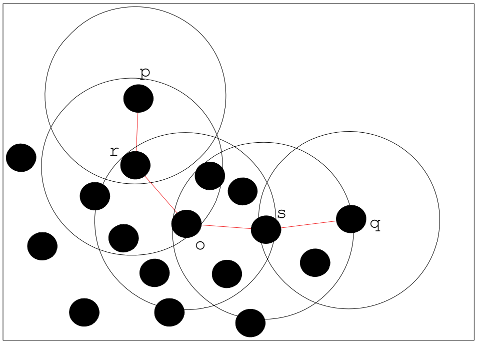
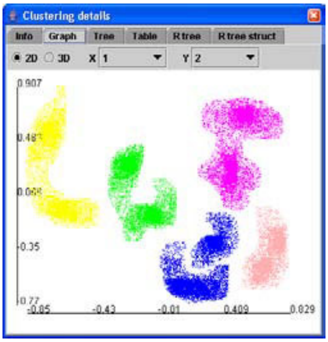


Fig. 1. Density-reachability of points (MinPts = 6)



● 2D ○ 3D X 1 Y 2

Fig. 2. Sample result of clustering with DBSCAN

A point p is defined as *density-reachable* from a point q with respect to Eps and MinPts if there is a sequence of points $p_1, \ldots, p_n$ such that $p_1 = q$, $p_n = p$ and p_{i+1} is directly density-reachable from p_i, i = 1..n-1.

Example 1. Let MinPts = 6. Point r in Figure 1 has 6 neighbors (including itself) in its neighborhood $N_{Eps}(r)$, so it is a core point. Point p has 2 neighbors in $N_{Eps}(p)$, so it is not a core point. Point p, however, belongs to $N_{Eps}(r)$, so it is directly density-reachable from r. To the contrary r is not directly density-reachable from p despite r belongs to $N_{Eps}(p)$. Point p in Figure 1 is density-reachable from point o, since there is a point (e.g. point r) such that p is directly density-reachable from it and it is directly density-reachable from o. Please note that p, which is density-reachable from core point o, is not a core point. □

A point p is defined as 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(o) determine all points in D that are density-reachable from point o. Clearly, if o is not a core point, then C(o) is empty. In Figure 1, points p and q are density-reachable from core point o. Hence, p and q belong to C(o).

A *cluster*[1] is defined as 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 [3]. Thus, a core point p belongs to exactly one cluster, namely to C(p). We note, however, that a border point may belong to more than one cluster.

Noise is defined as 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. Hence, each point that is neither a core point, nor border one, constitutes noise.

Fig. 2 presents the results of clustering with DBSCAN for a sample dataset.

3 Using the Triangle Inequality for Efficient Determination of Eps-Neighborhoods

Let us start with recalling the triangle inequality property:

Property 1. (Triangle inequality property). For any three points p, q, r:

$$\text{distance}(p,r) \leq \text{distance}(p,q) + \text{distance}(q,r)$$

Property 2 presents its equivalent form, which is more suitable for further considerations.

Property 2. (Triangle inequality property). For any three points p, q, r:

$$\text{distance}(p,q) \geq \text{distance}(p,r) - \text{distance}(q,r).$$

[1] This definition differs from the original one provided in [3]. However, it is equivalent to the original one by Lemma 1 in [3], and is more suitable for our presentation.

Lemma 1. Let D be a set of points. For any two points p, q in D and any point r:

$$\text{distance(p,r)} - \text{distance(q,r)} > \text{Eps} \Rightarrow q \notin N_{Eps}(p) \wedge p \notin N_{Eps}(q).$$

Proof. Let distance(p,r) – distance(q,r) > Eps (*). By Property 2, distance(p,q) ≥ distance(p,r) – distance(q,r) (**). By (*) and (**), distance(p,q) > Eps, and distance(q,p) = distance(p,q). Hence, $q \notin N_{Eps}(p)$ and $p \notin N_{Eps}(q)$. □

By Lemma 1, if we know that the difference of distances of two points p and q to some point r is greater than Eps, we are able to conclude that $q \notin N_{Eps}(p)$ without calculating the actual distance between p and q. Theorem 1 is our proposal of effective determination of points that do not belong to Eps-neighborhood of a given point p.

Theorem 1. Let r be any point and D be a set of points ordered in a non-decreasing way with respect to their distances to r. Let p be any point in D, q_f be a point following point p in D such that distance(q_f,r) – distance(p,r) > Eps, and q_b be a point preceding point p in D such that distance(p,r) – distance(q_b,r) > Eps. Then:

a) q_f and all points following q_f in D do not belong to $N_{Eps}(p)$.
b) q_b and all points preceding q_b in D do not belong to $N_{Eps}(p)$.

Proof. Let r be any point and D be a set of points ordered in a non-decreasing way with respect to their distances to r.
a) Let p be any point in D, q_f be a point following point p in D such that distance(q_f,r) – distance(p,r) > Eps (*), and s be either point q_f or any point following q_f in D. Then distance(s,r) ≥ distance(q_f,r) (**). By (*) and (**), distance(s,r) – distance(p,r) > Eps. Thus, by Lemma 1, $s \notin N_{Eps}(p)$.
b) The proof is analogous to the proof of Theorem 1a). □

Corollary 1. Let r be any point and D be a set of points ordered in a non-decreasing way with respect to their distances to r. Let p be any point in D, q_f be the first point following point p in D such that distance(q_f,r) – distance(p,r) > Eps, and q_b be the first point preceding point p in D such that distance(p,r) – distance(q_b,r) > Eps. Then, only the points that follow q_b in D and precede q_f in D have a chance to belong to $N_{Eps}(p)$, and p certainly belongs to $N_{Eps}(p)$.

Example 2. Let r be a point (0,0). Figure 3 shows sample set D of two dimensional points. Table 1 illustrates the same set D ordered in a non-decreasing way with respect to the distance of its points to point r. Let us consider the determination of the Eps-neighborhood of point p = F, where Eps = 0.5, by means of Corollary 1. We note that distance(F,r) = 3.2, the first point q_f following point F in D such that distance(q_f,r) – distance(F,r) > Eps is point C (distance(C,r) – distance(F,r) = 4.5 – 3.2 = 1.3 > Eps), and the first point q_b preceding point p in D such that distance(F,r) – distance(q_b,r) > Eps is G (distance(F,r) – distance(G,r) = 3.2 – 2.4 = 0.8 > Eps). By Corollary 1, only the points that follow G and precede C in D (here, points F and H) may belong to $N_{Eps}(F)$. Clearly, $F \in N_{Eps}(F)$. Hence, H is the only point for which it is necessary to calculate its actual distance to F in order to determine $N_{Eps}(F)$ properly. □

In the sequel, a point r to which the distances of all points in D have been determined will be called a *reference point*.

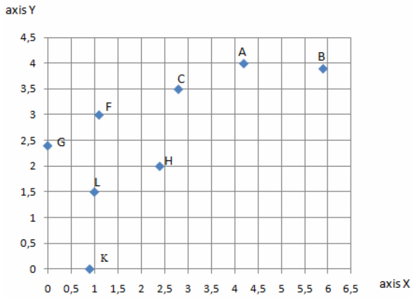


Fig. 3. Set of points D

Table. 1. Ordered set of points D from Fig. 3 with their distance to reference point r(0,0)

Q	X	Y	distance(q,r)
K	0,9	0,0	0,9
L	1,0	1,5	1,8
G	0,0	2,4	2,4
H	2,4	2,0	3,1
F	1,1	3,0	3,2
C	2,8	3,5	4,5
A	4,2	4,0	5,8
B	5,9	3,9	7,1

4 New Algorithms: TI-DBSCAN and TI-DBSCAN-REF

In this section, we propose a new clustering algorithm called TI-DBSCAN and its version TI-DBSCAN-REF. The result of clustering of core points and identifying of noise by our algorithms is the same as the one produced by the DBSCAN algorithm [3]. The border points may be assigned to different clusters[2]. In our algorithms we use Corollary 1 for efficient determination of the Eps-neighborhoods of points. In addition, we adopt the solution from [6] that consists in removing a point from the analyzed set D as soon as it is found to be a core point. Here, we remove each analyzed point, even if it not a core point. Let us start with the description of TI-DBSCAN.

Notation for TI-DBSCAN

- D – the set of points that is subject to clustering;
- Eps – the radius of the point neighborhood;
- MinPts – the required minimal number of points MinPts within Eps-neighborhood;
- r – a reference point assumed to be fixed, e.g. to the point with all coordinates equal to 0 or minimal values in the domains of all coordinates;
- fields of any point p in D:
  - p.ClusterId – label of a cluster to which p belongs; initially assigned the UNCLASSIFIED label;
  - p.dist – the distance of point p to reference point r;
  - p.NeighborsNo – the number of neighbors of p already found; initially assigned 1 to indicate that a point itself belongs to its own Eps-neighborhood;
  - Border – the information about neighbors of p that turned out non-core points for which it is not clear temporary if they are noise ones or border ones; initially assigned an empty set;
- D’ – the result of clustering of D (initially an empty set);

[2] Although a border point may belong to many clusters, DBSCAN assigns it arbitrarily only to one of them, so does TI-DBSCAN. It is easy to modify the algorithms so that border points are assigned to all including clusters.

TI-DBSCAN takes as an input a set of points D, a radius Eps and a threshold MinPts. Its output, i.e. clustered points, shall be stored in D'. A reference point r is assumed to be fixed, e.g. to the point with all coordinates equal to 0 or minimal values in the domains of all coordinates. With each point p in D, there are associated the following fields: ClusterId, dist, NeighborsNo, and Border.

TI-DBSCAN starts with the initialization of D' and the fields of all points in D. Then it sorts all points in D in a non-decreasing way w.r.t. their distance to reference point r. Next it generates a label for the first cluster to be found. Then it scans point after point in D. For each scanned point p, the TI-ExpandCluster function (described later) is called. If p is a core point, then the function assigns the current cluster's label to all points in C(p), moves them from D to D', and TI-DBSCAN generates a next label for a new cluster to be created. Otherwise, point p is assigned label NOISE and is moved from D to D'. After all points in D are scanned, each point is assigned either to a respective cluster identified by ClusterId or is identified as noise.

```
Algorithm TI-DBSCAN(set of points D, Eps, MinPts);
/* assert: r denotes a reference point */
D' = empty set of points;
for each point p in set D do
    p.ClusterId = UNCLASSIFIED;
    p.dist = Distance(p,r); p.NeighborsNo = 1; p.Border = ∅
endfor
sort all points in D non-decreasingly w.r.t. field dist;
ClusterId = label of first cluster;
for each point p in the ordered set D starting from
    the first point until last point in D do
     if TI-ExpandCluster(D, D', p, ClusterId, Eps, MinPts) then
         ClusterId = NextId(ClusterId)
     endif
endfor
return D'                            // D' is a clustered set of points
```

The TI-ExpandCluster function starts with calling TI-Neighborhood function (decribed later) to determine Eps-neighborhood of a given point p in, possibly reduced, set D (more precisely, TI-Neighborhood determines $N_{Eps}(p)\setminus\{p\}$ in D) and stores it in the seeds variable. Clearly, $N_{Eps}(p)$ determined in a reduced set D will not contain the neighboring points that were already moved from D to D'. In order to determine the real size of $N_{Eps}(p)$ in the original, non-reduced D, the auxiliary NeighborsNo field of point p is used. Whenever, a point p is moved from D to D', the NeighborsNo field of each of its neighboring points in D is incremented. As a result, the sum of the size of $N_{Eps}(p)$ found in the reduced D and p.NeighborsNo equals the size of $N_{Eps}(p)$ in the original, non-reduced set D.

If p is found not to be a core point, it is temporary labeled as a noise point, NeighborsNo field of each of its neighboring points in D is incremented, the information about p is added to the Border field of each of its neighboring points in D, p itself is moved from D to D', 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 Eps-neighborhood of the analyzed point are assigned a label (ClId) of the currently built cluster and their NeighborsNo fields are incremented. Next all non-core points indicated by the p.Border, which were stored in D’, are found to be border points, and are assigned cluster label ClId, p.Border is cleared, and p is moved from D to D’. Now, each core point in seeds further extends the seeds collection with the points in its Eps-neighborhood that are still unclassified. After processing a seed point, it is deleted from seeds. The function ends when all points found as cluster seeds are processed.

Note that TI-ExpandCluster calculates Eps-neighborhood for each point only once.

```
function TI-ExpandCluster(var D, var D′, point p, ClId, Eps, MinPts)
/* assert: D is ordered in a non-decreasing way w.r.t.  */
/* distances of points in D from the reference point r. */
/* assert: TI-Neighborhood does not return p.           */
seeds = TI-Neighborhood(D, p, Eps);
p.NeighborsNo = p.NeighborsNo + |seeds|;           // including p itself
if p.NeighborsNo < MinPts then    // p is either noise or a border point
    p.ClusterId = NOISE;
    for each point q in seeds do
        append p to q.Border; q.NeighborsNo = q.NeighborsNo + 1
    endfor
    p.Border = ∅; move p from D to D′;     // D′ stores analyzed points
    return FALSE
else
    p.ClusterId = ClId;
    for each point q in seeds do
        q.ClusterId = ClId; q.NeighborsNo = q.NeighborsNo + 1
    endfor
    for each point q in p.Border do
        D′.q.ClusterId = ClId;    //assign cluster id to q in D′
    endfor
    p.Border = ∅; move p from D to D′;     // D′ stores analyzed points
    while |seeds| > 0 do
        curPoint = first point in seeds;
        curSeeds = TI-Neighborhood(D, curPoint, Eps);
        curPoint.NeighborsNo = curPoint.NeighborsNo + |curSeeds|;
        if curPoint.NeighborsNo < MinPts then //curPoint is border point
            for each point q in curSeeds do
                q.NeighborsNo = q.NeighborsNo + 1
            endfor
        else                              // curPoint is a core point
            for each point q in curSeeds do
                q.NeighborsNo = q.NeighborsNo + 1
                if q.ClusterId = UNCLASSIFIED then
                    q.ClusterId = ClId;
                    move q from curSeeds to seeds
                else
                    delete q from curSeeds
                endif
            endfor
            for each point q in curPoint.Border do
                 D′.q.ClusterId = ClId;   //assign cluster id to q in D′
            endfor
        endif
        curPoint.Border = ∅; move curPoint from D to D′;
        delete curPoint from seeds
    endwhile
    return TRUE
endif
```

The TI-Neighborhood function takes the ordered point set D, point p in D, and Eps as input parameters. It returns $N_{Eps}(p)\setminus\{p\}$ as the set theoretical union of the point sets found by the TI-Backward-Neighborhood function and the TI-Forward-Neighborhood function. TI-Backward-Neighborhood examines points preceding currently analyzed point p, for which Eps-neighborhood is to be determined. The function applies Lemma 1 to identify first point, say q_b, preceding p in D such that distance(p,r) - distance(q_b,r) > Eps. All points preceding point q_b in D are not checked at all, since they are guaranteed not to belong to $N_{Eps}(p)$ (by Theorem 1). The points that precede p and, at the same time, follow q_b in D have a chance to belong to $N_{Eps}(p)$. For these points, it is necessary to calculate their actual distance to p (When using the Euclidean distance metric, the functions may apply the square of Distance and the square of Eps for efficiency reasons). The TI-Backward-Neighborhood function returns all points preceding p in D with the distance to p not exceeding Eps. The TI-Forward-Neighborhood function is analogous to TI-Backward-Neighborhood. Unlike TI-Backward-Neighborhood, TI-Forward-Neighborhood examines points following currently analyzed point p, for which Eps-neighborhood is to be determined. The TI-Forward-Neighborhood function returns all points following p in D with the distance to p not exceeding Eps.

```
function TI-Neighborhood(D, point p, Eps)
return TI-Backward-Neighborhood(D, p, Eps) ∪
       TI-Forward-Neighborhood(D, p, Eps)
```

```
function TI-Backward-Neighborhood(D, point p, Eps)
/* assert: D is ordered non-decreasingly w.r.t. dist */
seeds = {};
backwardThreshold = p.dist - Eps;
for each point q in the ordered set D starting from
    the point immediately preceding point p until
    the first point in D do
    if q.dist < backwardThreshold then        // p.dist – q.dist > Eps?
        break;
    endif
    if Distance(q, p) ≤ Eps then append q to seeds endif
endfor
return seeds
```

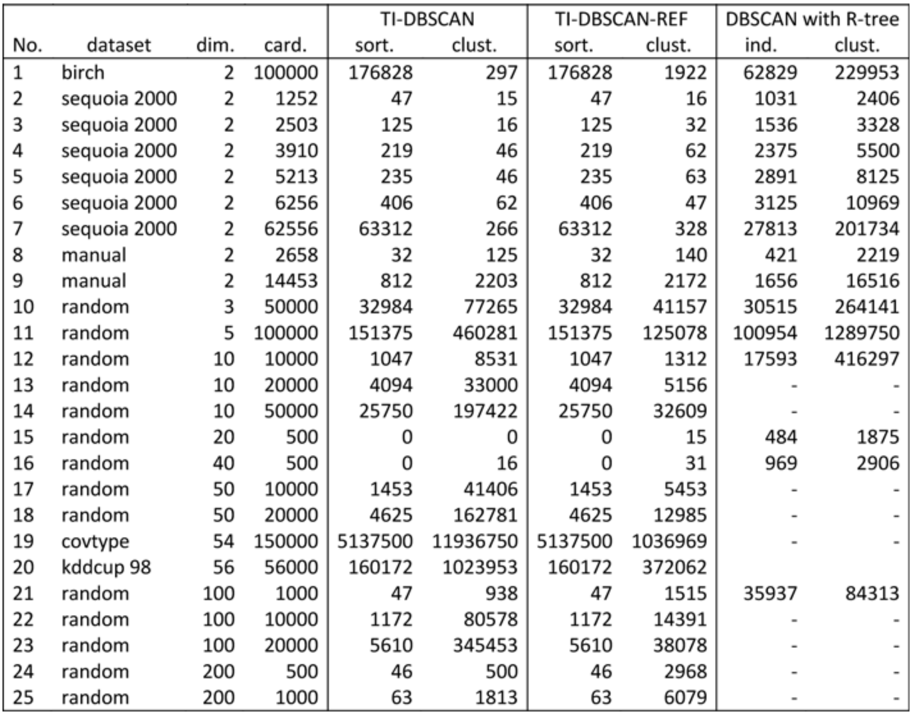
```
function TI-Forward-Neighborhood(D, point p, Eps)
/* assert: D is ordered non-decreasingly w.r.t. dist */
seeds = {};
forwardThreshold = Eps + p.dist;
for each point q in the ordered set D starting from
    the point immediately following point p until
    the last point in D do
    if q.dist > forwardThreshold then         // q.dist – p.dist > Eps?
        break;
    endif
    if Distance(q, p) ≤ Eps then append q to seeds endif
endfor
return seeds
```

Except for TI-DBSCAN, we have also implemented its variant TI-DBSCAN-REF [5] that uses many reference points instead of one for estimating the distance among pairs of points. Additional reference points are used only when the basic reference point according to which the points in D are sorted is not sufficient to estimate if a given point q belongs to Eps-neighborhood of another point p. The estimation of the distance between q and p by means of an additional reference point is based on Lemma 1. The actual distance between the two points q and p is calculated only when all reference points are not sufficient to estimate if q ∈ $N_{Eps}(p)$.

5 Performance Evaluation

In this section, we report the results of our experimental evaluation of TI-DBSCAN and TI-DBSCAN-REF as well as the original DBSCAN with R-Tree as an index. The number of reference points used by TI-DBSCAN-REF was set to the number of dimensions of a clustered dataset. In the experiments, we used a number of datasets (and/or their subsamples) of different cardinality and dimensionality. In particular, we used widely known datasets such as: birch [9], SEQUOIA 2000 [8], covtype [7] and kddcup 98 [10] as well as datasets generated automatically (random) or manually.

Table 2. Datasets used in experiments and run times (in milliseconds) of examined algorithms. Notation: dim. – number of point's dimensions, card. – number of points in a dataset, sort. – time of sorting of points, ind. – time of building of an index, clust. – clustering.



				TI-DBSCAN		TI-DBSCAN-REF		DBSCAN with R-tree	
No.	dataset	dim.	card.	sort.	clust.	sort.	clust.	ind.	clust.
1	birch	2	100000	176828	297	176828	1922	62829	229953
2	sequoia 2000	2	1252	47	15	47	16	1031	2406
3	sequoia 2000	2	2503	125	16	125	32	1536	3328
4	sequoia 2000	2	3910	219	46	219	62	2375	5500
5	sequoia 2000	2	5213	235	46	235	63	2891	8125
6	sequoia 2000	2	6256	406	62	406	47	3125	10969
7	sequoia 2000	2	62556	63312	266	63312	328	27813	201734
8	manual	2	2658	32	125	32	140	421	2219
9	manual	2	14453	812	2203	812	2172	1656	16516
10	random	3	50000	32984	77265	32984	41157	30515	264141
11	random	5	100000	151375	460281	151375	125078	100954	1289750
12	random	10	10000	1047	8531	1047	1312	17593	416297
13	random	10	20000	4094	33000	4094	5156	-	-
14	random	10	50000	25750	197422	25750	32609	-	-
15	random	20	500	0	0	0	15	484	1875
16	random	40	500	0	16	0	31	969	2906
17	random	50	10000	1453	41406	1453	5453	-	-
18	random	50	20000	4625	162781	4625	12985	-	-
19	covtype	54	150000	5137500	11936750	5137500	1036969	-	-
20	kddcup 98	56	56000	160172	1023953	160172	372062		
21	random	100	1000	47	938	47	1515	35937	84313
22	random	100	10000	1172	80578	1172	14391	-	-
23	random	100	20000	5610	345453	5610	38078	-	-
24	random	200	500	46	500	46	2968	-	-
25	random	200	1000	63	1813	63	6079	-	-

The run times of clustering with TI-DBSCAN, TI-DBSCAN-REF, and DBSCAN using R-Tree as an index are presented in Table 2. As follows from Table 2, TI-DBSCAN and TI-DBSCAN-REF are more efficient than DBSCAN even up to 600 times. TI-DBSCAN-REF tends to be faster than TI-DBSCAN for large high dimensional datasets. For small low dimensional datasets, TI-DBSCAN tends to be faster than TI-DBSCAN-REF.

6 Conclusions

In the paper, we have proposed two versions of our new algorithm: TI-DBSCAN and TI-DBSCAN-REF that produce the same results as DBSCAN, but use the triangle inequality to speed up the clustering process. TI-DBSCAN uses only one reference point, while TI-DBSCAN-REF uses many reference points. As follows from the experiments, both versions of our algorithm are much more efficient than the original DBSCAN algorithm, which is supported by a spatial index. Unlike DBSCAN, TI-DBSCAN and TI-DBSCAN-REF enable efficient clustering of high-dimensional data. The usage of many reference points is particularly useful in the case of large high dimensional datasets.

References

[1] Beckmann, N., Kriegel, H.P.: The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: Proc. of ACM SIGMOD, Atlantic City, pp. 322–331 (1990)

[2] Elkan, C.: Using the Triangle Inequality to Accelerate k-Means. In: Proc. of ICML 2003, Washington, pp. 147–153 (2003)

[3] Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A Density-Based Algorithm for Discovering Clusters in Large Spatial Database with Noise. In: Proc. of KDD 1996, Portland, pp. 226–231 (1996)

[4] Guttman, A.: R-Trees: A Dynamic Index Structure For Spatial Searching. In: Proc. of ACM SIGMOD, Boston, pp. 47–57 (1984)

[5] Kryszkiewicz, M., Lasek, P.: TI-DBSCAN: Clustering with DBSCAN by Means of the Triangle Inequality, ICS Research Report, Warsaw University of Technology (April 2010)

[6] Kryszkiewicz, M., Skonieczny, Ł.: Faster Clustering with DBSCAN. In: Proc. of IIPWM 2005, Gdańsk, pp. 605–614 (2005)

[7] Moore, A.W.: The Anchors Hierarchy: Using the Triangle Inequality to Survive High Dimensional Data. In: Proc. of UAI, Stanford, pp. 397–405 (2000)

[8] Stonebraker, M., Frew, J., Gardels, K., Meredith, J.: The SEQUOIA 2000 Storage Benchmark. In: Proc. of ACM SIGMOD, Washington, pp. 2–11 (1993)

[9] Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: A New Data Clustering Algorithm and its Applications. Data Mining and Knowledge Discovery 1(2), 141–182 (1997)

[10] http://kdd.ics.uci.edu/databases/kddcup98/kddcup98.html