Proc. ACM SIGMOD'99 Int. Conf. on Management of Data, Philadelphia PA, 1999.

OPTICS: Ordering Points To Identify the Clustering Structure

Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander

Institute for Computer Science, University of Munich
Oettingenstr. 67, D-80538 Munich, Germany
phone: +49-89-2178-2226, fax: +49-89-2178-2192
{ankerst | breunig | kriegel | sander}@dbs.informatik.uni-muenchen.de

Abstract

Cluster analysis is a primary method for database mining. It is either used as a stand-alone tool to get insight into the distribution of a data set, e.g. to focus further analysis and data processing, or as a preprocessing step for other algorithms operating on the detected clusters. Almost all of the well-known clustering algorithms require input parameters which are hard to determine but have a significant influence on the clustering result. Furthermore, for many real-data sets there does not even exist a global parameter setting for which the result of the clustering algorithm describes the intrinsic clustering structure accurately. We introduce a new algorithm for the purpose of cluster analysis which does *not* produce a clustering of a data set explicitly; but instead creates an augmented *ordering* of the database representing its density-based clustering structure. This cluster-ordering contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings. It is a versatile basis for both automatic and interactive cluster analysis. We show how to automatically and efficiently extract not only 'traditional' clustering information (e.g. representative points, arbitrary shaped clusters), but also the intrinsic clustering structure. For medium sized data sets, the cluster-ordering can be represented graphically and for very large data sets, we introduce an appropriate visualization technique. Both are suitable for interactive exploration of the intrinsic clustering structure offering additional insights into the distribution and correlation of the data.

Keywords

Cluster Analysis, Database Mining, Visualization

1. Introduction

Larger and larger amounts of data are collected and stored in databases increasing the need for efficient and effective analysis methods to make use of the information contained implicitly in the data. One of the primary data analysis tasks is cluster analysis which is intended to help a user to understand the natural grouping or structure in a data set. Therefore, the development of improved clustering algorithms has received a lot of attention in the last few years (c.f. section 2).

Roughly speaking, the goal of a clustering algorithm is to group the objects of a database into a set of meaningful subclasses. A clustering algorithm can be used either as a stand-alone tool to get insight into the distribution of a data set, e.g. in order to focus further analysis and data processing, or as a preprocessing step for other algorithms which operate on the detected clusters. Applications of clustering are, for instance, the creation of thematic maps in geographic information systems by clustering feature spaces [Ric 83], the detection of clusters of objects in geographic information systems and to explain them by other objects in their neighborhood ([NH 94] and [KN 96]), or the clustering of a Web-log database to discover groups of similar access patterns which may correspond to different user profiles [EKS+ 98].

Most of the recent research related to the task of clustering has been directed towards efficiency. The more serious problem, however, is effectivity, i.e. the quality or usefulness of the result. Although most traditional clustering algorithms do not scale well with the size and/or dimension of the data set, one way to overcome this problem is to use *sampling* in combination with a clustering algorithm (see e.g. [EKX 95]). This approach works well for many applications and clustering algorithms. The idea is to apply a clustering algorithm *A* only to a subset of the whole database. From the result of *A* for the subset, we can then *infer* a clustering of the whole database which does not differ much from the result obtained by applying *A* to the whole data set. However, this does not ensure that the result of the clustering algorithm *A* actually reflects the natural groupings in the data.

There are three interconnected reasons why the effectivity of clustering algorithms is a problem. First, almost all clustering algorithms require values for input parameters which are hard to determine, especially for real-world data sets containing high-dimensional objects. Second, the algorithms are very sensible to these parameter values, often producing very different partitionings of the data set even for slightly different parameter settings. Third, high-dimensional real-data sets often have a very skewed distribution that cannot be revealed by a clustering algorithm using only one global parameter setting.

In this paper, we introduce a new algorithm for the purpose of cluster analysis which does *not* produce a clustering of a data set explicitly; but instead creates an augmented *ordering* of the database representing its density-based clustering structure. This cluster-ordering contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings. It is a versatile basis for both automatic and interactive cluster analysis. We show how to automatically and efficiently extract not only 'traditional' clustering information (e.g. representative points, arbitrary shaped clusters), but also the intrinsic clustering structure. For medium sized data sets, the cluster-ordering can be represented graphically and for very large data sets, we introduce an appropriate visualization technique. Both are suitable for interactive exploration of the intrinsic clustering structure offering additional insights into the distribution and correlation of the data.

The rest of the paper is organized as follows. Related work on

clustering is briefly discussed in section 2. In section 3, the basic notions of density-based clustering are defined and our new algorithm OPTICS to create an ordering of a data set with respect to its density-based clustering structure is presented. The application of this cluster-ordering for the purpose of cluster analysis is demonstrated in section 4. Both, automatic as well as interactive techniques are discussed. Section 5 concludes the paper with a summary and a short discussion of future research.

2. Related Work

Existing clustering algorithms can be broadly classified into *hierarchical* and *partitioning clustering* algorithms (see e.g. [JD 88]). Hierarchical algorithms decompose a database *D* of *n* objects into several levels of nested partitionings (clusterings), represented by a *dendrogram*, i.e. 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*. Partitioning algorithms construct a flat (single level) partition of a database *D* of *n* objects into a set of *k* clusters such that the objects in a cluster are more similar to each other than to objects in different clusters.

The Single-Link method is a commonly used hierarchical clustering method [Sib 73]. Starting with the clustering obtained by placing every object in a unique cluster, in every step the two closest clusters in the current clustering are merged until all points are in one cluster. Other algorithms which in principle produce the same hierarchical structure have also been suggested (see e.g. [JD 88], [HT 93]) .

Another approach to hierarchical clustering is based on the clustering properties of spatial index structures. The GRID [Sch 96] and the BANG clustering [SE 97] apply the same basic algorithm to the data pages of different spatial index structures. A clustering is generated by a clever arrangement of the data pages with respect to their point density. This approach, however, is not well suited for high-dimensional data sets because it is based on the effectivity of these structures as spatial access methods. It is well-known that the performance i.e. the clustering properties of spatial index structures degenerate with increasing dimensionality of the data space (e.g. [BKK 96]).

Recently, the hierarchical algorithm CURE has been proposed in [GRS 98]. This algorithm stops the creation of a cluster hierarchy if a level consists of *k* clusters where *k* is one of several input parameters. It utilizes multiple representative points to evaluate the distance between clusters, thereby adjusting well to arbitrary shaped clusters and avoiding the single-link effect. This results in a very good clustering quality. To improve the scalability, random sampling and partitioning (pre-clustering) are used. The authors do provide a sensitivity analysis using one synthetic data set, showing that although some parameters can be varied without impacting the quality of the clustering. The parameter setting does have a profound influence on the result.

Optimization based partitioning algorithms typically represent clusters by a *prototype*. Objects are assigned to the cluster represented by the most similar (i.e. closest) prototype. An iterative control strategy is used to optimize the whole clustering such that, e.g., the average or squared distances of objects to its prototypes are minimized. Consequently, these clustering algorithms are effective in determining a good clustering if the clusters are of convex shape, similar size and density, and if their number *k* can be reasonably estimated.

Depending on the kind of prototypes, one can distinguish *k*-means, *k*-modes and *k*-medoid algorithms. For *k-means* algorithms (see e.g. [Mac 67]), the prototype is the mean value of all objects belonging to a cluster. The *k-modes* [Hua 97] algorithm extends the *k*-means paradigm to categorical domains. For *k-medoid* algorithms (see e.g. [KR 90]), the prototype, called the *medoid*, is one of the objects located *near* the "center" of a cluster. The algorithm CLARANS introduced by [NH 94] is an improved *k*-medoid type algorithm restricting the huge search space by using two additional user-supplied parameters. It is significantly more efficient than the well-known *k*-medoid algorithms PAM and CLARA presented in [KR 90], nonetheless producing a result of nearly the same quality.

Density-based approaches apply a local cluster criterion and are very popular for the purpose of database mining. Clusters are regarded as regions in the data space in which the objects are dense, and which are separated by regions of low object density (noise). These regions may have an arbitrary shape and the points inside a region may be arbitrarily distributed.

A common way to find regions of high-density in the dataspace is based on grid cell densities [JD 88]. A histogram is constructed by partitioning the data space into a number of non-overlapping regions or cells. Cells containing a relatively large number of objects are potential cluster centers and the boundaries between clusters fall in the "valleys" of the histogram. The success of this method depends on the size of the cells which must be specified by the user. Cells of small volume will give a very "noisy" estimate of the density, whereas large cells tend to overly smooth the density estimate.

In [EKSX 96], a density-based clustering method is presented which is not grid-based. The basic idea for the algorithm *DBSCAN* is that for each point of a cluster the neighborhood of a given radius (ε) has to contain at least a minimum number of points (*MinPts*) where ε and *MinPts* are input parameters.

Another density-based approach is WaveCluster [SCZ 98], which applies wavelet transform to the feature space. It can detect arbitrary shape clusters at different scales and has a time complexity of $O(n)$. The algorithm is grid-based and only applicable to low-dimensional data. Input parameters include the number of grid cells for each dimension, the wavelet to use and the number of applications of the wavelet transform.

In [HK 98] the density-based algorithm DenClue is proposed. This algorithm uses a grid but is very efficient because it only keeps information about grid cells that do actually contain data points and manages these cells in a tree-based access structure. This algorithm generalizes some other clustering approaches which, however, results in a large number of input parameters. Also the density- and grid-based clustering technique CLIQUE [AGG+ 98] has been proposed for mining in high-dimensional data spaces. Input parameters are the size of the grid and a global density threshold for clusters. The major difference to all other clustering approaches is that this method also detects subspaces of the highest dimensionality such that high-density clusters exist in those subspaces.

Another recent approach to clustering is the BIRCH method [ZRL 96] which cannot entirely be classified as a hierarchical or partitioning method. BIRCH constructs a CF-tree which is a

hierarchical data structure designed for a multiphase clustering method. First, the database is scanned to build an initial in-memory CF-tree which can be seen as a multi-level compression of the data that tries to preserve the inherent clustering structure of the data. Second, an arbitrary clustering algorithm can be used to cluster the leaf nodes of the CF-tree. Because BIRCH is reasonably fast, it can be used as a more intelligent alternative to data sampling in order to improve the scalability of clustering algorithms.

3. Ordering The Database With Respect To The Clustering Structure

3.1 Motivation

An important property of many real-data sets is that their intrinsic cluster structure cannot be characterized by *global* density parameters. Very different local densities may be needed to reveal clusters in different regions of the data space. For example, in the data set depicted in Figure 1, it is not possible to detect the clusters A, B, C_1, C_2, and C_3 simultaneously using one global density parameter. A global density-based decomposition would consist only of the clusters A, B, and C, or C_1, C_2, and C_3. In the second case, the objects from A and B are noise.

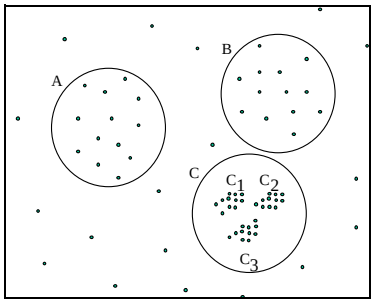


Figure 1. Clusters wrt. different density parameters

The first alternative to detect and analyze such clustering structures is to use a hierarchical clustering algorithm, for instance the single-link method. This alternative, however, has two drawbacks. First, in general it suffers considerably from the single-link effect, i.e. from the fact that clusters which are connected by a line of few points having a small inter-object distance are not separated. Second, the results produced by hierarchical algorithms, i.e. the dendrograms, are hard to understand or analyze for more than a few hundred objects.

The second alternative is to use a density-based partitioning algorithm with different parameter settings. However, there are an infinite number of possible parameter values. Even if we use a very large number of different values - which requires a lot of secondary memory to store the different cluster memberships for each point - it is not obvious how to analyze the results and we may still miss the interesting clustering levels.

The basic idea to overcome these problems is to run an algorithm which produces a special order of the database with respect to its density-based clustering structure containing the information about *every* clustering level of the data set (up to a "generating distance" ε), and is very easy to analyze.

3.2 Density-Based Clustering

The key idea of density-based clustering is that for each object of a cluster the neighborhood of a given radius (ε) has to contain at least a minimum number of objects (*MinPts*), i.e. the cardinality of the neighborhood has to exceed a threshold. The formal definitions for this notion of a clustering are shortly

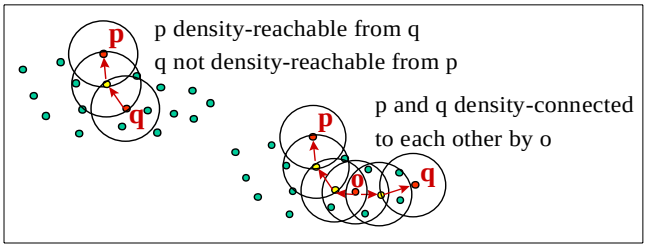


Figure 2. Density-reachability and connectivity

introduced in the following. For a detailed presentation see [EKSX 96].

Definition 1: (directly density-reachable)

Object p is *directly density-reachable* from object q wrt. ε and *MinPts* in a set of objects D if

1) $p \in N_\varepsilon(q)$ ($N_\varepsilon(q)$ is the subset of D contained in the ε-neighborhood of q.)

2) $Card(N_\varepsilon(q)) \geq MinPts$ ($Card(N)$ denotes the cardinality of the set N)

The condition $Card(N_\varepsilon(q)) \geq MinPts$ is called the "core object condition". If this condition holds for an object p, then we call p a "core object". Only from core objects, other objects can be directly density-reachable.

Definition 2: (density-reachable)

An object p is *density-reachable* from an object q wrt. ε and *MinPts* in the set of objects D if there is a chain of objects $p_1, ..., p_n$, $p_1 = q$, $p_n = p$ such that $p_i \in D$ and p_{i+1} is directly density-reachable from p_i wrt. ε and *MinPts*.

Density-reachability is the transitive hull of direct density-reachability. This relation is not symmetric in general. Only core objects can be mutually density-reachable.

Definition 3: (density-connected)

Object p is *density-connected* to object q wrt. ε and *MinPts* in the set of objects D if there is an object $o \in D$ such that both p and q are density-reachable from o wrt. ε and *MinPts* in D.

Density-connectivity is a symmetric relation. Figure 2 illustrates the definitions on a sample database of 2-dimensional points from a vector space. Note that the above definitions only require a distance measure and will also apply to data from a metric space.

A density-based *cluster* is now defined as a set of density-connected objects which is maximal wrt. density-reachability and the *noise* is the set of objects not contained in any cluster.

Definition 4: (cluster and noise)

Let D be a set of objects. A *cluster* C wrt. ε and *MinPts* in D is a non-empty subset of D satisfying the following conditions:

1) Maximality: $\forall p,q \in D$: if $p \in C$ and q is density-reachable from p wrt. ε and *MinPts*, then also $q \in C$.

2) Connectivity: $\forall p,q \in C$: p is density-connected to q wrt. ε and *MinPts in D*.

Every object not contained in any cluster is *noise*.

Note that a cluster contains not only core objects but also objects that do not satisfy the core object condition. These objects

- called "border objects" of the cluster - are, however, directly density-reachable from at least one core object of the cluster (in contrast to noise objects).

The algorithm DBSCAN [EKSX 96], which discovers the clusters and the noise in a database according to the above definitions, is based on the fact that a cluster is equivalent to the set of all objects in D which are density-reachable from an arbitrary core object in the cluster (c.f. lemma 1 and 2 in [EKSX 96]). The retrieval of density-reachable objects is performed by iteratively collecting *directly* density-reachable objects. DBSCAN checks the ε-neighborhood of each point in the database. If the ε-neighborhood $N_\varepsilon(p)$ of a point p has more than *MinPts* points, a new cluster C containing the objects in $N_\varepsilon(p)$ is created. Then, the ε-neighborhood of all points q in C which have not yet been processed is checked. If $N_\varepsilon(q)$ contains more than *MinPts* points, the neighbors of q which are not already contained in C are added to the cluster and their ε-neighborhood is checked in the next step. This procedure is repeated until no new point can be added to the current cluster C.

3.2.1 Density-Based Cluster-Ordering

To introduce the notion of a density-based cluster-ordering, we first make the following observation: for a constant *MinPts*-value, density-based clusters with respect to a higher density (i.e. a lower value for ε) are completely contained in density-connected sets with respect to a lower density (i.e. a higher value for ε). This fact is illustrated in figure 3, where C_1 and C_2 are density-based clusters with respect to $\varepsilon_2 < \varepsilon_1$ and C is a density-based cluster with respect to ε_1 completely containing the sets C_1 and C_2.

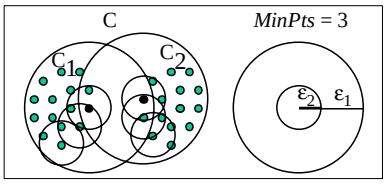


Figure 3. Illustration of "nested" density-based clusters

Consequently, we could extend the DBSCAN algorithm such that several distance parameters are processed at the same time, i.e. the density-based clusters with respect to different densities are constructed simultaneously. To produce a consistent result, however, we would have to obey a specific *order* in which objects are processed when expanding a cluster. We always have to select an object which is density-reachable with respect to the lowest ε value to guarantee that clusters with respect to higher density (i.e. smaller ε values) are finished first.

Our new algorithm OPTICS works in principle like such an extended DBSCAN algorithm for an infinite number of distance parameters ε_i which are smaller than a "generating distance" ε (i.e. $0 \le \varepsilon_i \le \varepsilon$). The only difference is that we do not assign cluster memberships. Instead, we store the *order* in which the objects are processed and the information which *would* be used by an extended DBSCAN algorithm to assign cluster memberships (if this were at all possible for an infinite number of parameters). This information consists of only two values for each object: the *core-distance* and a *reachability-distance*, introduced in the following definitions.

Definition 5: (core-distance of an object p)

Let p be an object from a database D, let ε be a distance value, let $N_\varepsilon(p)$ be the ε-neighborhood of p, let *MinPts* be a natural number and let *MinPts-distance(p)* be the distance from p to its *MinPts*' neighbor. Then, the *core-distance* of p is defined as *core-distance*$_{\varepsilon,MinPts}(p)$ =

$$\begin{cases} UNDEFINED, \text{ if } Card(N_\varepsilon(p)) < MinPts \\ MinPts\text{-}distance(p), \text{ otherwise} \end{cases}$$

The core-distance of an object p is simply the smallest distance ε' between p and an object in its ε-neighborhood such that p would be a core object with respect to ε' if this neighbor is contained in $N_\varepsilon(p)$. Otherwise, the core-distance is UNDEFINED.

Definition 6: (reachability-distance object p w.r.t. object o)

Let p and o be objects from a database D, let $N_\varepsilon(o)$ be the ε-neighborhood of o, and let *MinPts* be a natural number. Then, the *reachability-distance* of p with respect to o is defined as *reachability-distance*$_{\varepsilon,MinPts}(p, o)$ =

$$\begin{cases} UNDEFINED, \text{ if } |N_\varepsilon(o)| < MinPts \\ max(core\text{-}distance(o), distance(o, p)), \text{otherwise} \end{cases}$$

Intuitively, the reachability-distance of an object p with respect to another object o is the smallest distance such that p is directly density-reachable from o if o is a core object. In this case, the reachability-distance cannot be smaller than the core-distance of o because for smaller distances no object is directly density-reachable from o. Otherwise, if o is not a core object, even at the generating distance ε, the reachability-distance of p with respect to o is UNDEFINED. The reachability-distance of an object p depends on the core object with respect to which it is calculated. Figure 5 illustrates the notions of core-distance and reachability-distance.

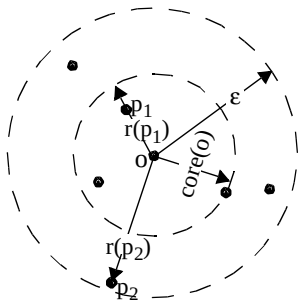


Figure 4. Core-distance(o), reachability-distances r(p_1,o), r(p_2,o) for *MinPts*=4

Our algorithm OPTICS creates an ordering of a database, additionally storing the core-distance and a suitable reachability-distance for each object. We will see that this information is sufficient to extract all density-based clusterings with respect to any distance ε' which is smaller than the generating distance ε from this order. Figure 5 illustrates the main loop of the algorithm OPTICS. At the beginning, we open a file OrderedFile for writing and close this file after ending the loop. Each object from a database SetOfObjects is simply handed over to a procedure ExpandClusterOrder if the object is not yet processed.

The pseudo-code for the procedure ExpandClusterOrder is depicted in figure 6 . The procedure ExpandClusterOrder first retrieves the ε-neighborhood of the object Object passed from the main loop OPTICS, sets its reachability-distance to UNDE-

```
OPTICS (SetOfObjects, ε, MinPts, OrderedFile)
  OrderedFile.open();
  FOR i FROM 1 TO SetOfObjects.size DO
    Object := SetOfObjects.get(i);
    IF NOT Object.Processed THEN
      ExpandClusterOrder(SetOfObjects, Object, ε,
                 MinPts, OrderedFile)
  OrderedFile.close();
END; // OPTICS
```

Figure 5. Algorithm OPTICS

FINED and determines its core-distance. Then, Object is written to OrderedFile. The IF-condition checks the core object property of Object and if it is not a core object at the generating distance ε, the control is simply returned to the main loop OPTICS which selects the next unprocessed object of the database. Otherwise, if Object is a core object at a distance ≤ ε, we iteratively collect directly density-reachable objects with respect to ε and *MinPts*. Objects which are directly density-reachable from a current core object are inserted into the seed-list OrderSeeds for further expansion. The objects contained in OrderSeeds are sorted by their reachability-distance to the closest core object from which they have been directly density-reachable. In each step of the WHILE-loop, an object currentObject having the smallest reachability-distance in the seed-list is selected by the method OrderSeeds:next(). The ε-neighborhood of this object and its core-distance are determined. Then, the object is simply written to the file OrderedFile with its core-distance and its current reachability-distance. If currentObject is a core object, further candidates for the expansion may be inserted into the seed-list OrderSeeds.

```
ExpandClusterOrder(SetOfObjects, Object, ε, MinPts,
OrderedFile);
  neighbors := SetOfObjects.neighbors(Object, ε);
  Object.Processed := TRUE;
  Object.reachability_distance := UNDEFINED;
  Object.setCoreDistance(neighbors, ε, MinPts);
  OrderedFile.write(Object);
  IF Object.core_distance <> UNDEFINED THEN
    OrderSeeds.update(neighbors, Object);
    WHILE NOT OrderSeeds.empty() DO
      currentObject := OrderSeeds.next();
      neighbors:=SetOfObjects.neighbors(currentObject, ε);
      currentObject.Processed := TRUE;
      currentObject.setCoreDistance(neighbors, ε, MinPts);
      OrderedFile.write(currentObject);
      IF currentObject.core_distance<>UNDEFINED THEN
        OrderSeeds.update(neighbors, currentObject);
END; // ExpandClusterOrder
```

Figure 6. Procedure ExpandClusterOrder

Insertion into the seed-list and the handling of the reachability-distances is managed by the method OrderSeeds::update(neighbors, CenterObject) depicted in figure 7. The reachability-distance for each object in the set neighbors is determined with respect to the center-object CenterObject. Objects which are not yet in the priority-queue OrderSeeds are simply inserted with their reachability-distance. Objects which are already in the queue are moved further to the top of the queue if their new reachability-distance is smaller than their previous reachability-distance.

Due to its structural equivalence to the algorithm DBSCAN, the run-time of the algorithm OPTICS is nearly the same as the run-time for DBSCAN. We performed an extensive performance test using different data sets and different parameter settings. It simply turned out that the run-time of OPTICS was almost constantly 1.6 times the run-time of DBSCAN. This is not surprising since the run-time for OPTICS as well as for DBSCAN is heavily dominated by the run-time of the ε-neighborhood queries which must be performed for each object in the database, i.e. the run-time for both algorithms is O(*n* * run-time of an ε-neighborhood query).

To retrieve the ε-neighborhood of an object *o*, a *region query* with the center *o* and the radius ε is used. Without any index support, to answer such a region query, a scan through the whole database has to be performed. In this case, the run-time of OPTICS would be $O(n^2)$. If a tree-based spatial index can be used, the run-time is reduced to $O(n \log n)$ since region queries are supported efficiently by spatial access methods such as the R*-tree [BKSS 90] or the X-tree [BKK 96] for data from a vector space or the M-tree [CPZ 97] for data from a metric space. The height of such a tree-based index is O(log *n*) for a database of *n* objects in the worst case and, at least in low-dimensional spaces, a query with a "small" query region has to traverse only a limited number of paths. Furthermore, if we have a direct access to the ε-neighborhood, e.g. if the objects are organized in a grid, the run-time is further reduced to $O(n)$ because in a grid the complexity of a single neighborhood query is $O(1)$.

Having generated the augmented cluster-ordering of a database with respect to ε and *MinPts*, we can extract any density-based clustering from this order with respect to *MinPts* and a clustering-distance ε' ≤ ε by simply "scanning" the cluster-ordering and assigning cluster-memberships depending on the reachability-distance and the core-distance of the objects. Figure 8 depicts the algorithm ExtractDBSCAN-Clustering which performs this task.

We first check whether the reachability-distance of the current object Object is larger than the clustering-distance ε'. In this case, the object is not density-reachable with respect to ε' and *MinPts* from any of the objects which are located before the current object in the cluster-ordering. This is obvious, because if Object had been density-reachable with respect to ε' and *MinPts* from a preceding object in the order, it would have been assigned a reachability-distance of at most ε'. Therefore, if the reachability-distance is larger than ε', we look at the core-distance of Object and start a new cluster if Object is a core object with respect to ε' and *MinPts*; otherwise, Object is assigned to

```
OrderSeeds::update(neighbors, CenterObject);
  c_dist := CenterObject.core_distance;
  FORALL Object FROM neighbors DO
    IF NOT Object.Processed THEN
      new_r_dist:=max(c_dist,CenterObject.dist(Object));
      IF Object.reachability_distance=UNDEFINED THEN
        Object.reachability_distance := new_r_dist;
        insert(Object, new_r_dist);
      ELSE // Object already in OrderSeeds
        IF new_r_dist<Object.reachability_distance THEN
          Object.reachability_distance := new_r_dist;
          decrease(Object, new_r_dist);
END; // OrderSeeds::update
```

Figure 7. Method OrderSeeds::update()

```
ExtractDBSCAN-Clustering (ClusterOrderedObjs,ε', MinPts)
// Precondition: ε' ≤ generating dist ε for ClusterOrderedObjs
   ClusterId := NOISE;
   FOR i FROM 1 TO ClusterOrderedObjs.size DO
      Object := ClusterOrderedObjs.get(i);
      IF Object.reachability_distance > ε' THEN
         // UNDEFINED > ε
         IF Object.core_distance ≤ ε' THEN
            ClusterId := nextId(ClusterId);
            Object.clusterId := ClusterId;
         ELSE
            Object.clusterId := NOISE;
      ELSE     // Object.reachability_distance ≤ ε'
         Object.clusterId := ClusterId;
END; // ExtractDBSCAN-Clustering
```

Figure 8. Algorithm ExtractDBSCAN-Clustering

NOISE (note that the reachability-distance of the first object in the cluster-ordering is always UNDEFINED and that we assume UNDEFINED to be greater than any defined distance). If the reachability-distance of the current object is smaller than ε', we can simply assign this object to the current cluster because then it is density-reachable with respect to ε' and *MinPts* from a preceding core object in the cluster-ordering.

The clustering created from a cluster-ordered data set by ExtractDBSCAN-Clustering is nearly indistinguishable from a clustering created by DBSCAN. Only some border objects may be missed when extracted by the algorithm ExtractDBSCAN-Clustering if they were processed by the algorithm OPTICS before a core object of the corresponding cluster had been found. However, the fraction of such border objects is so small that we can omit a postprocessing (i.e. reassign those objects to a cluster) without much loss of information.

To extract different density-based clusterings from the cluster-ordering of a data set is not the intended application of the OPTICS algorithm. That an extraction is possible only demonstrates that the cluster-ordering of a data set actually contains the information about the intrinsic clustering structure of that data set (up to the generating distance ε). This information can be analyzed much more effectively by using other techniques which are presented in the next section.

4. Identifying The Clustering Structure

The OPTICS algorithm generates the augmented cluster-ordering consisting of the ordering of the points, the reachability-values and the core-values. However, for the following interactive and automatic analysis techniques only the ordering and the reachability-values are needed. To simplify the notation, we specify them formally:

Definition 7: (results of the OPTICS algorithm)

Let *DB* be a database containing *n* points. The OPTICS algorithm generates an ordering of the points $\boldsymbol{o}:\{1..n\} \rightarrow DB$ and corresponding reachability-values $\boldsymbol{r}:\{1..n\} \rightarrow \boldsymbol{R}_{\geq 0}$.

The visual techniques presented below fall into two main categories. First, methods to get a general overview of the data. These are useful for gaining a high-level understanding of the way the data is structured. It is important to see most or even all of the data at once, making pixel-oriented visualizations the method of choice. Second, once the general structure is understood, the user is interested in zooming into the most interesting looking subsets. In the corresponding detailed view, single (small or large) clusters are being analyzed and their relationships examined. Here it is important to show the maximum amount of information which can easily be understood. Thus, we present different techniques for these two different tasks. Because the detailed technique is a direct graphical representation of the cluster-ordering, we present it first and then continue with the high-level technique.

A totally different set of requirements is posed for the automatic techniques. They are used to generate the intrinsic cluster structure automatically for further (automatic) processing steps.

4.1 Reachability Plots And Parameters

The cluster-ordering of a data set can be represented and understood graphically. In principle, one can *see* the clustering structure of a data set if the reachability-distance values $\boldsymbol{r}$ are plotted for each object in the cluster-ordering $\boldsymbol{o}$. Figure 9 depicts the reachability-plot for a very simple 2-dimensional data set. Note that the visualization of the cluster-order is independent from the dimension of the data set. For example, if the objects of a high-dimensional data set are distributed similar to the distribution of the 2-dimensional data set depicted in figure 9 (i.e. there are three "Gaussian bumps" in the data set), the "reachability-plot" would also look very similar.

A further advantage of cluster-ordering a data set compared to other clustering methods is that the reachability-plot is rather insensitive to the input parameters of the method, i.e. the generating distance ε and the value for *MinPts*. Roughly speaking, the values have just to be "large" enough to yield a good result. The concrete values are not crucial because there is a broad range of possible values for which we always can see the clustering structure of a data set when looking at the corresponding reachability-plot. Figure 10 shows the effects of different parameter settings on the reachability-plot for the same data set used in figure 9. In the first plot we used a smaller generating distance ε, for the second plot we set *MinPts* to the smallest possible value. Although, these plots look different from the plot depicted in figure 9, the overall clustering structure of the data set can be recognized in these plots as well.

The generating distance ε influences the number of clustering-levels which can be seen in the reachability-plot. The smaller we choose the value of ε, the more objects may have an UNDEFINED reachability-distance. Therefore, we may not see clusters of lower density, i.e. clusters where the core objects are core objects only for distances larger than ε.

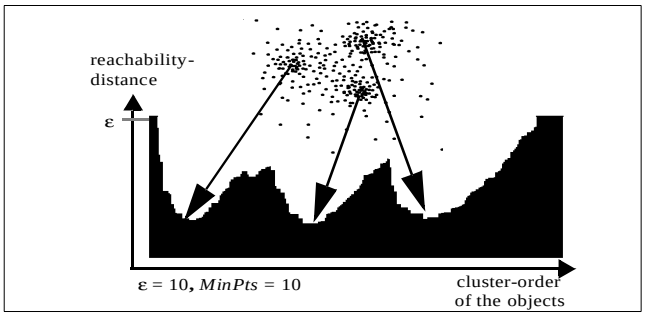


Figure 9. Illustration of the cluster-ordering

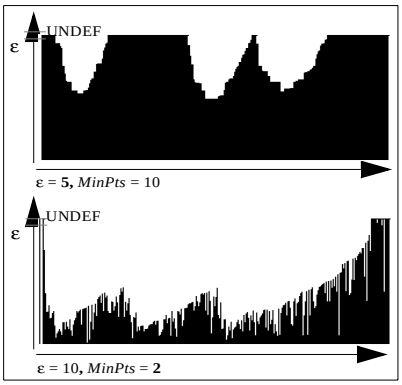


Figure 10. Effects of parameter settings on the cluster-ordering

The optimal value for ε is the smallest value so that a density-based clustering of the database with respect to ε and *MinPts* consists of only one cluster containing almost all points of the database. Then, the information of *all* clustering levels will be contained in the reachability-plot. However, there is a large range of values around this optimal value for which the appearance of the reachability-plot will not change significantly. Therefore, we can use rather simple heuristics to determine the value for ε, as we only need to guarantee that the distance value will not be too small. For example, we can use the expected *k*-nearest-neighbor distance (for $k = MinPts$) under the assumption that the objects are randomly distributed, i.e. under the assumption that there are no clusters. This value can be determined analytically for a data space *DS* containing *N* points. The distance is equal to the radius *r* of a *d*-dimensional hypersphere *S* in *DS* where *S* contains exactly *k* points. Under the assumption of a random distribution of the points, it holds that $Volume_S = \frac{Volume_{DS}}{N} \times k$ and the volume of a *d*-dimensional hypersphere *S* having a radius *r* is $Volume_{S(r)} = \frac{\sqrt{\pi^d}}{\Gamma(\frac{d}{2}+1)} \times r^d$, where Γ denotes the Gamma-function. The radius *r* can be computed as $r = \sqrt[d]{\frac{Volume_{DS} \times k \times \Gamma(\frac{d}{2}+1)}{N \times \sqrt{\pi^d}}}$.

The effect of the *MinPts*-value on the visualization of the cluster-ordering can be seen in figure 10. The overall shape of the reachability-plot is very similar for different *MinPts* values. However, for lower values the reachability-plot looks more jagged and higher values for *MinPts* smoothen the curve. Moreover, high values for *MinPts* will significantly weaken possible "single-link" effects. Our experiments indicate that we will always get good results using values between 10 and 20.

To show that the reachability-plot is very easy to understand, we will finally present some examples. Figure 11 depicts the reachability-plot for a very high-dimensional real-world data set containing 10,000 greyscale images of 32x32 pixels. Each object is represented by a vector containing the greyscale value for each pixel. Thus, the dimension of the vectors is equal to 1,024. The Euclidean distance function was used as similarity measure for these vectors.

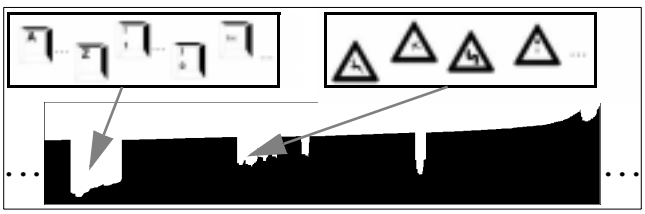


Figure 11. Part of the reachability-plot for 1,024-*d* image data

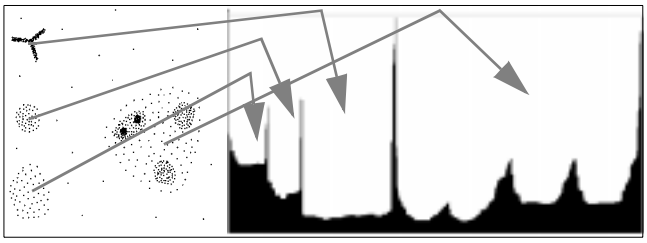


Figure 12. Reachability-plots for a data set with hierarchical clusters of different sizes, densities and shapes

Figure 12 shows a further example of a reachability-plot having characteristics which are very typical for real-world data sets. For a better comparison of the real distribution with the cluster-ordering of the objects, the data set was synthetically generated in two dimensions. Obviously, there is no global density-threshold (which is graphically a horizontal line in the reachability plot) that can reveal all the structure in the data set.

To make the simple reachability-plots even more useful, we can additionally show an *attribute-plot*. For every point it shows the attribute values (discretized into 256 levels) for every dimension. Underneath each reachability value we plot for each dimension one rectangle in a shade of grey corresponding to the value of this attribute. The width of the rectangle is the same as the width of the reachability bar above it, whereas the height can be chosen arbitrarily by the user. In figure 13, for example, we see the reachability-plot and the attribute-plot for 9-dimensional data from weather stations. The data is very clearly structured into a number of clusters with very little noise in between as we can see from the reachability-plot. From the attribute-plot, we can furthermore see that the points in each cluster are close to each other mainly because within the set of points belonging to a cluster, the attribute values in all but one attribute does not differ significantly. A domain expert knowing what each dimension represents will find this to be a very useful information.

To summarize, the reachability-plot is a very intuitive means for getting a clear understanding of the structure of the data. Its shape is mostly independent from the choice of the parameters ε and *MinPts*. If we supplement it further with attribute-plots, we can even gain information about the dependencies between the clustering structure and the attribute values.

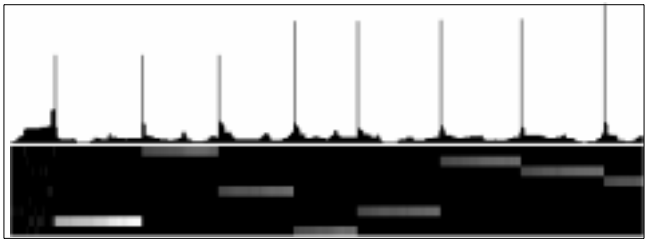


Figure 13. Reachability-plot and attribute-plot for 9-*d* data from weather stations

4.2 Visualizing Large High-*d* Data Sets

The applicability of the reachability-plot is obviously limited to a certain number of points as well as dimensions. After scrolling a couple of screens of information, it is hard to remember the overall structure in the data. Therefore, we investigate approaches for visualizing very large amounts of multidimensional data (see [Kei 96b] for an excellent classification of the existing techniques).

In order to increase the amount of both, the number of objects and the number of dimensions that can be visualized simultaneously, we could apply commonly used reduction techniques like the wavelet transform [GM 85] or the Discrete Fourier transform [PTVF 92] and display a *compressed* reachability-plot. The major drawback of this approach, however, is that we may loose too much information, especially with respect to the structural similarity of the reachability-plot to the attribute-plot. Therefore, we decided to extend a *pixel-oriented* technique [Kei 96a] which can visualize more data items at the same time than other visualization methods.

The basic idea of pixel-oriented techniques is to map each attribute value to one colored pixel and to present the attribute values belonging to different dimensions in separate subwindows. The color of a pixel is determined by the HSI color scale which is a slight modification of the scale generated by the HSV color model. Within each subwindow, the attribute values of the same record are plotted at the same relative position.

Definition 8: (pixel-oriented visualization technique)

Let *Col* be the HSI color scale, *d* the number of dimensions, dom_d the domain of the dimension *d* and $\aleph \times \aleph$ the pixelspace on the screen. Then a pixel-oriented visualization technique (PO) consists of the two independent mappings SO (*sorting*) and DV (*data-values*). SO maps the cluster-ordering to the arrangement, and DV maps the attribute-values to colors: PO=(SO,DV) with

$$\text{SO} : \{1 \ldots n\} \rightarrow \aleph \times \aleph \text{ and DV} : (dom_d)^d \rightarrow Col .$$

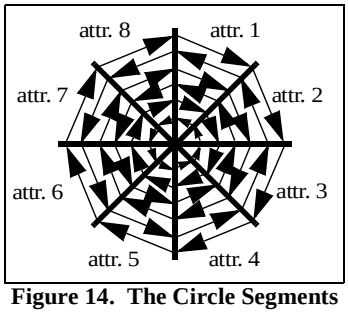


Figure 14. The Circle Segments technique for 8-*d* data

Existing pixel-oriented techniques differ only in the arrangements SO within the subwindows. For our application, we extended the Circle Segments technique introduced in [AKK 96]. The Circle Segments technique maps *n*-dimensional objects to a circle which is partitioned into *n* segments representing one attribute each. Figure 14 illustrates the partitioning of the circle as well as the arrangement within each segment. It starts in the middle of the circle and continues to the outer border of the corresponding segment in a line-by-line fashion. Since the attribute values of the same record are all mapped to the same relative position, their coherence is perceived as parts of a circle.

For the purpose of cluster analysis, we extend the Circle Segments technique as follows:

- **Discretization**. Discretization of the data values can obviously improve the distinctness of the cluster structure. We generate the mapping of the data values to the greyscale colormap dynamically, thus enabling the user to adapt the mapping to his domain-specific requirements. Since *Col* in the mapping DV is a user-specified colormap (e.g. greyscale), the discretization determines the number of different colors used.
- **Small clusters**. Potentially interesting clusters may consist of relatively few data points which should be perceptible even in a very large data set. Let $Resolution$ be the sidelength of a square of pixels used for visualizing one attribute value. We have extended the mapping SO to SO', with SO': $\{1 \ldots n\} \rightarrow |\aleph \times \aleph| \cdot Resolution^{-2}$. $Resolution$ can be chosen by the user.
- **Progression of the ordered data values**. The color scale *Col* in the mapping DV should reflect the progression of the ordered data values in a way that is well perceptible. Our experiments indicate that the greyscale colormap is most suitable for the detection of hierarchical clusters.

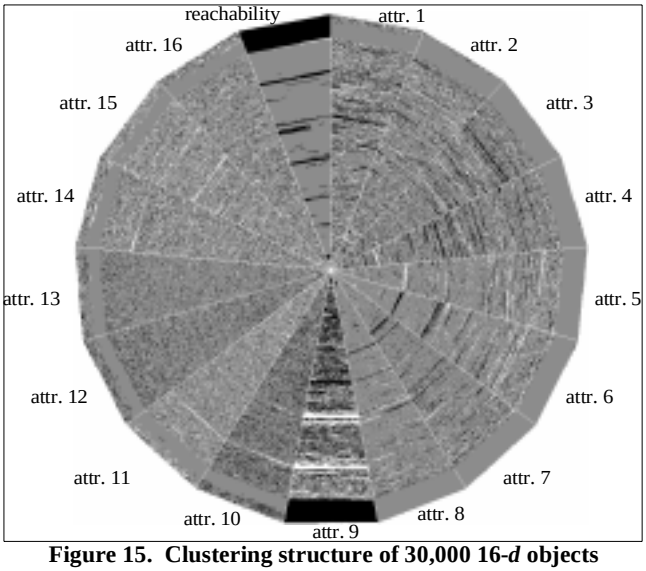


Figure 15. Clustering structure of 30,000 16-*d* objects

In the following example using real-world data, the cluster-ordering of both, the reachability values and the attribute values, is mapped from the inside of the circle to the outside. DV maps high values to light colors and low data values to dark colors. As far as the reachability values are concerned, the significance of a cluster is correlated to the darkness of the color since it reflects close distances. For all other attributes, the color represents the attribute value. Due to the same relative position of the attributes and the reachability for each object, the relations between the attribute values and the clustering structure can be easily examined.

In figure 15, 30,000 records consisting of 16 attributes of fourier-transformed data describing contours of industrial parts and the reachability attribute are visualized by setting the discretization to just three different colors, i.e. white, grey and black. The representation of the reachability attribute clearly shows the general clustering structure, revealing many small to medium sized clusters (regions with black pixels). Only the

outside of the segments which depicts the end of the ordering shows a large cluster surrounded by white-colored regions denoting noise. When comparing the progression of all attribute values within this large cluster, it becomes obvious that attributes 2 - 9 all show an (up to discretization) constant value, whereas the other attributes differ in their values in the last third part. Moreover, in contrast to all other attributes, attribute 9 has its lowest value within the large cluster and its highest value within other clusters. When focussing on smaller clusters like the third black stripe in the reachability attribute, the user identifies the attributes 5, 6 and 7 as the ones which have values differing from the neighboring attribute values in the most remarkable fashion. Many other data properties can be revealed when selecting a small subset and visualizing it with the reachability-plot in great detail.

To summarize, with the extended Circle Segments technique we are able to visualize large multidimensional data sets supporting the user in analyzing attributes in relation to the overall cluster structure and to other attributes. Note that attributes not used by the OPTICS clustering algorithm to determine the cluster structure can also be mapped to additional segments for the same kind of analysis.

4.3 Automatic Techniques

In this section, we will look at how to analyze the cluster-ordering automatically and generate a hierarchical clustering structure from it. The general idea of our algorithm is to identify potential start-of-cluster and end-of-cluster regions first, and then to combine matching regions into (nested) clusters.

4.3.1 Concepts And Formal Definition Of A Cluster

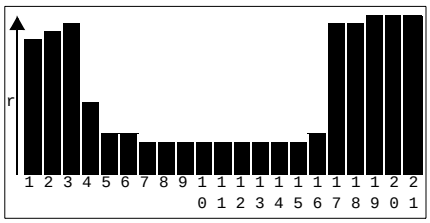


Figure 16. A cluster

In order to identify the clusters contained in the database, we need a notion of "clusters" based on the results of the OPTICS algorithm. As we have seen above, the reachability value of a point corresponds to the distance of this point to the set of its predecessors. From this (and the way OPTICS chooses the order) we know that clusters are dents in the reachability-plot. For example, in figure 16 we see a cluster starting at point #3 and ending at point #16. Note that point #3, which is the last point with a high reachability value, is part of the cluster, because its high reachability indicates that it is far away from the points #1 and #2. It has to be close to point #4, however, because point #4 has a low reachability value, indicating that it is close to one of the points #1, #2 or #3. Because of the way OPTICS chooses the next point in the cluster-ordering, it has to be close to #3 (if it were close to point #1 or point #2 it would have been assigned index 3 and not index 4). A similar argument holds for point #16, which is the last point with a low reachability value, and therefore is also part of the cluster.

Clusters in real-world data sets do not always start and end with extremely steep points. For example, in figure 17 we see three clusters that are very different. The first one starts with a very steep point A and ends with a very steep point B, whereas the second one ends with a number of not-quite-so-steep steps in point D. To capture these different degrees of steepness, we need to introduce a parameter ξ:

Definition 9: (ξ-steep points)

A point $p \in \{1 \ldots n-1\}$ is called a ξ-steep upward point iff it is ξ% lower than its successor:

$$UpPoint_\xi(p) \Leftrightarrow r(p) \le r(p+1) \times (1-\xi)$$

A point $p \in \{1 \ldots n-1\}$ is called a ξ-steep downward point iff p's successor is ξ% lower:

$$DownPoint_\xi(p) \Leftrightarrow r(p) \times (1-\xi) \le r(p+1)$$

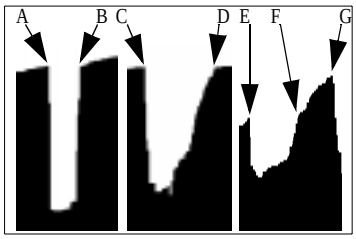


Figure 17. Real world clusters

Looking closely at figure 17, we see that all clusters start and end in a number of upward points, most of, but not all are ξ-steep. These 'regions' start with ξ-steep points, followed by a few points where the reachability values level off, followed by more ξ-steep points. We will call such a region a *steep area*. More precisely, a ξ-steep upward area starts at a ξ-steep upward point, ends at a ξ-steep upward point and always keeps on going upward or level. Also, it must not contain too many consecutive non-steep points. Because of the core condition used in OPTICS, a natural choice for "not too many" is "fewer than *MinPts*", because *MinPts* consecutive non-steep points can be considered as a separate cluster and should not be part of a steep area. Our last requirement is that ξ-steep areas be as large as possible, leading to the following definition:

Definition 10: (ξ-steep areas)

An interval $I = [s, e]$ is called a ξ-steep upward area $UpArea_\xi(I)$ iff it satisfies the following conditions:

- s is a ξ-steep upward point: $UpPoint_\xi(s)$

- e is a ξ-steep upward point: $UpPoint_\xi(e)$

- each point between s and e is at least as high as its predecessor:

$$\forall x, s < x \le e : r(x) \ge r(x-1)$$

- I does not contain more than *MinPts* consecutive points that are not ξ-steep upward:

$$\forall [\bar{s}, \bar{e}] \subseteq [s, e] : ((\forall x \in [\bar{s}, \bar{e}] : \neg UpPoint_\xi(x)) \Rightarrow \bar{e} - \bar{s} < MinPts)$$

- I is maximal: $\forall J : (I \subseteq J, UpArea_\xi(J) \Rightarrow I = J)$

A ξ-steep downward area is defined analogously ($DownArea_\xi(I)$).

The first and the second cluster in figure 17 start at the beginning of steep areas (points A and C) and end at the end of other steep areas (points B and D), while the third cluster ends at point F in the middle of a steep area extending up to point G.

Given these definitions, we can formalize the notion of a clus-

ter. The following definition of a ξ-cluster consists of 4 conditions, each will be explained in detail. Recall that the first point of a cluster (called the start of the cluster) is the last point with a high reachability value, and the last point of a cluster (called the end of the cluster) is the last point with a low reachability value.

Definition 11: (ξ-cluster)

An interval $C = [s, e] \subseteq [1, n]$ is called a ξ-cluster iff it satisfies conditions 1 to 4:

$\text{cluster}_\xi(C) \Leftrightarrow \exists D = [s_D, e_D], U = [s_U, e_U]$ with

1) $DownArea_\xi(D) \wedge s \in D$

2) $UpArea_\xi(U) \wedge e \in U$

3) a) $e - s \geq MinPts$

b) $\forall x, s_D < x < e_U : (r(x) \leq min(r(s_D), r(e_U)) \times (1 - \xi))$

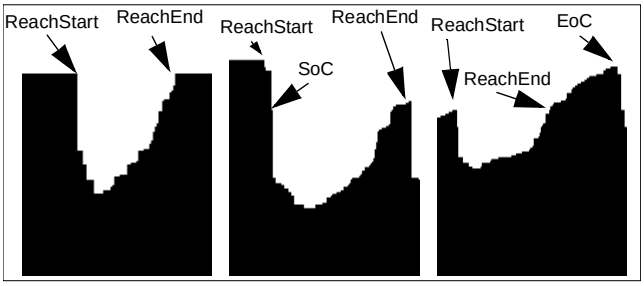
4) (*s*,*e*) =

$$\begin{cases} (max\{x \in D \mid r(x) > r(e_U + 1)\}, e_U) \text{ if } r(s_D) \times (1-\xi) \geq r(e_U + 1) & \text{(b)} \\ (s_D, min\{x \in U \mid r(x) < r(s_D)\}) \quad \text{if } r(e_U + 1) \times (1-\xi) \geq r(s_D) & \text{(c)} \\ (s_D, e_U) \qquad \text{otherwise} & \text{(a)} \end{cases}$$

Conditions 1) and 2) simply state that the start of the cluster is contained in a ξ-steep downward area *D* and the end of the cluster is contained in a ξ-steep upward area *U*.

Condition 3a) states that the cluster has to consist of at least *MinPts* points because of the core condition used in OPTICS. Condition 3b) states that the reachability values of all points in the cluster have to be at least ξ% lower than the reachability value of the first point of *D* and the reachability value of the first point after the end of U.

Condition 4) determines the start and the end point. Depending on the reachability values of the first point of *D* (called *ReachStart*) and the reachability of the first point after the end of *U* (called *ReachEnd*), we distinguish three cases (c.f. figure 18). First, if these two values are at most ξ% apart, the cluster starts at the beginning of *D* and ends at the end of *U* (figure 18a). Second, if *ReachStart* is more than ξ% higher than *ReachEnd*, the cluster ends at the end of *U*, but starts at that point in *D*, that has approximately the same reachability value as *ReachEnd* (figure 18b, cluster starts at SoC). Otherwise (i.e. if *ReachEnd* is more than ξ% higher than *ReachStart*), the cluster starts at the first point of *D* and ends at that point in *U*, that has approximately the same reachability value as *ReachStart* (figure 18c, cluster ends at EoC).



(a) **(b)** **(c)**

Figure 18. Three different types of clusters taken from an industrial parts data set

4.3.2 An Efficient Algorithm To Compute All ξ-Clusters

We will now present an efficient algorithm to compute all ξ-clusters using only one pass over the reachability values of all points. The basic idea is to identify all ξ-steep down and ξ-steep up areas (we will drop the ξ for the remainder of this section and just talk about steep up areas, steep down areas and clusters), and combine them into clusters if a combination satisfies all of the cluster conditions.

We will start out with a naïve implementation of definition 11. The resulting algorithm, however, is inefficient, so in a second step we will show how to transform it into a more efficient version while still finding all clusters. The naïve algorithm works as follows: We make one pass over all reachability values in the order computed by OPTICS. Our main data structure is a set of steep down regions SDASet which contains for each steep down region encountered so far its start-index and end-index. We start at index=0 with an empty SDASet.

While index < n do

① If a new steep down region starts at index, add it to SDASet and continue to the right of it.

② If a new steep up region starts at index, combine it with every steep down region in SDASet, check each combination for fulfilling the cluster conditions and save it if it is a cluster (computing *s* and *e* according to condition 4). Continue to the right of this steep up region.

③ Otherwise, increment index.

Obviously, this algorithm finds all clusters. It makes one pass over the reachability values and additionally one loop over each potential cluster to check for condition 3b in step ②. The number of potential clusters is quadratic in the number of steep up and steep down regions. Most combinations do not actually result in clusters, however, so we employ two techniques to overcome this inefficiency: First, we filter out most potential clusters which will not result in real clusters, and second, we get rid of the loop over all points in the cluster.

Condition 3b,

$\forall x, s_D < x < e_U : (r(x) \leq min(r(s_D), r(e_U)) \times (1 - \xi))$, is equivalent to

(sc1) $\forall x, s_D < x < e_U : (r(x) \leq r(s_D) \times (1 - \xi))$ $\wedge$

(sc2) $\forall x, s_D < x < e_U : (r(x) \leq r(e_U) \times (1 - \xi))$,

so we can split it and check the sub-conditions (sc1) and (sc2) separately. We can further transform (sc1) and (sc2) into the equivalent condition (sc1*) and (sc2*), respectively:

(sc1*) $max\{x \mid s_D < x < e_U\} \leq r(s_D) \times (1 - \xi)$

(sc2*) $max\{x \mid s_D < x < e_U\} \leq r(e_U) \times (1 - \xi)$

In order to make use of conditions (sc1*) and (sc2*), we need to introduce the concept of maximum-in-between values, or mib-values, containing the maximum value between a certain point and the current index. We will keep track of one mib-value for each steep down region in SDASet, containing the maximum value between the end of the steep down region and the current index, and one global mib-value containing the maximum between the end of the last steep (up or down) region found and the current index.

We can now modify the algorithm to keep track of all the mib-

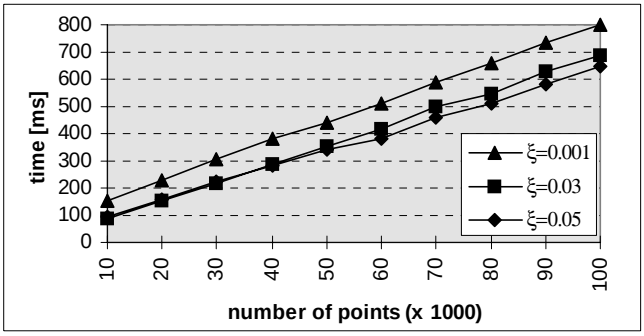


Figure 20. Scale-up of the ξ-clustering algorithm for the 64-*d* color histograms

values and use them to save the expensive operations identified above. The complete algorithm is shown in figure 19. Whenev-

```
SetOfSteepDownAreas = EmptySet
SetOfClusters = EmptySet
index = 0, mib = 0
WHILE(index < n)
  mib = max(mib, r(index))
  IF(start of a steep down area D at index)
    update mib-values and filter SetOfSteepDownAreas(*)
    set D.mib = 0
    add D to the SetOfSteepDownAreas
    index = end of D + 1; mib = r(index)
  ELSE IF(start of steep up area U at index)
    update mib-values and filter SetOfSteepDownAreas
    index = end of U + 1; mib = r(index)
    FOR EACH D in SetOfSteepDownAreas DO
      IF(combination of D and U is valid AND(**)
        satisfies cluster conditions 1, 2, 3a)
        compute [s, e] add cluster to SetOfClusters
  ELSE index = index + 1
RETURN(SetOfClusters)
```

Figure 19. Algorithm ExtractClusters

er we encounter a new steep (up or down) area, we filter all steep down areas from SDASet whose start multiplied by (1-ξ) is smaller than the global mib-value, thus reducing the number of potential clusters significantly and satisfying (sc1*) at the same time (c.f. line (*) in figure 19). In step ② (line (**) in figure 19), we compare the end of the steep up area U multiplied by (1-ξ) to the mib-value of the steep down area D, thus satisfying (sc2*).

What we have gained by using the mib-values to satisfy condition (sc1*) and (sc2*) (implying that condition 3b is satisfied) is that we reduced the number of potential clusters significantly and saved the loop over all points in each remaining potential cluster.

4.3.3 Experimental Evaluation

Figure 20 depicts the runtime of the cluster extraction algorithm for a data set containing 64-dimensional color histograms extracted from TV-snapshots. It shows the scale-up between 10,000 and 100,000 data objects for different ξ values, proving that the algorithm is indeed very fast and linear in the number of data objects. For higher ξ-values the number of clusters increases, resulting in a small (constant) increase in runtime. The parameter ξ can be used to control the steepness of the points a cluster starts with and ends with. Higher ξ-values can be used to find only the most significant clusters, lower ξ-values to find

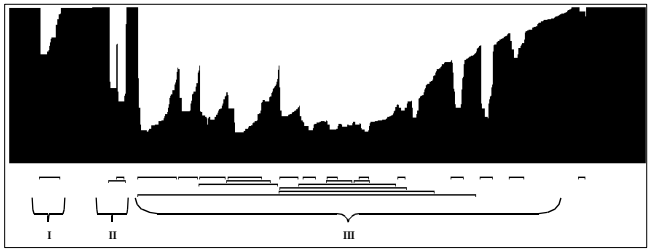


Figure 22. 64-*d* color histograms auto-clustered for ξ=0.02

less significant clusters which means that the choice depends on the intended granularity of the analysis. All experiments were performed on a 180 MHz PentiumPro with 96 MB RAM under Windows NT 4.0. The clustering algorithm was implemented in Java by using Sun's JDK version 1.1.6. Note that Java is interpreted byte-code, not compiled native code.

In section 4.1, we have identified clusters as "dents" in the reachability-plot. Here, we demonstrate that what we call a dent is, in fact, a ξ-cluster by showing synthetic, two-dimensional points as well as high-dimensional real-world examples.

In figure 21, we see an example of three equal size clusters, two of which are very close together, and some noise points. We see that the algorithm successfully identifies this hierarchical structure, i.e. it finds the two clusters and the higher-level cluster containing both of them. It also finds the third cluster, and even identifies an especially dense region within it.

In figure 22, a reachability-plot for 64-dimensional color histograms is shown. The cluster identified in region I contains screen shots from one talk show. The cluster in region II contains stock market data. It is very interesting to see that this cluster actually consists of 2 smaller sub-clusters (both of which the algorithm identifies; the first one, however, is filtered out because it does not contain enough points). These sub-clusters contain the *same* stock-market-data shown on two different TV-stations which (during certain times of the day) broadcast the same program. The only difference between these clusters is the TV-station symbol, which each station overlays in the top-left hand corner of the image. Region III are pictures from a tennis match, the subclusters are different camera angles. Note that the notion of ξ-clusters nicely captures this hierachical structure.

We have seen that it is possible to extract the hierarchical cluster structure from the augmented cluster-ordering generated by OPTICS, both visually and automatically. The algorithm for the automatic extraction is highly efficient and of a very high quality. Once we have the set of points belonging to a cluster, we can easily compute traditional clustering information like representative points (e.g. medoid or mean) or shape descriptions.

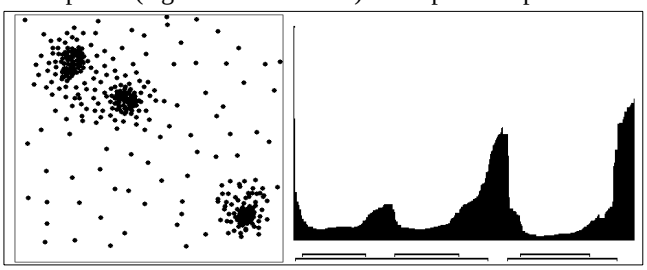


Figure 21. 2-*d* synthetic data set (left), the reachability-plot (right) and ξ=0.09**-clustering (below the reachability-plot)**

5. Conclusions

In this paper, we proposed a cluster analysis method based on the OPTICS algorithm. OPTICS computes an *augmented cluster-ordering* of the database objects. The main advantage of our approach, when compared to the clustering algorithms proposed in the literature, is that we do not limit ourselves to one global parameter setting. Instead, the augmented cluster-ordering contains information which is equivalent to the density-based clusterings corresponding to a broad range of parameter settings and thus is a versatile basis for both automatic and interactive cluster analysis.

We demonstrated how to use it as a stand-alone tool to get insight into the distribution of a data set. Depending on the size of the database, we either represent the cluster-ordering graphically (for small data sets) or use an appropriate visualization technique (for large data sets). Both techniques are suitable for interactively exploring the clustering structure, offering additional insights into the distribution and correlation of the data. We also presented an efficient and effective algorithm to automatically extract not only 'traditional' clustering information but also the intrinsic, hierarchical clustering structure.

There are several opportunities for *future research*. For very high-dimensional spaces, no index structures exist to efficiently support the hypersphere range queries needed by the OPTICS algorithm. Therefore it is infeasible to apply it in its current form to a database containing several million high-dimensional objects. Consequently, the most interesting question is whether we can modify OPTICS so that we can trade-off a limited amount of accuracy for a large gain in efficiency. Incrementally managing a cluster-ordering when updates on the database occur is another interesting challenge. Although there are techniques to update a 'flat' density-based decomposition [EKS+ 98] incrementally, it is not obvious how to extend these ideas to a density-based cluster-ordering of a data set.

6. References

[AGG+ 98] Agrawal R., Gehrke J., Gunopulos D., Raghavan P.: "*Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications*", Proc. ACM SIGMOD'98 Int. Conf. on Management of Data, Seattle, WA, 1998, pp. 94-105.

[AKK 96] Ankerst M., Keim D. A., Kriegel H.-P.: "'*Circle Segments': A Technique for Visually Exploring Large Multidimensional Data Sets*", Proc. Visualization'96, Hot Topic Session, San Francisco, CA, 1996.

[BKK 96] Berchthold S., Keim D., Kriegel H.-P.: "*The X-Tree: An Index Structure for High-Dimensional Data*", 22nd Conf. on Very Large Data Bases, Bombay, India, 1996, pp. 28-39.

[BKSS 90] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *"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, ACM Press, New York, 1990, pp. 322-331.

[CPZ 97] Ciaccia P., Patella M., Zezula P.: *"M-tree: An Efficient Access Method for Similarity Search in Metric Spaces"*, Proc. 23rd Int. Conf. on Very Large Data Bases, Athens, Greece, 1997, pp. 426-435.

[EKSX 96] Ester M., Kriegel H.-P., Sander J., Xu X.: *"A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise"*, Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, 1996, pp. 226-231.

[EKS+ 98] Ester M., Kriegel H.-P., Sander J., Wimmer M., Xu X.: "*Incremental Clustering for Mining in a Data Warehousing Environment*", Proc. 24th Int. Conf. on Very Large Data Bases, New York, NY, 1998, pp. 323-333.

[EKX 95] Ester M., Kriegel H.-P., Xu X.: *"Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification*", Proc. 4th Int. Symp. on Large Spatial Databases, Portland, ME, 1995, in: Lecture Notes in Computer Science, Vol. 951, Springer, 1995, pp. 67-82.

[GM 85] Grossman A., Morlet J.: "*Decomposition of functions into wavelets of constant shapes and related transforms*". Mathematics and Physics: Lectures on Recent Results, World Scientific, Singapore, 1985.

[GRS 98] Guha S., Rastogi R., Shim K.: *"CURE: An Efficient Clustering Algorithms for Large Databases"*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Seattle, WA, 1998, pp. 73-84.

[HK 98] Hinneburg A., Keim D.: *"An Efficient Approach to Clustering in Large Multimedia Databases with Noise"*, Proc. 4th Int. Conf. on Knowledge Discovery & Data Mining, New York City, NY, 1998.

[HT 93] Hattori K., Torii Y.: "*Effective algorithms for the nearest neighbor method in the clustering problem*", Pattern Recognition, 1993, Vol. 26, No. 5, pp. 741-746.

[Hua 97] Huang Z.: *"A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining*", Proc. SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tech. Report 97-07, UBC, Dept. of CS, 1997.

[JD 88] Jain A. K., Dubes R. C.: "*Algorithms for Clustering Data*," Prentice-Hall, Inc., 1988.

[Kei 96a] Keim D. A.: "*Pixel-oriented Database Visualizations*", in: SIGMOD RECORD, Special Issue on Information Visualization, Dezember 1996.

[Kei 96b] Keim D. A.: "*Databases and Visualization*", Proc. Tutorial ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada, 1996, p. 543.

[KN 96] Knorr E. M., Ng R.T.: "*Finding Aggregate Proximity Relationships and Commonalities in Spatial Data Mining*," IEEE Trans. on Knowledge and Data Engineering, Vol. 8, No. 6, December 1996, pp. 884-897.

[KR 90] Kaufman L., Rousseeuw P. J.: *"Finding Groups in Data: An Introduction to Cluster Analysis"*, John Wiley & Sons, 1990.

[Mac 67] MacQueen, J.: *"Some Methods for Classification and Analysis of Multivariate Observations"*, 5th Berkeley Symp. Math. Statist. Prob., Vol. 1, pp. 281-297.

[NH 94] Ng R. T., Han J.: *"Efficient and Effective Clustering Methods for Spatial Data Mining"*, Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, Morgan Kaufmann Publishers, San Francisco, CA, 1994, pp. 144-155.

[PTVF 92] Press W. H.,Teukolsky S. A., Vetterling W. T., Flannery B. P.: "*Numerical Recipes in C*", 2nd ed., Cambridge University Press, 1992.

[Ric 83] Richards A. J.: "*Remote Sensing Digital Image Analysis. An Introduction*", 1983, Berlin, Springer Verlag.

[Sch 96] Schikuta E.: "*Grid clustering: An efficient hierarchical clustering method for very large data sets*". Proc. 13th Int. Conf. on Pattern Recognition, Vol 2, 1996, pp. 101-105.

[SE 97] Schikuta E., Erhart M.: "*The bang-clustering system: Grid-based data analysis*". Proc. Sec. Int. Symp. IDA-97, Vol. 1280 LNCS, London, UK, Springer-Verlag, 1997.

[SCZ 98] Sheikholeslami G., Chatterjee S., Zhang A.: "*WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases"*, Proc. 24th Int. Conf. on Very Large Data Bases, New York, NY, 1998, pp. 428 - 439.

[Sib 73] Sibson R.: *"SLINK: an optimally efficient algorithm for the single-link cluster method"*.The Comp. Journal, Vol. 16, No. 1, 1973, pp. 30-34.

[ZRL 96] Zhang T., Ramakrishnan R., Linvy M.: *"BIRCH: An Efficient Data Clustering Method for Very Large Databases"*. Proc. ACM SIGMOD Int. Conf. on Management of Data, ACM Press, New York, 1996, pp.103-114.