Institut für Informatik
Lehr- und Forschungseinheit
für Datenbanksysteme

Ludwig
Maximilians
Universität
München

LMU

Diplomarbeit

Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects

Stefan Brecheisen

Aufgabensteller:	Prof. Dr. Hans-Peter Kriegel
Betreuer:	Martin Pfeifle
Abgabetermin:	21. Januar 2003

Erklärung

Hiermit versichere ich, daß ich diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 21. Januar 2003

Stefan Brecheisen

Zusammenfassung

Der Begriff der Ähnlichkeitssuche gewinnt für moderne Datenbankanwendungen, wie z.B. in den Bereichen Multimedia, Molekularbiologie, medizinische Bildverarbeitung und vielen anderen, zunehmend an Bedeutung. Besonders bei CAD-Anwendungen können geeignete Ähnlichkeitsmodelle und eine klare Repräsentation der Ergebnisse dabei helfen, die Kosten für die Entwicklung und Herstellung neuer Teile zu senken, indem bereits vorhandene Teile so weit wie möglich wiederverwendet werden. Die meisten der bekannten Ähnlichkeitsmodelle basieren auf der Idee der Extraktion von Feature-Vektoren. Wir passen vier dieser Modelle für die Anwendung auf 3D-Voxeldaten an. Anhand des vielversprechendsten dieser vier Modelle erklären wir, wie Mengen von Feature-Vektoren anstelle von einzelnen Feature-Vektoren für effektivere und dennoch effiziente Ähnlichkeitssuche benutzt werden können. Zunächst führen wir einen intuitiven Distanzbegriff zwischen Mengen von Feature-Vektoren ein und beschreiben einen effizienten Algorithmus zur Distanzberechnung. Desweiteren stellen wir eine Methode zur Beschleunigung der Anfragebearbeitung auf Datenbanken, die Vektormengen enthalten, vor. Wir nutzen dazu die Strategie der mehrstufigen Anfragebearbeitung. Die experimentelle Evaluierung wurde mit Hilfe zweier Testdatenbanken durchgeführt, die von der Auto- und Flugzeugindustrie zur Verfügung gestellt wurden. Unsere Experimente zeigen, daß unser neuer Ansatz zur Ähnlichkeitssuche in verhältnismäßig kurzer Zeit aussagekräftigere Ergebnisse liefert als die bisherigen Verfahren. Zur Evaluierung verwenden wir ein Verfahren zum hierarchischen Clustering. Dies ist eine neue und effektive Methode, um Ähnlichkeitsmodelle zu analysieren und miteinander zu vergleichen.

Abstract

Similarity search in database systems is becoming an increasingly important task in modern application domains such as multimedia, molecular biology, medical imaging and many others. Especially for CAD applications, suitable similarity models and a clear representation of the results can help to reduce the cost of developing and producing new parts by maximizing the reuse of existing parts. Most of the existing similarity models are based on the paradigm of feature vectors. We adapt four of these models to voxelized 3-D data. Based on the most promising of these four models, we explain how sets of feature vectors can be used for more effective and still efficient similarity search instead of single feature vectors. We first introduce an intuitive distance measure on sets of feature vectors together with an algorithm for its efficient computation. Furthermore, we present a method for accelerating the processing of similarity queries on vector set data by using a multi-step query processing strategy. The experimental evaluation is based on two real world test data sets provided by the car and aircraft industry and points out that our new similarity approach yields more meaningful results in comparatively short time. This evaluation is based on hierarchical clustering as a new and effective way to analyze and compare similarity models.

Contents

Chapter 1

Introduction

In the last ten years, an increasing number of database applications has emerged for which efficient and effective support for similarity search is substantial. The importance of similarity search grows in application areas such as multimedia, medical imaging, molecular biology, computer aided engineering, marketing and purchasing assistance, etc. [Jag91, AFS93, MG93, FBF+94, FRM94, ALSS95, BKK97, BK97, Kei99]. Particularly, the task of finding similar shapes in 2-D and 3-D becomes more and more important. Examples for new applications that require the retrieval of similar 3-D objects include databases for molecular biology, medical imaging and computer aided design.

Especially, the development, design, manufacturing and maintenance of modern engineering products is a very expensive and complex task. Effective similarity models are required for two- and three-dimensional CAD applications to cope with rapidly growing amounts of data. Shorter product cycles and a greater diversity of models are becoming decisive, competitive factors in the hard-fought automobile and aircraft market. These demands can only be met if the engineers have an overview of already existing CAD parts. In this diploma thesis, we introduce an effective and flexible similarity model for complex 3-D CAD data, which helps to find and group similar parts. This model is particularly suitable for voxelized data, which often occur in CAD applications. It is not based on the traditional approach of describing one object by a single feature vector but instead we map an object onto a *set of feature vectors*, i.e. an object is described by a *point set*.

The remainder of this diploma thesis is organized as follows: In Chapter 2 we shortly review already existing spatial similarity models and provide a classification of the techniques into feature-based models and direct geometric models. Moreover we provide the basis for similarity models based on voxelized

CAD objects. We discuss the concepts of translation, rotation, reflection and scaling invariances. In the rest of the chapter we adapt three known similarity models to voxelized 3-D data. These three models are based on a complete partitioning of the data space into disjoint cells. In Chapter 3 we introduce a fourth similarity model which is based on a more flexible object-oriented partitioning approach in greater detail. We present a known algorithm for finding good sequential descriptions of spatial objects using axis parallel rectangular covers which we adapted for the use on 3-D data. Furthermore we show how to transform these sequential descriptions into feature vectors. Based on the most promising of our four models, we explain in Chapter 4 our new approach based on sets of feature vectors. We introduce a distance measure on vector sets suitable for similarity search together with a method to compute this distance measure efficiently. Next, in Chapter 5, we address the issue of efficient query processing in large databases. We use a multi-step query processing strategy and show how to take advantage of already existing index structures. In Chapter 6, we introduce hierarchical clustering as a new and effective way to analyze similarity models. We show that our new approach efficiently generates more significant results compared to the traditional approaches based on single feature vectors. The experiments are based on two real-world test data sets of our industrial partners, a German car manufacturer and an American aircraft producer. The diploma thesis concludes in Chapter 7 with a short summary of our results and a few remarks on future work.

Chapter 2

Similarity Models for Voxelized CAD Objects

2.1 Related Work

In recent years, considerable work on similarity search in database systems has been published. Many of the previous approaches, however, deal with one- or two-dimensional data, such as time series, digital images or polygonal data, most of them do not support three-dimensional objects. In this section, we discuss some competing approaches to establish similarity measures. We provide a classification of the techniques into feature-based models and direct geometric models.

2.1.1 Feature-Based Similarity

A widely used class of similarity models is based on the paradigm of feature vectors. The basic idea is as follows: Using a feature transform, the objects are mapped onto a feature vector in an appropriate multidimensional feature space. The similarity of two objects is then defined as the proximity of their feature vectors in the feature space: The closer their feature vectors are located, the more similar two objects are considered.

Several reasons lead to the wide use of feature-based similarity models: First, the more complex the objects are, the more difficult it may be to find an appropriate similarity distance function. A second reason wherefore feature-based similarity models are quite popular is that they may be easily tuned to fit to specific applications. In general, this task is performed in close cooperation with domain experts who specify appropriate features and adapt them to the

specific requirements. Since the existing techniques for query processing are independent from the particular definition of the features, efficient support may be provided without an in-depth insight into the application domain.

Examples where the paradigm of feature-based similarity has been successfully applied to the retrieval of similar spatial objects include structural features of 2-D contours [GM93, MG93, MG95], angular profiles of polygons [BMH92], rectangular covers of regions [Jag91], algebraic moment invariants [TC91, FBF+94], and 2-D section coding [Ber97, BKK97]. Non-geometric applications include similarity search on time series [AFS93, FRM94], and on color histograms in image databases [NBE+93, FBF+94, HSE+95], among several others.

Agrawal et al. present a method for similarity search in a sequence database of one-dimensional data [AFS93]. The sequences are mapped onto points of a low-dimensional feature space using a Discrete Fourier Transform, and then a PAM is used for efficient retrieval. This technique was later generalized for subsequence matching [FRM94], and searching in the presence of noise, scaling, and translation [ALSS95]. However, it remains restricted to one-dimensional sequence data.

Mehrotra and Gary suggest the use of boundary features for the retrieval of shapes [MG93, GM93]. Here, a 2-D shape is represented by an ordered set of surface points, and fixed-sized subsets of this representation are extracted as shape features. All of these features are mapped to points in multidimensional space which are stored using a Point Access Method (PAM). This method is essentially limited to two dimensions.

Jagadish proposes a technique for the retrieval of similar shapes in two dimensions [Jag91]. He derives an appropriate object description from a rectilinear cover of an object, i.e. a cover consisting of axis-parallel rectangles. The rectangles belonging to a single object are sorted by size, and the largest ones serve as retrieval key for the shape of the object. This method can be generalized to three dimensions by using covers of hyperrectangles, as we will see in chapter 3.

Histograms as Feature Vectors

Histograms represent a quite general class of feature vectors which have been successfully applied to several applications. For any arbitrary distribution of objects, a histogram represents a more or less fine grained aggregation of the information. The general idea is to completely partition the space of interest into disjoint regions which are called cells, and to map every object onto a

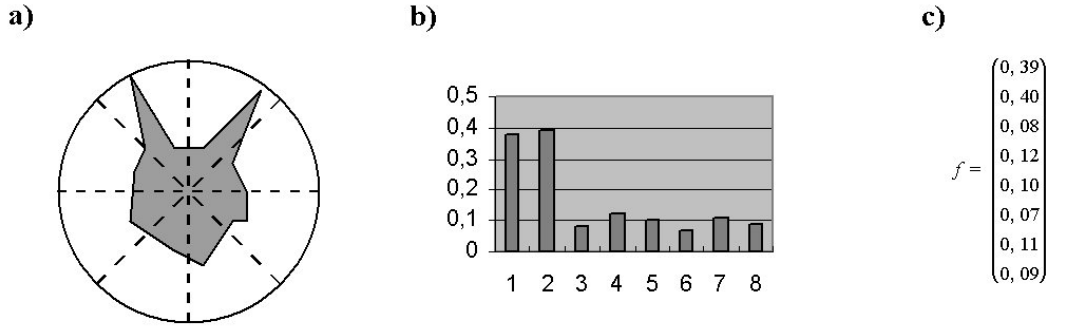


Figure 2.1: Section coding of 2-D regions: a) Original space and object. b) Corresponding histogram. c) Corresponding feature vector.

single bin or to distribute an object among a set of bins of the corresponding histogram. Then a histogram can be transformed directly into a feature vector by mapping each bin of the histogram onto one dimension (attribute) of the feature vector. The histogram approach applies to geometric spaces as well as to non-geometric spaces.

A popular example for the use of histograms to define the similarity of complex objects is the color histogram approach which is a core component of the QBIC system [NBE+93, FBF+94]. Among other techniques, color histograms are used to encode the percentage of colors in an image [SH94, HSE+95]. Our second example is taken from a spatial database application: The 2-D section coding approach [Ber97, BK97] represents a particular histogram technique that is used in the S3 system [BKK97] for the retrieval of similar mechanical parts. For each object, the circumscribing circle is decomposed into a fixed number of sectors around the center point. For each sector, the fraction of the area is determined that is overlapped by the object. Altogether, the resulting feature vector is a histogram over the 2-D, whose bins represent the corresponding 2-D sectors. Figure 2.1 illustrates the technique by an example with 8 sectors. This approach, however, is also limited to two dimensions since a linear ordering of the boundary is required. In the 3-D, there is no canonical linearization of the two-dimensional boundary of arbitrary solids.

In [KKS98, AKKS99] the retrieval of similar 3-D objects from a biomolecular database was investigated. The introduced models are based on 3-D shape histograms, where three different approaches were used for space partitioning: shell bins, section bins and combined bins (cf. Figure 2.2). Unfortunately, these models are not inherently suitable for voxelized data which are axis-parallel.

2.1.2 Geometry-Based Similarity

A class of models that is to be distinguished from the feature-based techniques are the similarity models that are defined by directly using the geometry. Two objects are considered similar if they minimize a distance criterion that is purely defined by the geometry of the objects. Examples include the similarity retrieval of mechanical parts, the difference volume approach, and the approximation-based similarity model for 3-D surface segments:

Rotational Symmetric Mechanical Parts. In [SKSH89], a method is presented to retrieve similar mechanical parts from a database. The similarity criterion is defined in terms of tolerance areas which are specified around the query object. All objects that fit into the tolerance area count for being similar. Although the parts are 3-D, only their 2-D contour is taken into account for the retrieval technique.

Difference Volume Approach. The difference volume or error volume of spatial objects is a promising approach which has been already successfully applied to medical images, for instance [Hig90, Vin91]. Furthermore, extensions such as the combination with methods from mathematical morphology have been investigated on a tumor database [KSF+96]. However, they considered only 2-D images. A competing approach is is based on a new geometric index structure as suggested in [Kei99]. The basic idea of this solution is to use the concept of hierarchical approximations of the 3D objects to speed up the search process.

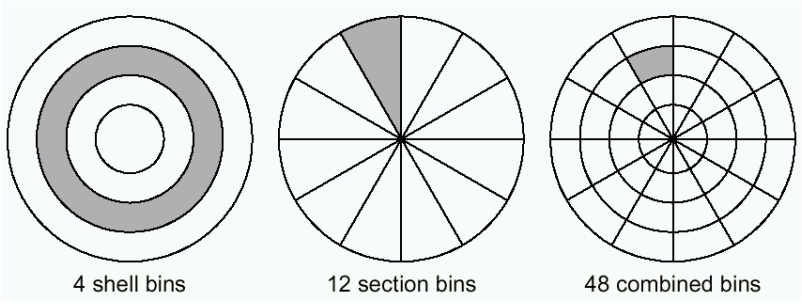


Figure 2.2: Shells and sections as basic models for shape histograms. In each of the 2-D examples, a single bin is marked.

Class	Definition of Similarity	Examples
feature-based similarity	similarity is proximity in the feature space	• rectangular cover of regions [Jag91] • algebraic moment invariants [TC91] • 2-D contour features [GM93, MG95] • angular profiles of polygons [BMH92] • section coding [Ber97, BKK97] • time series [AFS93, FRM94] • color histograms [NBE+93, FBF+94, HSE+95]
geometric similarity	similarity is directly defined by geometry	• symmetric mechanical parts [SKSH89] • difference volume [Vin91, KSF+96, Kei99] • 3-D surface segments [KSS97]

Table 2.1: Classification of complex similarity models.

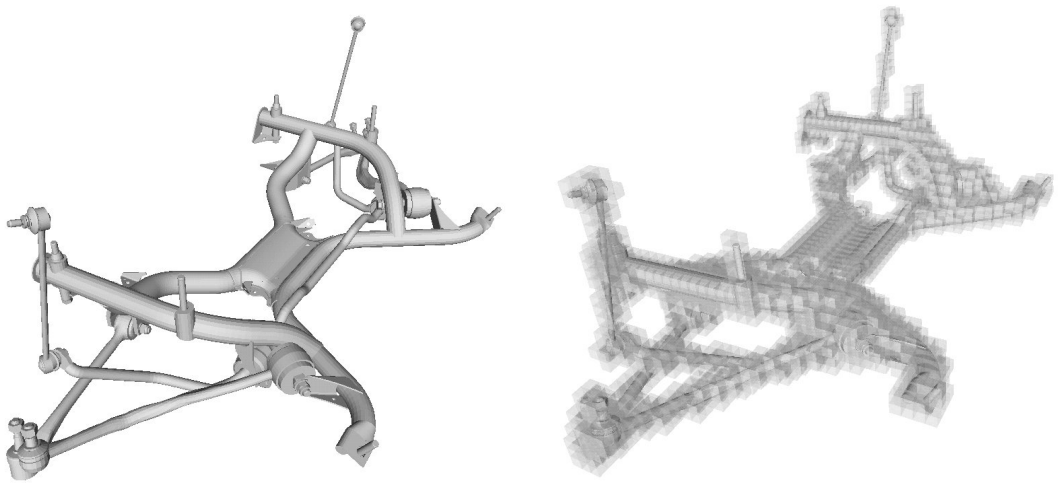
Approximation-based Similarity of 3-D Surface Segments. The retrieval of similar 3-D surface segments is a task that supports the docking search for proteins in biomolecular databases. Following the approximation-based model, the similarity of 3-D surface segments is measured by their mutual approximation error with respect to a given multi-parametric surface function which serves as the underlying approximation model. To state it simply, two segments are the more similar, the better they fit to the approximation of the partner segment [KSS97].

Summary

In Table 2.1, we summarize our classification of similarity models into feature-based approaches and direct geometry-based proposals. The list of examples is by no means complete but provides an impression of the potentials of both paradigms. In this work, we introduce effective similarity models for CAD objects, which rely on the feature based histogram approach.

2.2 Voxelized CAD Objects

Engineering products can be regarded as a collection of individual, three-dimensional parts. Each of these parts may consist of a complex and an intricate geometric shape with a very high precision. Accurate representations of CAD surfaces are typically implemented by parametric bicubic surfaces, including



(a) Triangulated surface

(b) Voxelized surface

Figure 2.3: Scan conversion on a triangulated surface.

Hermite, Bézier, and B-spline patches. For many operations, such as graphical display or the efficient computation of surface intersections, these parametric representations are too complex [MH99]. As a solution, approximative polygon (e.g. triangle) meshes can be derived from the accurate surface representation. These triangle meshes allow for an efficient and interactive display of complex objects. In order to apply spatial indexing, often, a coarser, conservative approximation of the parts, by means of voxels, is applied (cf. Figure 2.3).

A basic algorithm for the 3D scan-conversion of polygons into a voxel-based occupancy map has been proposed by Kaufmann [Kau87]. Similarly to the well-known 2D scan-conversion technique, the runtime complexity to voxelize a 3D polygon is $O(n)$, where n is the number of generated voxels. If we apply this conversion to the given triangle mesh of a CAD object (cf. Figure 2.3(a)), a conservative approximation of the part surface is produced (cf. Figure 2.3(b)). In the following, we assume a uniform three-dimensional voxel grid covering the global product space. The grid resolution determines the finest possible granularity for the approximation of the objects. By means of space filling curves, each cell of the grid can be encoded by a single integer number, and thus an extended CAD object is represented by a set of integers.

2.3 Shape Histograms

Histograms are usually based on a complete partitioning of the data space into disjoint cells which correspond to the bins of the histograms.

We divide the data space into axis parallel, equi-sized partitions (cf. Figure 2.4, which provides an illustration in 2-D). This kind of space partitioning is especially suitable for voxelized data, as cells and voxels are of the same shape, i.e. cells can be regarded as coarse voxels.

Each of these partitions is assigned to one or several bins in a histogram, depending on the specific similarity model. By scaling the number of partitions, the number of dimensions of the feature vector can be regulated (cf. Figure 2.4). Obviously, the more partitions we use, the more smaller differences between the objects become decisive.

By means of the resulting feature vectors, the similarity of two objects can be defined as follows.

Definition 1 (feature-based object similarity)
Let O be the domain of the objects and $F : O \to \mathbb{R}^d$ be a mapping of the objects into the d-dimensional feature space. Furthermore, let $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function between two d-dimensional feature vectors. Then a feature-based object similarity function $simdist : O \times O \to \mathbb{R}$ is defined as follows:

$$simdist(Obj_1, Obj_2) = dist(F(Obj_1), F(Obj_2)).$$

There exist a lot of distance functions which are suitable for similarity search. In the literature, often the *L_p-distance* is used, as for instance the Manhattan distance ($p = 1$) or the Euclidian distance ($p = 2$). Throughout

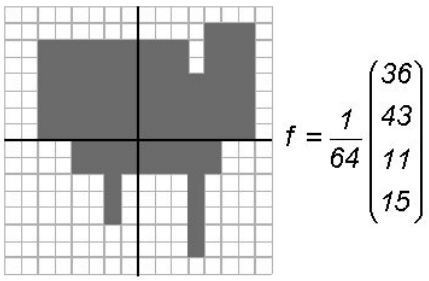


Figure 2.4: Space partitioning with 4 cells. The feature vector generated by the volume model is depicted on the right hand side.

our experiments (cf. Chapter 6), the common Euclidian distance was used.

$$d_{euclid}(\vec{x}, \vec{y}) = \|\vec{x} - \vec{y}\|_2 = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

2.4 Normalization

For effective similarity search it is often required to meet invariance properties with respect to a certain class of transformations, i.e. applying a transformation from this class to an object should have no influence on the result of the similarity function. This leads to the following definition.

Definition 2 (invariance)
Let O be the domain of the objects and $simdist : O \times O \rightarrow \mathbb{R}$ be a feature-based object similarity function. $simdist$ is invariant with respect to a class of transformations $\mathcal{T}$, if for all objects $Obj_1, Obj_2 \in O$ and all transformations $T \in \mathcal{T}$ holds:

$$simdist(Obj_1, Obj_2) = simdist(T(Obj_1), Obj_2) = simdist(Obj_1, T(Obj_2))$$

Invariance can be achieved by applying appropriate transformations to the objects in the database. This is called the normalization of data. Invariance properties relevant for similarity search in CAD databases are scaling, translation, rotation and reflection invariances. It depends on the similarity model as well as user choices which invariances have to be considered for a particular application. Taking normalization into account, we get the following extended similarity definition.

Definition 3 (extended feature-based object similarity)
Let O be the domain of the objects, $F : O \rightarrow \mathbb{R}^d$ a mapping of the objects into the d-dimensional feature space, and $dist : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ a distance function between two d-dimensional feature vectors. Furthermore, let $\mathcal{U}$ be the set of all user-dependent combinations of scaling, translation, rotation and reflection transformations. Then a feature-based object similarity function $simdist$: $O \times O \rightarrow \mathbb{R}$ is defined as follows:

$$simdist(Obj_1, Obj_2) = \min_{T \in \mathcal{U}}\{dist(F(Obj_1), F(T(Obj_2)))\}.$$

Note that we achieve invariance by taking the minimum of the distances between Obj_1 and all transformations of Obj_2 with respect to $\mathcal{U}$.

In the next four sections we discuss each of the aforementioned invariance properties in more detail. In particular we describe the corresponding transformation matrices using homogeneous coordinates [Gri92, NS86], together with their relevant parameters. To calculate these parameters we need the minimal bounding box mbb_o for each object o.

$$mbb_o = (x_{min}, y_{min}, z_{min}, x_{max}, y_{max}, z_{max})$$

The minimal bounding boxes have to be updated after each transformation in order to obtain the parameters for the next matrix.

2.4.1 Scaling Invariance

The actual size of the different objects in a CAD database can vary from a few millimeters to several meters. In order to compare the shape of the objects, we scale them to a uniform size. That is, we fit each voxelized object into a cubic voxel space with a predefined extension r in each of the three dimensions. This can be achieved by applying the following transformation with appropiate parameters s_x, s_y and s_z.

$$\left(\begin{array}{ccc|c} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

We distinguish between two scaling methods.

- Proportional scaling: The objects are scaled proportionally with respect to the three coordinate axes. The shape of the objects is preserved. Using this method the scaling factors s_x, s_y and s_z are equal and can be determined like this:

  $$s_x = s_y = s_z = \frac{r}{\Delta mbb_o}$$

  where Δmbb_o is the maximal extension of the minmal bounding box with respect to the three coordinate axes.

  $$\Delta mbb_o = \max\{x_{max} - x_{min},\ y_{max} - y_{min},\ z_{max} - z_{min}\}$$

- Non-proportional scaling: Here the extensions of the objects are adjusted to the size of the voxel space independently for each of the coordinate axes. This way the voxel space is used optimally, but the shape of the objects is distorted. Small differences of the shape in dimensions with low extension are amplified to a high degree. The scaling factors s_x, s_y and s_z result from the ratio of the extension of the voxel space and the extension of the object for each dimension.

$$s_x = \frac{r}{x_{max} - x_{min}}, \quad s_y = \frac{r}{y_{max} - y_{min}}, \quad s_z = \frac{r}{z_{max} - z_{min}}$$

We store each object normalized with respect to proportional scaling in the database. Furthermore, we store the scaling factor, so that we can (de)activate scaling invariance depending on the user's needs at runtime, as the actual size of the parts may or may not exert influence on the similarity model.

2.4.2 Translation Invariance

CAD objects are designed and constructed in a standardized position, normalized to the center of the coordinate system. So similarity models for CAD data should recognize similar parts, independently of their spatial location. The four, respectively five, tires of a car are similar, although they are located differently.

Therefore we move each object to a uniform position in the voxel space, so that the center of the minimal boundings box lies in the origin of the coordinate system. The corresponding transformation matrix looks like this:

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

The parameters t_x, t_y and t_z are the coordinates of the center of the minimal bounding box.

$$t_x = \frac{x_{min} + x_{max}}{2}, \quad t_y = \frac{y_{min} + y_{max}}{2}, \quad t_z = \frac{z_{min} + z_{max}}{2}$$

We store each object normalized with respect to translation in the database.

2.4.3 Rotation Invariance

In general, we can apply principal axis transformation in order to achieve invariance with respect to rotation. Here, the idea is to map the position of each object in such a way that the principal axis of each object is parallel to the coordinate system.

In the case of CAD applications, not all possible rotations are considered, but only 90°-rotations. This yields up to 24 different possible positions for each object. The transformation matrices for 90°-rotation around the X, Y and Z axes are listed here:

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \quad \left(\begin{array}{ccc|c} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \quad \left(\begin{array}{ccc|c} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

2.4.4 Reflection Invariance

Reflected parts, e.g. the right and left front door of a car, should be recognized as similar as far as design is concerned. If we look at the production, reflected parts are no longer similar and have to be treated differently.

To reflect an object with respect to the X, Y or Z axis, one of the following transformation matrices can be used:

$$\left(\begin{array}{ccc|c} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \quad \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right), \quad \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

Taking 90°-rotations as well as reflection into account, we may obtain up to $24 \cdot 2 = 48$ varying positions. We could achieve 90°-rotation and reflection invariance by storing 48 different feature vectors for each object in the database or by carrying out 48 different permutations of the query object at runtime. As we want to decide at runtime whether we want to consider reflection invariance or not, we chose the second variant.

To sum up, a similarity model for CAD data should take translation and rotation invariances into account whereas reflection and scaling invariances have to be tunable. Throughout our experiments, we considered invariance with respect to scaling, translation, reflection, and 90°-rotation.

2.5 Spatial Features

After partitioning the data space, we have to determine the spatial features of the objects for each grid cell depending on the chosen model. In order to do that we first have to introduce some notations:

The data space is partitioned in each dimension into p grid cells. Thus, our histogram will consist of $k \cdot p^3$ bins where $k \in \mathbb{N}$ depends on the model specifying the kind and number of features extracted from each cell. For a given object o, let $V^o = \{v \in V_i^o \mid 1 \leq i \leq p^3\}$ be the set of voxels that represents o where V_i^o are the voxels covered by o in cell i. $\bar{V}^o \subseteq V^o$ denotes the set of voxels at the surface of the objects and $\dot{V}^o \subseteq V^o$ denotes set of the voxels inside the object, such that $\bar{V}^o \cup \dot{V}^o = V^o$ and $\bar{V}^o \cap \dot{V}^o = \emptyset$ holds.

Let f_o be the computed feature vector of an object o. The i-th value of the feature vector of object o is denoted by $f_o^{(i)}$.

Let r be the number of voxels of the dataspace in each dimension. In order to ensure a unique assignment of the voxels to a grid cell, we assume that $\frac{r}{p} \in \mathbb{N}$.

2.5.1 The Volume Model

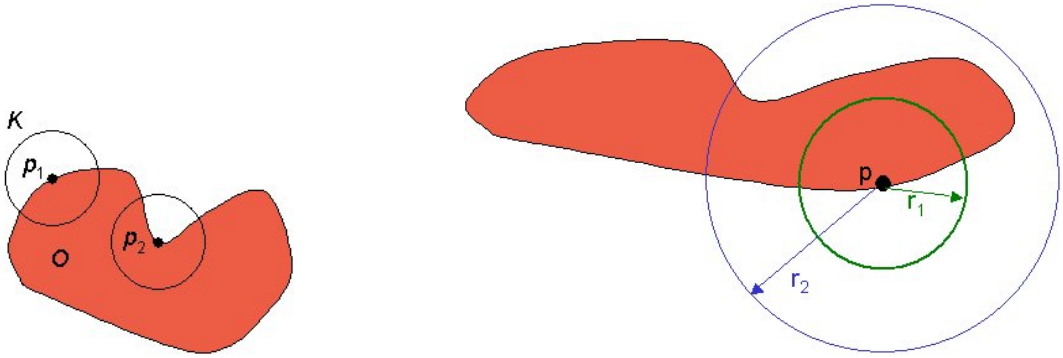
A simple and established approach to compare two objects is based on the number of the object voxels $|V_i^o|$ in each cell i of the partitioning. In the following, this model is refered to as the *volume model*. Each cell represents one dimension in the feature vector of the object. The i-th dimension of the feature vector ($1 \leq i \leq p^3$) of object o can be computed by the normalized number of voxels of o lying in cell i, formally:

$$f_o^{(i)} = \frac{|V_i^o|}{K} \quad \text{where} \quad K = \left(\frac{r}{p}\right)^3$$

Figure 2.4 illustrates the volume model for the 2-D case.

2.5.2 The Solid-Angle Model

The *Solid-Angle* method [Con86] measures the concavity and the convexity of geometric surfaces. It is therefore a good candidate for adequately modelling geometric shapes and has been used in different approaches to spatial similarity modelling. In the following, we describe a model that combines the Solid-Angle approach with our axis-parallel partitioning.



(a) (b)

Figure 2.5: (a) A sample object with different shapes at surface-points p_1 and p_2. (b) Effect of the radius on the Solid-Angle value: The object can be modeled more accurately when using radius r_1 instead of radius r_2.

Let $K_{c,r}$ be a set of voxels that describes a 3-D voxelized sphere with central voxel c and radius r. For each surface-voxel $\bar{v}$ of an object o the so called Solid-Angle value is computed as follows. The voxels of o which are inside $K_{\bar{v},r}$ are counted and divided by the size of $K_{\bar{v},r}$, i.e. the number of voxels of $K_{\bar{v},r}$. The resulting measure is called the Solid-Angle value $\mathrm{Sa}(\bar{v}, r)$ and can be computed as follows:

$$\mathrm{Sa}(\bar{v}, r) = \frac{|\, K_{\bar{v},r} \cap V^o \,|}{|\, K_{\bar{v},r} \,|}$$

where

$$K_{\bar{v},r} \cap V^o = \{w \in K_{\bar{v},r} \mid \exists v \in V^o : w.x = v.x \wedge w.y = v.y \wedge w.z = v.z\}$$

A small Solid-Angle value $\mathrm{Sa}(\bar{v}, r)$ indicates that an object is convex at voxel $\bar{v}$. Otherwise, a high value of $\mathrm{Sa}(\bar{v}, r)$ denotes a concave shape of an object at voxel $\bar{v}$. Figure 2.5(a) illustrates this behavior.

The choice of the radius of the measurement sphere is a crucial parameter. A particular radius could approximate a given object very well, whereas another object is not really well approximated at this radius. This effect is visualized in Figure 2.5(b).

The Solid-Angle values of the cells are transfered into the according histogram bins as described in the following. We distinguish between three different types of cells:

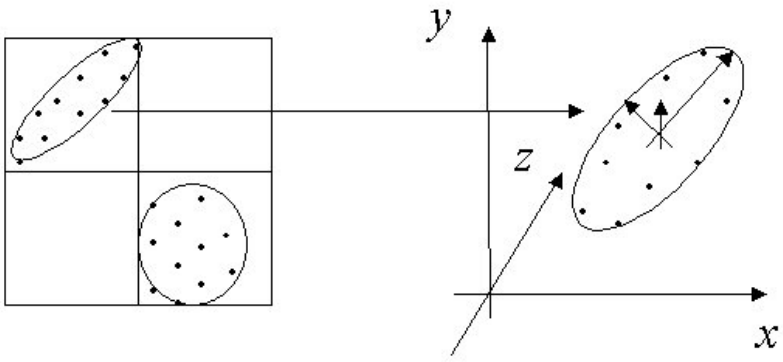


Figure 2.6: Computation of the ellipsoids based on their eigen values

1. Cell i contains surface-voxels of object o, i.e. $\bar{V}_i^o \neq \emptyset$. The mean of all Sa-values of the surface-voxels is computed as the feature value of this cell:
$$f_o^{(i)} = \frac{1}{m} \sum_{j=1}^{m} \mathrm{Sa}(\bar{v}_{i_j}, r) \quad \text{where} \quad \bar{V}_i^o = \{\bar{v}_{i_1}, \ldots \bar{v}_{i_m}\}$$

2. Cell i contains only inside-voxels of object o, i.e. $\bar{V}_i^o = \emptyset$ and $V_i^o \neq \emptyset$. The feature value of this cell is set to 1 (i.e. $f_o^{(i)} = 1$).

3. Cell i contains no voxels of object o, i.e. $V_i^o = \emptyset$. The value of the according bin of the histogram is 0 (i.e. $f_o^{(i)} = 0$).

2.5.3 The Eigen Value Model

In the following, we introduce a new approach to extract local features which is based on eigen values. The set of voxels of an object can be considered as a set of points in the 3-D data space following a particular scattering. The *Eigen Value Model* uses this scattering of the voxel sets in each cell of the partitioning to distinguish the objects by computing the minimum bounding ellipsoid of the voxel set.

A minimum bounding ellipsoid in the 3-D space can be described by three vectors (cf. Figure 2.6). In order to compute these vectors, we consider each voxel v of the object o as a Euclidian vector

$$\vec{v}^o = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

in the 3-D data space and apply principal axis transformation. To determine the principal axis of the vectors in cell i, we first compute their centroid $\vec{C}_i^o$:

$$\vec{C}_i^o = \begin{pmatrix} x_C \\ y_C \\ z_C \end{pmatrix} = \frac{1}{|V_i^o|} \begin{pmatrix} \sum_{j=1}^{|V_i^o|} x_j \\ \sum_{j=1}^{|V_i^o|} y_j \\ \sum_{j=1}^{|V_i^o|} z_j \end{pmatrix}$$

After that, for each vector $\vec{v}^o$ in cell i, the following translation is carried out:

$$\vec{v}^o \mapsto \vec{v}^o - \vec{C}_i^o$$

Based on these transformed vectors $\vec{v}^o$, the covariance matrix Cov_i^o for each cell i can be computed as follows:

$$\mathrm{Cov}_i^o = \frac{1}{|\,V_i^o\,| - 1} \begin{pmatrix} \sum_{j=1}^{|V_i^o|} x_j^2 & \sum_{j=1}^{|V_i^o|} x_j y_j & \sum_{j=1}^{|V_i^o|} x_j z_j \\ \sum_{j=1}^{|V_i^o|} x_j y_j & \sum_{j=1}^{|V_i^o|} y_j^2 & \sum_{j=1}^{|V_i^o|} y_j z_j \\ \sum_{j=1}^{|V_i^o|} x_j z_j & \sum_{j=1}^{|V_i^o|} y_j z_j & \sum_{j=1}^{|V_i^o|} z_j^2 \end{pmatrix}$$

The three eigen vectors $\vec{e}_i^{\,j}$ $(j = 1, 2, 3)$ of the matrix Cov_i^o correspond to the vectors spanning the minimum bounding ellipsoid of the voxel set V_i^o. The eigen values λ_i^j represent the scaling factors for the eigen vectors (cf. Figure 2.7). Both eigen values and eigen vectors are determined by the following equation:

$$\mathrm{Cov}_i^o \cdot \vec{e}_i^{\,j} = \lambda_i^j \, \vec{e}_i^{\,j}$$

The interesting values that are inserted in the bins of the histogram are the eigen values which describe the scattering along the principal axis of the voxel set. These three values can be computed using the characteristic polynomial:

$$\det(\mathrm{Cov}_i^o - \lambda_i^j \, Id) = 0 \quad \text{for} \quad j = 1, 2, 3$$

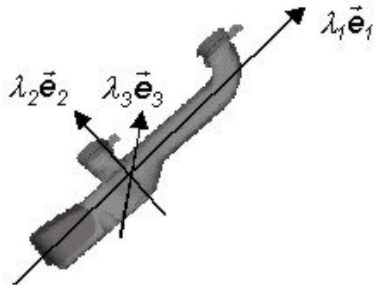


Figure 2.7: Principal axis of a sample object.

Using this equation we obtain three eigen values which are sorted in descending order in the vector $\vec{\lambda_i}$. The highest value represents the variance along the first principal axis, the second value represents the variance along the second principal axis, and the third value represents the variance along the third principal axis.

For each cell i of the partitioning we compute the vector $\vec{\lambda_i}$ of the three eigen values as described right above and register it in the according bins of the histogram:

$$f_o^{(i)} = \vec{\lambda_i} = \begin{pmatrix} \lambda_i^1 \\ \lambda_i^2 \\ \lambda_i^3 \end{pmatrix}$$

Note that for p^3 cells we obtain a feature vector with $3 \cdot p^3$ dimensions.

Chapter 3

Cover Sequence Approximation

The three models described in Section 2.5 are based on a complete partitioning of the data space into disjoint cells. In this chapter, we adapt a known model [Jag91, JB91] to voxelized 3-D data which is not restricted to this rigid space partitioning but rather uses a more flexible object-oriented partitioning approach. This model is in the following referred to as *cover sequence model*.

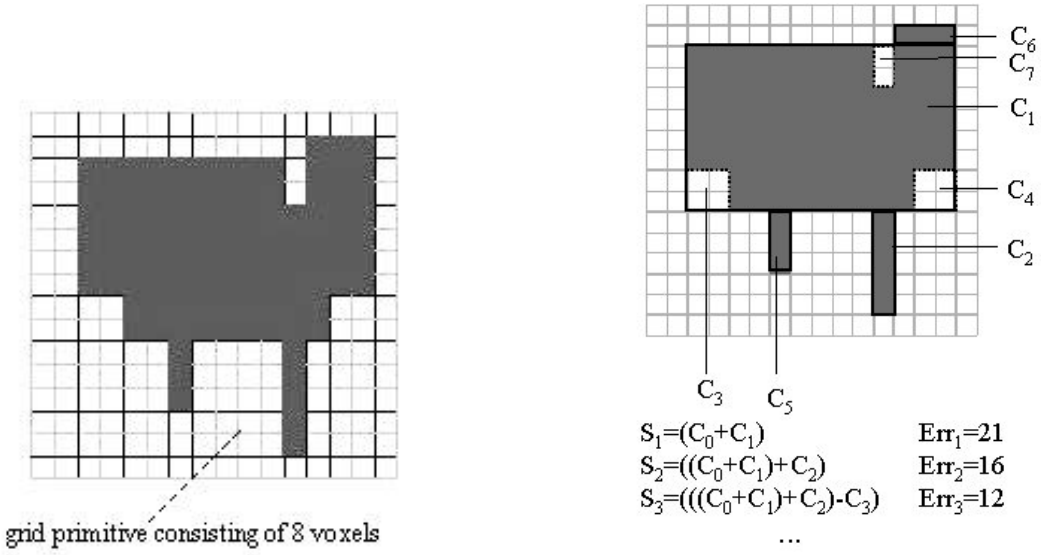
3.1 The Cover Sequence Model

As depicted in Figure 3.1(a) each edge of an object can be extended infinitely in either direction, to obtain a grid of lines. Each rectangle in this grid is called a *grid primitive*, and is located either entirely inside the object, or entirely outside of the object. Furthermore, any pair of adjacent grid primitives must also form a rectangle, respectively a cuboid in the 3-D data space. The basic idea of this model is to find large clusters of grid primitives, called *covers*, which approximate the object as good as possible [JB91]. These covers are organized in a *cover sequence* which provides a sequential description of the object.

Let o be the object being approximated. The quality of a cover sequence S_k of some length $k \in \mathbb{N}$ is measured by the symmetric volume difference Err_k between the object o and the sequence S_k (cf. Figure 3.1(b)). Formally, let the covers be drawn from the set $\mathcal{C}$ of all possible rectangular covers. Then each unit i of the cover sequence comprises a pair $(C_i \in \mathcal{C}, \sigma_i \in \{+, -\})$, where "+" represents set union and "−" represents set difference. The sequence after k units is:

$$S_k = (((C_0\, \sigma_1\, C_1)\, \sigma_2\, C_2)\, \ldots\, \sigma_k\, C_k)$$

where C_0 is an initial empty cover at the zero point. The symmetric volume



(a) Grid primitives.

(b) A cover sequence.

Figure 3.1: The cover sequence model.

difference after k units is:

$$Err_k = |\, o \,\mathrm{XOR}\, S_k \,|$$

Note that there exists some natural number N such that $S_k = o$ and $Err_k = 0$ for all $k \geq N$. At this point an exact description of the object o has been obtained.

3.2 Approximation

Jagadish and Bruckstein [JB91] suggest two algorithms for the retrieval of a cover sequence S_k: a *branch and bound* algorithm with exponential runtime complexity, and a *greedy* algorithm with polynomial runtime complexity which tries to minimize Err_i in each step $i \leq k$. Throughout our experiments we used this second algorithm.

Let us call a grid primitive *black* if it is in the object, and *white* if it is not. Also, a grid primitive is *in* if it is included in the current description of the object, and *out* if it is not. Thus, to start with, all grid primitives are either

out-black or out-white. At the end, when an exact description of the object is obtained, all grid primitives are either in-black or out-white. The total volume of the grid primitives that are out-black and those that are in-white gives the error in the current description.

To speed up the retrieval of the cover sequence, not every possible cover has to be considered in each step, but only those that are not *dominated* for addition or subtraction. The notion of domination is defined as follows.

Cover X is said to dominate cover Y for addition iff X contains every out-black grid primitive in Y, Y contains every out-white grid primitive in X, and $X - Y$ is either empty or has at least one out-black grid primitive. Cover X is said to dominate cover Y for subtraction iff X contains every in-white grid primitive in Y, Y contains every in-black grid primitive in X, and $X - Y$ is either empty or has at least one in-white grid primitive.

If cover X dominates cover Y for addition or subtraction, we are guaranteed that the error at the current step is less if cover X is added or subtracted rather than Y, and that the error will continue to be no greater for all future steps.

The important point is that it is possible, within time proportional to the perimeter of the cover, to determine whether a cover can be dominated by another cover, which is shown in [JB91]. This is faster than calculating the symmetric volume difference for every cover, which takes time proportional to the volume of the cover.

3.3 Feature Extraction

In [Jag91], Jagadish sketches how a 3-D cover sequence

$$S_k = (((C_0\,\sigma_1\,C_1)\,\sigma_2\,C_2)\,\cdots\,\sigma_k\,C_k)$$

of an object o, can be transformed into a $6 \cdot k$-dimensional feature vector. Thereby, each cover C_{i+1} with $0 \leq i \leq k-1$ is mapped onto 6 values in the feature vector f_o in the following way:

$$\begin{aligned}
f_o^{6i+1} &= x\text{-position of } C_i \\
f_o^{6i+2} &= y\text{-position of } C_i \\
f_o^{6i+3} &= z\text{-position of } C_i \\
f_o^{6i+4} &= x\text{-extension of } C_i \\
f_o^{6i+5} &= y\text{-extension of } C_i \\
f_o^{6i+6} &= z\text{-extension of } C_i
\end{aligned}$$

If an object o can be described by a sequence S_j with $j < k$ covers and $Err_j = 0$, we assign $((S_j\, \sigma_{j+1}\, C_0) \cdots \sigma_k\, C_0)$ to S_k. These dummy covers C_0 do not distort our similarity notion (cf. Definition 3), but guarantee that all feature vectors are of the same dimensionality. Thus we can use spatial index structures [BKK96, LJF94, BBJ+00] in order to accelerate similarity queries.

Initially, a cover $C = (x_L, y_L, z_L, x_U, y_U, z_U)$ is given by the coordinates of its corner points. Rather than use these coordinates directly, we apply a few transformations to them. First, we obtain distinct position and size values. The position of the box is given in terms of the mean of the L and U corner points, i.e. the point

$$(\frac{x_L + x_U}{2}, \frac{y_L + y_U}{2}, \frac{z_L + z_U}{2})$$

The size of the box is obtained as the difference between the L and U corner points, i.e. the triple

$$(x_U - x_L, y_U - y_L, z_U - z_L)$$

Thus, we still have six values to store for each cover. However, after this transformation they respresent the position and size of the cover rather than the location of the corner points.

Second, the minimal bounding box mbb_o of the approximated object o is used to normalize the positions of the boxes. That is, the center of the bounding box is placed at the origin, and all coordinates are taken with respect to this origin. This transformation is represented by a shift, which is a triple of constants that has to be subtracted from all the X, Y and Z coordinates respectivly of the position values of the cover. The size values remain unaffected.

Third, the size of the minimal bounding box mbb_o is used to normalize the positions and sizes of the boxes. For this, the X, Y and Z extensions of the bounding box are used to divide the X, Y and Z (both size and position) parameters respectively of the cover. Further, we take the (natural) logarithms of the normalized size values, thus making them "additive" like the position values (no logs are reqired for the position values). Then the difference between two size values provides information about the ratio between these size values.

$$\left(\ln \frac{a}{b}\right)^2 = (\ln a - \ln b)^2 \quad \text{for} \quad a, b \in \mathbb{R}^+$$

This way common distance functions such as the Euclidian distance can be used to measure the similarity of cover sequences.

Chapter 4

Vector Set Representation

4.1 Motivation

As described in Section 3.3 a data object is represented as a feature vector which consists of values obtained from a cover sequence approximation. For similarity queries this method yields a major problem. Always comparing the two covers having the same ranking according to the symmetric volume difference, does not make sense in all cases. Two objects can be considered very different, because of the order of their covers, although they are very similar by intuition. The reason for this effect is that the order of the covers does not guarantee that the most similar covers due to size and position will be stored in the same dimensions. Especially for objects generating two or more covers having almost the same volume, the intuitive notion of similarity can be seriously disturbed. Thus, the possibility to match the covers of two compared objects with more degrees of freedom, might offer a better similarity measure. Figure 4.1 displays a 2-dimensional example of a comparison between a query object and a very similar database object. The first sequence (cf. Figure 4.1(a) represents the covers of the query object in the order given by the symmetric volume difference. Let us note that the covers C_2, C_3 and C_4 are not very similar to the corresponding covers of the database object and therefore, the calculated similarity is relatively weak. By rearranging the order of these covers the total distance between the query object and the database object is considerably decreasing, which is displayed in Figure 4.1(b). Thus, the new order preserves the similarity between the objects much better.

To overcome the problem, the author in [Jag91] proposes to generate several good representations of the query object and then process a query for each of the representations. Afterwards the union of the returned database objects is

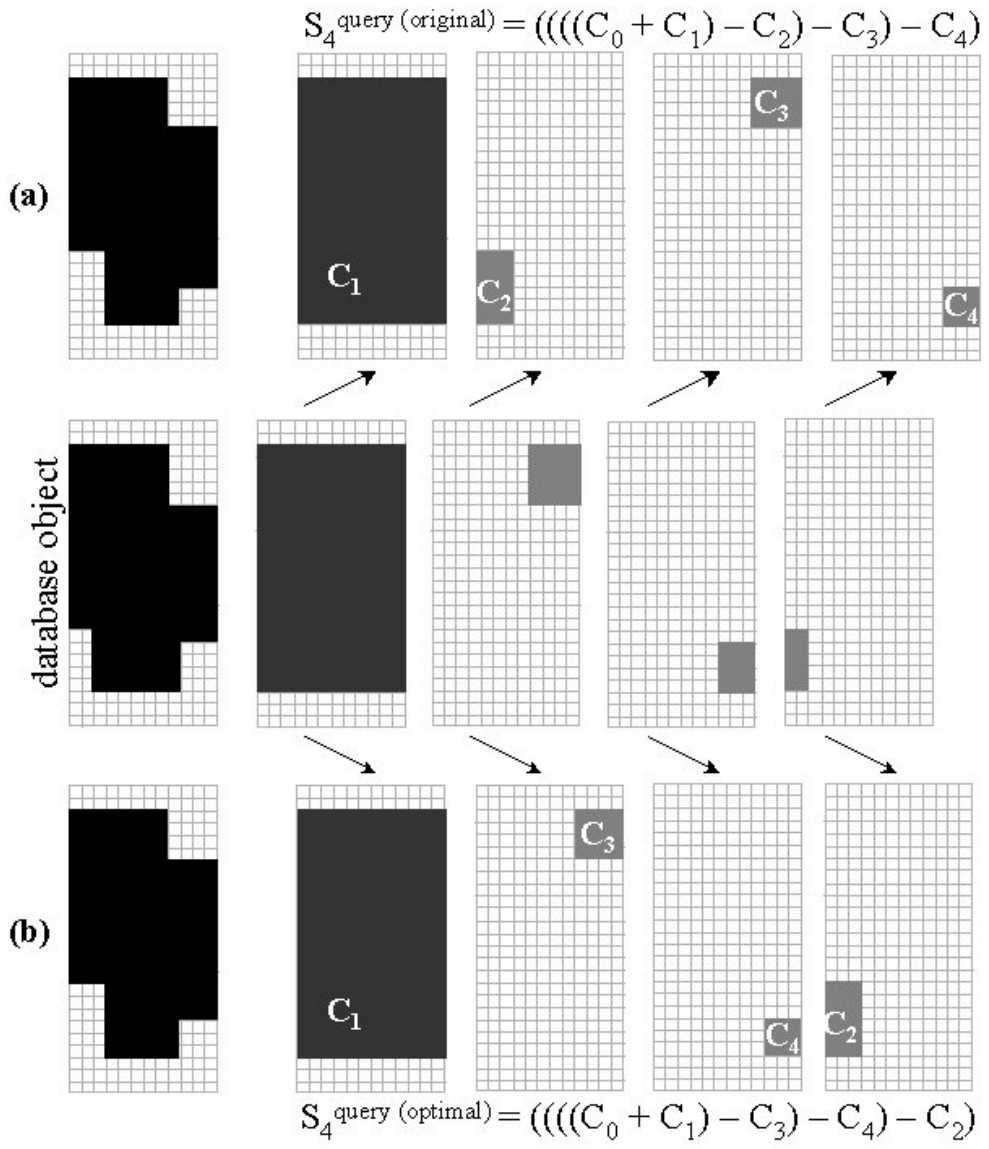


Figure 4.1: Examples demonstrating the advantage of free permutations.

taken as a result. We can obtain different representations by permuting the order of the found covers and choose the most "promising" orderings to create the query vectors. Though, the method may offer reasonable results in many cases, there is no guarantee that the ordering offering the minimum distance is included within this selection. Thus, the whole similarity measure is dependent on the criteria used to select the most "promising" orderings. Since there is no well defined selection criterion known so far, the solution does not necessarily offer a precisely defined similarity measure.

Another solution for the problem is to consider all possible permutations. Since the distance between two objects can now be considered as the minimum distance over all possible orderings, the distance is defined precisely this way.

Definition 4 (minimum Euclidian distance under permutation)
Let $exch : \mathbb{N}\times\mathbb{N}\times\mathbb{R}^{(k\cdot d)} \to \mathbb{R}^{(k\cdot d)}$ *be a function, where* $exch(i, j, \vec{x})$ *exchanges the* d *successive components beginning with dimension* $i \cdot d+1$ $(0 \leq i \leq k-1)$ *with the* d *successive components beginning with dimension* $j \cdot d+1 (0 \leq j \leq k-1)$ *of a vector* $\vec{x} \in \mathbb{R}^{(k\cdot d)}$. *Let* $Exch : \mathbb{R}^{(k\cdot d)} \to 2^{\mathbb{R}^{(k\cdot d)}}$ *be the function, that generates the set of all vectors that can be generated by applying* $exch(i, j, \vec{x})$ *arbitrary many times to a vector* $\vec{x}$ *using any combination for* i *and* j.

Then the minimum Euclidian distance under permutation $d_{\pi\text{-}euclid}$: $\mathbb{R}^{(k\cdot d)} \times \mathbb{R}^{(k\cdot d)} \to \mathbb{R}$ *is defined as follows:*

$$d_{\pi\text{-}euclid}(\vec{x}, \vec{y}) = \min_{\vec{z}\in Exch(\vec{y})}\{d_{euclid}(\vec{x}, \vec{z})\}$$

With a growing number of describing covers k, the processing time of considering all possible permutations increases exponentially, since there are $k!$ many permutations. With computation cost rising this rapidly, it is obvious that the description length k has to be kept low, which is not acceptable for all applications.

To guarantee that the permutation with the minimal distance is used, our approach does not work with one single feature vector, but with a set of feature vectors in lower dimensions. By treating the data objects as sets of d-dimensional feature vectors with a maximum cardinality of k, we introduce a new model for representing data objects in similarity search systems, the so called *vector set model*. Having obtained cover sequence approximations for our objects with a maximal length of k as described in Chapter 3, we represent each object by a vector set $X \subset \mathbb{R}^d$ with $|X| \leq k$.

The representation of extracted features as a set of vectors is a generalization of the use of just one large feature vector. It is always possible to restrict the model to a feature space, in which a data object will be completely represented

by just one feature vector. But in our application the possibilities of vector set representation allow us to model the dependencies between the extracted features more precisely. As the development of conventional database systems in the recent two decades has shown, the use of more sophisticated ways to model data can enhance both the effectiveness and efficiency for applications using large amounts of data. In our application the vector set representation is able to avoid the problems that occur by storing a set of covers according to a strict order. Therefore, it is possible to compare two objects more intuitively, causing a relatively small rise of calculation cost compared to the distance calculation in the one-vector model. Another advantage of our new approach is the better storage utilization. It is not necessary to force objects into a common size, if they are represented by sets of different cardinality. For our current application there is no need for dummy covers to fill up the feature vectors. If the quality of the approximation is optimal with less than the maximum number of covers, only this smaller number of vectors has to be stored and loaded. In the case of a one-vector representation avoiding dummies is not possible without further modifications of the access structures used. Furthermore, we are able to distinguish between the distance measure used on the feature vectors of a set and the way we combine the resulting distances between the single feature vectors. For example, this possibility might be useful when defining partial similarity, where it is only necessary to compare the closest $i < k$ vectors of a set.

In the following sections, we will discuss the concept of vector set representation in detail, with the goal of providing a high quality distance measure for vector-set-represented data and an algorithm for its efficient computation.

4.2 Distance Measures on Sets of Objects

There are already several distance measures proposed on sets of objects. Ideally a distance measure has the properties of a metric.

Definition 5 (metric)
Given a nonempty set of objects O, a metric is a mapping $d : O \times O \to \mathbb{R}$ such that for all $x, y, z \in O$:

1. $d(x, y) \geq 0$ *(definity)*
2. $d(x, y) = 0 \Leftrightarrow x = y$ *(reflexivity)*
3. $d(x, y) = d(y, x)$ *(symmetry)*
4. $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality)*

In [EM97] the authors survey the following distance functions, which are computable in polynomial time: the Hausdorff distance, the sum of minimal distances, the (fair-)surjection distance and the link distance. The Hausdorff distance is a metric, but does not seem to be suitable as a similarity measure, because it relies too much on the extreme positions of the elements of both sets. The last three distance measures are suitable for modelling similarity, but are not metric. This circumstance makes them unattractive, since there are only limited possibilities for processing similarity queries efficiently when using a non-metric distance function. In [EM97] the authors also introduce a method for expanding the distance measures into metrics, but as a side effect the complexity of distance calculation becomes exponential. Furthermore, the possibility to match several elements in one set to just one element in the compared set, is questionable when comparing sets of covers like in our application.

4.3 The Minimal Matching Distance

A distance measure on vector sets that demonstrates to be suitable for defining similarity in our application is based on the *minimal weight perfect matching* of sets. This well known graph problem can be applied here. First, let us introduce some notations.

Definition 6 (weighted complete bipartite graph)
A graph $G = (V, E)$ *consists of a (finite) set of vertices* V *and a set of edges* $E \subseteq V \times V$. *A* weighted graph *is a graph* $G = (V, E)$ *together with a weight function* $w : E \to \mathbb{R}$. *A* bipartite graph *is a graph* $G = (X \cup Y, E)$ *with* $X \cap Y = \emptyset$ *and* $E \subseteq X \times Y$. *A bipartite graph* $G = (X \cup Y), E)$ *is called* complete *if* $E = X \times Y$.

Definition 7 (perfect matching)
Given a bipartite graph $G = (X \cup Y, E)$ *a* matching *of* X *to* Y *is a set of edges* $M \subseteq E$ *such that no two edges in* M *share an endpoint, i.e.*

$$\forall (x_1, y_1), (x_2, y_2) \in M : x_1 = x_2 \Leftrightarrow y_1 = y_2.$$

A matching M *of* X *to* Y *is* maximal *if there is no matching* M' *of* X *to* Y *such that* $|M| < |M'|$. *A maximal matching* M *of* X *to* Y *is called a* complete *matching if* $|M| = \min\{|X|, |Y|\}$. *In the case* $|X| = |Y|$ *a complete matching is also called a* perfect *matching.*

Definition 8 (minimal weight matching)
Given a weighted bipartite graph $G = (X \cup Y, E)$ *with the weight function* $w : E \to \mathbb{R}$ *the* weight $w(M)$ *of a matching* M *of* X *to* Y *is the sum of the weights of the edges in* M*.*

$$w(M) = \sum_{(x,y)\in M} w(x, y)$$

A matching M *of* X *to* Y *is a* minimal weight *matching if there is no matching* M' *of* X *to* Y *such that* $w(M) > w(M')$*.*

In our application we build a complete bipartite graph $G = (X' \cup Y', X' \times Y')$ between two vector sets $X, Y \subset \mathbb{R}^d$ with $|X|, |Y| \leq k$. We set $X' = X \times \{1\}$ and $Y' = Y \times \{2\}$ in order to fulfill the property $X' \cap Y' = \emptyset$. The weight of each edge $((\vec{x}, 1), (\vec{y}, 2)) \in X' \times Y'$ in this graph G is defined by the distance $dist(\vec{x}, \vec{y})$ between the vectors $\vec{x} \in X$ and $\vec{y} \in Y$. For example the Euclidian distance can be used here. A perfect matching is a subset $M \subseteq X' \times Y'$ that connects each $\vec{x} \in X$ to exactly one $\vec{y} \in Y$ and vice versa. A minimal weight perfect matching is a matching with maximum cardinality and a minimum sum of weights of its edges. Since a perfect matching can only be found for sets of equal cardinality, it is necessary to introduce weights for unmatched nodes when defining a distance measure.

Definition 9 (enumeration of a set)
Let S *be any finite set of arbitrary elements. Then* π *is a mapping that assigns* $s \in S$ *a unique number* $i \in \{1, \ldots |S|\}$*. This is written as* $\pi(S) = (s_1, \ldots s_{|S|})$*. The set of all possible enumerations of* S *is named* $\Pi(S)$*.*

Definition 10 (minimal matching distance)
Let $V \subset \mathbb{R}^d$ *and let* $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ *be a distance function between two d-dimensional feature vectors. Let* $X, Y \in 2^V$ *with* $|X| = m$, $X = \{\vec{x}_1, \ldots \vec{x}_m\}$, $|Y| = n$, $Y = \{\vec{y}_1, \ldots \vec{y}_n\}$*. Furthermore, let* $w : V \to \mathbb{R}$ *be a weight function for the unmatched elements. Then the* minimal matching distance $d_{mm}^{w,dist}$: $2^V \times 2^V \to \mathbb{R}$ *is defined as follows: If* $|X| = m \geq n = |Y|$*, set*

$$d_{mm}^{w,dist}(X, Y) = \min_{\pi\in\Pi(X)} \left(\sum_{i=1}^{n} dist(\vec{x}_{\pi(i)}, \vec{y}_i) + \sum_{l=n+1}^{m} w(\vec{x}_{\pi(l)}) \right)$$

Otherwise, set $d_{mm}^{w,dist}(X, Y) = d_{mm}^{w,dist}(Y, X)$*.*

The weight function $w : V \to \mathbb{R}$ provides the penalty given to every unassigned element of the set having larger cardinality. Let us note that *minimal matching distance* is a specialization of the *netflow distance* which is introduced in [RB00]. The authors in [RB00] show that the netflow distance is a metric and that it is computable in polynomial time. Therefore, we derive the following lemma without further proof.

Lemma 1 *Let $V \subset \mathbb{R}^d$. The minimal matching distance $d_{mm}^{w,dist} : 2^V \times 2^V \to \mathbb{R}$ is a metric if the underlying distance function $dist : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a metric and the weight function $w : V \to \mathbb{R}$ meets the following conditions for all $\vec{x}, \vec{y} \in V$:*

1. $w(\vec{x}) > 0$
2. $w(\vec{x}) + w(\vec{y}) \geq dist(\vec{x}, \vec{y})$

The *minimum Euclidian distance under permutation* can be derived from the *minimal matching distance*. By selecting the squared Euclidian distance as distance measure on V and taking the squared Euclidian norm as weight function, the distance value calculated by the minimal matching distance is the same as the squared value of the minimum Euclidian distance under permutation. This follows exactly from the definitions of both distance measures. Let us note that it is necessary to extract the square root from this distance value to preserve the metric character.

4.4 The Kuhn-Munkres Algorithm

Though it was shown in [RB00] that the netflow distance can be calculated in polynomial time, it is not obvious how to achieve it. Since we are only interested in the minimal matching distance, it is sufficient to calculate a minimal weight perfect matching. Therefore, we apply the method proposed by Kuhn [Kuh55] and Munkres [Mun57].

For the remainder of this section we assume that we are given a weighted complete bipartite graph $G = (X \cup Y, X \times Y)$ with the weight function $w : X \times Y \to \mathbb{R}$. As we can provide weights for unmatched elements, we assume w.l.o.g. that X and Y have equal cardinality k.

The goal of the Kuhn-Munkres algorithm is to find a *maximal* weight matching in G. To obtain a *minimal* weight matching the following trick can be used. We replace the weight function w by the function w' with $w'(x, y) = -w(x, y)$ and apply the algorithm to G and w'.

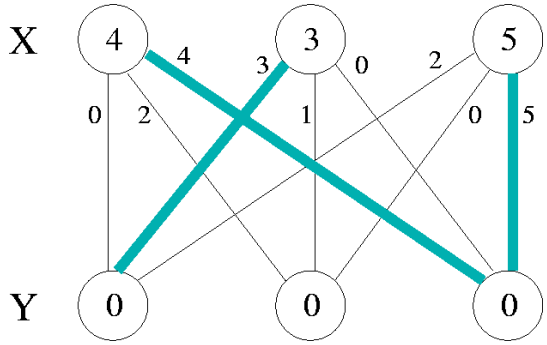


Figure 4.2: An example of a feasible vertex labeling and an equality subgraph.

Definition 11 (M-alternating path)
Given a matching M of X to Y an edge $(x, y) \in X \times Y$ is called matched *if $(x, y) \in M$,* unmatched *otherwise. An M-*alternating path *starts with an unmatched edge, then alternates between matched and unmatched edges and ends with an unmatched edge.*

If the edges of an M-alternating path are flipped in M, i.e. matched edges become unmatched and vice versa, a matching M' is obtained with $|M'| = |M| + 1$.

Definition 12 (feasible vertex labeling)
A feasible vertex labeling *in G is a function $l : X \cup Y \to \mathbb{R}$ such that*

$$\forall x \in X, y \in Y : l(x) + l(y) \geq w(x, y)$$

It is always possible to find a feasible vertex labeling. One way to do this is to set $l(y) = 0$ for all $y \in Y$ and for each $x \in X$ take the maximum value in the corresponding row of edge weights, i.e.

$$\begin{array}{ll} l(x) = \max_{y \in Y} w(x, y) & \text{for } x \in X \\ l(y) = 0 & \text{for } y \in Y \end{array}$$

An example of a feasible vertex labeling is depicted in Figure 4.2. If l is a feasible labeling, we denote by G_l the subgraph of G which contains those edges where $l(x) + l(y) = w(x, y)$, together with the endpoints of these edges. This graph G_l is called the *equality subgraph* for l. In Figure 4.2 the edges of G_l are shaded. For $S \subseteq X$ we denote by $J(G_l, S)$ the set $\{y \in Y \mid x \in S \wedge (x, y) \in G_l\}$ of all vertices in Y which are adjacent to the vertices in S.

Lemma 2 *If l is a feasible vertex labeling for G, and M is a perfect matching of X to Y with $M \subseteq G_l$, then M is a maximal weight perfect matching of X to Y.*

PROOF: We must show that no other complete matching can have weight greater than M. Let any complete matching M' of X to Y be given. Then

$$\begin{aligned} w(M') &= \sum_{(x,y)\in M'} w(x,y) \\ &\leq \sum_{(x,y)\in M'} (l(x)+l(y)) \quad \text{(feasibility of l)} \\ &= \sum_{(x,y)\in M} (l(x)+l(y)) \quad \text{(all } l(x), l(y) \text{ summed in either matching)} \\ &= \sum_{(x,y)\in M} w(x,y) \quad \text{(since } M \subseteq G_l) \\ &= w(M) \end{aligned}$$

□

Thus the problem of finding an maximal weight perfect matching is reduced to the problem of finding a feasible vertex labeling whose equality subgraph contains a perfect matching of X to Y. Using this result, the Kuhn-Munkres algorithm (also known as the Hungarian method) works like this:

1. Start with an arbitrary feasible vertex labeling l, determine G_l, and choose an arbitrary matching M in G_l.

2. If M is complete for G, then M is optimal. Stop. Otherwise, there is some unmatched $x \in X$. Set $S = \{x\}$ and $T = \emptyset$.

3. If $J(G_l, S) \neq T$, go to step 4. Otherwise, $J(G_l, S) = T$. Find

$$\alpha_l = \min_{x\in S, y\in Y\setminus T} \{l(x) + l(y) - w(x,y)\}$$

and construct a new labeling l' by

$$l'(v) = \begin{cases} l(v) - \alpha_l & \text{for } v \in S \\ l(v) + \alpha_l & \text{for } v \in T \\ l(v) & \text{otherwise} \end{cases}$$

Note that $\alpha_l > 0$ and $J(G'_l, S) \neq T$. Replace l by l' and G_l by G'_l.

4. Choose a vertex y in $J(G_l, S)$, not in T. If y is matched in M, say with $z \in X$, replace S by $S \cup \{z\}$ and T by $T \cup \{y\}$, and go to step 3. Otherwise, there will be an M-alternating path from x to y, and we may use this path to find a larger matching M' in G_l. Replace M by M' and go to step 2.

When necessary, edges are added to the equality subgraph G_l by constructing a new feasible labeling in step 3. This way a perfect matching will eventually be found. Since there are at most k phases in which an M-alternating path is constructed and each phase can be computed in $O(k^2)$ the all over complexity of a distance calculation using the method of Kuhn and Munkres is $O(k^3)$ in the worst case. Let us note that for larger numbers of k this is far better than the previously mentioned method on $k!$ many permutations.

Chapter 5

Efficient Query Processing on Vector Set Data

Though we discussed the time for a single distance calculation between vector-set-represented objects in Section 4.2, the problem of efficiently answering similarity queries in large databases is still open. Since it is necessary here, to locate the objects belonging to the result in comparably short time, the use of index structures that avoid comparing a query object to the complete database is mandatory. For one-vector-represented data objects there exists a wide variety of index structures that are suitable for answering similarity queries efficiently like the the TV-tree [LJF94], the X-tree [BKK96] or the IQ-tree [BBJ+00]. But unfortunately, those index structures cannot be used directly to retrieve vector-set-represented objects.

To accelerate similarity queries on vector-set-represented objects, the simplest approach is the use of more general access structures. Since the minimal matching distance is a metric for the right choice of distance and weight function, the use of index structures for metric objects like the M-tree [CPZ97] offers a good possibility here. Another approach is the use of the above mentioned high dimensional index structures, for querying subtasks of the complete similarity query.

5.1 Query Types

There are two types of queries that are relevant with respect to similarity search: similarity range queries and k-nearest neighbor queries (k-nn queries). We provide formal definitions for these fundamental similarity query types. Let O be the domain of the objects that may occur as database objects or

query objects. Let $simdist : O \times O \rightarrow \mathbb{R}$ be a similarity distance function (cf. Definition 3). By $\mathrm{DB} \subseteq O$, let us denote an actual database.

5.1.1 Similarity Range Queries

We specify similarity range queries by a query object q and a range value ε, and the answer set is defined to contain all the objects from the database that have a distance to the query object q of less than or equal to ε:

Definition 13 (similarity range query)
For a query object q and a query range ε, the similarity range query returns the set:

$$sim_q(\varepsilon) = \{o \in \mathrm{DB} \mid simdist(o, q) \leq \varepsilon\}$$

From a geometric point of view, the given distance function and the range value ε define a region around the query object q. Thus, the similarity range query reports all data objects which are contained in this region. Processing range queries on a multidimensional access method is performed as follows: The search algorithm starts from the root and then traverses the tree recursively. At each directory node, the entries (Minimal Bounding Boxes) which intersect the query region are identified and the search is directed downwards. At data nodes, all objects which are contained in the query region are finally reported.

Note that for the similarity range query, the distance values of the resulting objects is bounded by the query range ε, but the number of answers is previously unknown. The result may be empty if no object has a similarity distance to the query object that is less or equal to the query range, and it may enclose the overall database if no object has a distance to the query object that is greater than the query range.

5.1.2 k-Nearest Neighbor Queries

Since similarity distance functions are quite abstract, the user must be experienced with typical similarity distances in order to specify useful similarity range queries. This is the reason why k-nearest neighbor queries are becoming more and more important for similarity search in large databases of complex objects. The k-nearest neighbor query retrieves, for any query object, the k most similar objects from the database and can be defined as follows:

Definition 14 (k-nearest neighbor query)
For a query object $q \in O$ and a query parameter k, the k-nearest neighbor query returns the set $\mathrm{NN}_q(k) \subseteq \mathrm{DB}$ that exactly contains k objects from the

database, and for which the following condition holds:

$$\forall o_1 \in \mathrm{NN}_q(k), o_2 \in \mathrm{DB} \setminus \mathrm{NN}_q(k) : simdist(o_1, q) \leq simdist(o_2, q)$$

Note that possibly several objects in the database exist which have the same distance to the query object as the k-th object in the answer set. In this case, the k-th object is a non-deterministic selection of one of those equally distanced objects. An approach to process k-nearest neighbor queries available from the literature is the similarity ranking algorithm proposed in [HS95] which can easily be adapted to process k-nearest neighbor queries by ranking exactly k data objects. The basic idea of this algorithm is to visit nodes in the order of their minimum distance from the query object to any possible object inside a node. The algorithm is generally designed for multidimensional access methods that hierarchically manage page regions.

5.2 Multi-Step Query Processing

To speed up similarity search in presence of complex similarity distance functions like our minimal matching distance, the use of a multi-step query processing architecture is recommended. Following this paradigm, a *feature distance function* (also called *filter distance function*) is employed that serves as a approximation of the complex object distance function. One or more filter steps produce sets of candidates that are exactly evaluated in one or more subsequent refinement steps. Refinement steps discard false positive candidates (*false hits*), but do not reconstruct false negatives (*false drops*) that have been dismissed by the filter step. Therefore, the crucial correctness requirement for any filter step is to prevent false drops. In other words, no actual result may be dismissed from the set of candidates. The *lower-bounding property* of filter and object distance functions is a criterion to ensure the correctness of a filter step.

Definition 15 (lower-bounding property)
Let O be a set of objects. A feature distance function d_f and an object distance function d_o fulfill the lower-bounding property *if d_f underestimates d_o in any case, i.e. for all objects $o_1, o_2 \in O$:*

$$d_f(o_1, o_2) \leq d_o(o_1, o_2)$$

An algorithm that obeys this principle has been developed for k-nearest neighbor queries [SK98]. The algorithm is proven to be optimal, i.e. it produces only the minimum number of candidates. Thus, expensive evaluations

k-NearestNeighborSearch (q, k) // optimal algorithm

1 initialize *ranking* = index.increm_ranking ($F(q)$, d_f)
2 initialize *result* = new sorted_list ⟨key, object⟩
3 initialize $d_{\max} = \infty$
4 **while** o = *ranking*.getnext **and** $d_f(o, q) \leq d_{\max}$ **do**
5 **if** $d_o(o, q) \leq d_{\max}$ **then** *result*.insert ($d_o(o, q)$, o)
6 **if** *result*.length $\geq k$ **then** $d_{\max}$ = *result*[k].key
7 remove all entries from *result* where key > $d_{\max}$
8 **endwhile**
9 report all entries from *result* where key $\leq d_{\max}$

Figure 5.1: Optimal multi-step k-nearest neighbor algorithm.

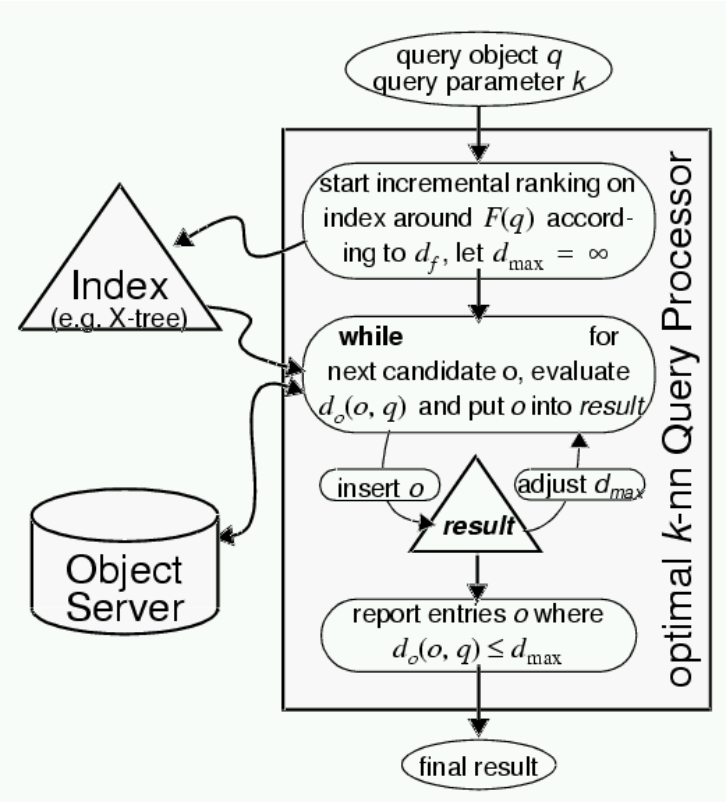


Figure 5.2: Optimal multi-step query processor for k-nearest neighbor search.

```
method RTree :: ranking (Object query)
1  PriorityQueue queue;
2  queue.insert (0, root);
3  wait (getnext_is_called);
4  while not queue.isempty() do
5     Element first = queue.pop();
6     case first isa
7        DirNode:
8           foreach child in first do
9              queue.insert (mindist (query, child.box), child);
10       DataNode:
11          foreach object in first do
12             queue.insert (distance (query, object), object);
13       Object:
14          report (first);
15          wait (getnext_is_called);
16    end
17 enddo
```

Figure 5.3: Incremental ranking query processing on R-trees.

of unnecessary candidates are avoided. Figure 5.1 provides a pseudocode description of the procedure whereas in Figure 5.2, the algorithm is illustrated schematically.

The algorithm has two basic components: By the *incremental ranking query* on the underlying access method, candidates are iteratively generated in ascending order according to their feature distance d_f to the query object. The second major component is the *result* list that manages the k nearest neighbors of the query object q within the current candidate set, keeping step with the candidate generation. The current k-th distance is held in d_{max} which is set to infinity until the first k candidates are retrieved from the index and evaluated. The termination is controlled by the refinement step in order to guarantee the minimum number of candidates

For the incremental ranking query an algorithm derived from [HS95] is used. Pseudocode is given in Figure 5.3. Incremental similarity ranking is a similarity query type that corresponds to a *give-me-more* facility. After an initialization, the ranked objects may be retrieved by a sequence of *getnext* calls. Formally,

performing an incremental similarity ranking means the partial materialization of a q-ranking which may be defined as follows.

Definition 16 (q-ranking)
Let O be a set of objects. Given a query object $q \in O$ and a database DB $\subseteq O$ *containing $N = |\text{DB}|$ objects, a q-ranking of the database* DB *is a bijection $ranked_q : I_N \rightarrow \text{DB}$ that maps the index set $I_N = \{1, \ldots N\}$ d_q-monotonously onto the database* DB*, i.e. ascendingly ordered by the distance of the objects to the query object q.*

We simplify our notation by writing $ranked_q(i) = o_i$ for the object o_i that is ranked to position i and denote the image of the index set I_k by $ranked_q(I_k) = \{o_1, \ldots o_k\}$. Using this abbreviation, the d_q-monotony appears as follows:

$$\forall i, j \in I_N : i < j \Rightarrow d(o_i, q) \leq d(o_j, q)$$

When processing incremental similarity ranking queries, the object $o_k = ranked_q(k)$ is reported as response to the k-th *getnext* call. Note that the q-ranking is not totally determined if some objects share the same distance to the query object q.

5.3 A Filter Step for Vector Set Data

In the following we will introduce a filter step for query processing on vector-set-represented data objects that is based on the relation between a set of d-dimensional vectors and its *extended centroid*.

Definition 17 (extended centroid)
Let $V \subset \mathbb{R}^d$ and $\vec{\omega} \in \mathbb{R}^d \setminus V$. Let $X \in 2^V$ be a vector set with $|X| \leq k$ and $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$ Then the extended centroid *$C_{k,\vec{\omega}}(X)$ is defined as follows:*

$$C_{k,\vec{\omega}}(X) = \frac{\sum_{i=1}^{|X|} \vec{x}_i + (k - |X|) \cdot \vec{\omega}}{k}$$

Note how the vector $\vec{\omega}$ is used as a "dummy" vector to fill up vector sets having a cardinality of less than k.

Lemma 3 *Let $V \subset \mathbb{R}^d$ and $\vec{\omega} \in \mathbb{R}^d \setminus V$. Let $w_{\vec{\omega}} : V \rightarrow \mathbb{R}$, $w_{\vec{\omega}}(\vec{x}) = \|\vec{x} - \vec{\omega}\|_2$, be a weight function. Furthermore, let $X, Y \in 2^V$, $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\}$ be two vector sets with $|X|, |Y| \leq k$, let $C_{k,\vec{\omega}}(X)$, $C_{k,\vec{\omega}}(Y)$ be their*

extended centroids and let $d_{mm}^{d_{euclid},w_{\vec{\omega}}} : 2^V \times 2^V \to \mathbb{R}$ *be the minimal matching distance using* $w_{\vec{\omega}}$ *as a weight function defined on* V*. Then the following inequality holds:*

$$k \cdot \|C_{k,\vec{\omega}}(X) - C_{k,\vec{\omega}}(Y)\|_2 \leq d_{mm}^{d_{euclid},w_{\vec{\omega}}}(X,Y)$$

PROOF: Let π be the enumeration of the indices of X that groups the x_i to y_i according to the minimum weight perfect matching. We assume w.l.o.g. $|X| = m \geq n = |Y|$.

$$\begin{aligned}
& k \cdot \|C_{k,\vec{\omega}}(X) - C_{k,\vec{\omega}}(Y)\|_2 \\
= \; & k \cdot \| \tfrac{\sum_{i=1}^{m} \vec{x}_{\pi(i)} + (k-m)\cdot\vec{\omega}}{k} - \tfrac{\sum_{i=1}^{n} \vec{y}_i + (k-n)\cdot\vec{\omega}}{k} \|_2 \\
= \; & \| \textstyle\sum_{i=1}^{m} \vec{x}_{\pi(i)} - \sum_{i=1}^{n} \vec{y}_i - (m-n)\cdot\vec{\omega} \|_2 \\
= \; & \| \textstyle\sum_{i=1}^{n} \vec{x}_{\pi(i)} - \sum_{i=1}^{n} \vec{y}_i + \sum_{i=n+1}^{m} \vec{x}_{\pi(i)} - \sum_{i=n+1}^{m} \vec{\omega} \|_2 \\
\overset{\text{tri. ineq.}}{\leq} \; & \| \textstyle\sum_{i=1}^{n} (\vec{x}_{\pi(i)} - \vec{y}_i) \|_2 + \| \sum_{i=n+1}^{m} (\vec{x}_{\pi(i)} - \vec{\omega}) \|_2 \\
\overset{\text{tri. ineq.}}{\leq} \; & \textstyle\sum_{i=1}^{n} \|\vec{x}_{\pi(i)} - \vec{y}_i\|_2 + \sum_{i=n+1}^{m} \|\vec{x}_{\pi(i)} - \vec{\omega}\|_2 \\
= \; & \textstyle\sum_{i=1}^{n} \|\vec{x}_{\pi(i)} - \vec{y}_i\|_2 + \sum_{i=n+1}^{m} w_{\vec{\omega}}(\vec{x}_{\pi(i)}) \\
= \; & d_{mm}^{d_{euclid},w_{\vec{\omega}}}(X,Y)
\end{aligned}$$

□

The lemma proves that the Euclidian distance between the extended centroids multiplied with the cardinality of the larger set is a lower bound (cf. Definition 15) for the minimal matching distance under the named preconditions. Therefore, when computing e.g. ε-range queries, we do not need to examine objects whose extended centroids have a distance to the query object q that is larger than ε/k. A good choice of $\vec{\omega}$ for our application is $\vec{0}$, since it has the shortest average distance within the position and has no volume. Since there are no covers having no volume in any data object, the conditions for the metric character of minimum matching distance are satisfied.

5.4 Implementation

To implement the filter step, we store the extended centroids in a 6-dimensional X-tree [BKK96]. The X-tree (eXtended node tree) is an index structure supporting efficient query processing of high-dimensional data. The goal is to

support not only point data but also extended spatial data and therefore, the X-tree uses the concept of overlapping regions. It is crucial to avoid overlap in the directory in order to improve the indexing of high-dimensional data. The X-tree therefore avoids overlap whenever it is possible without allowing the tree to degenerate; otherwise, the X-tree uses extended variable size directory nodes, so-called supernodes. In addition to providing a directory organization which is suitable for high-dimensional data, the X-tree uses the available main memory more efficiently in comparison to using a cache.

The X-tree may be seen as a hybrid of a linear array-like and a hierarchical directory. It is well established that in low dimensions the most efficient organization of the directory is a hierarchical organization. The reason is that the selectivity in the directory is very high which means that, e.g. for point queries, the number of required page accesses directly corresponds to the height of the tree. This, however, is only true if there is no overlap between directory rectangles which is the case for a low dimensionality. It is also reasonable, that for very high dimensionality a linear organization of the directory is more efficient. The reason is that due to the high overlap, most of the directory if not the whole directory has to be searched anyway. If the whole directory has to be searched, a linearly organized directory needs less space and may be read much faster from disk than a blockwise reading of the directory. For medium dimensionality, an efficient organization of the directory would probably be partially hierarchical and partially linear. The problem is to dynamically organize the tree such that portions of the data which would produce high overlap are organized linearly and those which can be organized hierarchically without too much overlap are dynamically organized in a hierarchical form. The algorithms used in the X-tree are designed to automatically organize the directory as hierarchical as possible, resulting in a very efficient hybrid organization of the directory.

The overall structure of the X-tree is presented in Figure 5.4. The data nodes of the X-tree contain rectilinear minimum bounding rectangles (MBRs) together with pointers to the actual data objects, and the directory nodes contain MBRs together with pointers to sub-MBRs. The X-tree consists of three different types of nodes: data nodes, normal directory nodes, and supernodes. Supernodes are large directory nodes of variable size (a multiple of the usual block size). The basic goal of supernodes is to avoid splits in the directory that would result in an inefficient directory structure. The alternative to using larger node sizes are highly overlapping directory nodes which would require to access most of the son nodes during the search process. This, however, is more inefficient than linearly scanning the larger supernode. The X-tree only

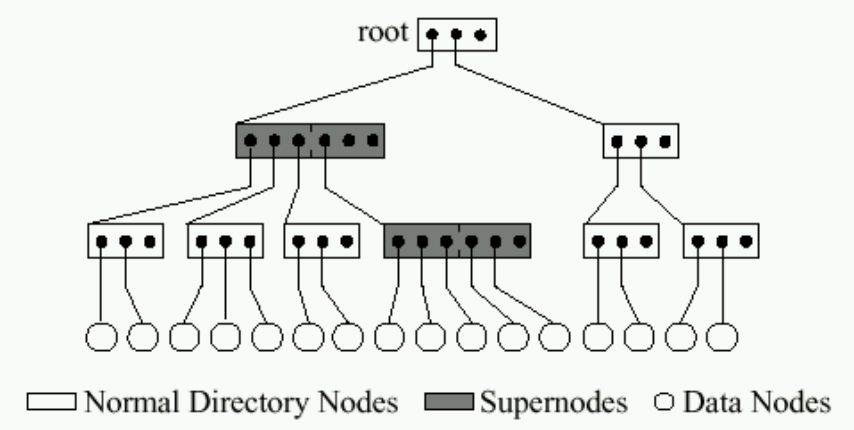


Figure 5.4: Structure of the X-tree.

consists of larger nodes where actually necessary. As a result, the structure of the X-tree may be rather heterogeneous as indicated in Figure 5.4.

Since this index structure provides high performance for similarity queries, it offers an efficient way to determine the keys of the candidate sets of feature vectors during the filter step. Afterwards we load the vector sets themselves to determine the membership of the object within the result. Since there exist established algorithms for ε-range [KSF+96] and k-nearest neighbor queries [SK98] using filter steps, the method is able to speed up both kinds of queries.

Chapter 6

Experimental Evaluation

In this chapter, we present the results of an exhaustive evaluation based on nearest neighbor queries and clustering. We also show how the use of the hierarchical clustering algorithm *OPTICS* (*O*rdering *P*oints *T*o *I*dentify the *C*lustering *S*tructure) [ABKS99] is applicable for a more objective evaluation of similarity models than sample k-nearest neighbor queries.

6.1 Data Sets

We evaluated the three proposed models on the basis of two real-world datasets. The first one – in the following refered to as *Car Dataset* – contains approximately 200 CAD objects from a German car maufacturer. The Car Dataset contains several groups of intuitively similar objects, e.g. a set of tires, doors, fenders, engine blocks and kinematic envelopes of seats.

The second dataset contains 5,000 CAD objects from an American aircraft producer and in the following is called *Aircraft Dataset*. This dataset contains many small objects (e.g. nuts, bolts, etc.) and a few large ones (e.g. wings).

For the volume model and the solid-angle model, we used a raster resolution of $r = 30$. Thus, $|V^o|$ ranges from 1 to $30^3 = 27,000$ for each object o. Furthermore, the data space is partitioned into $p = 3$ cells in each dimension. We retrieve $3^3 = 27$-dimensional feature vectors for the volume model and the solid-angle model. The eigen value model yields feature vectors of dimensionality $3 \cdot 3^3 = 81$. Using the cover sequence model and the vector set model, the data space of both datasets contains objects represented as voxel approximations using a raster resolution of $r = 15$. Here, $|V^o|$ ranges from 1 to $15^3 = 3375$ for each object o. These values were optimized to the quality of the evaluation results.

6.2 Methods for Evaluating the Effectiveness of Similarity Models

In general, similarity models can be evaluated by computing k-nearest neighbor queries. A drawback of this evaluation approach is that the quality measure of the similarity model depends on the results of few similarity queries and, therefore, on the choice of the query objects. A model may perfectly reflect the intuitive similarity according to the chosen query objects and would be evaluated as "good" although it produces disastrous results for other query objects. As a consequence, the evaluation of similarity models with sample k-nn queries is subjective and error-prone.

A better way to evaluate and compare several similarity models is to apply a clustering algorithm. Clustering groups a set of objects into classes where objects within one class are similar and objects of different classes are dissimilar to each other. The result can be used to evaluate which model is best suited for which kind of objects.

We evaluated our models using both approaches. In the following, we first discuss the results of the evaluation based on k-nn queries (see Section 6.2.1). In Section 6.2.2 we then evaluate our models using the hierachical clustering algorithm OPTICS [ABKS99].

6.2.1 k-Nearest Neighbor Queries

The notion of k-nn queries has been defined in Section 5.1. We evaluated the three models described in Section 2.5 using k-nn queries with $k = 5$. We performed the 5-nn queries on the Car Dataset and evaluated the resulting objects according to our intuitive notion of similarity.

The results of a 5-nn query for the volume model and the solid-angle model are presented in Figure 6.1. We achieved satisfying results for each model depending on the query object. For a tire, for example, the volume model performs very well, yielding objects that are intuitively very similar to the query object (cf. Figure 6.1(a)). Comparably good results are also produced by the solid-angle model for a part of the fender (cf. Figure 6.1(b)).

Although all three models deliver rather accurate results for the chosen query objects, we also see in Figure 6.1 that these results are delusive. Figure 6.1(c) shows a nearest neighbor query for an object which belongs to a cluster (cf. Figure 6.9), i.e. there exist several similar parts to this object. The volume model does not recognize this. Furthermore, there might be objects where a nearest neighbor query does not yield any intuitively similar parts (cf. Figure

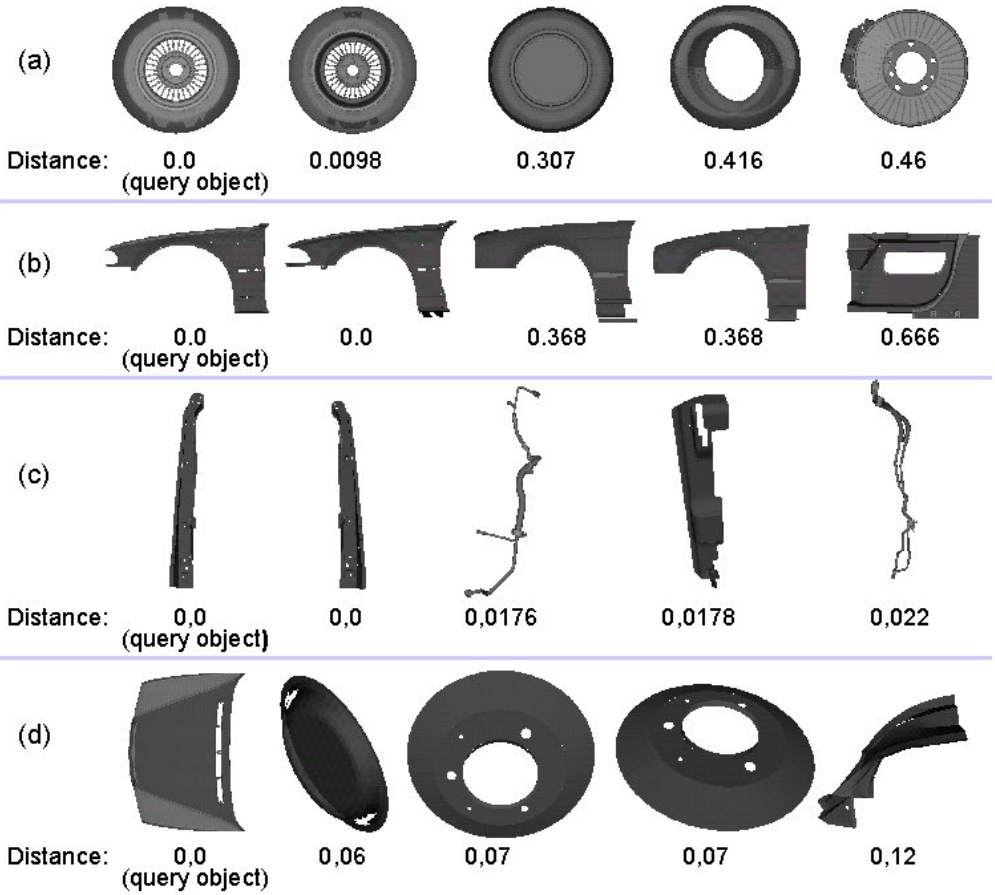


Figure 6.1: Results of sample 5-nn queries. The particular distances to the query object are depicted below each object.

6.1(d)). Obviously, we should not discard a similarity model if the chosen query object belongs to noise. This confirms the assumption that the method of evaluating similarity models using several k-nn queries is subjective and error-prone, due to its dependency on the choice of the query objects.

In the next section, we introduce hierarchical clustering in order to overcome the above described difficulties.

6.2.2 Clustering

A more objective way to evaluate and compare several similarity models is to apply a clustering algorithm. Clustering groups a set of objects into classes

where objects within one class are similar and objects of different classes are dissimilar to each other. The result can be used to evaluate which model is best suited for which kind of objects. In addition, using clustering the evaluation of the models is based on the whole data set and not only on few sample objects.

For evaluation we used the density-based, hierarchical algorithm OPTICS [ABKS99]. The algorithm is similar to hierarchical Single-Link clustering methods [JD88] and is an extension of the clustering algorithm DBSCAN [EKSX96].

We choose OPTICS due to the following reasons. First, OPTICS is – in contrast to most other algorithms – relatively insensitive to its two input parameters. The authors in [ABKS99] state that the input parameters just have to be big enough to retrieve good results. Second, 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 [McQ67]).

The output of OPTICS is a linear ordering of the database objects minimizing a binary relation called *reachability* which is in most cases equal to the minimum distance of each database object to one of its predecessors in the ordering. This idea is similar to the Single-Link method but instead of a dendrogram, the resulting reachability-plot is much easier to analyse. The reachability values can be plotted for each object of the cluster-ordering computed by OPTICS. Valleys in this plot indicate clusters: objects having a small reachability value are more similar to their predecessor objects than objects having a higher reachability value.

The reachability plot generated by OPTICS can be cut at any level ε parallel to the abscissa. It represents the density-based clusters according to the density threshold ε: A consecutive subsequence of objects having a smaller reachability value than ε belong to the same cluster. An example is presented in Figure 6.2: For a cut at the level ε_1 we retrieve 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 as A_1 and A_2 and cluster B has decreased its size. Usually, for evaluation purposes, a good value for ε would yield as many clusters as possible.

6.3 The OPTICS Algorithm

In this section we provide a detailed description of the clustering algorithm OPTICS. The algorithm is based on the idea of density-based clustering. We give formal definitions of the underlying concepts and discuss the effects of parameter settings on the resulting cluster ordering.

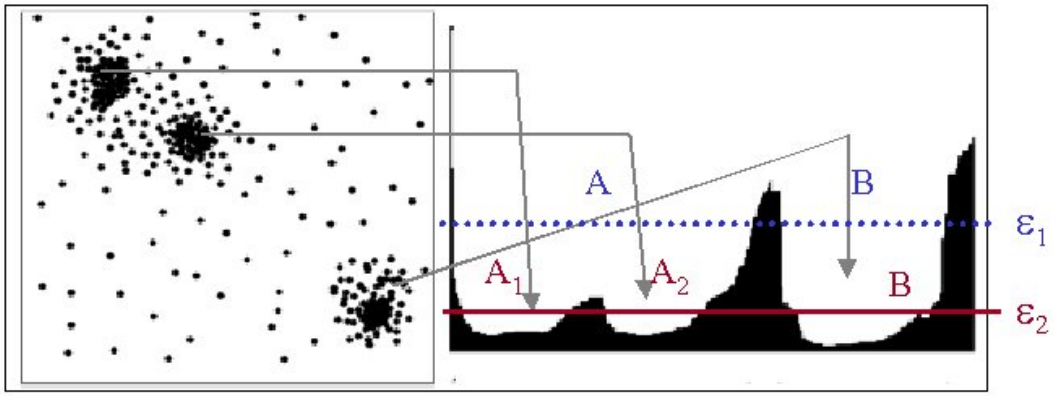


Figure 6.2: Reachability plot (right) computed by OPTICS for a sample 2-D dataset (left).

6.3.1 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 introduced in the following.

Definition 18 (directly density-reachable)
Object p is directly density-reachable *from object q with regard to ε and $MinPts$ in a set of objects O if*

1. $p \in N_\varepsilon(q)$
2. $|N_\varepsilon(q)| \geq MinPts$

The condition $| N_\varepsilon(q) | \geq MinPts$ 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 19 (density-reachable)
An object p is density-reachable *from an object q with regard to ε and $MinPts$ in the set of objects O, if there is a chain of objects $p_1, \ldots p_n$ with $p_1 = q, p_n = p$, such that $p_i \in O$ and p_{i+1} is directly density-reachable from p_i with regard to ε and $MinPts$.*

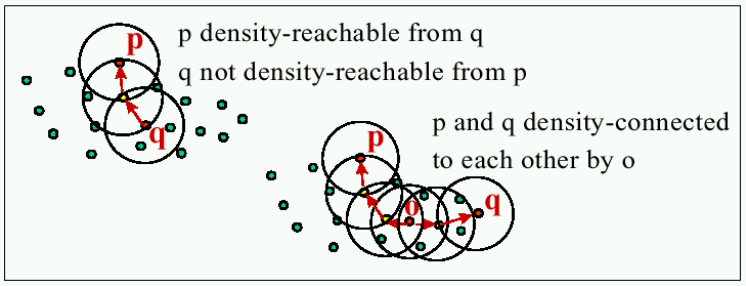


Figure 6.3: Density-reachability and connectivity.

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 20 (density-connected)
Object p is density-connected *to object q with regard to ε and $MinPts$ in the set of objects O, if there is an object $o \in O$ such that both p and q are density-reachable from o with regard to ε and $MinPts$ in O.*

Density-connectivity is a symmetric relation. Figure 6.3 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 with regard to density-reachability and the noise is the set of objects not contained in any cluster.

Definition 21 (cluster and noise)
Let O be a set of objects. A cluster *C with regard to ε and $MinPts$ in O is a non-empty subset of O satisfying the following conditions:*

1. *Maximality: for all $p, q \in O$: if $p \in C$ and q is density-reachable from p with regard to ε and $MinPts$, then also $q \in C$.*

2. *Connectivity: for all $p, q \in C$: p is density-connected to q with regard to ε and $MinPts$ in O.*

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

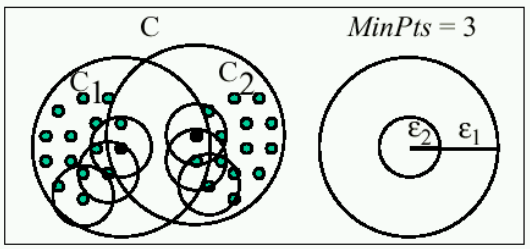


Figure 6.4: "Nested" density-based clusters.

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

The algorithm DBSCAN [EKSX96], 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 O which are density-reachable from an arbitrary core object in the cluster (c.f. lemma 1 and 2 in [EKSX96]). 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.

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 6.4, 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.

Consequently, the DBSCAN algorithm could be extended 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.

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. $0 < \varepsilon_i \leq \varepsilon$). 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 22 (core-distance)
Let p be an object from a database DB*, 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 $MinPts$-th neighbor. Then, the* core-distance *of p is defined as*

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

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

Definition 23 (reachability-distance)
Let p and o be objects from a database DB*, 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*

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

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

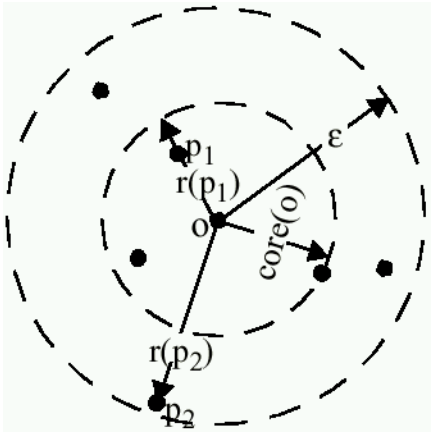


Figure 6.5: core-distance $core(o)$, reachability distances $r(p_1, o)$, $r(p_2, o)$ for $MinPts = 4$.

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 6.5 illustrates the notions of core-distance and reachability-distance.

6.3.2 Reachability Plots and Parameters

The OPTICS algorithm generates the augmented cluster-ordering consisting of the ordering of the points, the reachability-values and the core-values. In the following only the ordering and the reachability-values are needed. To simplify the notation, we specify them formally.

Definition 24 (results of the OPTICS algorithm)
Let DB *be a database containing n points. The OPTICS algorithm generates an ordering of the points* $o : \{1, \ldots n\} \rightarrow$ DB *and corresponding reachability-values* $r : \{1, \ldots n\} \rightarrow \mathbb{R}_0^+$.

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 r are plotted for each object in the cluster-ordering o. Figure 6.2 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

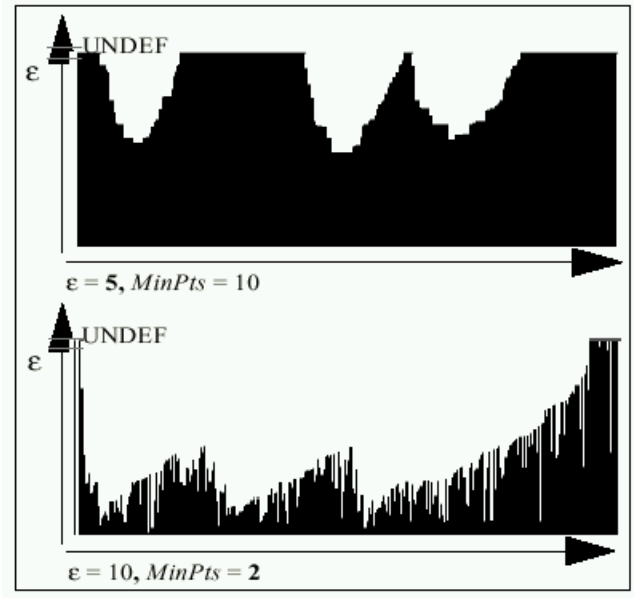


Figure 6.6: Effects of parameter settings on the cluster ordering.

set depicted in figure 6.2 (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 6.6 shows the effects of different parameter settings on the reachability-plot for the same data set used in figure 6.2. 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 6.2, 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.

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

$$Volume_S = \frac{Volume_{DS}}{N} \cdot k$$

and the volume of a d-dimensional hypersphere S having a radius r is

$$Volume_{S(r)} = \frac{\sqrt{\pi^d}}{\Gamma(\frac{d}{2}+1)} \cdot r^d$$

where Γ denotes the Gamma-function known from analysis, which interpolates the factorial function.

$$\Gamma(x) := \int_0^\infty t^{x-1} e^{-t}\, dt \quad \text{for} \quad x > 0$$

The radius r can be computed as

$$r = \sqrt[d]{\frac{Volume_{DS} \cdot k \cdot \Gamma(\frac{d}{2}+1)}{N \cdot \sqrt{\pi^d}}}$$

The effect of the $MinPts$-value on the visualization of the cluster-ordering can be seen in figure 6.6. 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.

6.4 Evaluation of the Effectiveness

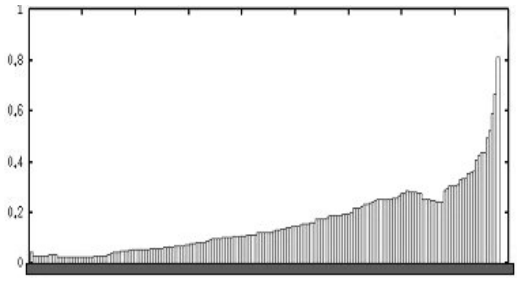
The reachability plots generated by OPTICS for all models are depicted in Figure 6.7 and 6.8.

Obviously, the volume model performs rather ineffective. The plots computed by OPTICS when applying the model on the Car Dataset and the Aircraft Dataset are depicted in Figure 6.7(a) and 6.7(b). Both plots show a minimum of structure indicating that the volume model cannot satisfyingly represent the intuitive notion of similarity.

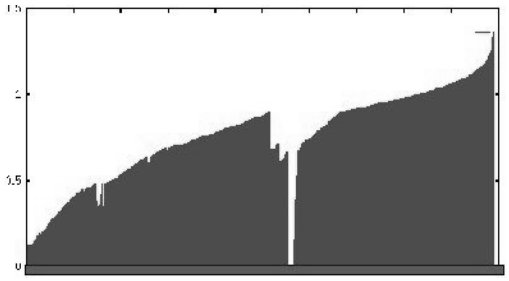
The solid-angle model performs slightly better. On the Car Dataset, OPTICS found three clusters denoted as A, B, and C in Figure 6.7(c). We analyzed these clusters by picking out some samples of the objects grouped in each cluster. The result of this evaluation on the Car Dataset is presented in Figure 6.9. As it can be seen, the objects in clusters A and C are intuitively similar but the objects in B are not. Furthermore, there are clusters of intuitively similar objects (e.g. doors), which are not detected. Evaluating the solid-angle model using the Aircraft Dataset we made similar observations. The reachability plot computed by OPTICS (cf. Figure 6.7(d)) yields a clustering with a large number of hierarchical classes. But the analysis of the objects within each cluster displays that intuitively dissimilar objects are treated as similar. A further observation is the following: objects, that are intuitively similar, are clustered in different groups. This suggests the conclusion that the solid-angle model is also rather unsuitable as a similarity model for our real-world test datasets.

In contrast to the volume model and the solid-angel model, the eigen value model yields valuable results. The plots computed by OPTICS for the eigen value Model are presented in Figure 6.7(e) and 6.7(f). On the Car Dataset (cf. Figure 6.7(e)) OPTICS finds six clusters which are analysed in Figure 6.10. Each class consists of intuitive similar objects. Class A represents a large number of small and thin objects (similar to the solid-angle Model – cf. Figure 6.9). Class B consists of fenders, class C represents doors, all objects in class D are seats, class E consists of engine blocks and class F represents kinematic envelopes of seats. Analysing the Aircraft Dataset with OPTICS based on the eigen value Model yields affirmative results as well. The reachability plot (cf. Figure 6.7(f)) depicts five clusters containing similar parts.

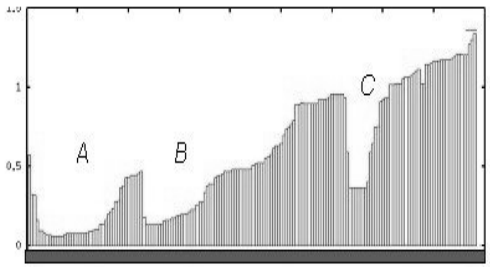
The plots computed by OPTICS for the cover sequence model and the vector set model (cf. Figure 6.8) look even better than the plots computed for the eigen value model. We will confirm this observation in the following, evaluating the effectiveness of the cover sequence model compared to the vector set model.



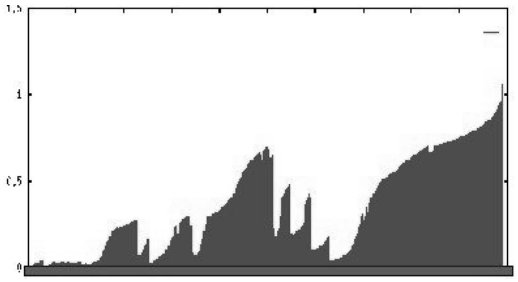
(a) volume model on the car dataset



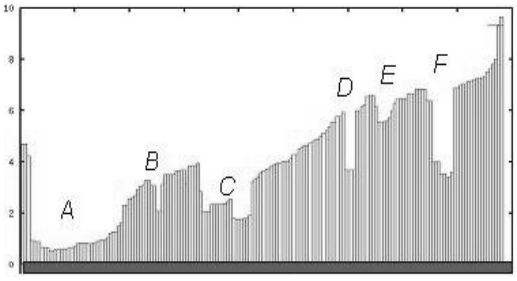
(b) volume model on the aircraft dataset



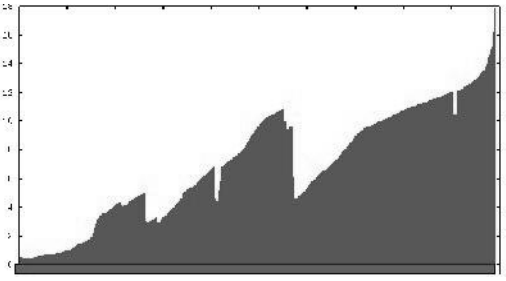
(c) solid-angle model on the car dataset



(d) solid-angle model on the aircraft dataset

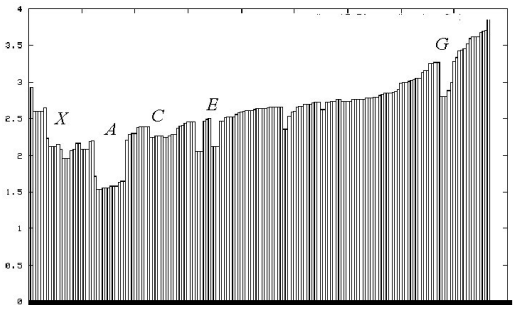


(e) eigen value model on the car dataset

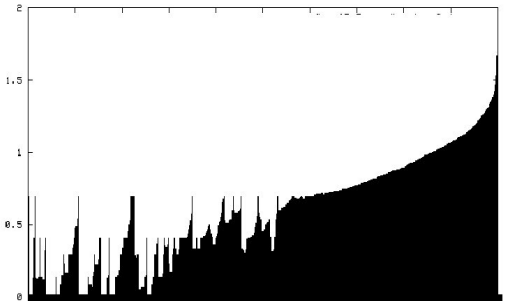


(f) eigen-value model on the aircraft dataset

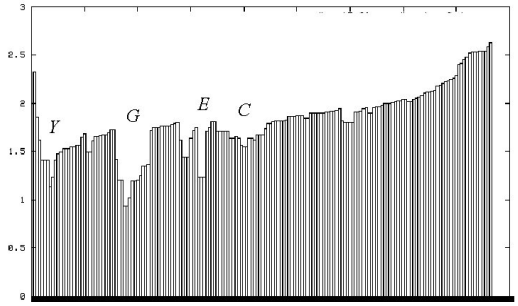
Figure 6.7: Reachability plots computed by OPTICS for the Car Dataset and the Aircraft Dataset using the volume, solid-angle and eigen value model.



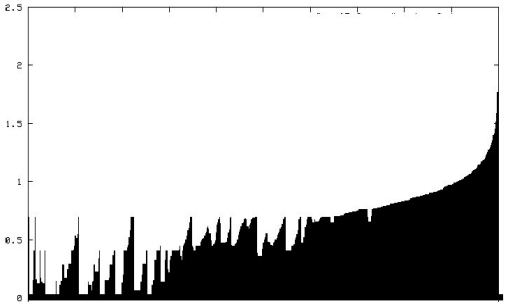
(a) cover sequence model (using 7 covers) on the Car Dataset



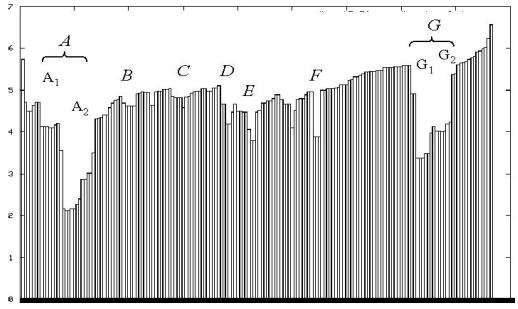
(b) cover sequence model (using 7 covers) on the Aircraft Dataset



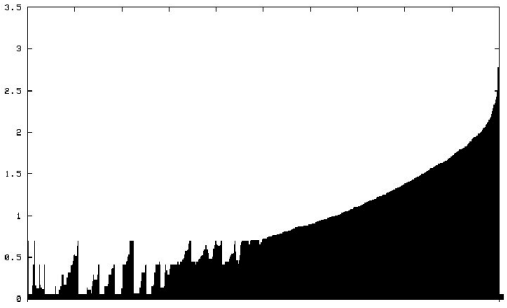
(c) vector set model (using 3 covers) on the Car Dataset



(d) vector set model (using 3 covers) on the Aircraft Dataset



(e) vector set model (using 7 covers) on the Car Dataset



(f) vector set model (using 7 covers) on the Aircraft Dataset

Figure 6.8: Reachability plots computed by OPTICS for the Car Dataset and the Aircraft Dataset using the cover sequence model and vector set model.

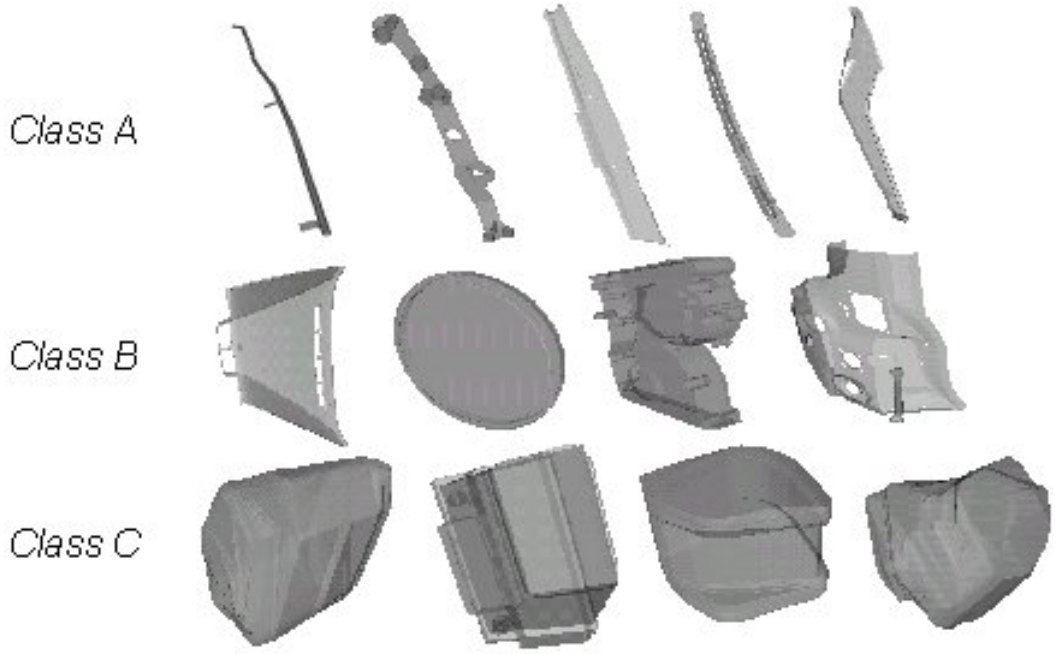


Figure 6.9: Classes found by OPTICS in the Car Dataset using the solid-angle model (cf. Figure 6.7(c)).

We apply the vector set model using only 3 covers as well as 7 covers.

Comparing the vector set model with the cover sequence model on the Car Dataset (cf. Figure 6.8(a),6.8(c), and 6.8(e)) we conclude, that the vector set model is superior. All plots look similar on the first glance. When evaluating the clusters (cf. Figure 6.11 and 6.12), it turned out that there are clusters which are detected by both approaches and thus appear in both plots, e.g. classes E in Figure 6.11 and 6.12. Nevertheless, we observed the following three failures of the cover sequence model:

1. Meaningful hierarchies of clusters detected by the vector set model, e.g. G_1 and G_2 in Figure 6.8(e) which are visualized in Figure 6.12 are lost in the plot of the cover sequence model (Class G in Figure 6.8(a) evaluated in Figure 6.11).

2. Some clusters found by the vector set model are not modeled using the cover sequence model, e.g. cluster G in Figure 6.12.

3. Using the cover sequence model, objects that are intuitively not similar

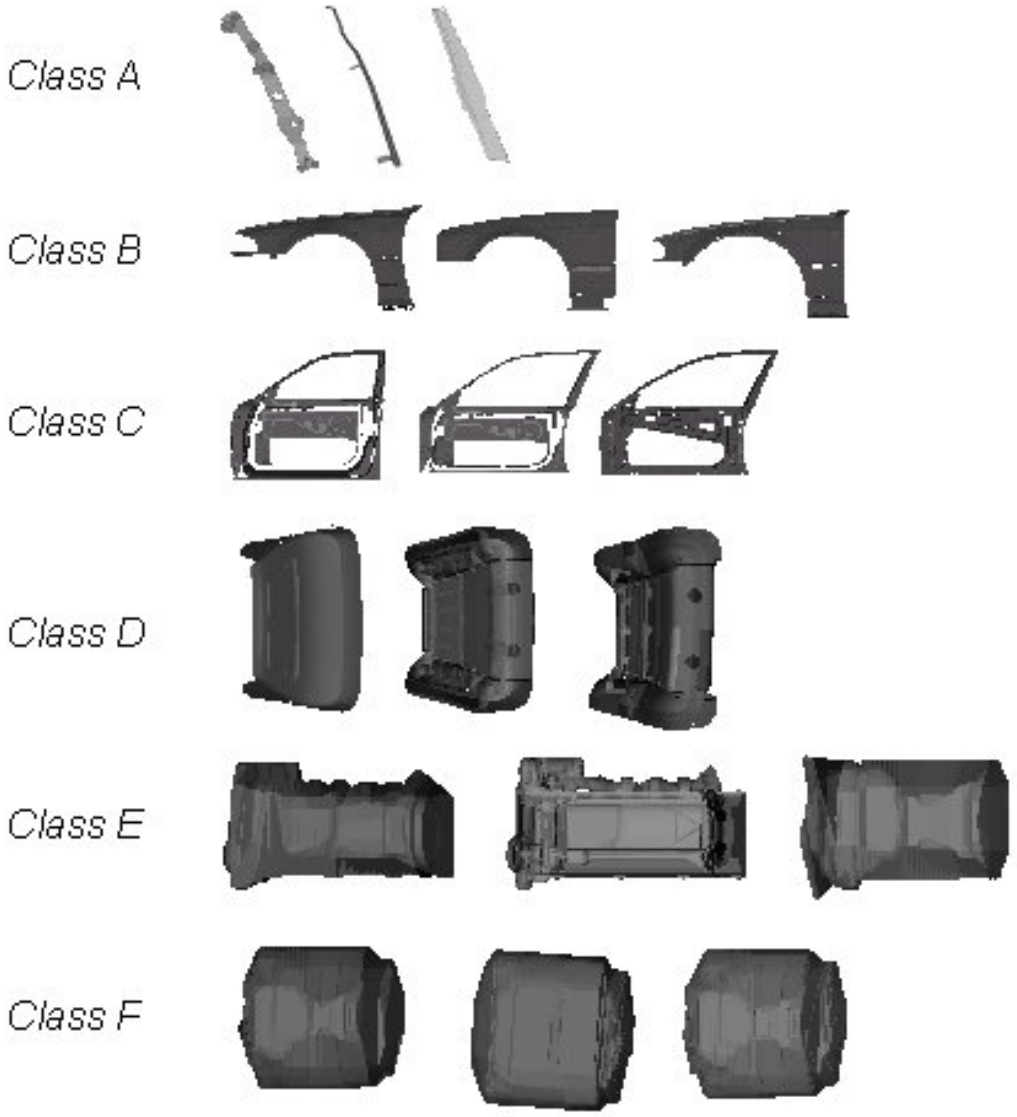


Figure 6.10: Classes found by OPTICS in the Car Dataset using the eigen value model (cf. Figure 6.7(e)).

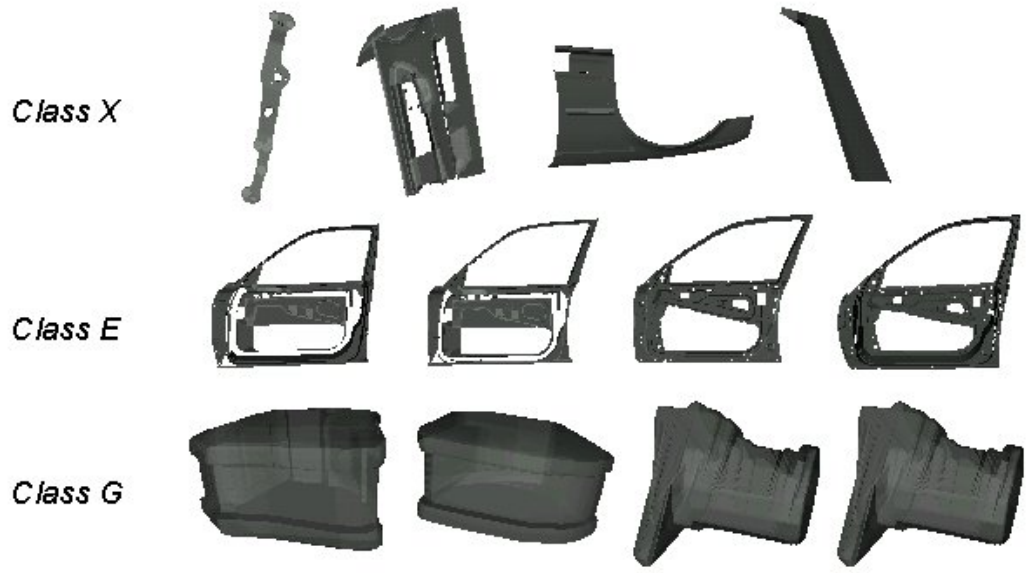


Figure 6.11: Classes found by OPTICS in the Car Dataset using the cover sequence model (cf. Figure 6.8(a)).

are clustered together in one class (e.g. class X in Figure 6.8(a) which is evaluated in Figure 6.11). This is not the case when using the vector set model.

A reason for the superior effectiveness of the vector set model compared to the cover sequence model is the role of permutations of the covers. This is supported by the observations which are depicted in Table 6.1. In more than 95% of all distance calculations during an OPTICS run there was at least one permutation necessary to compute the minimal matching distance.

The plots in Figure 6.8(c) and 6.8(e) compare the influence of the number of covers used to generate the vector sets on the quality of the similarity

Nr. of covers	permutations
3	68%
5	95%
7	99.0%
9	99.4%

Table 6.1: Percentage of proper permutations.

Figure 6.12: Classes found by OPTICS in the Car Dataset using the vector set model with 7 covers (cf. Figure 6.8(e)).

model. An evaluation of the clusters yields the observation, that 7 covers are necessary to model real-world CAD objects accurately. Using only 3 covers we observed basicly the same three problems already noticed when applying the cover sequence model.

All the results of the evaluation on the Car Dataset can also be observed evaluating the models on the Aircraft Dataset. As a consequence, the evaluation shows that the vector set model outperforms the other models with respect to effectiveness. Furthermore, we see that we need about 7 covers to model similarity most accurately.

6.5 Evaluation of the Efficiency

The most effective results on our test datasets were generated with $k = 7$ covers, entailing an average permutation rate of 99% (cf. Table 6.1). This leads to the conclusion, that the cover sequence model can only compete with the vector set model with respect to quality, if all permutations are taken into account. Obviously, the vector set model using the minimal matching distance approach is much more efficient than the cover sequence model (one-vector model) using the minimum Euclidian distance under permutation.

To analyze the performance of the filter step, introduced in Chapter 5, we evaluated k-nn queries, which are the most common query type in similarity search systems. Since the Car Dataset consists of only some 200 objects, it is not suitable for efficiency evaluation. Thus, we ran our efficiency experiments on the Aircraft Dataset only. We took 100 random query objects from the database and examined 10-nn queries. Our test machine was equipped with an INTEL XEON 1.7 GHz processor and 2 GByte main memory. Since data and access structures fitted easily into the main memory, we calculated the I/O cost. One page access was counted as 8 ms and for the costs of reading one byte we counted 200 μs. The results are shown in Table 6.2.

It turns out that the filter step yields a speed-up of factor 10 on the CPU time, but suffers from a higher I/O-time. Nevertheless it provides a speed up factor of about 2 for total time. Furthermore, Table 6.2 demonstrates that the run time using the vector set model with filter step is in the same order of magnitude as the one-vector model even without permutation. In our experiments, the vector set approach even outperformed the one-vector model in both CPU time and I/O time. Let us note that in our experiments we based the implementation of the one-vector model on the X-tree [BKK96], which is penalized by the simulation of I/O time. Since it does not take the idea of page caches into account, an implementation of the one-vector model using the

Model	CPU time	I/O time	total time
1-vector	142.82	2632.06	2774.88
vector set with filter	105.88	932.80	1038.68
vector set seq. scan	1025.32	806.40	1831.72

Table 6.2: Run times for sample 10-nn queries in seconds.

sequential scan exhibited slightly better performance for some combinations of dimensionality and data set size, but the performance was still in the same order of magnitude.

Chapter 7

Conclusions

7.1 Results

In this diploma thesis, we surveyed four feature transformations that are suitable for the use on voxelized CAD data: the volume model, the solid angle model, the eigen value model and the cover sequence model. The cover sequence model generates a set of covers of a 3-dimensional object that can be stored in a feature vector. In comparison to the other three models it offers a better notion of similarity. A major problem of the cover sequence model is the order in which the covers are stored within the feature vector. For calculating the similarity of two objects the order realizing minimum distance offers a better similarity measure, but is prohibitive in calculation cost. Our new approach to represent an object as a set of feature vectors instead of a single feature vector avoids this problem. Furthermore, it offers a more general approach for applications working with set-valued objects. Then we formally described the distance measure on vector sets we used, called minimal matching distance. Minimal matching distance is a metric and computable in $O(k^3)$. To demonstrate how similarity queries can be answered efficiently, we introduced a highly selective filter step that is able to speed up similarity queries by the use of spatial index structures. To evaluate our system we used two CAD data sets. To demonstrate the good notion of similarity provided by the combination of the cover sequence model and the vector set representation, we introduced an algorithm for hierarchical clustering called OPTICS as a comparably more objective way to examine similarity measures. Since k-nearest neighbor queries are the most common operation in similarity search systems, we evaluated the efficiency of the filter step using 100 sample k-nearest neighbor queries. It turned out that our new approach yields more meaningful results without sacrificing efficiency.

7.2 Future Work

Since vector set representation provides many advantages for applications working with set-valued objects, it is desirable to develop a more general system for managing vector-set-represented objects. With the more general system we plan to examine various other applications for similarity search, such as the retrieval of bio-molecular data and images. Another essential goal is the development of fast and flexible algorithms for processing similarity queries on vector set representations.

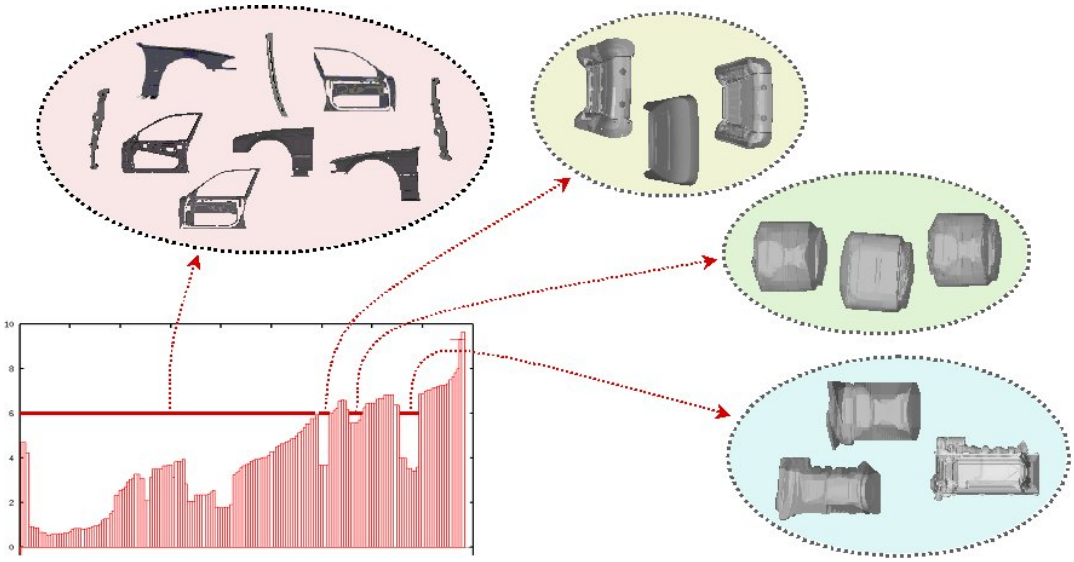
As shown above, the evaluation method based on hierarchical clustering yields enormous advantages. Based on both our new evaluation procedure and our effective similarity model, we sketch a prototype suitable for industrial use, which helps the user to cope with rapidly growing amounts of data, and helps thereby to reduce the cost of developing and producing new parts. This prototype is called *BOSS* (*B*rowsing *O*PTICS-Plots for *S*imilarity *S*earch).

BOSS is an interactive data browsing tool which depicts the reachability plot computed by OPTICS in a user friendly way together with appropriate representatives of the clusters. This clear illustration supports the user in his time-consuming task to find similar parts. From the industrial user's point of view, BOSS meets the following two requirements:

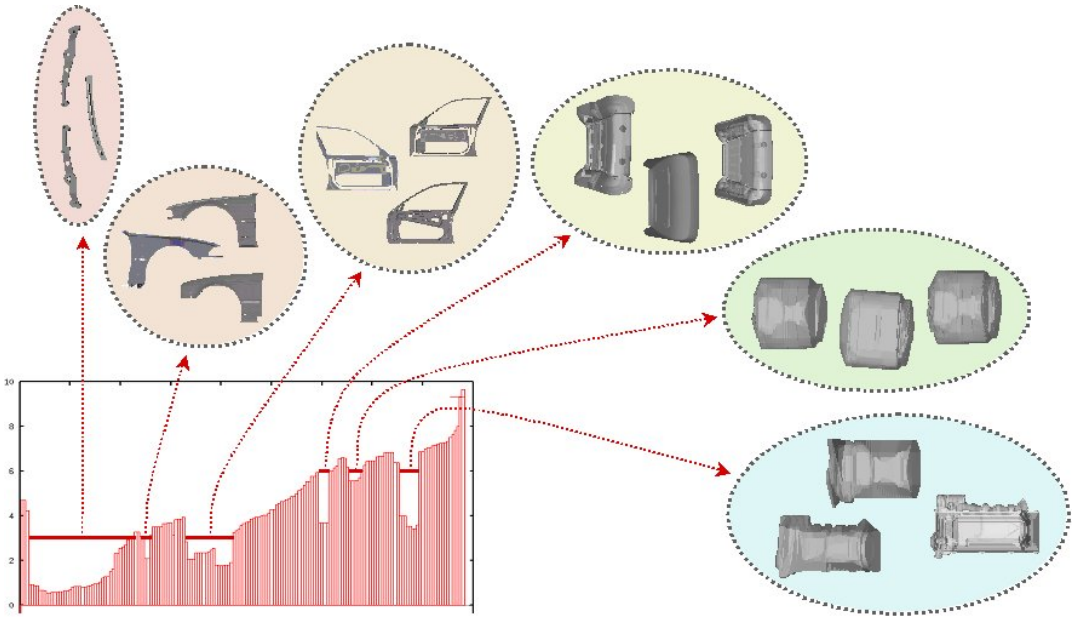
- The hierarchical clustering structure of the dataset is revealed at a glance. The reachability plot is an intuitive visualisation of the clustering hierarchy which helps to assign each object to its corresponding cluster or to noise, respectively. Furthermore, the hierarchical representation of the clusters by the reachability plot helps the user to get a quick overview over all clusters and their relation to each other. As each entry in the reachabiltity plot is assigned to one object, we can easily illustrate some representatives of the clusters belonging to the current ε-value (cf. Figure 7.1).

- The user is not only interested in the shape and the number of the clusters, but also in the specific parts building up a cluster. As for large clusters it is rather difficult to depict all objects, BOSS also displays representatives of each cluster. These representatives are simply constructed by superimposing all parts belonging to the regarded cluster. We can browse through the hierarchy of the representatives in the same way as through the OPTICS-Plots (cf. Figure 7.2).

BOSS helps to reduce the cost of developing and producing new parts by maximizing the reuse of existing parts because it allows the user to browse



(a) High density threshold ε



(b) Low density threshold ε

Figure 7.1: BOSS: Hierarchical OPTICS-Plots.

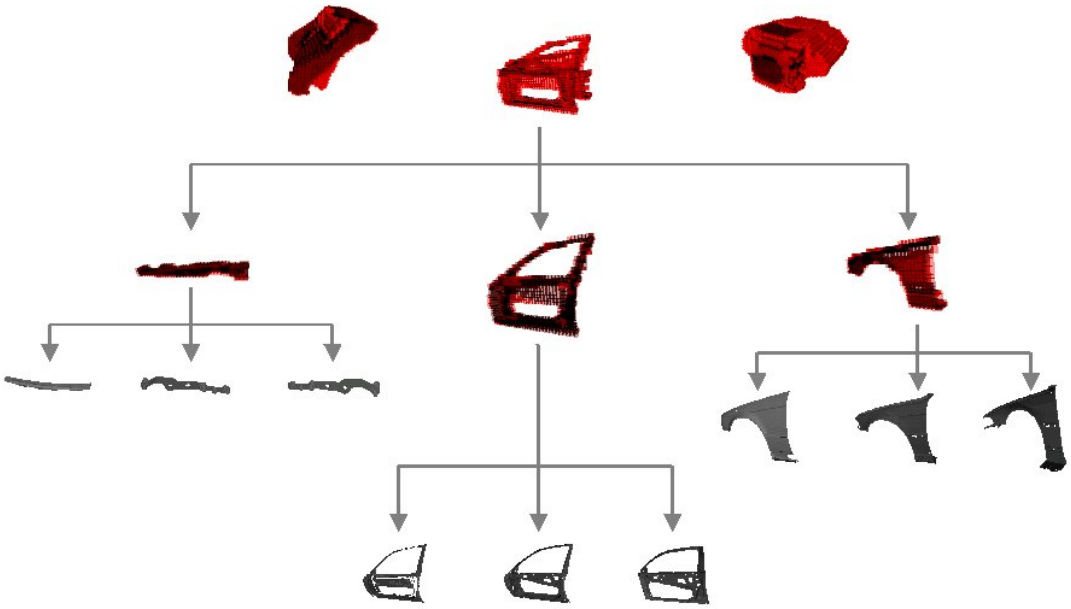


Figure 7.2: BOSS: Hierarchically ordered representatives.

through the hierarchical structure of the clusters in a top-down way. Thus the engineers get an overview of already existing parts and are able to navigate their way through the diversity of existing variants of products, such as cars.

In our future work, we plan to advance our prototype, so that the information contained in the representatives of the clusters is better perceivable. Such a tool would greatly benefit the similarity evaluation technique presented in this work.

List of Figures

List of Tables

List of Definitions

Bibliography

[ABKS99] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. *OPTICS: Ordering Points to Identify the Clustering Structure*. In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA, pages 49–60, 1999.

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. *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)*, pages 69–84. Springer, 1993.

[AKKS99] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl. *3D Shape Histograms for Similarity Search and Classification in Spatial Databases*. In Proc. 6th Int. Symposium on Large Spatial Databases (SSD'99), Hong Kong, China, volume 1651 of *Lecture Notes in Computer Science (LNCS)*, pages 207–226. Springer, 1999.

[ALSS95] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Shim. *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), pages 490–501, 1995.

[BBJ+00] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. *Independent Quantization: An Index Compression Technique for High-Dimensional Data Spaces*. In Proc. Int. Conf. on Data Engineering (ICDE 2000), San Diego, CA, pages 577–588, 2000.

[Ber97] S. Berchtold. *Geometry-based Search for Similar Mechanical Parts*. PhD thesis, Institute for Computer Science, University of Munich, 1997. (in German).

[BK97] S. Berchtold and H.-P. Kriegel. *S3: Similarity Search in CAD Database Systems.* In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'97), Tucson, AZ, pages 564–567, 1997.

[BKK96] S. Berchtold, D. A. Keim, and H.-P Kriegel. *The X-Tree: An Index Structure for High-Dimensional Data.* In Proc. 22th Int. Conf. on Very Large Databases (VLDB'96), Bombay, India, pages 28–39, 1996.

[BKK97] S. Berchtold, D. A. Keim, and H.-P. Kriegel. *Using Extended Feature Objects for Partial Similarity Retrieval.* VLDB Journal, 6(4):333–348, 1997.

[BMH92] A. Badel, J. P. Mornon, and S. Hazout. *Searching for Geometric Molecular Shape Complementarity using Bidimensional Surface Profiles.* Journal of Molecular Graphics, 10:205–211, 1992.

[Con86] M. L. Connolly. *Shape Complementarity at the Hemoglobin a1b1 Subunit Interface.* Biopolymers, 25:1229–1247, 1986.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. *M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces.* In Proc. 23rd Int. Conf. of Very Large Databases (VLDB'97), Athens, Greece, pages 426–435, 1997.

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. *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, pages 226–231, 1996.

[EM97] T. Eiter and H. Mannila. *Distance Measures for Point Sets and Their Computation.* Acta Informatica, 34(2):103–133, 1997.

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, et al. *Efficient and Effective Querying by Image Content.* Journal of Intelligent Information Systems, 3:231–262, 1994.

[FRM94] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. *Fast Subsequence Matching in Time-Series Databases.* In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'94), Minneapolis, MN, pages 419–429, 1994.

[GM93] J. E. Gary and R. Mehrotra. *Similar Shape Retrieval using a Structural Feature Index*. Information Systems, 18(7):525–537, 1993.

[Gri92] I. Grieger. *Graphische Datenverarbeitung*. Springer Verlag, 1992.

[Hig90] W. E. Higgins. *Automatic Analysis of 3-D and 4-D Radiological Images*. grant application to the Department of Health and Human Services, December 1990.

[HS95] G. R. Hjaltason and H. Samet. *Ranking in Spatial Databases*. In Proc. 4th Int. Symposium on Large Spatial Databases (SSD'95), volume 951 of *Lecture Notes in Computer Science (LNCS)*, pages 83–95. Springer, 1995.

[HSE+95] J. Hafner, H. S. Sawhney, W. Equitz, M. Flickner, and Niblack W. *Efficient Color Histogram indexing for Quadratic Form Distance Functions*. IEEE Trans. on Pattern Analysis and Machine Intelligence, 17(7):729–736, 1995.

[Jag91] H. V. Jagadish. *A Retrieval Technique for Similar Shapes*. In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'91), Denver, CO, pages 208–217, 1991.

[JB91] H. V. Jagadish and A. M. Bruckstein. *On sequential shape descriptions*. Pattern Recognition, 1991.

[JD88] A. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[Kau87] A. Kaufman. *An Algorithm for 3D Scan-Conversion of Polygons*. In Proc. Eurographics, pages 197–208, 1987.

[Kei99] D. A. Keim. *Efficient Geometry-based Similarity Search of 3D Spatial Databases*. In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA, pages 419–430, 1999.

[KKS98] G. Kastenmüller, H.-P. Kriegel, and T. Seidl. *Similarity Search in 3D Protein Databases*. In Proc. German Conf. on Bioinformatics (GCB'98), Köln, Germany, 1998.

[KSF+96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. *Fast Nearest Neighbor Search in Medical Image Databases*. In Proc. 22th Int. Conf. on Very Large Databases (VLDB'96), Bombay, India, pages 215–226, 1996.

[KSS97] H.-P. Kriegel, T. Schmidt, and T. Seidl. *3D Similarity Search by Shape Approximation*. In Proc. 5th Int. Symposium on Large Spatial Databases (SSD'97), Berlin, Germany, volume 1262 of *Lecture Notes in Computer Science (LNCS)*, pages 11–28. Springer, 1997.

[Kuh55] H. W. Kuhn. *The Hungarian Method for the Assignment Problem*. Naval Research Logistics Quarterly, 2:83–97, 1955.

[LJF94] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. *The TV-Tree: An Index Structure for High-Dimensional Data*. VLDB Journal, 3(4):517–542, 1994.

[McQ67] J. McQueen. *Some Methods for Classification and Analysis of Multivariate Observations*. In 5th Berkeley Symp. Math. Statist. Prob., volume 1, pages 281–297, 1967.

[MG93] R. Mehrotra and J. E. Gary. *Feature-Based Retrieval of Similar Shapes*. In Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, pages 108–115, 1993.

[MG95] R. Mehrotra and J. E. Gary. *Feature-Index-Based Similar Shape Retrieval*. In Proc. 3rd Working Conf. on Visual Database Systems, 1995.

[MH99] T. Möller and E. Haines. *Real-Time Rendering*. A K Peters, Natick, MA, 1999.

[Mun57] J. Munkres. *Algorithms for the Assignment and Transportation Problems*. Journal of the SIAM, 6:32–38, 1957.

[NBE+93] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasmann, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. *The QBIC Project: Querying Images by Content Using Color, Texture, and Shape*. In SPIE 1993 Int. Symposium on Electronic Imaging: Science and Technology Conference 1908, Storage and Retrieval for Image and Video Databases, San Jose, CA, 1993.

[NS86] W. M. Newman and R. F. Sproull. *Grundzüge der interaktiven Computergraphik*. McGraw-Hill Book Company GmbH Hamburg, 1986.

[RB00] J. Ramon and M. Bruynooghe. *A Polynomial Time Computable Metric Between Point Sets*. Technical Report CW 301, Katholieke Universiteit Leuven, Department of Computer Science, October 2000.

[SH94] H. Sawhney and J. Hafner. *Efficient Color Histogram Indexing*. In Proc. Int. Conf. on Image Processing, pages 66–70, 1994.

[SK98] T. Seidl and H.-P. Kriegel. *Optimal Multi-Step k-Nearest Neighbor Search*. In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'98), Seattle, WA, pages 154–165, 1998.

[SKSH89] R. Schneider, H.-P. Kriegel, B. Seeger, and S. Heep. *Geometry-based Similarity Retrieval of Rotational Parts*. In Proc. Int. Conf. on Data and Knowledge Systems for Manufacturing and Engineering, Gaithersburg, ML, pages 150–160, 1989.

[TC91] G. Taubin and D. B. Cooper. *Recognition and Positioning of Rigid Objects Using Algebraic Moment Invariants*. Geometric Methods in Computer Vision, 1570:175–186, 1991.

[Vin91] L. Vincent. *New Trends in Morphological Algorithms*. In SPIE Proceedings on Non-linear Image Processing II, San Jose, CA, 1991.