Published in Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)

A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise

Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 München, Germany
{ester | kriegel | sander | xwxu}@informatik.uni-muenchen.de

Abstract

Clustering algorithms are attractive for the task of class identification in spatial databases. However, the application to large spatial databases rises the following requirements for clustering algorithms: minimal requirements of domain knowledge to determine the input parameters, discovery of clusters with arbitrary shape and good efficiency on large databases. The well-known clustering algorithms offer no solution to the combination of these requirements. In this paper, we present the new clustering algorithm DBSCAN relying on a density-based notion of clusters which is designed to discover clusters of arbitrary shape. DBSCAN requires only one input parameter and supports the user in determining an appropriate value for it. We performed an experimental evaluation of the effectiveness and efficiency of DBSCAN using synthetic data and real data of the SEQUOIA 2000 benchmark. The results of our experiments demonstrate that (1) DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS, and that (2) DBSCAN outperforms CLARANS by a factor of more than 100 in terms of efficiency.

Keywords: Clustering Algorithms, Arbitrary Shape of Clusters, Efficiency on Large Spatial Databases, Handling Noise.

1. Introduction

Numerous applications require the management of *spatial* data, i.e. data related to space. *Spatial Database Systems (SDBS)* (Gueting 1994) are database systems for the management of spatial data. Increasingly large amounts of data are obtained from satellite images, X-ray crystallography or other automatic equipment. Therefore, automated knowledge discovery becomes more and more important in spatial databases.

Several tasks of *knowledge discovery in databases* (KDD) have been defined in the literature (Matheus, Chan & Piatetsky-Shapiro 1993). The task considered in this paper is *class identification*, i.e. the grouping of the objects of a database into meaningful subclasses. In an earth observation database, e.g., we might want to discover classes of houses along some river.

Clustering algorithms are attractive for the task of class identification. However, the application to large spatial databases rises the following requirements for clustering algorithms:

(1) Minimal requirements of domain knowledge to determine the input parameters, because appropriate values are often not known in advance when dealing with large databases.

(2) Discovery of clusters with arbitrary shape, because the shape of clusters in spatial databases may be spherical, drawn-out, linear, elongated etc.

(3) Good efficiency on large databases, i.e. on databases of significantly more than just a few thousand objects.

The well-known clustering algorithms offer no solution to the combination of these requirements. In this paper, we present the new clustering algorithm DBSCAN. It requires only one input parameter and supports the user in determining an appropriate value for it. It discovers clusters of arbitrary shape. Finally, DBSCAN is efficient even for large spatial databases. The rest of the paper is organized as follows. We discuss clustering algorithms in section 2 evaluating them according to the above requirements. In section 3, we present our notion of clusters which is based on the concept of density in the database. Section 4 introduces the algorithm DBSCAN which discovers such clusters in a spatial database. In section 5, we performed an experimental evaluation of the effectiveness and efficiency of DBSCAN using synthetic data and data of the SEQUOIA 2000 benchmark. Section 6 concludes with a summary and some directions for future research.

2. Clustering Algorithms

There are two basic types of clustering algorithms (Kaufman & Rousseeuw 1990): partitioning and hierarchical algorithms. *Partitioning algorithms* construct a partition of a database *D* of *n* objects into a set of *k* clusters. *k* is an input parameter for these algorithms, i.e some domain knowledge is required which unfortunately is not available for many applications. The partitioning algorithm typically starts with an initial partition of *D* and then uses an iterative control strategy to optimize an objective function. Each cluster is represented by the gravity center of the cluster *(k-means algorithms*) or by one of the objects of the cluster located near its center (*k-medoid algorithms*). Consequently, partitioning algorithms use a two-step procedure. First, determine *k* representatives minimizing the objective function. Second, assign each object to the cluster with its representative "closest" to the considered object. The second step implies that a partition is equivalent to a voronoi diagram and each cluster is contained in one of the voronoi cells. Thus, the shape of all

clusters found by a partitioning algorithm is convex which is very restrictive.

Ng & Han (1994) explore partitioning algorithms for KDD in spatial databases. An algorithm called CLARANS (Clustering Large Applications based on RANdomized Search) is introduced which is an improved k-medoid method. Compared to former k-medoid algorithms, CLARANS is more effective and more efficient. An experimental evaluation indicates that CLARANS runs efficiently on databases of thousands of objects. Ng & Han (1994) also discuss methods to determine the "natural" number k_{nat} of clusters in a database. They propose to run CLARANS once for each k from 2 to n. For each of the discovered clusterings the silhouette coefficient (Kaufman & Rousseeuw 1990) is calculated, and finally, the clustering with the maximum silhouette coefficient is chosen as the "natural" clustering. Unfortunately, the run time of this approach is prohibitive for large n, because it implies O(n) calls of CLARANS.

CLARANS assumes that all objects to be clustered can reside in main memory at the same time which does not hold for large databases. Furthermore, the run time of CLARANS is prohibitive on large databases. Therefore, Ester, Kriegel &Xu (1995) present several focusing techniques which address both of these problems by focusing the clustering process on the relevant parts of the database. First, the focus is small enough to be memory resident and second, the run time of CLARANS on the objects of the focus is significantly less than its run time on the whole database.

Hierarchical algorithms create a hierarchical decomposition of *D*. The hierarchical decomposition is represented by a *dendrogram*, a tree that iteratively splits *D* into smaller subsets until each subset consists of only one object. In such a hierarchy, each node of the tree represents a cluster of *D*. The dendrogram can either be created from the leaves up to the root (*agglomerative approach*) or from the root down to the leaves (*divisive approach*) by merging or dividing clusters at each step. In contrast to partitioning algorithms, hierarchical algorithms do not need *k* as an input. However, a *termination condition* has to be defined indicating when the merge or division process should be terminated. One example of a termination condition in the agglomerative approach is the critical distance D_{min} between all the clusters of *Q*.

So far, the main problem with hierarchical clustering algorithms has been the difficulty of deriving appropriate parameters for the termination condition, e.g. a value of D_{min} which is small enough to separate all "natural" clusters and, at the same time large enough such that no cluster is split into two parts. Recently, in the area of signal processing the hierarchical algorithm Ejcluster has been presented (García, Fdez-Valdivia, Cortijo & Molina 1994) automatically deriving a termination condition. Its key idea is that two points belong to the same cluster if you can walk from the first point to the second one by a "sufficiently small" step. Ejcluster follows the divisive approach. It does not require any input of domain knowledge. Furthermore, experiments show that it is very effective in discovering non-convex clusters. However, the computational cost of Ejcluster is $O(n^2)$ due to the distance calculation for each pair of points. This is acceptable for applications such as character recognition with moderate values for n, but it is prohibitive for applications on large databases.

Jain (1988) explores a density based approach to identify clusters in k-dimensional point sets. The data set is partitioned into a number of nonoverlapping cells and histograms are constructed. Cells with relatively high frequency counts of points are the potential cluster centers and the boundaries between clusters fall in the "valleys" of the histogram. This method has the capability of identifying clusters of any shape. However, the space and run-time requirements for storing and searching multidimensional histograms can be enormous. Even if the space and run-time requirements are optimized, the performance of such an approach crucially depends on the size of the cells.

3. A Density Based Notion of Clusters

When looking at the sample sets of points depicted in figure 1, we can easily and unambiguously detect clusters of points and noise points not belonging to any of those clusters.

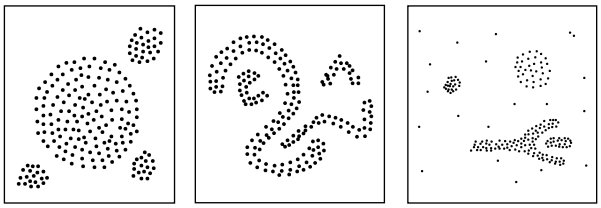


figure 1: Sample databases

The main reason why we recognize the clusters is that within each cluster we have a typical density of points which is considerably higher than outside of the cluster. Furthermore, the density within the areas of noise is lower than the density in any of the clusters.

In the following, we try to formalize this intuitive notion of "clusters" and "noise" in a database *D* of points of some k-dimensional space *S*. Note that both, our notion of clusters and our algorithm DBSCAN, apply as well to 2D or 3D Euclidean space as to some high dimensional feature space. The key idea is that for each point of a cluster the neighborhood of a given radius has to contain at least a minimum number of points, i.e. the density in the neighborhood has to exceed some threshold. The shape of a neighborhood is determined by the choice of a distance function for two points p and q, denoted by *dist(p,q)*. For instance, when using the Manhattan distance in 2D space, the shape of the neighborhood is rectangular. Note, that our approach works with any distance function so that an appropriate function can be chosen for some given application. For the purpose of proper visualization, all examples will be in 2D space using the Euclidean distance.

Definition 1: (Eps-neighborhood of a point) The *Eps-neighborhood* of a point p, denoted by $N_{Eps}(p)$, is defined by $N_{Eps}(p) = \{q \in D \mid dist(p,q) \leq Eps\}$.

A naive approach could require for each point in a cluster that there are at least a minimum number (*MinPts*) of points in an Eps-neighborhood of that point. However, this ap-

proach fails because there are two kinds of points in a cluster, points inside of the cluster (*core points*) and points on the border of the cluster (*border points*). In general, an Eps-neighborhood of a border point contains significantly less points than an Eps-neighborhood of a core point. Therefore, we would have to set the minimum number of points to a relatively low value in order to include all points belonging to the same cluster. This value, however, will not be characteristic for the respective cluster - particularly in the presence of noise. Therefore, we require that for every point p in a cluster C there is a point q in C so that p is inside of the Eps-neighborhood of q and $N_{Eps}(q)$ contains at least MinPts points. This definition is elaborated in the following.

Definition 2: (directly density-reachable) A point p is *directly density-reachable* from a point q wrt. Eps, MinPts if

1) $p \in N_{Eps}(q)$ and

2) $|N_{Eps}(q)| \geq$ MinPts (core point condition).

Obviously, directly density-reachable is symmetric for pairs of core points. In general, however, it is not symmetric if one core point and one border point are involved. Figure 2 shows the asymmetric case.

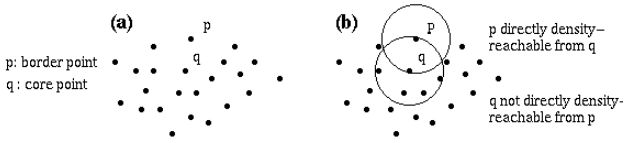


figure 2: core points and border points

Definition 3: (density-reachable) A point p is *density-reachable* 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.

Density-reachability is a canonical extension of direct density-reachability. This relation is transitive, but it is not symmetric. Figure 3 depicts the relations of some sample points and, in particular, the asymmetric case. Although not symmetric in general, it is obvious that density-reachability is symmetric for core points.

Two border points of the same cluster C are possibly not density reachable from each other because the core point condition might not hold for both of them. However, there must be a core point in C from which both border points of C are density-reachable. Therefore, we introduce the notion of density-connectivity which covers this relation of border points.

Definition 4: (density-connected) A point p is *density-connected* 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.

Density-connectivity is a symmetric relation. For density reachable points, the relation of density-connectivity is also reflexive (c.f. figure 3).

Now, we are able to define our density-based notion of a cluster. Intuitively, a cluster is defined to be a set of density-connected points which is maximal wrt. density-reachability. Noise will be defined relative to a given set of clusters. Noise is simply the set of points in *D* not belonging to any of its clusters.

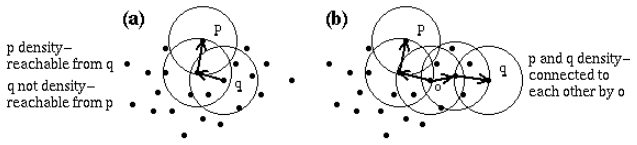


figure 3: density-reachability and density-connectivity

Definition 5: (cluster) Let *D* be a database of points. 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)

Definition 6: (noise) Let C_1 ,. . ., C_k be the clusters of the database *D* wrt. parameters Eps_i and $MinPts_i$, i = 1, . . ., k. Then we define the *noise* as the set of points in the database *D* not belonging to any cluster C_i , i.e. noise = {p ∈ *D* | ∀ i: p ∉ C_i}.

Note that a cluster C wrt. Eps and MinPts contains at least MinPts points because of the following reasons. Since C contains at least one point p, p must be density-connected to itself via some point o (which may be equal to p). Thus, at least o has to satisfy the core point condition and, consequently, the Eps-Neighborhood of o contains at least MinPts points.

The following lemmata are important for validating the correctness of our clustering algorithm. Intuitively, they state the following. Given the parameters Eps and MinPts, we can discover a cluster in a two-step approach. First, choose an arbitrary point from the database satisfying the core point condition as a seed. Second, retrieve all points that are density-reachable from the seed obtaining the cluster containing the seed.

Lemma 1: Let p be a point in *D* and $|N_{Eps}(p)| \geq$ MinPts. Then the set *O* = {o | o ∈ D and o is density-reachable from p wrt. Eps and MinPts} is a cluster wrt. Eps and MinPts.

It is not obvious that a cluster *C* wrt. Eps and MinPts is uniquely determined by *any* of its core points. However, each point in *C* is density-reachable from any of the core points of *C* and, therefore, a cluster *C* contains exactly the points which are density-reachable from an arbitrary core point of *C*.

Lemma 2: Let *C* be a cluster wrt. Eps and MinPts and let p be any point in *C* with $|N_{Eps}(p)| \geq$ MinPts. Then *C* equals to the set *O* = {o | o is density-reachable from p wrt. Eps and MinPts}.

4. DBSCAN: Density Based Spatial Clustering of Applications with Noise

In this section, we present the algorithm DBSCAN (Density Based Spatial Clustering of Applications with Noise) which is designed to discover the clusters and the noise in a spatial database according to definitions 5 and 6. Ideally, we would have to know the appropriate parameters Eps and MinPts of each cluster and at least one point from the respective cluster. Then, we could retrieve all points that are density-reachable from the given point using the correct parameters. But

there is no easy way to get this information in advance for all clusters of the database. However, there is a simple and effective heuristic (presented in section section 4.2) to determine the parameters Eps and MinPts of the "thinnest", i.e. least dense, cluster in the database. Therefore, DBSCAN uses global values for Eps and MinPts, i.e. the same values for all clusters. The density parameters of the "thinnest" cluster are good candidates for these global parameter values specifying the lowest density which is not considered to be noise.

4.1 The Algorithm

To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p wrt. Eps and MinPts. If p is a core point, this procedure yields a cluster wrt. Eps and MinPts (see Lemma 2). If p is a border point, no points are density-reachable from p and DBSCAN visits the next point of the database.

Since we use global values for Eps and MinPts, DBSCAN may merge two clusters according to definition 5 into one cluster, if two clusters of different density are "close" to each other. Let the *distance between two sets of points* S_1 and S_2 be defined as $dist(S_1, S_2) = min\{dist(p,q) \mid p \in S_1, q \in S_2\}$. Then, two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than Eps. Consequently, a recursive call of DBSCAN may be necessary for the detected clusters with a higher value for MinPts. This is, however, no disadvantage because the recursive application of DBSCAN yields an elegant and very efficient basic algorithm. Furthermore, the recursive clustering of the points of a cluster is only necessary under conditions that can be easily detected.

In the following, we present a basic version of DBSCAN omitting details of data types and generation of additional information about clusters:

```
DBSCAN (SetOfPoints, Eps, MinPts)

// SetOfPoints is UNCLASSIFIED
  ClusterId := nextId(NOISE);
  FOR i FROM 1 TO SetOfPoints.size DO
    Point := SetOfPoints.get(i);
    IF Point.ClId = UNCLASSIFIED THEN
      IF ExpandCluster(SetOfPoints, Point,
            ClusterId, Eps, MinPts) THEN
        ClusterId := nextId(ClusterId)
      END IF
    END IF
  END FOR
END; // DBSCAN
```

`SetOfPoints` is either the whole database or a discovered cluster from a previous run. `Eps` and `MinPts` are the global density parameters determined either manually or according to the heuristics presented in section 4.2. The function `SetOfPoints.get(i)` returns the i-th element of `SetOfPoints`. The most important function used by `DBSCAN` is `ExpandCluster` which is presented below:

```
ExpandCluster(SetOfPoints, Point, ClId, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeClId(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeClIds(seeds,ClId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                        Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.ClId
              IN {UNCLASSIFIED, NOISE} THEN
            IF resultP.ClId = UNCLASSIFIED THEN
              seeds.append(resultP);
            END IF;
            SetOfPoints.changeClId(resultP,ClId);
          END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

A call of `SetOfPoints.regionQuery(Point,Eps)` returns the `Eps`-Neighborhood of `Point` in `SetOfPoints` as a list of points. Region queries can be supported efficiently by spatial access methods such as R*-trees (Beckmann et al. 1990) which are assumed to be available in a SDBS for efficient processing of several types of spatial queries (Brinkhoff et al. 1994). The height of an R*-tree is O(log n) for a database of n points in the worst case and a query with a "small" query region has to traverse only a limited number of paths in the R^*-tree. Since the Eps-Neighborhoods are expected to be small compared to the size of the whole data space, the average run time complexity of a single region query is O(log n). For each of the n points of the database, we have at most one region query. Thus, the average run time complexity of DBSCAN is O(n * log n).

The `ClId` (clusterId) of points which have been marked to be `NOISE` may be changed later, if they are density-reachable from some other point of the database. This happens for border points of a cluster. Those points are not added to the `seeds`-list because we already know that a point with a ClId of `NOISE` is not a core point. Adding those points to `seeds` would only result in additional region queries which would yield no new answers.

If two clusters C_1 and C_2 are very close to each other, it might happen that some point p belongs to both, C_1 and C_2. Then p must be a border point in both clusters because otherwise C_1 would be equal to C_2 since we use global parame-

ters. In this case, point p will be assigned to the cluster discovered first. Except from these rare situations, the result of DBSCAN is independent of the order in which the points of the database are visited due to Lemma 2.

4.2 Determining the Parameters Eps and MinPts

In this section, we develop a simple but effective heuristic to determine the parameters Eps and MinPts of the "thinnest" cluster in the database. This heuristic is based on the following observation. Let d be the distance of a point p to its k-th nearest neighbor, then the d-neighborhood of p contains exactly k+1 points for almost all points p. The d-neighborhood of p contains more than k+1 points only if several points have exactly the same distance d from p which is quite unlikely. Furthermore, changing k for a point in a cluster does not result in large changes of d. This only happens if the k-th nearest neighbors of p for k= 1, 2, 3, . . . are located approximately on a straight line which is in general not true for a point in a cluster.

For a given k we define a function *k-dist* from the database *D* to the real numbers, mapping each point to the distance from its k-th nearest neighbor. When sorting the points of the database in descending order of their k-dist values, the graph of this function gives some hints concerning the density distribution in the database. We call this graph the *sorted k-dist graph*. If we choose an arbitrary point p, set the parameter Eps to k-dist(p) and set the parameter MinPts to k, all points with an equal or smaller k-dist value will be core points. If we could find a *threshold point* with the maximal k-dist value in the "thinnest" cluster of *D* we would have the desired parameter values. The threshold point is the first point in the first "valley" of the sorted k-dist graph (see figure 4). All points with a higher k-dist value (left of the threshold) are considered to be noise, all other points (right of the threshold) are assigned to some cluster.

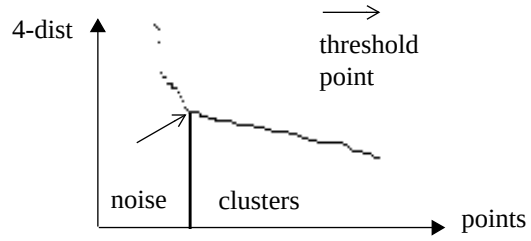


figure 4: sorted 4-dist graph for sample database 3

In general, it is very difficult to detect the first "valley" automatically, but it is relatively simple for a user to see this valley in a graphical representation. Therefore, we propose to follow an interactive approach for determining the threshold point.

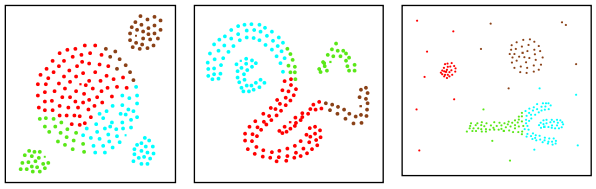
DBSCAN needs two parameters, Eps and MinPts. However, our experiments indicate that the k-dist graphs for k > 4 do not significantly differ from the 4-dist graph and, furthermore, they need considerably more computation. Therefore, we eliminate the parameter MinPts by setting it to 4 for all databases (for 2-dimensional data). We propose the following interactive approach for determining the parameter Eps of DBSCAN:

- The system computes and displays the 4-dist graph for the database.
- If the user can estimate the percentage of noise, this percentage is entered and the system derives a proposal for the threshold point from it.
- The user either accepts the proposed threshold or selects another point as the threshold point. The 4-dist value of the threshold point is used as the Eps value for DBSCAN.

5. Performance Evaluation

In this section, we evaluate the performance of DBSCAN. We compare it with the performance of CLARANS because this is the first and only clustering algorithm designed for the purpose of KDD. In our future research, we will perform a comparison with classical density based clustering algorithms. We have implemented DBSCAN in C++ based on an implementation of the R*-tree (Beckmann et al. 1990). All experiments have been run on HP 735 / 100 workstations. We have used both synthetic sample databases and the database of the SEQUOIA 2000 benchmark.

To compare DBSCAN with CLARANS in terms of effectivity (accuracy), we use the three synthetic sample databases which are depicted in figure 1. Since DBSCAN and CLARANS are clustering algorithms of different types, they have no common quantitative measure of the classification accuracy. Therefore, we evaluate the accuracy of both algorithms by visual inspection. In sample database 1, there are four ball-shaped clusters of significantly differing sizes. Sample database 2 contains four clusters of nonconvex shape. In sample database 3, there are four clusters of different shape and size with additional noise. To show the results of both clustering algorithms, we visualize each cluster by a different color (see www availability after section 6). To give CLARANS some advantage, we set the parameter *k* to 4 for these sample databases. The clusterings discovered by CLARANS are depicted in figure 5.

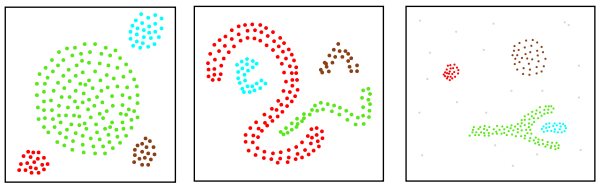


database 1 **database 2** **database 3**

figure 5: Clusterings discovered by CLARANS

For DBSCAN, we set the noise percentage to 0% for sample databases 1 and 2, and to 10% for sample database 3, respectively. The clusterings discovered by DBSCAN are depicted in figure 6.

DBSCAN discovers all clusters (according to definition 5) and detects the noise points (according to definition 6) from all sample databases. CLARANS, however, splits clusters if they are relatively large or if they are close to some other cluster. Furthermore, CLARANS has no explicit notion of noise. Instead, all points are assigned to their closest medoid.



database 1 **database 2** **database 3**

figure 6: Clusterings discovered by DBSCAN

To test the efficiency of DBSCAN and CLARANS, we use the SEQUOIA 2000 benchmark data. The SEQUOIA 2000 benchmark database (Stonebraker et al. 1993) uses real data sets that are representative of Earth Science tasks. There are four types of data in the database: raster data, point data, polygon data and directed graph data. The point data set contains 62,584 Californian names of landmarks, extracted from the US Geological Survey's Geographic Names Information System, together with their location. The point data set occupies about 2.1 M bytes. Since the run time of CLARANS on the whole data set is very high, we have extracted a series of subsets of the SEQUIOA 2000 point data set containing from 2% to 20% representatives of the whole set. The run time comparison of DBSCAN and CLARANS on these databases is shown in table 1.

Table 1: run time in seconds

number of points	1252	2503	3910	5213	6256
DBSCAN	3.1	6.7	11.3	16.0	17.8
CLARANS	758	3026	6845	11745	18029
number of points	7820	8937	10426	12512	
DBSCAN	24.5	28.2	32.7	41.7	
CLARANS	29826	39265	60540	80638	

The results of our experiments show that the run time of DBSCAN is slightly higher than linear in the number of points. The run time of CLARANS, however, is close to quadratic in the number of points. The results show that DBSCAN outperforms CLARANS by a factor of between 250 and 1900 which grows with increasing size of the database.

6. Conclusions

Clustering algorithms are attractive for the task of class identification in spatial databases. However, the well-known algorithms suffer from severe drawbacks when applied to large spatial databases. In this paper, we presented the clustering algorithm DBSCAN which relies on a density-based notion of clusters. It requires only one input parameter and supports the user in determining an appropriate value for it. We performed a performance evaluation on synthetic data and on real data of the SEQUOIA 2000 benchmark. The results of these experiments demonstrate that DBSCAN is significantly more effective in discovering clusters of arbitrary shape than the well-known algorithm CLARANS. Furthermore, the experiments have shown that DBSCAN outperforms CLARANS by a factor of at least 100 in terms of efficiency.

Future research will have to consider the following issues. First, we have only considered point objects. Spatial databases, however, may also contain extended objects such as polygons. We have to develop a definition of the density in an Eps-neighborhood in polygon databases for generalizing DBSCAN. Second, applications of DBSCAN to high dimensional feature spaces should be investigated. In particular, the shape of the k-dist graph in such applications has to be explored.

WWW Availability

A version of this paper in larger font, with large figures and clusterings in color is available under the following URL: `http://www.dbs.informatik.uni-muenchen.de/dbs/project/publikationen/veroeffentlichungen.html`.

References

Beckmann N., Kriegel H.-P., Schneider R, and Seeger B. 1990. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

Brinkhoff T., Kriegel H.-P., Schneider R., and Seeger B. 1994 Efficient Multi-Step Processing of Spatial Joins, Proc. ACM SIGMOD Int. Conf. on Management of Data, Minneapolis, MN, 1994, pp. 197-208.

Ester M., Kriegel H.-P., and Xu X. 1995. A Database Interface for Clustering in Large Spatial Databases, Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, Montreal, Canada, 1995, AAAI Press, 1995.

García J.A., Fdez-Valdivia J., Cortijo F. J., and Molina R. 1994. A Dynamic Approach for Clustering Data. *Signal Processing*, Vol. 44, No. 2, 1994, pp. 181-196.

Gueting R.H. 1994. An Introduction to Spatial Database Systems. *The VLDB Journal* 3(4): 357-399.

Jain Anil K. 1988. *Algorithms for Clustering Data*. Prentice Hall.

Kaufman L., and Rousseeuw P.J. 1990. *Finding Groups in Data: an Introduction to Cluster Analysis*. John Wiley & Sons.

Matheus C.J.; Chan P.K.; and Piatetsky-Shapiro G. 1993. Systems for Knowledge Discovery in Databases, *IEEE Transactions on Knowledge and Data Engineering* 5(6): 903-913.

Ng R.T., and Han J. 1994. Efficient and Effective Clustering Methods for Spatial Data Mining, Proc. 20th Int. Conf. on Very Large Data Bases, 144-155. Santiago, Chile.

Stonebraker M., Frew J., Gardels K., and Meredith J.1993. The SEQUOIA 2000 Storage Benchmark, Proc. ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, 1993, pp. 2-11.