In: Petrushin V. A., Khan L. (eds.): Multimedia Data Mining and Knowledge Discovery, pp. 94–115, 2006. © Springer-Verlag London Ltd 2006

Density-Based Data Analysis and Similarity Search

Hans-Peter Kriegel, Stefan Brecheisen, Peer Kröger, Martin Pfeifle, Matthias Schubert, and Arthur Zimek

Department "Institute for Informatics", University of Munich
Oettingenstr. 67, 80538 Munich, Germany
{kriegel,brecheis,kroegerp,pfeifle,schubert,zimek}@dbs.ifi.lmu.de

Abstract. Similarity search in database systems is becoming an increasingly important task in modern application domains such as multimedia, molecular biology, medical imaging, computer aided engineering, marketing and purchasing assistance as well as many others. Furthermore, the feature transformations and distance measures used in similarity search build the foundation of sophisticated data analysis and mining techniques. In this chapter, we show how visualizing cluster hierarchies describing a database of objects can aid the user in the time consuming task to find similar objects and discover interesting patterns. We present related work and explain its shortcomings which led to the development of our new methods. Based on reachability plots, we introduce methods for visually exploring a data set in multiple representations and comparing multiple similarity models. Furthermore, we present a new method for automatically extracting cluster hierarchies from a given reachability plot which allows a user to browse the database for similarity search. We integrated our new method in a prototype which serves two purposes, namely visual data analysis and a new way of object retrieval called navigational similarity search.

1 Introduction

In recent years, an increasing number of database applications has emerged for which efficient and effective similarity search and data analysis is substantial. Important application areas are multimedia, medical imaging, molecular biology, computer aided engineering, marketing and purchasing assistance, etc. [1–8]. In these applications, there usually exist various feature representations and similarity models that can be used to retrieve similar data objects or derive interesting patterns from a given database. Hierarchical clustering was shown to be effective for evaluating similarity models [9, 10]. Especially, the reachability plot generated by *OPTICS* [11] is suitable for assessing the quality of similarity models and compare the meaning of different representations to each other. To further extract patterns and allow new methods of similarity search, cluster extraction algorithms can extract cluster hierarchies representing a concrete categorization of all data objects.

In this chapter, we present methods that employ hierarchical clustering and visual data mining techniques to fulfill various tasks for comparing and evaluating distance models and feature extractions methods. Furthermore, we introduce an algorithm for automatically detecting hierarchical clusters and use this hierarchy for navigational similarity search. In ordinary similarity search systems, a user is usually obliged to provide an example query object to which the retrieved database objects should be similar. In contrast, navigational similarity search allows a user to browse the database using the extracted cluster hierarchy to navigate between groups of similar objects. In order to evaluate our ideas, we developed a research prototype. Its basic functionality is to display the cluster structure of a given data set and to allow navigational similarity search. Furthermore, we integrated two components called VICO and CLUSS. *VICO* (*VI*sually *C*onnected *O*bject Orderings) is a tool for evaluating and comparing feature representations and similarity models. The idea of VICO is to compare multiple reachabilty plots of one and the same data set. *CLUSS* (*CLU*ster Hierarchies for *S*imilarity *S*earch) is an alternative hierarchical clustering algorithm which was especially developed to generate cluster hierarchies being well suited for navigational similarity search.

To sum up, the main topics of this chapter are as follows:

- We describe methods for evaluating data representations and similarity models. Furthermore, we sketch possibilities to visually compare these representations and models.
- We present an alternative approach to the retrieval of similar objects, called navigational similarity search. Unlike conventional similarity queries, the user does not need to provide a query object but can interactively browse the data set.
- We introduce a new cluster recognition algorithm for the reachability plots generated by OPTICS. This algorithm generalizes all the other known cluster recognition algorithms for reachability plots. Although our new algorithm does not need a sophisticated and extensive parameter setting, it outperforms the other cluster recognition algorithms w.r.t. quality and number of recognized clusters and subclusters. The derived cluster hierarchy enables us to employ OPTICS for navigational similarity search.
- We introduce an alternative method for generating cluster hierarchies which provides a more intuitive access to the database for navigational similarity search. The advantage of this method is that each of the derived clusters is described by a set of well selected representative objects giving the user a better impression of the objects contained in the cluster.

The remainder of the chapter is organized as follows: We briefly introduce the clustering algorithm OPTICS in Section 2. In Section 3, we present the main application areas of our new methods for data analysis and navigational similarity search. Section 4 introduces a novel algorithm for extracting cluster hierarchies, together with an experimental evaluation. An alternative way to derive cluster hierarchies for navigational similarity search is presented in Section 5. The chapter concludes in Section 6 with a short summary.

2 Hierarchical Clustering

In the following, we will briefly review the hierarchical density-based clustering algorithm OPTICS which is the foundation of the majority of the methods described in this chapter.

The key idea of density-based clustering is that for each object o of a cluster the neighborhood $\mathcal{N}_\varepsilon(o)$ of a given radius ε has to contain at least a minimum number *MinPts* of objects. Using the density-based hierarchical clustering algorithm OPTICS yields several advantages due to the following reasons:

- OPTICS is – in contrast to most other algorithms – relatively insensitive to its two input parameters, ε and *MinPts*. The authors in [11] state that the input parameters just have to be large enough to produce good results.
- OPTICS is a hierarchical clustering method which yields more information about the cluster structure than a method that computes a flat partitioning of the data (e.g. k-means [12]).
- There exists a very efficient variant of the OPTICS algorithm which is based on a sophisticated data compression technique called "Data Bubbles" [13], where we have to trade only very little quality of the clustering result for a great increase in performance.
- There exists an efficient incremental version [14] of the OPTICS algorithm.

OPTICS emerges from the algorithm DBSCAN [15] which computes a flat partitioning of the data. The clustering notion underlying DBSCAN is that of density-connected sets (cf. [15] for more details). It is assumed that there is a metric distance function on the objects in the database (e.g. one of the L_p-norms for a database of feature vectors).

In contrast to DBSCAN, OPTICS does not assign cluster memberships but computes an *ordering* in which the objects are processed and additionally generates the information which would be used by an extended DBSCAN algorithm to assign cluster memberships. This information consists of only two values for each object, the *core-distance* and the *reachability-distance*.

Definition 1 (core-distance). *Let $o \in DB$, $MinPts \in \mathbb{N}$, $\varepsilon \in \mathbb{R}$, and MinPts-dist($o$) be the distance from o to its MinPts-nearest neighbor. The* core-distance *of o w.r.t. ε and MinPts is defined as follows:*

$$\textit{Core-Dist}(o) := \begin{cases} \infty & \textit{if} \quad |\mathcal{N}_\varepsilon(o)| < \textit{MinPts} \\ \textit{MinPts-dist(o)} & \textit{otherwise.} \end{cases}$$

Definition 2 (reachability-distance). *Let $o \in DB$, $MinPts \in \mathbb{N}$ and $\varepsilon \in \mathbb{R}$. The* reachability distance *of o w.r.t. ε and MinPts from an object $p \in DB$ is defined as follows:*

$$\textit{Reach-Dist}(p, o) := \max\left(\textit{Core-Dist}(p), \textit{distance}(p, o)\right).$$

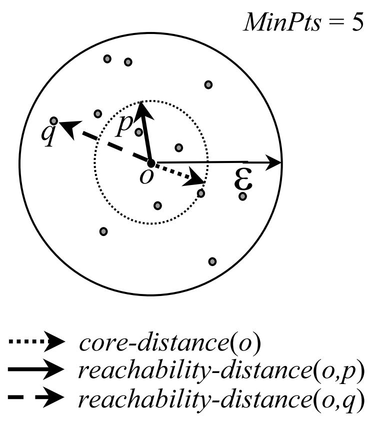


Fig. 1. Illustration of core-level and reachability-distance.

Figure 1 illustrates both concepts: The reachability-distance of p from o equals to the core-distance of o and the reachability-distance of q from o equals to the distance between q and o.

The original output of OPTICS is an ordering of the objects, a so called *cluster ordering*:

Definition 3 (cluster ordering). *Let MinPts* $\in \mathbb{N}$, $\varepsilon \in \mathbb{R}$, *and* CO *be a totally ordered permutation of the database objects. Each* $o \in D$ *has additional attributes* $o.P$, $o.C$ *and* $o.R$, *where* $o.P \in \{1, \ldots, |CO|\}$ *symbolizes the position of* o *in* CO. *We call* CO *a* cluster ordering *w.r.t.* ε *and MinPts if the following three conditions hold:*

(1) $\forall p \in CO : p.C = \mathit{Core\text{-}Dist}(p)$
(2) $\forall o, x, y \in CO :$
$o.P < x.P \wedge x.P < y.P \Rightarrow \mathit{Reach\text{-}Dist}(o, x) \leq \mathit{Reach\text{-}Dist}(o, y)$
(3) $\forall p, o \in CO : R(p) = \min\{\mathit{Reach\text{-}Dist}(o, p) \mid o.P < p.P\}$, *where* $\min \emptyset = \infty$.

Intuitively, Condition (2) states that the order is built on selecting at each position i in CO that object o having the minimum reachability to any object before i. $o.C$ symbolizes the core-distance of an object o in CO whereas $o.R$ is the reachability-distance assigned to object o during the generation of CO. We call $o.R$ the *reachablity* of object o throughout the chapter. Note that $o.R$ is only well-defined in the context of a cluster ordering.

The cluster structure can be visualized through so called reachability plots which are 2D plots generated as follows: the clustered objects are ordered along the x-axis according to the cluster ordering computed by OPTICS and the reachabilities assigned to each object are plotted along the abscissa. An example reachability plot is depicted in Figure 2. Valleys in this plot indicate clusters: objects having a small reachability value are closer and thus more similar to their predecessor objects than objects having a higher reachability value.

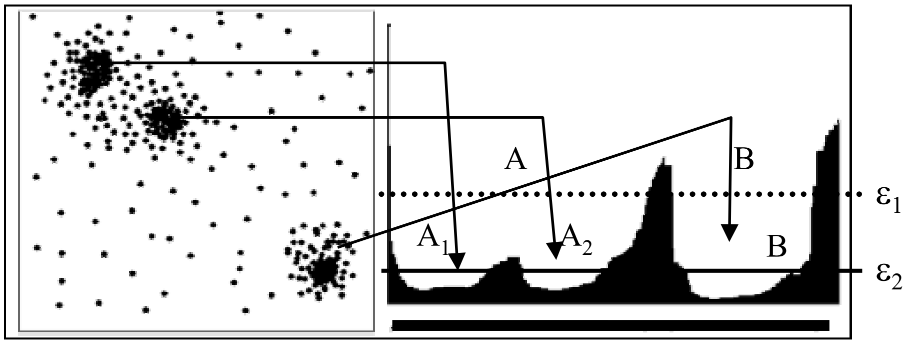


Fig. 2. Reachability plot (right) computed by OPTICS for a sample 2-D data set (left).

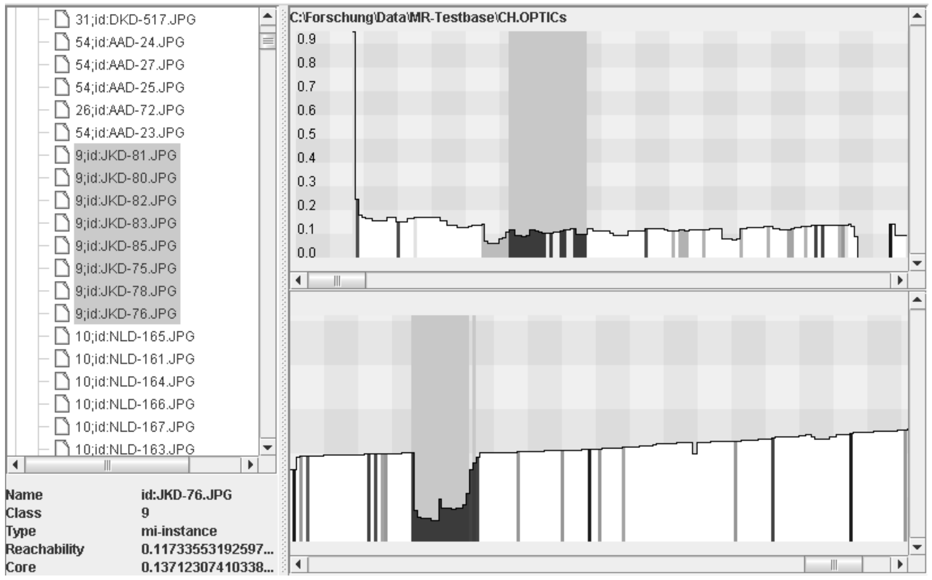
The reachability plot generated by OPTICS can be cut at any level ε_{cut} parallel to the abscissa. It represents the density-based clusters according to the density threshold ε_{cut}: A consecutive subsequence of objects having a smaller reachability value than ε_{cut} belongs to the same cluster. An example is presented in Figure 2: For a cut at the level ε_1 we find two clusters denoted as A and B. Compared to this clustering, a cut at level ε_2 would yield three clusters. The cluster A is split into two smaller clusters denoted by A_1 and A_2 and cluster B decreased its size. Usually, for evaluation purposes, a good value for ε_{cut} would yield as many clusters as possible.

3 Application Ranges

The introduced methods combine techniques from hierarchical clustering and data visualization for two main purposes, data analysis and similarity search. In the following, we will propose several applications in both areas for which our introduced methods are very useful.

3.1 Data Analysis

The data analysis part of our prototype is called VICO. It allows a user to cluster data objects in varying representations and using varying similarity models. The main purpose of VICO is to compare different feature spaces that describe the same set of data. For this comparison, VICO relies on the interactive visual exploration of reachability plots. Therefore, VICO displays any available view on a set of data objects as adjacent reachability plots and allows comparisons between the local neighborhoods of each object. Figure 3 displays the main window of VICO. The left side of the window contains a so-called tree control that contains a subtree for each view of the data set. In each subtree, the keys are ordered w.r.t. the cluster order of the corresponding view. The tree control allows a user to directly search for individual data objects. In addition to the


31;id:DKD-517.JPG
54;id:AAD-24.JPG
54;id:AAD-27.JPG
54;id:AAD-25.JPG
26;id:AAD-72.JPG
54;id:AAD-23.JPG
9;id:JKD-81.JPG
9;id:JKD-80.JPG
9;id:JKD-82.JPG
9;id:JKD-83.JPG
9;id:JKD-85.JPG
9;id:JKD-75.JPG
9;id:JKD-78.JPG
9;id:JKD-76.JPG
10;id:NLD-165.JPG
10;id:NLD-161.JPG
10;id:NLD-164.JPG
10;id:NLD-166.JPG
10;id:NLD-167.JPG
10;id:NLD-163.JPG

Name	id:JKD-76.JPG
Class	9
Type	mi-instance
Reachability	0.11733553192597...
Core	0.13712307410338...

C:\Forschung\Data\MR-Testbase\CH.OPTICs

Fig. 3. VICO displaying OPTICS plots of multi-represented data.

object keys displayed in the tree control, VICO displays the reachability plot of each view of the data set.

Since valleys in the reachability plot represent clusters in the underlying representation, the user gets an instant impression of the richness of the cluster structure in each representation. However, to explore the relationships between the representations, we need to find out whether objects that are clustered in one representation are also similar in the other representation. To achieve this type of comparison, VICO allows the user to select any data object in any reachability plot or the tree control. By selecting a set of objects in one view, the objects are highlighted in any other view as well. For example, if the user looks at the reachability plot in one representation and selects a cluster within this plot, the corresponding object keys are highlighted in the tree control and identify the objects that are contained in the cluster. Let us note that it is possible to visualize the selected objects as well, as long as there is a viewable object representation. In addition to the information about which objects are clustered together, the set of objects is highlighted in the reachability plots of the other representations as well. Thus, we can easily decide whether the objects in one representation are placed within a cluster in another representation as well or if they are spread among different clusters or are part of the noise. If there exist contradicting reachability plots for the same set of data objects, it is interesting to know which of these representations is closer to the desired notion of similarity. Thus, VICO allows the user to label data objects w.r.t. predefined class values. The different class values for the objects are displayed by different colors in the reachability plot. Thus, a reachability plot of a data space that matches the user's notion of similarity should display clusters containing objects of the same color. Figure 3 displays a comparison of two feature spaces for an image data set. Each image is labelled w.r.t. the displayed motive.

Another feature of VICO is the ability to handle multi-instance objects. In a multi-instance representation, one data object is given by a set of separated feature objects. An example are CAD parts that can be decomposed to a set of spatial primitives, which can be represented by a single feature vector. This way, the complete CAD part is represented by a set of feature vectors, which can be compared by a variety of distance functions. To find out which instances are responsible for clusters of multi-instance objects, VICO allows us to cluster the instances without considering the multi-instance object they belong to. Comparing this instance plot to the plot derived on the complete multi-instance objects allows us to analyze which instance clusters are typical for the clusters on the complete multi-instance object. Thus, for multi-instance settings, VICO highlights all instances belonging to some selected multi-instance object. To conclude, VICO allows even non-expert users to evaluate similarity models, directly compares different similarity models to each other, and helps exploring the connection between multi-instance distance functions and the underlying distance metrics in the feature space of instances.

3.2 Navigational Similarity Search

For similarity search, the idea of our system is to provide navigational access to a database. This way, it is possible to browse a database of objects in an explorer-like application, instead of posing separated similarity queries. A main problem of these similarity queries is that a user always has to provide a query object to which the retrieved objects should be as similar as possible. However, in many application scenarios finding a query object is not easy. For example, an engineer querying a CAD database would have to specify the 3D shape of a CAD part before finding similar parts. Sketching more complicated parts might cause a considerable effort. Thus, similarity queries are often quite time-consuming. The alternative idea of navigational similarity search offers an easier way to retrieve the desired objects. The idea is to use an extracted hierarchy of clusters as a navigation tree. The root represents the complete data set. Each node in the tree represents a subcluster consisting of a subset of data objects for which the elements are more similar to each other than in the father cluster. The more specialized a cluster is the more similar its members are to each other. To describe the members of a cluster one or more representative objects are displayed. Browsing starts at a general level. Afterwards, the user follows the path in the cluster hierarchy for which the displayed representatives resemble the image in the user's mind in a best possible way. The browsing terminates when the user reaches a cluster for which the contained objects match the user's expectation. Another advantage of this approach is that the cluster usually contains the complete set of similar objects. For similarity queries, the number of retrieved results can either be specified in the case of kNN queries or is implicitly controlled by a specified query range. However, both methods usually do not retrieve all objects that could be considered as similar.

4 Cluster Recognition for OPTICS

In this section, we address the task of automatically extracting clusters from a reachability plot. Enhancing the resulting cluster hierarchy with representative objects for each extracted cluster, allows us to use the result of OPTICS for navigational similarity search. After a brief discussion of recent work in that area, we propose a new approach for hierarchical cluster recognition based on reachability plots called *Gradient Clustering*.

4.1 Recent Work

To the best of our knowledge, there are only two methods for automatic cluster extraction from hierarchical representations such as reachability plots or dendrograms – both are also based on reachability plots. Since clusters are represented as valleys (or dents) in the reachability plot, the task of automatic cluster extraction is to identify significant valleys.

The first approach proposed in [11] called ξ-clustering is based on the steepness of the valleys in the reachability plot. The steepness is defined by means of an input parameter ξ. The method suffers from the fact that this parameter is difficult to understand and hard to determine. Rather small variations of the value ξ often lead to drastic changes of the resulting clustering hierarchy. As a consequence, this method is unsuitable for the purpose of automatic cluster extraction.

The second approach was proposed by Sander et al. [16]. The authors describe an algorithm called Tree Clustering that automatically extracts a hierarchical clustering from a reachability plot and computes a cluster tree. It is based on the idea that *significant* local maxima in the reachability plot separate clusters. Two parameters are introduced to decide whether a local maximum is significant: The first parameter specifies the minimum cluster size, i.e. how many objects must be located between two significant local maxima. The second parameter specifies the ratio between the reachability of a significant local maximum m and the average reachabilities of the regions to the left and to the right of m. The authors in [16] propose to set the minimum cluster size to 0.5% of the data set size and the second parameter to 0.75. They empirically show, that this default setting approximately represents the requirements of a typical user.

Although the second method is rather suitable for automatic cluster extraction from reachability plots, it has one major drawback. Many real-world data sets consist of narrowing clusters, i.e. clusters each consisting of exactly one smaller subcluster (cf. Figure 4).

Since the Tree Clustering algorithm runs through a list of all local maxima (sorted in descending order of reachability) and decides at each local maximum m, whether m is significant to split the objects to the left of m and to the right of

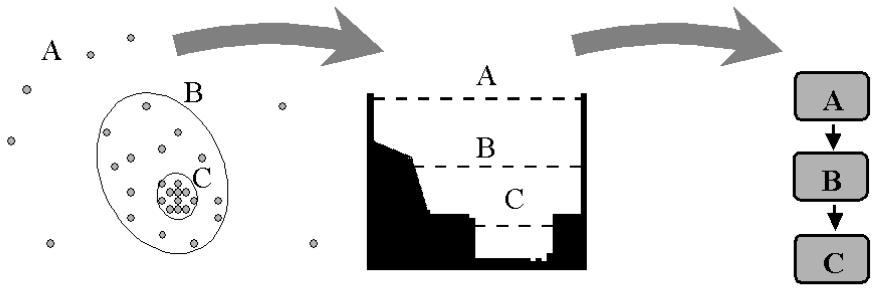


Fig. 4. Sample narrowing clusters: data space (left); reachability plot (middle); cluster hierarchy (right)

m into two clusters, the algorithm cannot detect such narrowing clusters. These clusters cannot be split by a significant maximum. Figure 4 illustrates this fact. The narrowing cluster A consists of one cluster B which is itself narrowing consisting of one cluster C (the clusters are indicated by dashed lines). The Tree Clustering algorithm will only find cluster A since there are no local maxima to split clusters B and C. The ξ-clustering will detect only one of the clusters A, B or C depending on the parameter ξ but also fails to detect the cluster hierarchy.

A new cluster recognition algorithm should meet the following requirements:

- It should detect all kinds of subclusters, including narrowing subclusters.
- It should create a clustering structure which is close to the one which an experienced user would manually extract from a given reachability plot.
- It should allow an easy integration into the OPTICS algorithm. We do not want to apply an additional cluster recognition step after the OPTICS run is completed. In contrast, the hierarchical clustering structure should be created on-the-fly during the OPTICS run without causing any noteworthy additional cost.
- It should be integrable into the incremental version of OPTICS [14], as most of the discussed application ranges benefit from such an incremental version.

4.2 Gradient Clustering

In this section, we introduce our new *Gradient Clustering* algorithm which fulfills all of the above mentioned requirements. The idea behind the new cluster extraction algorithm is based on the concept of *inflexion points*. During the OPTICS run, we decide for each point added to the result set, i.e. the reachability plot, whether it is an inflexion point or not. If it is an inflexion point we might be at the start or at the end of a new subcluster. We store the possible starting points of the subclusters in a list, called *startPts*. This stack consists of pairs

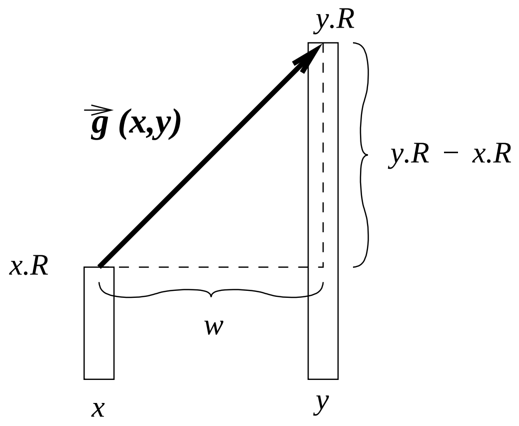


Fig. 5. Gradient vector $\boldsymbol{g}(x, y)$ of two objects x and y adjacent in the cluster ordering.

$(o.P, o.R)$. The Gradient Clustering algorithm can easily be intergrated into OPTICS and is described in full detail, after we have formally introduced the new concept of inflexion points.

In the following, we assume that CO is a cluster ordering as defined in Definition 3. We call two objects $o_1, o_2 \in CO$ *adjacent* in CO if $o_2.P = o_1.P + 1$. Let us recall, that $o.R$ is the reachability of $o \in CO$ assigned by OPTICS while generating CO. For any two objects $o_1, o_2 \in CO$ adjacent in the cluster ordering, we can determine the gradient of the reachability values $o_1.R$ and $o_2.R$. The gradient can easily be modelled as a 2D vector where the y-axis measures the reachability values ($o_1.R$ and $o_2.R$) in the ordering, and the x-axis represent the ordering of the objects. If we assume that each object in the ordering is separated by width w, the gradient of o_1 and o_2 is the vector

$$\boldsymbol{g}(o_1, o_2) = \begin{pmatrix} w \\ o_2.R - o_1.R \end{pmatrix}.$$

An example for a gradient vector of two objects x and y adjacent in a cluster ordering is depicted in Figure 5.

Intuitively, an inflexion point should be an object in the cluster ordering where the gradient of the reachabilities changes significantly. This significant change indicates a starting or an end point of a cluster.

Let $x, y, z \in CO$ be adjacent, i.e.

$$x.P + 1 = y.P = z.P - 1.$$

We can now measure the difference between the gradient vectors $\boldsymbol{g}(x, y)$ and $\boldsymbol{g}(y, z)$ by computing the cosine of the angle between the vectors $\boldsymbol{g}(x, y)$ and $\boldsymbol{g}(z, y)$ $(= -\boldsymbol{g}(y, z))$. The cosine of this angle is equal to -1 if the angle is $180°$, i.e. the vectors have the same direction. On the other hand, if the gradient vectors differ a lot, the angle between them will be clearly smaller than $180°$ and thus

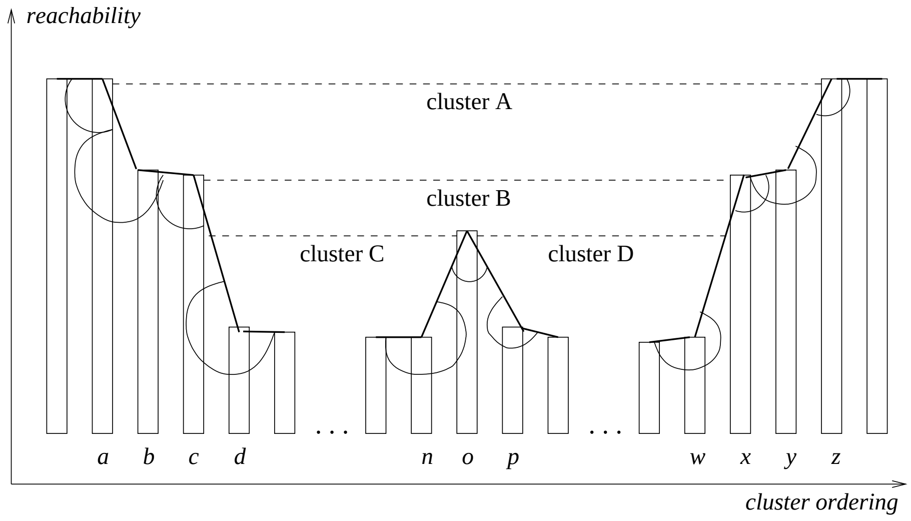


Fig. 6. Illustration of inflexion points measuring the angle between the gradient vectors of objects adjacent in the ordering.

the cosine will be significantly greater than -1. This observation motivates the concepts of inflexion index and inflexion points:

Definition 4 (inflexion index). *Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO. The* inflexion index *of y, denoted by $II(y)$, is defined as the cosine of the angle between the gradient vector of x, y ($\boldsymbol{g}(x,y)$) and the gradient vector of z, y ($\boldsymbol{g}(z,y)$), formally:*

$$II(y) = \cos\varphi_{(\boldsymbol{g}(x,y),\boldsymbol{g}(z,y))} = \frac{-w^2 + (y.R - x.R)(y.R - z.R)}{\|\boldsymbol{g}(x,y)\|\,\|\boldsymbol{g}(z,y)\|},$$

where $\|\boldsymbol{v}\| := \sqrt{v_1^2 + v_2^2}$ is the length of the vector $\boldsymbol{v}$.

Definition 5 (inflexion point). *Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO and let $t \in \mathbb{R}$. Object y is an* inflexion point *iff*

$$II(y) > t.$$

The concept of inflexion points is suitable to detect objects in CO which are interesting for extracting clusters.

Definition 6 (gradient determinant). *Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO. The* gradient determinant *of the gradients $\boldsymbol{g}(x,y)$ and $\boldsymbol{g}(y,z)$ is defined as*

$$gd(\boldsymbol{g}(x,y), \boldsymbol{g}(z,y)) := \begin{vmatrix} w & -w \\ x.R - y.R & z.R - y.R \end{vmatrix}$$

```
algorithm gradient_clustering(ClusterOrdering CO, Integer MinPts, Real t)
  startPts := emptyStack;
  setOfClusters := emptySet;
  currCluster := emptySet;
  o := CO.getFirst();                              // first object is a starting point
  startPts.push(o);

  WHILE o.hasNext() DO                             // for all remaining objects
    o := o.next;
    IF o.hasNext() THEN
      IF II(o) > t THEN                          // inflexion point
        IF gd(o) > 0 THEN
          IF currCluster.size() >= MinPts THEN
             setOfClusters.add(currCluster);
          ENDIF
          currCluster := emptySet;
          IF startPts.top().R <= o.R THEN
            startPts.pop();
          ENDIF
          WHILE startPts.top().R < o.R DO
            setOfClusters.add(set of objects from startPts.top() to last end point);
            startPts.pop();
          ENDDO
          setOfClusters.add(set of objects from startPts.top() to last end point);
          IF o.next.R < o.R THEN             // o is a starting point
             startPts.push(o);
          ENDIF
        ELSE
          IF o.next.R > o.R THEN             // o is an end point
            currCluster := set of objects from startPts.top() to o;
          ENDIF
        ENDIF
      ENDIF
    ELSE                                          // add clusters at end of plot
      WHILE NOT startPts.isEmpty() DO
        currCluster := set of objects from startPts.top() to o;
        IF (startPts.top().R > o.R) AND (currCluster.size() >= MinPts) THEN
          setOfClusters.add(currCluster);
        ENDIF
        startPts.pop();
      ENDDO
    ENDIF
  ENDDO

  RETURN setOfClusters;
END. // gradient_clustering
```

Fig. 7. Pseudo code of the Gradient Clustering algorithm.

If x, y, z are clear from the context, we use the short form $gd(y)$ for the gradient determinant $gd(\boldsymbol{g}(x,y), \boldsymbol{g}(y,z))$.

The sign of $gd(y)$ indicates whether $y \in CO$ is a starting point or an end point of a cluster. In fact, we can distinguish the following two cases which are visualized in Figure 6:

- $II(y) > t$ and $gd(y) > 0$:
  Object y is either a starting point of a cluster (e.g. object a in Figure 6) or the first object outside of a cluster (e.g. object z in Figure 6).
- $II(y) > t$ and $gd(y) < 0$:
  Object y is either an end point of a cluster (e.g. object n in Figure 6) or the second object inside a cluster (e.g. object b in Figure 6).

Let us note that a local maximum $m \in CO$ which is the cluster separation point in [16] is a special form of the first case (i.e. $II(m) > t$ and $gd(m) > 0$).

The threshold t is independent from the absolut reachability values of the objects in CO. The influence of t is also very comprehensible because if we know which values for the angles between gradients are interesting, we can easily compute t. For example, if we are interested in angles $< 120°$ and $> 240°$ we set $t = \cos 120° = -0.5$.

Obviously, the Gradient Clustering algorithm is able to extract narrowing clusters. Experimental comparisons with the methods in [16] and [11] are presented in Section 4.3.

The pseudo code of the Gradient Clustering algorithm is depicted in Figure 7, which works like this. Initially, the first object of the cluster ordering CO is pushed to the stack of starting points *startPts*. Whenever a new starting point is found, it is pushed to the stack. If the current object is an end point, a new cluster is created containing all objects between the starting point on top of the stack and the current end point. Starting points are removed from the stack if their reachablity is lower than the reachability of the current object. Clusters are created as described above for all removed starting points as well as for the starting point which remains in the stack. The input parameter *MinPts* determines the minimum cluster size and the parameter t was discussed above. Finally the parameter w influences the gradient vectors and proportionally depends on the reachability values of the objects in CO.

After extracting a meaningful hierarchy of clusters from the reachability plot of a given data set, we still need to enhance the found clustering with suitable representations. For this purpose, we can display the medoid of each cluster, i.e. the object having the minimal average distance to all the other objects in the cluster.

4.3 Evaluation

Automatic cluster recognition is very desirable when analyzing large sets of data. In the following, we will first evaluate the quality and then the efficiency of the three cluster recognition algorithms using two real-world test data sets. The first data set contains approximately 200 CAD objects from a German car manufacturer, and the second one is a sample of the Protein Databank [17] containing approximately 5000 protein structures. We tested on a workstation featuring a 1.7 GHz CPU and 2 GB RAM.

Effectivity Both the car and the protein data set exhibit the commonly seen quality of unpronounced but nevertheless to the observer clearly visible clusters. The corresponding reachability plots of the two data sets are depicted in Figure 8.

Figure 8c shows that the Tree Clustering algorithm does not find any clusters at all in the car data set, with the suggested default ratio parameter of 75% [16]. In order to detect clusters in the car data set, we had to adjust the ratio

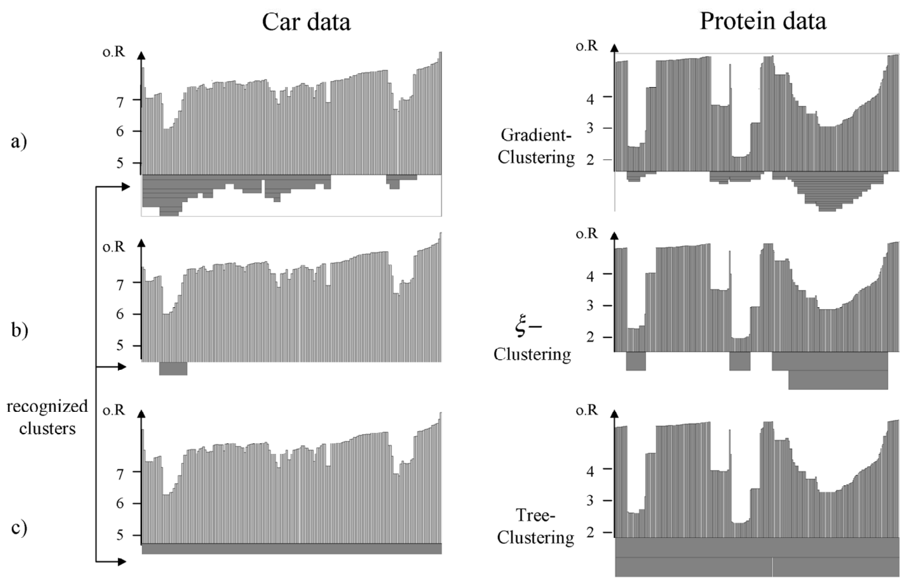


Fig. 8. Sample cluster of car parts. a) Gradient Clustering, b) ξ-Clustering, c) Tree Clustering.

parameter to 95%. In this case Tree Clustering detected some clusters but missed some other important clusters and did not detect any cluster hierarchies at all. If we have rather high reachability values, e.g. values between 5 and 7 as in Figure 8 for the car data set, the ratio parameter for the Tree Clustering algorithm should be set higher than for smaller values. In the case of the protein data set we detected three clusters with the default parameter setting, but again missed out on some important clusters. Generally, in cases where a reachability graph consists of rather high reachability values or does not present spikes at all, but clusters are formed by smooth troughs in the waveform, this cluster recognition algorithm is unsuitable. Furthermore, it is inherently unable to detect narrowing clusters where a cluster has one subcluster of increased density (cf. Figure 4).

On the other hand, the ξ-clustering approach successfully recognizes some clusters while also missing out on significant subclusters (cf. Figure 8b). This algorithm has some trouble recognizing cluster structures with a significant differential of “steepness”. For instance, in Figure 4 it does not detect the narrowing cluster B inside of cluster A because it tries to create steep down-areas containing as many points as possible. Thus, it will merge the two steep edges if their steepness exceeds the threshold ξ. On the other hand, it is able to detect cluster C within cluster A.

Finally, we look at our new Gradient Clustering algorithm. Figure 8a shows that the recognized cluster structure is close to the intuitive one, which an experienced user would manually derive. Clusters which are clearly distinguishable and contain more than *MinPts* elements are detected by this algorithm. Not

Table 1. CPU time for cluster recognition.

	Car data (200 parts)	Protein data (5,000 molecules)
ξ-clustering	0.221 s	5.057 s
Tree Clustering	0.060 s	1.932 s
Gradient Clustering	0.310 s	3.565 s

only does it detect a lot of clusters, but it also detects a lot of meaningful cluster hierarchies, consisting of narrowing subclusters.

To sum up, in all our tests the Gradient Clustering algorithm detected much more clusters than the other two approaches, without producing any redundant and unnecessary cluster information.

Efficiency In all tests, we first created the reachability plots and then applied the algorithms for cluster recognition and representation. Let us note that we could also have integrated the Gradient Clustering into the OPTICS run without causing any noteworthy overhead.

The overall runtimes for the three different cluster recognition algorithms are depicted in Table 1. Our new Gradient Clustering algorithm does not only produce the most meaningful results, but also in sufficiently short time. This is due to its runtime complexity of $O(n)$.

It theoretically and empirically turned out, that the Gradient Clustering algorithm seems to be more practical than recent work for automatic cluster extraction from hierarchical cluster representations.

5 Extracting Cluster Hierarchies for Similarity Search

5.1 Motivation

So far, our prototype works fine by computing, extracting, and visualizing the hierarchical density-based cluster structure of a data set. The density-based cluster model has been chosen due to several criteria. One very important aspect among these criteria is its effectivity in finding clusters of different size and shape. However, this clustering notion needs not to be the best cluster model for navigational similarity search. For this application, the use of OPTICS may have two limitations. In this section, we will discuss these limitations and a possible solution for them.

The first limitation of using the density-based clustering notion is that OPTICS may place two objects that are rather similar, i.e. near in the feature space, into two separate clusters. As a consequence, these two objects may be displayed in completely different subtrees of the cluster hierarchy, i.e. the relationship between these two points in the cluster hierarchy is rather weak. This problem is visualized in Figure 9: object A is obviously much more similar to object B than to object C. However, since A and C belong to the same cluster, both objects will be considered as similar by OPTICS. A and C will be placed in a similar

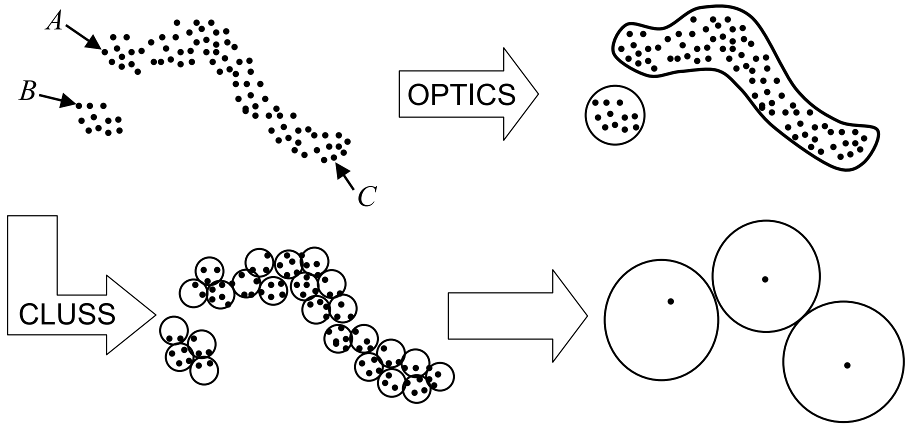


Fig. 9. A new cluster model for navigational similarity search.

subtree of the cluster hierarchy, whereas B may end up in a completely different subtree. This is against the intuitive notion of similarity that would expect A and B having a closer relationship in the cluster hierarchy than A and C.

The second limitation is that a cluster of complex shape and huge size can usually not be represented by one representative object. However, the idea of navigational similarity search depends on the suitability of the object which is displayed in order to represent the objects in a cluster.

In order to solve these limitations, we propose a novel way of computing the cluster hierarchy and suitable representations. The general idea is that we are interested in small spherically shaped clusters rather than in clusters of arbitrary size and shape. Intuitively, we can select a given number of representative objects that represent their spherical neighborhood in a best possible way. These representatives can build the lowest level of a cluster hierarchy tree. The next level of the tree (above a given level) can then be built by choosing again the most representative objects from the representatives in the level below until we do not have enough representatives and, thus, have reached the root of the hierarchy. The most important aspect in this strategy is the definition of the representative power of an object. We will discuss this issue and outline a novel procedure to generate the cluster hierarchy in the following.

The idea of this new approach is sketched in Figure 9: a data set with two clusters of different size and shape is clustered. Using OPTICS, both clusters are well separated and we can observe both of the mentioned problems: objects that are member of the larger cluster but are quite near (i.e. similar) to the objects in the smaller cluster will have a significantly poor relationship in the cluster hierarchy to the objects in the smaller cluster. On the other hand, objects that are much less similar but are members of the same (larger) cluster will have a strong relationship to each other in the hierarchy. In addition, it is not clear how to represent the larger cluster with its complex shape by a meaningful representative. Our new approach generates representatives for small convex clusters step by step. As it can be seen from Figure 9, this results in a hierarchy that is much more suitable for interactive similarity search. For example, the objects of the smaller cluster will be represented by the same object in the root node of the hierarchy as the objects that belong to the large cluster but are located at the border of that cluster near the smaller cluster. This reflects the intuitive notion of similarity more accurately.

5.2 Basic Definitions

The basic idea is to extract a sufficient amount of dedicated objects that represents the other objects in a best possible way. Such objects are called *representatives* and should be associated with a set of non-representatives, called the *border-shadow* of a representative.

Definition 7 (border-shadow). *Let REP be a set of representative objects and* $\varepsilon \in \mathbb{R}$*. The* border-shadow *of a representative* $r \in REP$ *is defined by:*

$$r.bordershadow := \{o \in REP \mid dist(o, r) \leq \varepsilon\}.$$

The border-shadow of r contains the set of objects in the (hyper-) sphere around r with radius ε. Obviously, a global value for the size of the representative area for each representative is not appropriate, since it does not reflect the local data distribution. Thus, we define an additional, adaptive set of representative objects, the so called *core-shadow* of a representative.

Definition 8 (core-distance). *Let REP be a set of representative objects,* $k \in \mathbb{N}$ *and* $r \in REP$*. The* core-distance *of* r *is defined by:*

$$r.coredist := \begin{cases} 0 & \textit{if } |\mathcal{N}_\varepsilon(r)| < k \\ k\text{-}nn\text{-}dist(r) & \textit{else.} \end{cases}$$

Definition 9 (core-shadow). *Let REP be a set of representative objects and* $r \in REP$*. The* core-shadow *of* r *is defined by:*

$$r.coreshadow := \{o \in REP \mid dist(o, r) \leq r.coredist\}.$$

The core-shadow of r contains the set of objects in the (hyper-) sphere around r with radius k-nn-$dist(r)$. Obviously, this radius adapts to the local density of objects around r.

Both the border-shadow and the core-shadow can be used to define the quality of a representative.

Definition 10 (quality of a representative). *Let REP be a set of representative objects and* $r \in REP$*. The* quality *of* r *is defined by:*

$$r.quality := \begin{cases} 0 & \textit{if } |\mathcal{N}_\varepsilon(r)| < k \\ \frac{\mathcal{N}_\varepsilon(r)}{1+k\text{-}nn\text{-}dist(r)} & \textit{else.} \end{cases}$$

```
algorithm cluster_representatives(SetOfObjects DB, Integer k)
  REP_0 := emptySet;
  REP_1 := DB;
  i := 1;

  WHILE REP_i != REP_i-1 DO
    compute new epsilon;
    send = emptySpatialIndex;
    wait = emptySpatialIndex;
    sort REP_i in descending order w.r.t. quality values;
    r := REP_i.removeFirst();

    WHILE r.quality == 0 AND REP_i.size > 0 DO
      IF r.coreshadow does not intersect the core shadow of any object in send THEN
        send.add(r);
        FOR EACH s IN wait DO
          IF s is in r.bordershadow DO
            wait.remove(s);
          ENDIF
        ENDDO
      ELSE
        IF r is not in border-shadow of any object in send DO
          wait.add(r);
        ENDIF
      ENDIF
      r := REP_i.removeFirst();
    ENDDO

    REP_i+1 := REP_i;
    REP_i+1.addAll(send);
    REP_i+1.addAll(wait);
    i++;
  ENDDO
END. // cluster_representatives
```

Fig. 10. Pseudo code of the Cluster Representatives algorithm.

The key idea of our hierarchical clustering approach is to find on each level of the hierarchy an optimal (w.r.t. quality) set of representatives such that the representatives have non-overlapping core-shadows (overlapping border-shadows are allowed).

5.3 Algorithm

The general idea of the algorithm is to start with the whole database as the initial set of representatives. In the i-th iteration, we take the set of current representatives REP_i and compute the new set of representatives REP_{i+1}. To ensure, that we get the best representatives, we sort REP_i by descending quality values (cf. Definition 10). Theoretically, we can recursively select the representative r having the highest quality from the sorted REP_i list, add r with its border-shadow to REP_{i+1}, and remove all further objects from REP_i that are in the core-shadow of r.

However, we have to take care, that core-shadows of representatives in REP_{i+1} do not overlap. For that purpose, we test for each not yet selected representative $r \in REP_i$ whether its core-shadow overlaps the core-shadow of any object in REP_{i+1}. To support this intersection-query efficiently, we can organize the

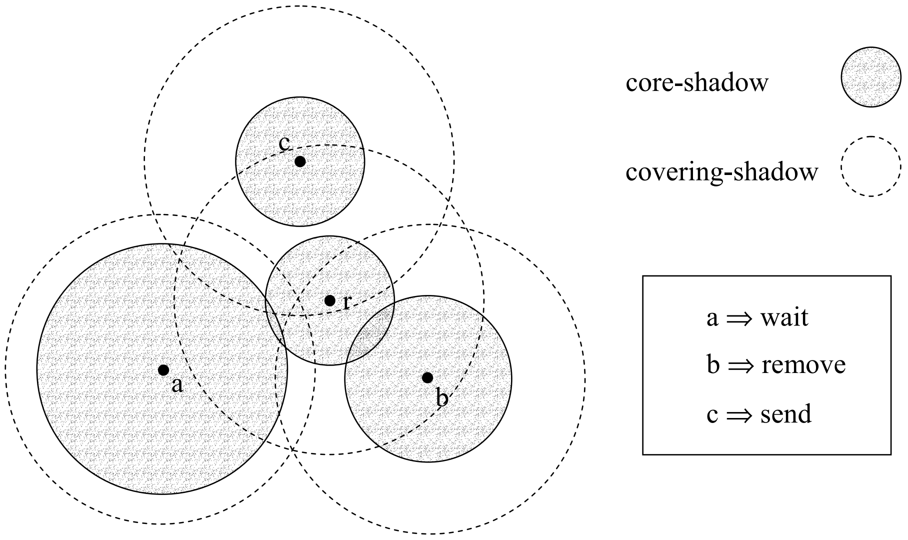


Fig. 11. Sorting of representatives acording to intersections of their core-shadows and border-shadows.

objects in REP_{i+1} in a spatial index structure (data structure *send*). If there is no such overlap, we add r and its border-shadow to REP_{i+1} (and *send*) and remove all objects in the core-shadow of r from REP_i. If the core-shadow of r intersects the core-shadow of any already selected representatives in REP_{i+1}, we have to distinguish two cases. (1) If r is within the border-shadow of any representative in REP_{i+1} we can remove r because it is represented by at least one representative. (2) If r is not within the border-shadow of any representative in REP_{i+1} we cannot remove r since it is not yet represented. We will have to test for r at a later time, whether it is in the border-shadow of a representative chosen later. For that purpose, we add those points to an additional data structure called *wait*. Thus, when adding a new representative r to REP_{i+1}, we have to test whether there are objects in *wait* which are in the border-shadow of r. If so, we delete those objects from *wait*. The algorithm terminates if we gain no more new representatives after an iteration, i.e. $REP_{i+1} = REP_i$.

In summary, in each iteration, we do the following (cf. the pseudo code in Figure 10):

1. Sort REP_i by descending quality values.
2. As long as there are representatives in REP_i having a quality greater than 0, take and remove the first object r from the sorted REP_i list:
   If *r.coreshadow* does not intersect the core-shadow of any object in REP_{i+1} (cf. object c in Figure 11):
   - add r along with *r.coreshadow* and *r.bordershadow* to REP_{i+1},
   - add r to *send*,

  - remove all objects in *wait* which are in the border-shadow of r (range-query against *wait*).

  Else (i.e. *r.coreshadow* intersects the core-shadow of at least one object in REP_{i+1}):
  - If r is not in the border-shadow of any object in REP_{i+1}, add r to *wait* (cf. object a in Figure 11).
  - If r is in the border-shadow of any object in REP_{i+1}, do nothing (r is already removed from REP_i; cf. object b in Figure 11).
3. Add all remaining objects in REP_i and *wait* to REP_{i+1}.

5.4 Choice of ε in the i-th Iteration

The quality of a representative r depends not only on the parameter k (specifying the number of neighbors of r which are taken into account for quality computation) but also on the radius ε of the local area around r. A global value for ε is not appropriate since the dataspace is getting sparser in each iteration. A dynamically adapting value for ε is deemed more appropriate.

If we assume that DB is a set of n feature vectors of dimensionality d (i.e. $DB \subseteq \mathbb{R}^d$) and each attribute value of all $o \in DB$ is normalized (i.e. falls into the range $[0, MAX]$ for a specified, fixed MAX) the volume of the data space can be computed by:

$$Vol(DB) = MAX^d.$$

If the n objects of DB are uniformly distributed in $Vol(DB)$, we expect one object in the volume $Vol(DB)/n$ and k objects in the volume $k \cdot Vol(DB)/n$. Thus, the expected k-nearest neighbor distance of any object $o \in DB$ is equal to the radius r of a hypersphere having this volume $k \cdot Vol(DB)/n$. Since the volume of a hypersphere with radius r can be computed by:

$$V_{Sphere}(r) = \frac{\sqrt{\pi}^d}{\Gamma(d/2+1)} \cdot r^d,$$

where Γ denotes the well-known Gamma function, we can compute the expected k-nn-distance $\hat{r}$ of the objects in DB by solving the following equation:

$$\frac{\sqrt{\pi}^d}{\Gamma(d/2+1)} \cdot \hat{r}^d = k \cdot \frac{MAX^d}{n}.$$

Simple algebraic transformations yield:

$$\hat{r} = MAX \cdot \sqrt[d]{\frac{k \cdot \Gamma(d/2+1)}{n \cdot \sqrt{\pi}^d}}.$$

For simplicity reasons, we can also compute:

$$\hat{r} = MAX \cdot \sqrt[d]{\frac{k}{n}}.$$

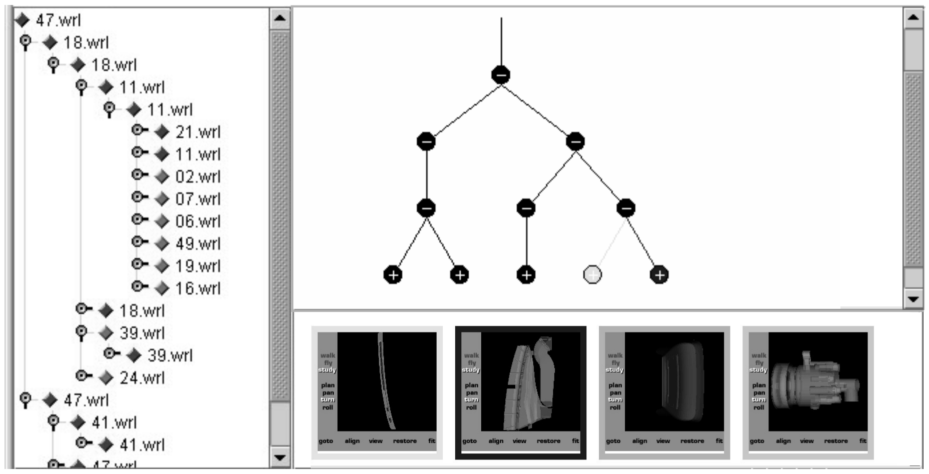


Fig. 12. Screenshot of CLUSS.

Let us note that this is the correct value of the expected k-nn-distance if we use the L_∞-norm instead of the L_2-norm.

In the i-th iteration, an appropriated choice for ε as the expected k-nn-distance of the objects in REP_i:

$$\varepsilon = MAX \cdot \sqrt[d]{\frac{k}{|REP_i|}}.$$

If we further assume $MAX = 1$ (i.e. all attributes have normalized values in $[0, 1]$), we have:

$$\varepsilon = \sqrt[d]{\frac{k}{|REP_i|}}.$$

5.5 The Extended Prototype CLUSS

We have implemented the proposed ideas in Java and integrated a GUI to visualize the cluster hierarchy and cluster representatives. The resulting prototype is called CLUSS which uses the newly proposed method for generating the cluster hierarchy and suitable cluster representatives. A sample sceenshot of CLUSS is depicted in Figure 12. The hierarchy is now visualized by means of a tree (upper right frame in Figure 12). For clearness, the subtrees of each node of the tree is not visualized per default but can be expanded for browsing by clicking on the according node. The hierarchy tree is also visualized in the frame on the left-hand side of the GUI. The frame on the lower right side in Figure 12 displays the representatives at the nodes that are currently selected.

We performed some sample visual similarity search queries using CLUSS and compared it to the cluster hierarchy created by OPTICS and Gradient

Clustering. In fact, it turned out that using CLUSS allows a more accurate interactive similarity search. The hierarchy generated by CLUSS differs from that generated by its comparison partner and is more meaningful. So CLUSS provided better results for navigational similarity search, for applications of visual data mining, employing OPTICS is more appropriate, especially, if the application calls for clustering solutions that detect clusters of different sizes and shapes.

6 Conclusions

In this chapter, we combined hierarchical clustering and data visualization techniques to allow data analysis, comparison of data models and navigational similarity search. The idea of comparing data spaces using hierarchical density-based clustering is to display connected reachability plots and compare these to reference class models using color coding. Navigational similarity search describes a novel approach to retrieve similar data objects. Instead of posing similarity queries our new approach extracts a cluster hierarchy from a given set of data objects and uses the resulting cluster tree to navigate between sets of similar objects. To extract a cluster hierarchy from a reachability plot generated by the density-based hierarchical clustering algorithm OPTICS, we introduced a new method for cluster extraction called Gradient Clustering. OPTICS produces a meaningful picture of the density distribution of a given data set and is thus well suited for data analysis. However, in many applications the cluster hierarchy derived from the reachability plot might not provide intuitive access for navigational similarity search. Therefore, we described an alternative hierarchical clustering approach called CLUSS which facilitates the interactive similarity search in large collections of data.

References

1. Jagadish, H.V.: "A Retrieval Technique for Similar Shapes". In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'91), Denver, CO. (1991) 208–217
2. Agrawal, R., Faloutsos, C., Swami, A.: "Efficient Similarity Search in Sequence Databases". In: Proc. 4th. Int. Conf. on Foundations of Data Organization and Algorithms (FODO'93), Evanston, ILL. Volume 730 of Lecture Notes in Computer Science (LNCS)., Springer (1993) 69–84
3. Faloutsos, C., Barber, R., Flickner, M., Hafner, J., et al.: "Efficient and Effective Querying by Image Content". Journal of Intelligent Information Systems **3** (1994) 231–262
4. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: "Fast Subsequence Matching in Time-Series Databases". In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94), Minneapolis, MN. (1994) 419–429
5. Agrawal, R., Lin, K.I., Sawhney, H.S., Shim, K.: "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases". In: Proc. 21th Int. Conf. on Very Large Databases (VLDB'95). (1995) 490–501

6. Berchtold, S., Keim, D.A., Kriegel, H.P.: “Using Extended Feature Objects for Partial Similarity Retrieval”. VLDB Journal **6** (1997) 333–348
7. Berchtold, S., Kriegel, H.P.: “S3: Similarity Search in CAD Database Systems”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’97), Tucson, AZ. (1997) 564–567
8. Keim, D.A.: “Efficient Geometry-based Similarity Search of 3D Spatial Databases”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’99), Philadelphia, PA. (1999) 419–430
9. Kriegel, H.P., Kröger, P., Mashael, Z., Pfeifle, M., Pötke, M., Seidl, T.: “Effective Similarity Search on Voxelized CAD Objects”. In: Proc. 8th Int. Conf. on Database Systems for Advanced Applications (DASFAA’03), Kyoto, Japan. (2003) 27–36
10. Kriegel, H.P., Brecheisen, S., Kröger, P., Pfeifle, M., Schubert, M.: “Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’03), San Diego, CA. (2003) 587–598
11. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: “OPTICS: Ordering Points to Identify the Clustering Structure”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’99), Philadelphia, PA. (1999) 49–60
12. McQueen, J.: “Some Methods for Classification and Analysis of Multivariate Observations”. In: 5th Berkeley Symp. Math. Statist. Prob. Volume 1. (1967) 281–297
13. Breunig, M.M., Kriegel, H.P., Kröger, P., Sander, J.: “Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering”. In: Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’01), Santa Barbara, CA. (2001) 79–90
14. Achtert, E., Böhm, C., Kriegel, H.P., Kröger, P.: “Online Hierarchical Clustering in a Data Warehouse Environment”. In: Proc. 5th IEEE Int. Conf. on Data Mining (ICDM’05), Houston, TX. (2005) 10–17
15. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining (KDD’96), Portland, OR, AAAI Press (1996) 291–316
16. Sander, J., Qin, X., Lu, Z., Niu, N., Kovarsky, A.: “Automatic Extraction of Clusters from Hierarchical Clustering Representations”. In: Proc. 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003), Seoul, Korea. (2003) 75–87
17. Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T., Weissig, H., Shindyalov, I.N., Bourne, P.E.: “The Protein Data Bank”. Nucleic Acids Research **28** (2000) 235–242