Efficient and Effective Similarity Search on Complex Objects

Dissertation im Fach Informatik
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig-Maximilians-Universität München

von

Stefan Brecheisen

Tag der Einreichung: 16.01.2007
Tag der mündlichen Prüfung: 22.02.2007

Berichterstatter:
Prof. Dr. Hans-Peter Kriegel, Ludwig-Maximilians-Universität München
Prof. Dr. Ralf Hartmut Güting, Fernuniversität Hagen

Acknowledgments

Many people supported and encouraged me in the past years while I was working on this dissertation. I would like to thank them here, even if I cannot mention them all by name.

First of all, I extend my warmest thanks to my supervisor, Prof. Dr. Hans-Peter Kriegel. He initiated and supported this work with his long standing experience and the organizational background and gave me the opportunity to work on this challenging domain. Without the inspiring, productive and supportive working environment, he created in the database research group, this work could never have come into existence. I am also very grateful to Prof. Dr. Ralf Hartmut Güting for his interest in my work and his immediate willingness to act as the second referee.

This work would not have been initiated and matured without the cooperation of and discussion with my colleagues in the database research group. In particular, I want to thank Peer Kröger, Peter Kunath, Martin Pfeifle, Alexey Pryakhin, and Matthias Schubert for constructive and productive teamwork, as well as Elke Achtert, Prof. Dr. Christian Böhm, Eshref Januzaj, Karin Kailing, Matthias Renz, Stefan Schönauer, and Arthur Zimek for many helpful discussions.

I also appreciate the substantial help of the students who worked on tasks like implementation, data processing, and testing while preparing their project thesis or diploma thesis. In particular, I wish to thank Felix Leis, Maximilian Viermetz, Christian Mahrt, and Michael Gruber.

I am extremely grateful for the background support of Susanne Grienberger, who managed much of the administrative work. Furthermore, I want

to express special thanks to Franz Krojer for taking care of our technical environment and unhesitatingly providing me with all the technical tools aiding the progress of this work.

Last but not least, I want to thank my family and friends for their support and encouragement during the time that I was engaged in this study. In particular, I thank my parents who always supported my career and encouraged me to find my way.

Stefan Brecheisen

Munich, January 2007

Abstract

Due to the rapid development of computer technology and new methods for the extraction of data in the last few years, more and more applications of databases have emerged, for which an efficient and effective similarity search is of great importance. Application areas of similarity search include multimedia, computer aided engineering, marketing, image processing and many more. Special interest adheres to the task of finding similar objects in large amounts of data having complex representations. For example, set-valued objects as well as tree or graph structured objects are among these complex object representations. The grouping of similar objects, the so-called clustering, is a fundamental analysis technique, which allows to search through extensive data sets.

The goal of this dissertation is to develop new efficient and effective methods for similarity search in large quantities of complex objects. Furthermore, the efficiency of existing density-based clustering algorithms is to be improved when applied to complex objects.

The first part of this work motivates the use of vector sets for similarity modeling. For this purpose, a metric distance function is defined, which is suitable for various application ranges, but time-consuming to compute. Therefore, a filter refinement technology is suggested to efficiently process range queries and k-nearest neighbor queries, two basic query types within the field of similarity search. Several filter distances are presented, which approximate the exact object distance and can be computed efficiently. Moreover, a multi-step query processing approach is described, which can be directly integrated into the well-known density-based clustering algorithms

DBSCAN and OPTICS.

In the second part of this work, new application ranges for density-based hierarchical clustering using OPTICS are discussed. A prototype is introduced, which has been developed for these new application areas and is based on the aforementioned similarity models and accelerated clustering algorithms for complex objects. This prototype facilitates interactive semi-automatic cluster analysis and allows visual search for similar objects in multimedia databases. Another prototype extends these concepts and enables the user to analyze multi-represented and multi-instance data. Finally, the problem of music genre classification is addressed as another application supporting multi-represented and multi-instance data objects.

An extensive experimental evaluation examines efficiency and effectiveness of the presented techniques using real-world data and points out advantages in comparison to conventional approaches.

Abstract (in German)

Aufgrund der rasanten Entwicklung der Computertechnik und der neuen Methoden der Datengewinnung sind in den letzten Jahren immer mehr Datenbankanwendungen entstanden, für die eine effiziente und effektive Ähnlichkeitssuche von großer Bedeutung ist. Zu den Anwendungsgebieten der Ähnlichkeitssuche gehören Multimedia, Computer Aided Engineering, Marketing, Bildverarbeitung und viele weitere Bereiche. Besonderes Interesse kommt dabei der Aufgabenstellung zu, ähnliche Objekte in großen Mengen von Daten mit komplexer Darstellung zu finden. Zu diesen komplexen Objektdarstellungen zählen beispielsweise mengenwertige Objekte und Objekte mit Baum- oder Graph-Struktur. Das Zusammenfassen ähnlicher Objekte, das sogenannte Clustering, stellt eine wichtige Analysetechnik dar, um umfangreiche Datenmengen durchsuchen zu können.

Das Ziel dieser Doktorarbeit ist, neue effiziente und effektive Verfahren für die Ähnlichkeitssuche in großen Mengen komplexer Objekte zu entwickeln. Außerdem soll die Effizienz vorhandener dichtebasierter Clustering-Verfahren bei Anwendung auf komplexen Objekten verbessert werden.

Der erste Teil der Arbeit motiviert zunächst den Einsatz von Vektormengen zur Ähnlichkeitsmodellierung. Dazu wird eine metrische Distanzfunktion definiert, die für verschiedene Anwendungsbereiche geeignet ist, deren Berechnung allerdings aufwendig ist. Zur effizienten Beantwortung von Bereichsanfragen und k-nächste-Nachbarn-Anfragen, zwei grundlegenden Anfragetypen im Bereich der Ähnlichkeitssuche, wird deshalb eine Filterverfeinerungstechnik vorgeschlagen. Mehrere Filterdistanzen werden präsentiert, die die exakte Objektdistanz abschätzen und effizient berechnet werden

können. Des Weiteren wird ein Verfahren zur mehrstufigen Anfragebearbeitung beschrieben, das direkt in die bekannten dichtebasierten Clustering-Algorithmen DBSCAN und OPTICS integriert werden kann.

Im zweiten Teil der Arbeit werden neue Anwendungsbereiche für das dichtebasierte hierarchische Clustering mit OPTICS diskutiert. Es wird ein Prototyp vorgestellt, der für diese neuen Anwendungsbereiche entwickelt wurde und auf den entwickelten Ähnlichkeitsmodellen und beschleunigten Clustering-Verfahren für komplexe Objekte basiert. Dieser Prototyp erleichtert die interaktive semi-automatische Clusteranalyse und ermöglicht die visuelle Suche nach ähnlichen Objekten in Multimedia-Datenbanken. Ein weiterer Prototyp entwickelt diese Konzepte weiter und ermöglicht die Analyse von multirepräsentierten und multiinstanziierten Datenbeständen. Schließlich wird das Problem der Genre-Klassifikation von Musikstücken behandelt, eine weitere Anwendung mit Unterstützung für Datenobjekte mit mehreren Repräsentationen und Instanzen.

Effizienz und Effektivität der vorgestellten Techniken werden ausführlich untersucht und die Vorteile gegenüber herkömmlichen Verfahren werden mittels Realdaten experimentell belegt.

Survey of Chapters

Contents

Part I

Preliminaries

Chapter 1

Introduction

Database systems are key components of today's information technology infrastructure. With the enormous growth of this infrastructure in the past decade, new challenges for database systems have arisen. In both, science and industry, new applications of database systems have been developed and their importance in practice is rapidly increasing. In this introductory chapter, we will discuss some of the new challenges for database systems, present our approach to tackle these challenges and outline the scope of this thesis. Furthermore, we will introduce the basic concepts behind our approach and some example applications in the following chapters.

1.1 Challenges for Modern Database Systems

The challenges for modern database systems are manifold, including topics like increased need for data security in e-commerce or integration of world-wide distributed databases. One very important challenge is the support for tasks like knowledge discovery in databases (KDD). KDD is the process of extracting new, valid and potentially useful knowledge from databases [FPSS96]. Particularly in a world of large and fast growing databases, a process to automatically or at least semi-automatically extract knowledge from those databases is essential.

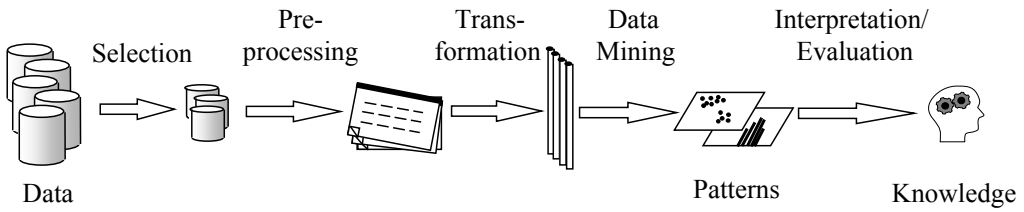


Figure 1.1: The KDD process.

The KDD process, as defined by Fayyad, Piatetsky-Shapiro and Smyth [FPSS96], has several steps which are depicted in Figure 1.1. After a selection and preprocessing of the relevant data, it is transformed in a suitable format. In the data mining step, patterns in the data are extracted and later evluated by the user, to gain knowlwedge. At the center of the KDD process is the data mining step, where the automatic detection of the information takes place. Several different subtasks of data mining have been identified, including clustering and object classification. Clustering is the task of grouping objects, where the similarity of objects within a group has to be maximized, while the similarity of objects in different groups has to be minimized. Obviously, the clustering of objects in a database depends on efficient and effective methods to identify similar objects in the database, or in other words, it depends on similarity search methods. But those methods also play a major role in object classification, where new objects have to be assigned to a class based on the knowledge extracted from a database of already classified objects. In this context, so-called nearest neighbor classifiers were successfully used, which assign an object to the class of its nearest neighbors in the database. This means that similarity search is an important basic technique for data mining in general. In this thesis, we will concentrate on the efficient and effective support for complex data types in advanced similarity search and data mining applications.

1.2 Complex Objects

In recent years, an increasing number of applications has emerged, processing large amounts of complex, application specific data objects [Jag91, AFS93, FRM94, BK97, KKS98, AKKS99a, KBK+03, KKM+03]. To provide a notion of similarity among database objects, an appropriate similarity measure must be defined for each application domain. However, defining the similarity of complex objects, such as car parts, proteins or text documents, is not a trivial task. In the following, we will shortly review two common techniques to define the similarity between complex objects. A widely used class of similarity models is based on the paradigm of feature vectors. The basic idea is that by a *feature transformation*, the objects are mapped onto a feature vector in an appropriate multidimensional feature space. The similarity between two objects is then measured through the proximity of the respective feature vectors.

If this feature-based approach is not able to capture the intuitive notion of similarity between objects, more complex similarity measures like the edit distance for graphs or trees are necessary. Usually, complex objects are then represented in some sort of application specific metric space. In this thesis, we concentrate on application domains which belong to one of the two approaches and do not regard application domains where non-metric data spaces are involved.

1.2.1 Representation as Vector Data

A common solution in application domains such as multimedia, medical imaging, molecular biology, computer aided design, marketing, purchasing assistance, etc. is the so-called *feature transformation*. For each data object, a given number d of numeric features is extracted (see Figure 1.2 for an illustration). Thus, the objects of a database are transformed into d-dimensional feature vectors, i.e. data objects are represented by points in a d-dimensional vector space. Then, the similarity between two objects is measured through the proximity of the respective feature vectors, e.g. using the Euclidean dis-

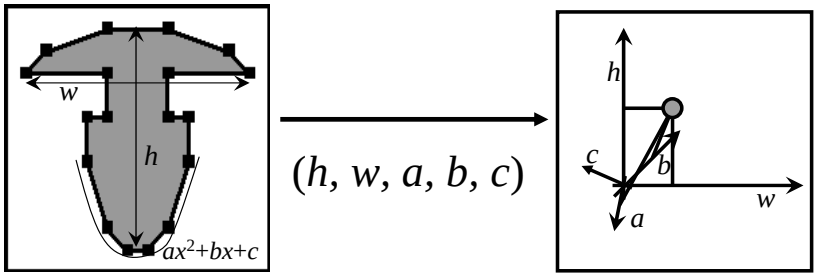


Figure 1.2: The idea of feature transformation.

tance measure. Examples of feature-based similarity include color histograms for image data [HSE+95], Fourier coefficients for time series data [AFS93] or 3D shape histograms for 3D objects [AKKS99a].

1.2.2 Representation as Arbitrary Metric Data

Sometimes the similarity between complex objects can not be captured by a feature transformation. The internal structure of complex data objects varies from application to application, but often it can be described by using the abstract concepts of graphs and trees. In this case, the use of more complex similarity models like the edit distance for graphs or trees is necessary. The remainder of this section presents three metric similarity models for complex objects. As this is an extremely broad field, we do not make any claim to completeness. The main purpose of this section is to motivate that there are lots of applications where the objects can no longer be represented as one single feature vector. In the following, we shortly review three examples of complex similarity models used in the evaluation parts of this thesis.

Sets of Feature Vectors. For CAD applications, suitable similarity models can help to reduce the cost of developing and producing new parts by maximizing the reuse of existing parts. In [KBK+03], an effective and flexible similarity model for complex 3D CAD data is introduced which helps to find and group similar parts. It is not based on the traditional approach of describing one object by a single feature vector. Instead an object is mapped onto a set of feature vectors, i.e. an object is described by a vector set (see

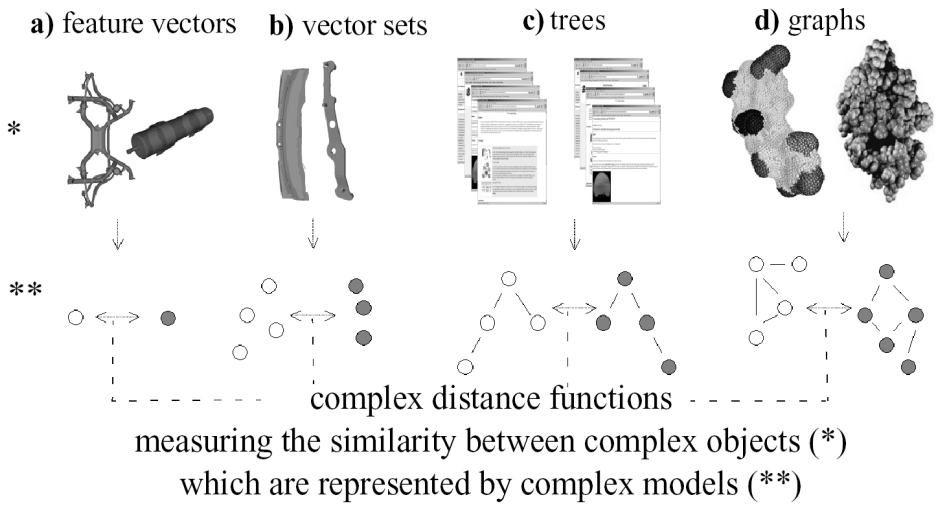


Figure 1.3: Examples of complex metric data.

Figure 1.3 left for an illustration). The cover sequence model introduced in [Jag91, JB92] is extended by generating several representations for each object, resulting in a set of feature vectors for each object. The experimental evaluation shows that this approach is superior to techniques using only one feature vector for each object.

Tree-Structured Data. In addition to a variety of content-based attributes, complex objects typically carry some kind of internal structure which often forms a hierarchy. Examples of such tree-structured data include chemical compounds, CAD drawings, XML documents or websites (see Figure 1.3 center for an illustration). For similarity search and therefore clustering, it is important to take into account both, the structure and the content features of such objects. A successful approach is to use the edit distance for tree structured data. However, as the computation of this measure is NP-complete [ZSS92], constrained edit distances like the degree-2 edit distance [ZWS96] have been introduced. They were successfully applied to trees for web site analysis [WZCS02], structural similarity of XML documents [NJ02], shape recognition [SKK01] or chemical substructure search [WZCS02].

Graphs. Attributed graphs are another natural way to model structured data (see Figure 1.3 right for an illustration). As graphs are a very general object model, graph similarity has been studied in many fields. Similarity

Table 1.1: Overview of publications the chapters are based on.

Part II Multi-Step Similarity Search and Clustering
4 Efficient Similarity Search on Vector Sets [BKP05]
5 Multi-Step Density-Based Clustering [BKP04, BKP06a]
6 Parallel Density-Based Clustering of Complex Objects [BKP06b]
Part III Advanced Similarity Search Applications
7 Visual Density-Based Data Analysis [BKKP04, BKK$^+$04, BKSG06]
8 Hierarchical Music Genre Classification [BKKP06, BKK$^+$06]

measures for graphs have been used in systems for shape retrieval [HCH99], object recognition [KKV90] or face recognition [WFKvdM97]. For all those measures, graph features, specific to the graphs in the application, are exploited in order to define graph similarity. Most known similarity measures for attributed graphs are either limited to a special type of graph or computationally extremely complex, i.e. NP-complete. Therefore, they are unsuitable for searching or clustering large collections. In [KS03], the authors present a new similarity measure for attributed graphs, called *edge matching distance*. They demonstrate how the edge matching distance can be used for efficient similarity search in attributed graphs.

1.3 Outline

In this chapter, we presented some of the challenges of modern database systems. Those challenges include support for complex data types and new applications for database systems. The aim of this thesis is to improve the efficiency of known similarity search methods and to provide new approaches to solve the efficiency and effectiveness problems of existing methods. Many of the algorithms and ideas discussed in the different chapters have already been published. For clearness and convenience, we list these publications by chapter in Table 1.1, while refraining from citing them repeatedly throughout the thesis.

The remainder of this thesis is organized as follows.

Chapter 2 presents important concepts of similarity search. This includes query types, similarity models and index structures to support efficient query processing in similarity search systems. Furthermore, we develop a set of requirements which similarity search methods for complex objects have to fulfill in order to meet the demands of modern database applications.

Chapter 3 provides an overview of the notion of density-based clustering. After introducing basic notations and concepts, we present an partitioning density-based clustering algorithm as well as an hierarchical extension thereof.

Part II presents innovative filter refinement techniques to efficiently process similarity queries and to accelerate clustering algorithms.

Chapter 4 motivates the use of sets of feature vectors as a promising way between too simple and too complex object representations for complete object similarity search as well as for partial object similarity search. After introducing a distance measure between vector sets, suitable for many different application ranges, we present and discuss different filters which are indispensable for efficient query processing. In an experimental evaluation based on artificial and real-world test datasets, we show that our approach considerably outperforms both the sequential scan and metric index structures.

Chapter 5 demonstrates how the paradigm of multi-step query processing which relies on exact as well as on lower-bounding approximated distance functions can be integrated into the two density-based clustering algorithms DBSCAN and OPTICS resulting in a considerable efficiency boost. We also extend our approach to approximated clustering allowing the user to find an individual trade-off between quality and efficiency. In order to assess the quality of the resulting clusterings, we introduce suitable quality measures which can be used generally for evaluating the quality of approximated partitioning and hierarchical clusterings. In a broad experimental evaluation, we demonstrate that our approach accelerates the generation of exact density-based clusterings by more than one order of magnitude. Furthermore, we

show that our approximated clustering approach results in high quality clusterings where the desired quality is scalable w.r.t. the overall number of exact distance computations.

In Chapter 6, we will show how lower-bounding distance functions can be used to parallelize the density-based clustering algorithm DBSCAN. First, the data is partitioned based on an enumeration calculated by the hierarchical clustering algorithm OPTICS, so that similar objects have adjacent enumeration values. We use the fact that clustering based on lower-bounding distance values conservatively approximates the exact clustering. By integrating the multi-step query processing paradigm directly into the clustering algorithms, the clustering on the slaves can be carried out very efficiently. Finally, we show that the different result sets computed by the various slaves can effectively and efficiently be merged to a global result by means of cluster connectivity graphs. In an experimental evaluation based on real-world test data sets, we demonstrate the benefits of our approach.

Part III presents new application ranges for similarty search and density-based hierarchical clustering.

Chapter 7 shows how visualizing the hierarchical clustering structure of a database of objects can aid the user in his time consuming task to find similar objects. We present related work and explain its shortcomings which led to the development of our new methods. Based on reachability plots, we introduce approaches which automatically extract the significant clusters in a hierarchical cluster representation along with suitable cluster representatives. We implemented our algorithms resulting in prototype systems which were used for the experimental evaluation. This evaluation is based on real world test data sets and points out that our new approaches to automatic cluster recognition and extraction of cluster representatives create meaningful and useful results in comparatively short time.

In Chapter 8, we propose a novel approach for hierarchical classification of pieces of music into a genre taxonomy. To provide a versatile description of the music content, several kinds of features like rhythm, pitch or timbre characteristics are commonly used. Taking the highly dynamic nature of music into account, each of these features should be calculated up to several

hundreds of times per second. Thus, a piece of music is represented by a complex object given by several large sets of feature vectors. Our approach is able to handle multiple characteristics of music content and achieves a high classification accuracy efficiently, as shown in our experiments. Furthermore, we present MUSCLE, a prototype tool which allows the user to organize large music collections in a genre taxonomy and to modify class assignments on the fly.

Part IV concludes the thesis.

Chapter 9 summarizes and discusses the major contributions of this work and concludes the thesis by pointing out some potentials for future research.

Chapter 2

Similarity Search

The basic task of a similarity search application is to find objects in the database which are similar to a query object. In this chapter, we will discuss the different aspects of this task.

2.1 Similarity Models

The first important aspect of similarity search is the concept of similarity itself. A formal concept of similarity is a necessary basis for any application in this field. In the literature, two concepts of similarity have been applied successfully which are the feature vector approach and the concept of distance-based similarity. We will present the two concepts in this section and discuss invariance and adaptability issues of similarity models.

2.1.1 The Feature Vector Approach

A very common way to define the similarity of objects is the feature vector approach. For this approach, a domain expert chooses a set of single-valued object features that describe an object from that application domain. Those features span a so-called feature space and objects are represented as points in this space. This is done by creating a feature vector for each object which contains the feature values of the specific object. Then, the similarity or

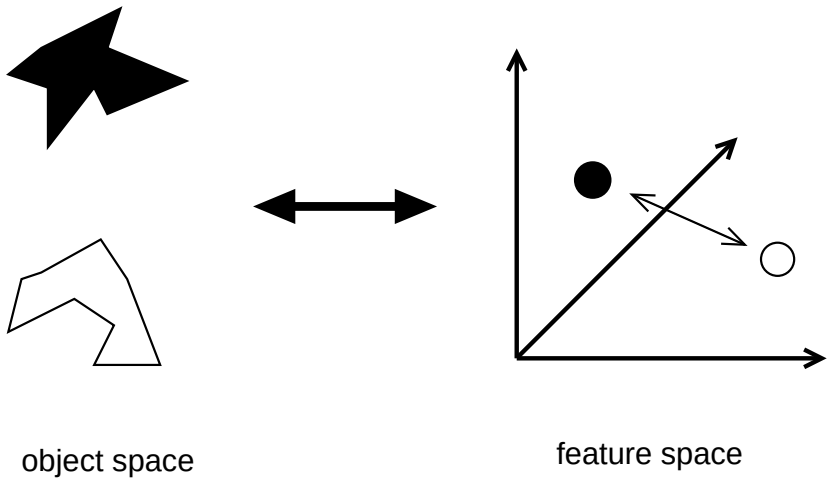


Figure 2.1: Similarity based on the feature vector approach.

dissimilarity of two objects is defined as their distance in the feature space. The feature vector approach for similarity, whose idea is illustrated in Figure 2.1, has been successfully applied in several application domains like medical imaging [KSF+98] and protein similarity [AKKS99b].

To determine the distance between two points in the feature space, several measures are used. Most often it is a variant of the L_p norms, which are defined as follows:

Definition 1 (L_p norms). Let there be two vectors $x = (x_1, \ldots, x_n)$, $x \in \mathbb{R}^n$, and $y = (y_1, \ldots, y_n)$, $y \in \mathbb{R}^n$. The L_p norms between x and y are defined as:

$$L_p(x, y) = \|x - y\|_p = (\sum_{i=1}^{n} |x_i - y_i|^p)^{\frac{1}{p}}$$

For $p = 1$ and $p = 2$ the L_p norms are the well-known Manhattan distance and the Euclidean distance, respectively. Most often, the Euclidean distance is used in similarity search applications based on the feature vector approach.

A problem of the L_p norms is that all dimensions of the feature space are considered to be independent of each other. Consequently, no relationships between the features, for example substitutability, may be regarded by the similarity process. But often such relationships exist, like in the case of color features where orange is certainly more similar to red or yellow than to blue.

To overcome this disadvantage, Niblack et al. [NBE+93] suggested to use the quadratic form distance instead of the usual Euclidean distance. The quadratic form distance of two vectors x and y is defined as

$$d_A^2(x,y) = (x-y) \cdot A \cdot (x-y)^T$$

where A is a positive definite similarity matrix and $(x-y)^T$ is the transpose of $(x-y)$. When using the identity matrix as similarity matrix, the quadratic form distance becomes the classic Euclidean distance since

$$(L_2(x,y))^2 = (x-y) \cdot (x-y)^T$$

By altering the similarity matrix A, it is possible to express relationships between the dimensions of the feature space which is the desired effect. For methods to ensure efficient query processing with the quadratic form distance see [Sei97].

2.1.2 Feature Vectors of Complex Objects

It is often difficult to extract useful feature vectors from datasets consisting of complex objects. For example, the set-like internal structure of a graph makes it difficult to apply the feature vector approach to data modeled as attributed graphs. This internal structure prevents a unique description of the graph structure with few feature values. The same is the case for the attribute part of an attributed graph. Consequently, many features have to be extracted from a graph in order to yield a description with sufficient discriminatory power to distinguish between separate objects. This leads to extremely high-dimensional feature vectors. But the high dimensionality of the feature vectors can make efficient similarity search in the database impossible due to a number of effects. For example, an increasing dimensionality leads to a larger volume of the data space and to higher distances between the data objects. Those and other effects are usually described by the term "curse of dimensionality".

Additionally, when choosing the features one has to take into account that any of the simple L_p norms or the quadratic forms distance yield sensible

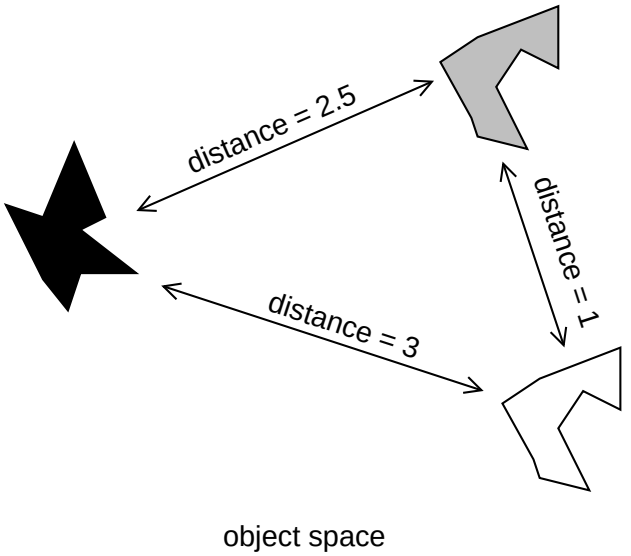


Figure 2.2: The concept of distance-based similarity.

results for a similarity search. This fact even worsens the problem of picking the right features. The distance-based similarity model, which we describe in the following section, avoids the choice of any features at all.

2.1.3 Distance-Based Similarity

The distance-based similarity model is a generalization of the feature vector model. Instead of transforming the data objects into a feature space and measuring the distance of the objects in the feature space, a distance measure for the data objects themselves is defined. This means that no feature extraction and no choice of features is necessary. Furthermore, a distance measure which is defined for complex data objects can take all object properties into account. The concept of distance-based similarity is illustrated in Figure 2.2.

Obviously, the increased flexibility also leads to a higher complexity, since the complete objects have to be managed and, therefore, the computational complexity of the similarity measure has to be chosen carefully to ensure efficiency.

The great flexibility of the distance-based approach is founded in the similarity distance measure. If O is the domain of the objects in the database, a similarity distance $d_{sim} : O \times O \mapsto \mathbb{R}^0_+$ is needed which means the only restriction for the similarity measure is positivity. While this very high flexibility may be useful in certain special applications, it usually makes sense to impose some restrictions on the similarity distance measure in order to ensure that efficient query processing is possible.

The restrictions imposed on the similarity measure can be summarized by demanding the measure to be a metric, which also justifies to call it a similarity distance. This requirment implies that the similarty measure has to fulfill the four metric properties:

1. Positivity: $\forall x, y : d_{sim}(x, y) \geq 0$
2. Definiteness: $\forall x, y : d_{sim}(x, y) = 0 \Leftrightarrow x = y$
3. Symmetry: $\forall x, y : d_{sim}(x, y) = d_{sim}(y, x)$
4. Triangle inequality: $\forall x, y, z : d_{sim}(x, z) \leq d_{sim}(x, y) + d_{sim}(y, z)$

The requirements of positivity and definiteness for the similarity distance reflect the idea that a low distance means high similarity and, therefore, identical objects should be assigned the lowest possible similarity distance. The idea that objects are mutually similar is expressed by the symmetry requirement. The triangle inequality ensures that no object can be very similar to two very dissimilar objects at the same time.

Demanding metric properties from a similarity distance also has the advantage that efficient access methods and search algorithms can be applied, as described in Section 2.3.

2.1.4 Invariance against Transformations

Another important topic in the context of similarity models is robustness against geometric transformations of the original data objects. Similarity

search is often done in databases containing geometric descriptions of real-world objects, like molecules, images or mechanical parts. Our example applications are also from such application domains, so we discuss the robustness against geometric transformations.

By "robustness against geometric transformations" we mean invariance against transformations such as translation, rotation or scaling. Depending on the application, specific invariances are either necessary or have to be avoided. An example application is similarity search in a database of proteins. Since there is no standard position or orientation of proteins defined, the proteins in the database have arbitrary orientation and position in 3D space. Consequently, invariance against translation and rotation are essential to identify similar proteins. On the other hand, invariance against scaling is unwanted, because proteins with different size but similar shape have different properties and should not be considered as similar.

2.1.5 Adaptable Similarity Search

In the previous sections, the adaptability of the different models and techniques was highlighted several times. This adaptability is of great importance for similarity search applications, because the exact definition of what is to be considered similar depends on two factors, which are the application domain and the user. An example of application requirements is our protein docking application, where we saw that invariance against translation and rotation is necessary while invariance against scaling has to be avoided. Therefore, the similarity model and the similarity measure have to provide enough flexibility to allow adaption to the specific needs of an application.

Apart from the application needs, the notion of similarity can differ between individual users or even for a single user in different situations. Similarity search is often an explorative process during which the user refines his notion of similarity more and more. The adaption to the application's needs can be considered during the design phase of the application and an adaption of the similarity model is possible in this phase. This approach can not be followed for the adaption to the users needs, since those can change between

two similarity queries. Consequently, the similarity measure has to provide the flexibility to allow the necessary adaption at runtime. Obviously, this should be possible with as little influence on query runtimes as possible, to support the explorative nature of the similarity search process. We already discussed the quadratic form distance as an example for such a measure. In [Sei97] efficient query processing techniques are presented which allow an adaption of the similarity matrix for this measure without influencing the processing time negatively.

But for a purposeful adaption of the similarity measure, another point gains importance. The user has to be able to understand why objects are considered similar by the application in order to change parameters appropriately. Consequently, the user should be provided with an explanation of the similarity distance value to support his understanding. Obviously, a simple numerical value does not fulfill this requirement. Instead, an explanation how this value comes about is necessary, which is preferably presented visually for a quick and easy understanding.

2.2 Similarity Query Types

In similarity search applications, the query types differ from those in standard database applications. Questions like which database objects are most similar to a query object or which database objects are similar to a certain degree, cannot be answered by using exact-match or partial-match queries. Instead, query algorithms returning database objects in a certain similarity distance to a query object are needed. In this section, we will present those query types which are most important in similarity search applications. For the presentation of the query types, we assume that O is the universe of all objects that may appear in a database and that a similarity distance function $d_{sim} : O \times O \mapsto \mathbb{R}^0_+$ is defined on the universe O. Furthermore, we presume that there is a database $DB \subseteq O$ given. It has to be noted that we do not assume a specific similarity model and the discussions below hold for applications based on the feature vector model as well as for applications using the distance-based similarity model.

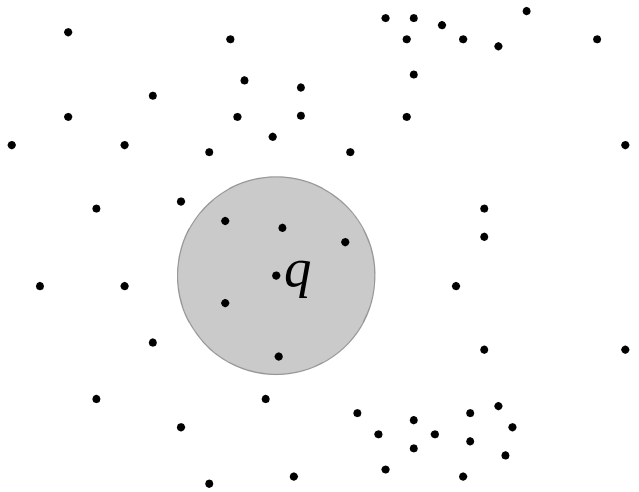


Figure 2.3: Result of a range query for object q.

2.2.1 Similarity Range Query

A basic task in similarity search is to find all objects which are within a certain similarity distance from a query object. Examples where this problem has to be solved are density-based clustering methods like DBSCAN [EKSX96] or OPTICS [ABKS99], which are described further in Chapter 3. In density-based clustering, an object o is put into a cluster if there are enough other objects within a predefined similarity distance to o. To determine a clustering of a database, for each object in the database the objects within the predefined similarity distance have to be found. This is done by using similarity range queries. Figure 2.3 illustrates the idea of the similarity range query.

With this intuitive understanding of a similarity range query, we can define it formally in the following way:

Definition 2 (similarity range query). For a query object $q \in O$ and a query range $\varepsilon \in \mathbb{R}^0_+$, the result of a similarity range query is defined as

$$N_\varepsilon(q) = \{o \in DB \mid d_{sim}(q, o) \leq \varepsilon\}$$

Obviously, with this definition the number of results for a similarity range query is not fixed in advance, but can be anything between zero and the size

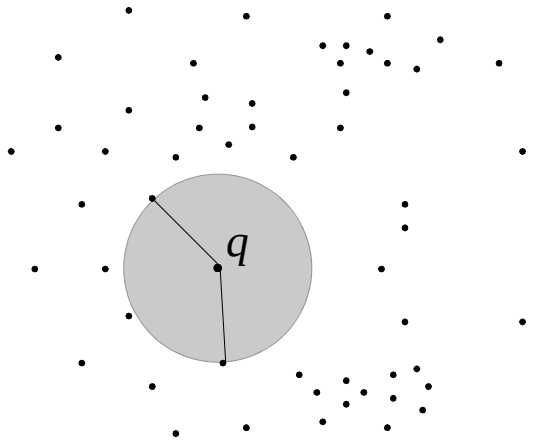


Figure 2.4: Result of a nearest-neighbor query with two nearest neighbors for query object q. The gray circle represents the equivalent range query.

of the database. Consequently, the choice of an inappropriate value for the query range ε leads to very few or too many query results and it remains to the user to rerun the query with an adapted query range. This problem is another reason, why a similarity measure should also include an explanation of the distance value to allow an adaption of the query range.

2.2.2 Nearest-Neighbor Query

Another important task in similarity search applications is to find the database object which is most similar to a query object. An example for this query type is to find the most similar protein with known function in a database, given a query protein with unknown function. This type of query is called nearest-neighbor query and can be defined informally as the task to find the database object with the smallest similarity distance to the query object. Figure 2.4 illustrates the idea of the nearest-neighbor query.

But this informal definition ignores the problem that the database object with the smallest distance may not be unique. In this case, one of the objects with the smallest similarity distance to the query object may be chosen randomly. But then query processing is no longer deterministic and important results may be missed. Therefore, the nearest-neighbor query is defined

in a way that allows a set of results which possibly contains more than one element.

Definition 3 (nearest-neighbor query). For a query object q, the result of a nearest-neighbor query is defined as

$$NN(q) = \{o \in DB \mid \forall p \in DB : d_{sim}(q, o) \leq d_{sim}(q, p)\}$$

With this definition, it remains to the user to resolve the ambiguity problem, but still, the result is at least a non-empty set. Especially when exploring a database manually, the guaranteed result is an advantage over the similarity range query for the user. The following lemma reveals another relationship between nearest-neighbor and range queries.

Lemma 1. *For every query object $q \in O$, the following holds:*

$$\varepsilon_{nn} = min\{d_{sim}(q, o), o \in DB\} \Rightarrow NN(q) = N_{\varepsilon_{nn}}(q)$$

Proof. For every object $o \in DB$ the following equivalences hold:

$$\begin{aligned} & o \in NN(q) \\ \Leftrightarrow\ & \forall p \in DB : d_{sim}(q, o) \leq d_{sim}(q, p) \\ \Leftrightarrow\ & d_{sim}(q, o) \leq min\{d_{sim}(q, p), p \in DB\} \\ \Leftrightarrow\ & d_{sim}(q, o) \leq \varepsilon_{nn} \\ \Leftrightarrow\ & o \in N_{\varepsilon_{nn}}(q) \end{aligned}$$

□

The lemma shows that every nearest-neighbor query can be transformed into a similarity range query, although the nearest-neighbor distance ε_{nn} is generally not known in advance.

2.2.3 k-Nearest-Neighbor Query

The k-nearest-neighbor query is an extension of the nearest-neighbor query in case, a result set with more than one element is desired. An example of such

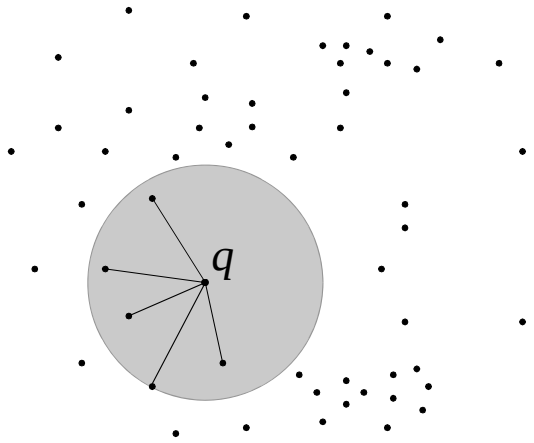


Figure 2.5: Result of a k-nearest-neighbor query for object q and $k = 5$. The gray circle represents the equivalent range query.

a case is the functional classification of proteins. To improve classification accuracy for nearest-neighbor classification, a protein is not assigned to the functional class of the most similar protein in the database but to the class of the majority of the k most similar proteins. The idea of the k-nearest-neighbor query is illustrated in Figure 2.5.

Like the nearest neighbor for a query object, the k-th nearest neighbor may not be unique and, therefore, the result of a k-nearest-neighbor query may contain more than k elements.

Definition 4 (k-nearest-neighbor query). For a query object $q \in O$ and a query parameter k, the k-nearest-neighbor query returns the smallest set $NN_k(q) \subseteq DB$ that contains (at least) k objects from the database, and for which the following condition holds:

$$\forall o \in NN_k(q) \forall p \in (DB - NN_k(q)) : d_{sim}(q, o) < d_{sim}(q, p)$$

Obviously, Lemma 1 holds analogously for the k-nearest-neighbor query which means that every k-nearest-neighbor query can also be transformed into a similarity range query with the same result.

2.2.4 Similarity Ranking Query

A final important similarity query type is the similarity ranking query which is needed in cases where the exact number of desired results is not known in advance. The idea of this query type is to iteratively retrieve the next closest objects of a query object from the database, starting at the nearest neighbor. This type of query appears, for example when the user interactively explores the database and retrieves the nearest neighbors of a query object one after another. Such queries could be done by issuing k-nearest-neighbor queries with increasing parameter k. But this would result in retrieving the nearest neighbor and other objects several times, i.e. again and again for each k-nearest-neighbor query. Therefore, an algorithm for similarity ranking queries should not start over again for each request of a new object and should not perform all the similarity searching while processing the first request to ensure interactive response times. Hijaltason and Samet presented an algorithm with those properties in [HS95].

2.3 Efficient Similarity Search

The size of modern databases and the complexity of the similarity searching task make efficiency an important issue for any similarity search application. In this section, we will present two techniques to speed up the query processing in similarity search applications. The two techniques, the use of index structures, and the use of a multi-step query processing architecture, are not meant to be mutually exclusive. Instead, they can both be applied in parallel or at different stages of the query processing.

2.3.1 Index Structures

The use of index structures is a standard technique to improve query processing times in database systems. Numerous different index structures have been proposed for many different data types and applications. For similarity search applications two types of structures are important: structures for high-

dimensional vector spaces and for metric spaces. The first category is primarily useful whenever the feature vector approach is used as the similarity model.

Metric index structures, on the other hand, can be applied if the distance-based similarity model is chosen, provided that the similarity measure fulfills the metric properties. But especially for the distance-based similarity model, where the similarity measure is often complex, speeding up the query processing is essential.

In the following, we will present the principles of important index structures for vector spaces as well as metric spaces.

Indexing Vector Spaces

The two main paradigms for index structures are hashing and tree structures. While there exist hashing approaches for vector spaces [NHS84, KS86], the vast majority of index structures for vector spaces are hierarchical data organizing structures. The idea behind those structures is to organize the vector data in a tree like directory to ensure logarithmic time complexity of index updates and search accesses. To achieve a tree structure for the index, the data vectors are grouped into pages which are described by a page region covering the entire subspace occupied by the data vectors on the page. The data pages are grouped into directory pages in the same manner until this recursive process yields a single root page. The many index structures following this approach differ in the shape and size of the page regions, the strategy for splitting pages and the insertion strategies. Examples of index structures following this paradigm are, among many others, the members of the R-tree family [Gut84, BKSS90], the X-tree [BKK96] and the IQ-tree [BBJ+00].

Indexing Metric Spaces

Index structures for metric spaces are more general than structures for vector spaces in the sense that they can also be applied to vector spaces, since every vector space is also a metric space. Like structures for vector spaces, index

structures for metric spaces also group the data objects into data pages. But since there is only a distance measure given between pairs of objects, no arbitrarily formed page regions are possible. The limitation of the distance measure results in ball-shaped or ring-shaped page regions. For the description of the page regions, one or more representatives from the data objects together with a radius have to be chosen. The many index structures for metric spaces mainly differ in the way, those representatives are chosen. Examples of index structures for metric spaces are GNAT [Bri95] or the family of vantage-point trees [Uhl91, Yia93, BÖ97]. Chávez et al. give an overview over existing approaches for indexing metric spaces in [CNBYM01].

Since even in data mining applications regular updates of the database are common, dynamic index structures for metric spaces are the most important variants for our similarity search applications. The M-tree [CPZ97] and its variant the Slim-tree [TJTSF00] are specifically designed to allow dynamic updates. Furthermore, those structures are also designed to reduce the number of similarity distance calculations which is especially important for costly similarity measures like they are common for complex data.

2.3.2 Multi-Step Query Processing

The complexity of the similarity distance measure is often a problem for efficient query processing in similarity search applications. Index structures are one way to exclude unnecessary parts of the database from scanning, which reduces the number of necessary similarity distance calculations. Another way to reach this reduction goal is to employ a multi-step query processing architecture.

To reduce the number of necessary distance calculations, the query processing in a multi-step query processing architecture, as depicted in Figure 2.6, is performed in two or more steps. The first step is a filter step which returns a number of candidate objects from the database. For those candidate objects, the exact similarity distance is then determined in the refinement step and the objects fulfilling the query predicate are reported. To reduce the overall search time, the filter step has to fulfill certain constraints. First, it is

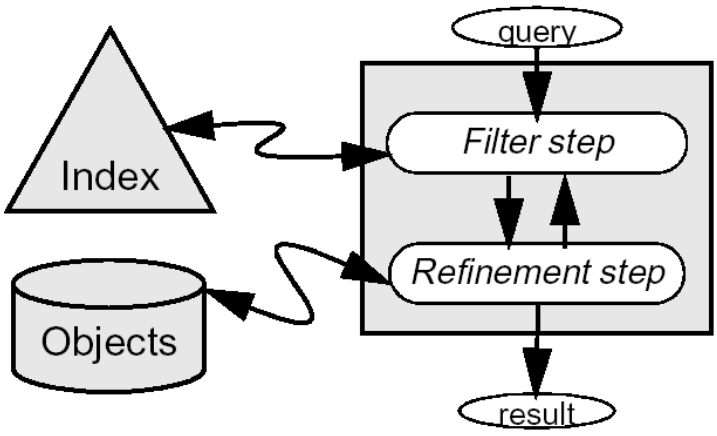


Figure 2.6: Schema of a multi-step query processing architecture.

essential that the filter predicate is considerably easier to evaluate than the exact similarity measure. Second, a substantial part of the database objects must be filtered out. Obviously, it depends on the complexity of the similarity measure which filter selectivity is sufficient. Only if both conditions are satisfied, the performance gain through filtering is greater than the cost for the extra processing step.

Additionally, the completeness of the filter step is essential. Completeness in this context means that all database objects satisfying the query condition are included in the candidate set or in other words, it must be guaranteed that no false drops occur during the filter step. Available similarity search algorithms guarantee completeness if the distance function in the filter step fulfills the lower-bounding property.

Definition 5 (lower-bounding property). For any two objects p and q, a lower-bounding distance function $D_f(p, q)$ in the filter step has to return a value that is not larger than the exact object distance D_o of p and q, i.e. $\forall p, q : d_{lb}(p, q) \leq d_e(p, q)$.

With a lower-bounding distance function it is possible to safely filter out all database objects which have a filter distance larger than the current query range, because the similarity distance of those objects cannot be less than the query range.

Using a multi-step query architecture requires efficient algorithms that actually use the filter steps. Agrawal, Faloutsos and Swami proposed such an algorithm for range queries [AFS93]. In [SK98] and [KSF+98] multi-step algorithms for k-nearest-neighbor search were presented which are optimal in the sense that the minimal number of exact distance calculations are performed during query processing.

2.4 Requirements for Similarity Measures

In the preceeding sections, we discussed several aspects of similarity search applications. From those discussion, we can now derive a few requirements which a similarity measure for complex data should fulfill.

One requirement for a similarity measure for complex data is that structural as well as content-related information has to be taken into account. Therefore, in the case of graphs, the measure should also be defined for attributed graphs and not only for simple graphs.

In Section 2.1.5, we showed that the similarity measure should be adaptable to the needs of specific applications and to the needs of the users. This adaption should be possible between two queries without negative effects on the performance of the query processing step.

Another requirement is closely related to the first one. It is necessary to provide an explanation of the similarity distance value between two data objects, to allow the user a purposeful and easy adaption of the parameters of the similarity distance measure.

The final two requirements are concerned with the efficiency of the query processing in similarity search applications. First, the measure should be of moderate time complexity, since it has to be evaluated often, especially in today's large and fast growing databases. Finally, a similarity distance measure should be a metric in order to allow the use of index structures and multi-step query processing techniques.

2.5 Summary

In this chapter, we discussed several aspects of similarity search applications. In the beginning, we presented two different models for the similarity of objects, namely the feature vector approach and the distance-based model. We discussed the strengths and weaknesses of those models and showed that the distance-based model has advantages especially for complex data. Furthermore, the problems of invariance against transformations and of adaptability to application and user needs were discussed.

Afterwards, we presented query types which are important in similarity search applications. Those query types form the basis for the evaluation of the similarity measures in the later chapters. Two different techniques to ensure efficient query processing were presented in Section 2.3.

Finally, the discussions lead to five requirements which a similarity measure for complex objects should fulfill in order to be useful in modern database systems.

Chapter 3

Density-Based Clustering

Clustering is the task of grouping objects of a database into classes, such that objects within one cluster are most similar to each other while object of different clusters are most dissimilar to each other. Many clustering algorithms have been proposed in recent years. This thesis will base on density-based clustering which turned out to be one of the most effective and also efficient clustering approaches. The clustering algorithms described in this chapter have in common that they are based on the successive computation of similarity range queries as introduced in Section 2.2.1 for each object in the database. Therefore, clustering relies on computing the distance between objects and, thus, the complexity of the underlying similarity model has a severe influence on the efficiency of clustering algorithms. Especially for density-based clustering, similarity range queries must be supported efficiently to reduce the runtime of clustering.

In this chapter, we will first give basic notations in Section 3.1 to establish the foundations of density-based clustering. After that, in Sections 3.2 and 3.3, we provide a detailed introduction to the density-based notions of clusters. In particular, we introduce the notion of *flat density-connected sets* as proposed in [EKSX96] providing the basis of the algorithm DBSCAN and discuss the hierarchical extensions leading to the notion of *density-based cluster orderings* as proposed in [ABKS99] which constitutes the central concept of the algorithm OPTICS.

3.1 Foundations

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 *MinPts* of objects, i.e. the cardinality of the neighborhood has to exceed a given threshold. In the following, we will present the basic definitions of density-based clustering.

Definition 6 (directly density-reachable). An object p is *directly density-reachable* from an object q w.r.t. ε and *MinPts* in a set of objects DB, if $p \in N_\varepsilon(q)$ and $|N_\varepsilon(q)| \geq MinPts$, where $N_\varepsilon(q)$ denotes the subset of DB contained in the ε-neighborhood of q.

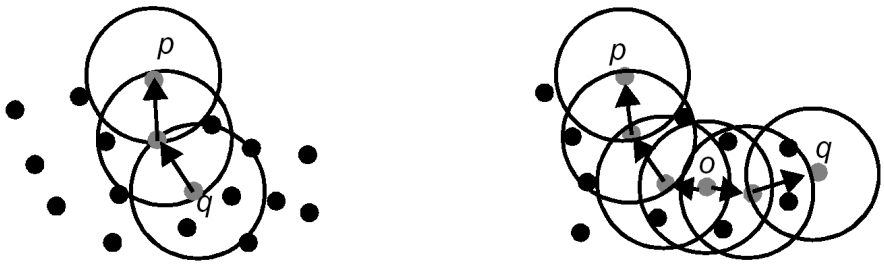
The condition $|N_\varepsilon(q)| \geq MinPts$ is called the core object condition. If this condition holds for an object q, then we call q a core object. Other objects can be directly density-reachable only from core objects.

Definition 7 (density-reachable and density-connected). An object p is *density-reachable* from an object q w.r.t. ε and *MinPts* in a set of objects DB, if there is a chain of objects $p_1, \ldots p_n$, $p_1 = q$, $p_n = p$, such that $p_i \in DB$ and p_{i+1} is directly density-reachable from p_i w.r.t. ε and *MinPts*. Object p is *density-connected* to object q w.r.t. ε and *MinPts* in a set of objects DB, if there is an object $o \in DB$, such that both p and q are density-reachable from o in DB w.r.t. ε and *MinPts*.

Density-reachability is the transitive closure of direct density-reachability and does not have to be symmetric. On the other hand, density-connectivity is symmetric (cf. Figure 3.1).

3.2 Partitioning Clustering

A flat density-based cluster is defined as a set of density-connected objects which is maximal w.r.t. density-reachability. Then the noise is the set of objects not contained in any cluster. A cluster contains not only core objects but also objects that do not satisfy the core object condition. These border



(a) p density-reachable from q, but q not density-reachable from p.

(b) p and q density-connected to each other by o.

Figure 3.1: Density-reachability and density-connectivity.

objects are directly density-reachable from at least one core object of the cluster.

The algorithm DBSCAN [EKSX96], which discovers the clusters and the noise in a database, is based on the fact that a cluster is equivalent to the set of all objects in DB which are density-reachable from an arbitrary core object in the cluster (cf. Lemmas 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 $N_\varepsilon(q)$ of a point q has more than *MinPts* elements, q is a so-called core point, and a new cluster C containing the objects in $N_\varepsilon(q)$ is created. Then, the ε-neighborhood of all points p in C which have not yet been processed is checked. If $N_\varepsilon(p)$ contains more than *MinPts* points, the neighbors of p 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. Then the algorithm continues with a point which has not yet been processed trying to expand a new cluster.

3.3 Hierarchical Clustering

While the partitioning density-based clustering algorithm DBSCAN can only identify a "flat" clustering, the newer algorithm OPTICS [ABKS99] com-

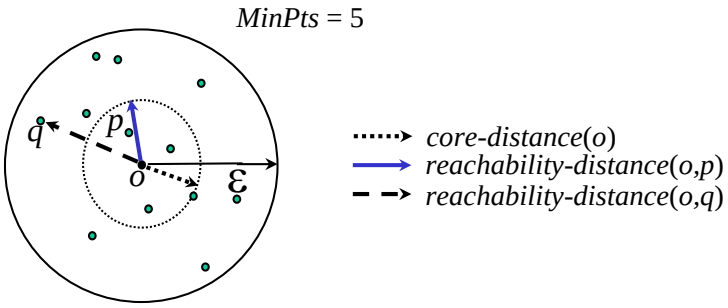


Figure 3.2: Illustration of core-distance and reachability-distance.

putes an ordering of the points augmented by additional information. In the following, we will shortly introduce the definitions underlying the OPTICS algorithm, the core-distance of an object p and the reachability-distance of an object p w.r.t. a predecessor object o.

Definition 8 (core-distance). Let p be an object from a database DB, let $N_\varepsilon(p)$ be the ε-neighborhood of p, let *MinPts* be a natural number and let $MinPtsDist(p)$ be the distance of p to its *MinPts*-th neighbor. Then, the *core-distance* of p, denoted as $CoreDist(p)$ is defined as $MinPtsDist(p)$ if $|N_\varepsilon(p)| \geq MinPts$ and INFINITY otherwise.

Definition 9 (reachability-distance). Let p and o be objects from a database DB, let $N_\varepsilon(o)$ be the ε-neighborhood of o, let $dist(o,p)$ be the distance between o and p, and let *MinPts* be a natural number. Then the *reachability-distance* of p w.r.t. o, denoted as $ReachDist(p,o)$, is defined as $\max(CoreDist(o), dist(o,p))$.

Figure 3.2 illustrates both concepts: The reachability-distance of p from o equals to the core-distance of o and the reachability-distance of q from o equals to the distance between q and o.

The OPTICS algorithm is given in (cf. Figure 3.3). It creates an ordering of a database, along with a reachability-distance for each object. The main data structure is the so-called *seedlist*, containing tuples of points and reachability-distances. The seedlist is organized w.r.t. ascending reachability-distances. Initially the seedlist is empty and all points are marked as *not-done*.

```
algorithm OPTICS
begin
   repeat
      if the seedlist is empty
         if all points are marked "done", terminate;
         choose "not-done" point q;
         add (q, INFINITY) to the seedlist;
      end if;
      (o1, r) = seedlist entry having the smallest reachability value;
      remove (o1, r) from seedlist;
      mark o1 as "done";
      output (o1, r);
      update-seedlist(o1);
   end repeat;
end;
```

Figure 3.3: The OPTICS algorithm.

The procedure update-seedlist(o_1) executes an ε-range query around the point o_1, i.e. the first object of the sorted seedlist, at the beginning of each cycle. For every point p in the result of the range query, it computes $r = \mathit{ReachDist}(p, o_1)$. If the seedlist already contains an entry (p, s), it is updated to $(p, \min(r, s))$, otherwise (p, r) is added to the seedlist. Finally, the order of the seedlist is reestablished.

In contrast to DBSCAN, OPTICS does not assign cluster memberships but computes an ordering in which the objects are processed and additionally generates the information which would be used by an extended DBSCAN algorithm to assign cluster memberships, i.e. the core-distance and the reachability-distance. The original output of OPTICS is the so-called *cluster ordering*:

Definition 10 (cluster ordering). Let $\mathit{MinPts} \in \mathbb{N}$, $\varepsilon \in \mathbb{R}$, and CO be a totally ordered permutation of the database objects. Each $o \in D$ has additional attributes $o.P$, $o.C$ and $o.R$, where $o.P \in \{1, \ldots, |CO|\}$ symbolizes the position of o in CO. We call CO a *cluster ordering* w.r.t. ε and MinPts if the following three conditions hold:

1. $\forall p \in CO : p.C = \mathit{CoreDist}(p)$

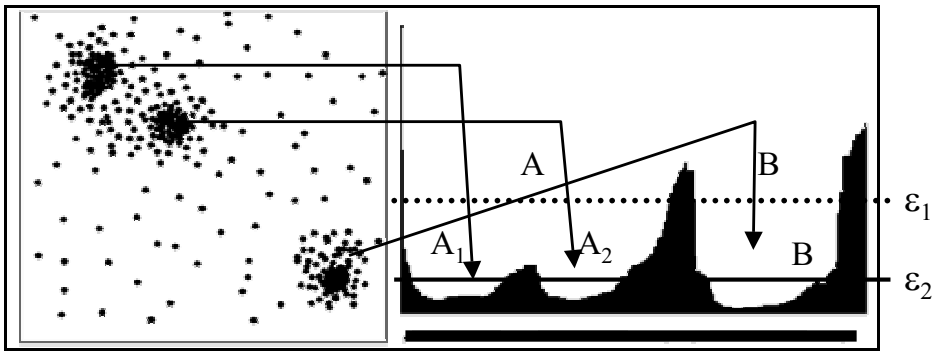


Figure 3.4: Reachability plot computed by OPTICS for a 2D data set.

2. $\forall o, x, y \in CO : o.P < x.P \wedge x.P < y.P \Rightarrow$
   $ReachDist(o, x) \leq ReachDist(o, y)$

3. $\forall p, o \in CO : R(p) = \min\{ReachDist(o, p) \mid o.P < p.P\}$,
   where $\min \emptyset = \infty$.

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

The cluster structure can be visualized by so called reachability plots which are 2D plots generated as follows: the clustered objects are ordered along the x-axis according to the cluster ordering computed by OPTICS and the reachabilitiy distances assigned to each object are plotted along the abscissa. An example reachability plot is depicted in Figure 3.4. Valleys in this plot indicate clusters: objects having a small reachability value are closer and thus more similar to their predecessor objects than objects having a higher reachability value.

The reachability plot generated by OPTICS can be cut at any level ε_{cut} parallel to the abscissa. It represents the density-based clusters according to the density threshold ε_{cut}: A consecutive subsequence of objects having a smaller reachability value than ε_{cut} belongs to the same cluster. An example

is presented in Figure 3.4: For a cut at the level ε_1 we find two clusters denoted as A and B. Compared to this clustering, a cut at level ε_2 would yield three clusters. The cluster A is split into two smaller clusters denoted by A_1 and A_2 and cluster B decreased its size. Usually, for evaluation purposes, a good value for ε_{cut} would yield as many clusters as possible.

Part II

Multi-Step Similarity Search and Clustering

Chapter 4

Efficient Similarity Search on Vector Sets

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, purchasing assistance, and others [Jag91, AFS93, FBF+94, FRM94, ALSS95].

As distance functions form the foundation of similarity search, we need an object representation which allows efficient and meaningful distance computations. A common approach is to represent an object by a numerical feature vector. In this case, a feature transformation extracts distinguishable characteristics which are represented by numerical values and grouped together in a feature vector. On the basis of such a feature transformation and under the assumption that similarity corresponds to feature distance, it is possible to define a distance function between the corresponding feature vectors as a similarity measure for two data objects. Thus, searching for data objects similar to a given query object is transformed into proximity search in the feature space. Most applications use the Euclidean metric (L_2) to evaluate the feature distance, but there are several other metrics commonly used, e.g. the Manhattan metric (L_1) and the maximum metric (L_∞).

Furthermore, there exist quite a few much more complex similarity models based on graphs [KS03] and trees [KKSS04]. Generally, the more complex and precise these models are, the more exact are the results of a similarity search, but at the same time, its computation cost rises as well.

In this chapter, we present a distance measure for an approach somewhere in between single feature vectors and complex trees and graphs. We model an object by a *set of feature vectors* which is a very suitable object representation for many different application ranges. In order to achieve efficient query processing we present three different lower-bounding filters and discuss their properties.

The remainder of this chapter is organized as follows. In Section 4.1, we motivate the use of vector set represented objects by presenting various application ranges which benefit from this modeling approach. In Section 4.2, we introduce the minimal matching distance between vector sets which is a suitable distance measure for partial and complete similarity search. In Section 4.3, we sketch the paradigm of multi-step query processing and present appropriate filter techniques for the minimal matching distance on vector sets. In Section 4.4, we present the results of our experimental evaluation. The chapter concludes in Section 4.5 with a short summary.

4.1 Application Ranges for Vector Sets

Using sets of feature 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 some applications the properties of vector set representations 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. Another advantage of using sets of feature vectors 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. In the following, we will shortly sketch different application ranges which benefit from the use of vector set data.

CAD databases. In [KBK+03] voxelized spatial objects were modeled by sets of feature vectors, where each feature vector represents a 3D rectangular cover which approximates the object as good as possible. The vector set representation is able to avoid the problems that occur by storing a set of covers according to a strict order, i.e. in one high-dimensional feature vector. Thereby, it is possible to compare two objects more intuitively compared to the distance calculation in the one-vector model. In a broad experimental evaluation it was shown that the use of sets of feature vectors greatly enhances the quality of the similarity model compared to the use of a single feature vector.

Soccer teams. As another example, let us assume that we want to measure the similarity between two soccer teams. It is beneficial to represent each player by a feature vector and the complete team as a set of feature vectors. A feature vector for one player may consist of attributes like his age, his salary, the number of goals in the last season, etc. We can compare two players by computing the Euclidean distance between the corresponding feature vectors. This measures the similarity between two players rather well. But, what is a suitable distance for comparing two teams? Assuming we have a team A consisting of 10 very young players having a low salary and having scored only a few goals in the last season. Furthermore, team A has one highly paid, rather experienced and successful player. On the other hand, we have a team B where we have 10 rather old, highly paid successful players and one young low-budget player. If we compare each player of team A to the most similar player in team B and vice versa, this yields that the two teams are very similar. This straightforward approach does not reflect the intuitive notion of similarity. On the other hand, if we compare each player from team A to a different player in team B trying to minimize the average distance between two "matched" players, this results in a very accurate similarity measure.

For partial similarity, it is advisable not to compare all players from team A to a different player in team B, but only the s most similar players. For low values of s, e.g. $s = 2$, the two teams A and B are very similar, as each team has an old player with a high salary and a young low-budget player. In this case, the distance between the teams A and B would be very small. For higher values of s, the two teams become more and more dissimilar. Let us note that for $s = 11$ the two notions of partial and complete similarity coincide. This behavior reflects the intuitive perception of similarity. To sum up, the use of vector sets allows us to adjust the degree of the partial similarity in k discrete steps, if we represent the objects by vector sets of cardinality k.

Further application areas. There exist a lot of further possible application fields for sets of feature vectors, e.g.:

- *stock portfolios*, where each stock is represented by the value of one share, the overall number of shares, how many days ago the shares were bought, the risk category, etc.

- *shopping carts*, where each consumer product corresponds to a feature vector containing the category, the price, the quantity, etc.

- *multimedia CDs*, where each media file is represented by the publisher, the artist, the title, the filesize, the kind of content, etc.

- *research teams*, where each researcher is modeled by the number of publications, his age, his salary, etc.

- *school classes*, where each student is represented by a feature vector consisting of his marks in Mathematics, English, etc.

- *car manufacturers*, where each car model is represented by the list price of a new car, the number of produced cars from this model, the average mileage, etc.

- *galaxies*, where each star is modeled by a single feature vector containing attributes reflecting the luminance of the star, its size, its age, etc.

- *married couples*, where each person is modeled by his/her size, age, weight, skin color, nationality, education, salary, etc.

To sum up, sets of feature vectors are a natural way to model a lot of complex real-world objects.

4.2 Distance Measures on Vector Sets

Effective distance functions which allow both complete and partial similarity search as well as suitable filter techniques for efficient query processing are indispensable for the general use of the powerful concept of "sets of feature vectors".

There are already several distance measures proposed on sets of vectors. In [EM97] the authors survey the following four measures, which are computable in polynomial time: the Hausdorff distance, the sum of minimum distances, the (fair-)surjection distance and the link distance. The Hausdorff distance 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 modeling 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 in the application areas presented in Section 4.1.

A distance measure on vector sets that demonstrates to be suitable for defining similarity is based on the *minimum weight perfect matching* of sets.

This well known graph problem can be applied here by building a complete bipartite graph $G = (X \cup Y, E)$ between the vector sets X and Y. The weight of each edge $(x, y) \in E$, where $x \in X$ and $y \in Y$, in this graph G is defined by the distance $d(x, y)$. A perfect matching is a subset $M \subseteq E$ that connects each $x \in X$ to exactly one $y \in Y$ and vice versa. A minimum weight perfect matching is a matching with a minimum sum of weights of its edges. Contrary to the second example of Section 4.1, where we considered vector sets of equal cardinality, i.e. soccer teams consisting of 11 players, there are a lot of application ranges, where objects are naturally represented by a varying number of vectors. Since a perfect matching can only be found for sets of equal cardinality, we need to introduce suitable weights as a penalty for the unmatched vectors when defining a distance measure between objects of varying cardinality.

Definition 11 (permutation of a set). Let A be any finite set of arbitrary elements. Then π is a mapping that assigns $a \in A$ a unique number $i \in \{1, .., |A|\}$. This is written as $\pi(A) = (a_1, .., a_{|A|})$. The set of all possible permutations of A is denoted by $\Pi(A)$.

Definition 12 (minimal matching distance). Let $V \subset \mathbb{R}^d$ and let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function between two d-dimensional feature vectors. Furthermore, let $W : V \to \mathbb{R}$ be a weight function for unmatched elements. Then the *minimal matching distance* $D_{\text{mm}}^{D,W} : 2^V \times 2^V \to \mathbb{R}$ is defined as follows:

$$D_{\text{mm}}^{D,W}(X, Y) = \min_{\pi \in \Pi(Y)} \left(\sum_{i=1}^{|X|} D(\vec{x}_i, \vec{y}_{\pi(i)}) + \sum_{i=|X|+1}^{|Y|} W(\vec{y}_{\pi(i)}) \right)$$

The weight function W provides the penalty given to every unassigned element of the set having larger cardinality. Let us note that the minimal matching distance is a specialization of the *netflow distance* which is proven to be a metric in [RB01]. The minimal matching distance $D_{\text{mm}}^{D,W}$ is a metric, if the distance function D is a metric and the weight function W meets the following conditions:

(1) $W(\vec{x}) > 0$ for $\vec{x} \in V$
(2) $W(\vec{x}) + W(\vec{y}) \geq D(\vec{x}, \vec{y})$ for $\vec{x}, \vec{y} \in V$

The Kuhn-Munkres algorithm [Kuh55, Mun57] can be used to calculate the minimal matching distance in polynomial time. In a primary initialization step, a distance matrix between the two vector sets containing k d-dimensional vectors is computed. If D is an L_p-distance, this initialization takes $O(k^2 d)$ time. The method itself is based on the successive augmentation of an alternating path between both sets. Since it is guaranteed that this path can be expanded by one further match within each step taking $O(k^2)$ time and there is a maximum of k steps, the overall complexity of a distance calculation is $O(k^3 + k^2 d)$ in the worst case.

The minimal matching distance can be adapted for partial similarity search in vector set represented data. The distance measure defined in the following is based on a partial minimal matching. Given two vector sets X and Y, $|X| \leq |Y|$, we only match $s \leq |X|$ vectors to calculate the distance between X and Y.

Definition 13 (partial minimal matching distance). Let $V \subset \mathbb{R}^d$ and let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function between two d-dimensional feature vectors. Let $s \leq |X|$. Then the *partial minimal matching distance* $D^{D,s}_{\text{pmm}} : 2^V \times 2^V \to \mathbb{R}$ is defined as follows:

$$D^{D,s}_{\text{pmm}}(X, Y) = \min_{\pi_1 \in \Pi(X), \pi_2 \in \Pi(Y)} \left(\sum_{i=1}^{s} D(\vec{x}_{\pi_1(i)}, \vec{y}_{\pi_2(i)}) \right)$$

Unlike the minimal matching distance the partial variant is not a metric. As the Kuhn-Munkres algorithm produces a partial minimal matching in each step as an intermediate result, we can use it to calculate the partial minimal matching distance $D^{D,s}_{\text{pmm}}(X, Y)$. But we have to take into account all $\binom{|X|}{s}$ combinations of vectors in X to match with vectors in Y. Therefore, the time complexity for a single distance calculation is $O(\binom{k}{s} sk^2 + k^2 d)$. Thus, a filtering technique to speed up query processing is essential.

4.3 Filters for Vector Sets

Complete similarity search on vector set data can be accelerated by using metric index structures, e.g. the M-tree [CPZ97]. For a detailed survey on metric index structures we refer the reader to [CNBYM01]. Another approach is to use the multi-step query processing paradigm which, in contrast to metric index structures, is also suitable for partial similarity search. The main goal of multi-step query processing is to reduce the number of complex and therefore time consuming distance calculations in the query process. In order to guarantee that there occur no false drops the used filter distances have to fulfill a lower-bounding distance criterion. As defined in Definition 5, for any two objects o_1 and o_2, a lower-bounding distance function D_f in the filter step has to return a value that is not greater than the exact object distance D_o of o_1 and o_2. With a lower-bounding distance function, it is possible to safely filter out all database objects which have a filter distance greater than the current query range because the exact similarity distance of those objects cannot be less than the query range.

The computation of the minimal matching distance on vector sets is a rather expensive operation. Thus, the employment of selective and efficiently computable filter distance functions for similarity search is very important. In the following, we present three different filter types for query processing on data objects represented by vector sets, namely the *closest pair filter*, the *centroid filter* and the *norm vector filter*.

4.3.1 Closest Pair Approach

The *closest pair distance* between two vector sets X and Y can be used as a filter distance for the minimal matching distance $D_{\text{mm}}^{D,W}$ and is defined as follows.

Definition 14 (closest pair distance). Let $V \subset \mathbb{R}^d$ and $\vec{\omega} \in \mathbb{R}^d \setminus V$. Let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function. Let $X' = \{\vec{x}_1, \ldots \vec{x}_{|Y|}\}$ be a multiset where $\vec{x}_i = \vec{\omega}$ for $i \in \{|X|+1, \ldots |Y|\}$. Then

the *closest pair distance* $D_{\text{cp}}^{D,\vec{\omega}}(X,Y) : 2^V \times 2^V \to \mathbb{R}$ is defined as follows.

$$D_{\text{cp}}^{D,\vec{\omega}}(X,Y) = \max\left(\sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_i,\vec{y}_j), \sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_j,\vec{y}_i)\right)$$

Let us note that the closest pair filter works directly on the set of vectors, i.e. on the original data, and not on approximated data. The filter distance can be computed by scanning the matrix of distance values between each pair of vectors in X and Y for the closest pairs. We will now show that the closest pair distance between two vector sets is a lower bound for the minimal matching distance.

Theorem 1. *Let $V \subset \mathbb{R}^d$ and $\vec{\omega} \in \mathbb{R}^d \setminus V$. Let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function. Furthermore, let $W_{\vec{\omega}} : V \to \mathbb{R}$, $W_{\vec{\omega}}(\vec{v}) = D(\vec{v},\vec{\omega})$, be a weight function for unmatched elements. Then the following inequality holds:*

$$D_{\text{cp}}^{D,\vec{\omega}}(X,Y) \leq D_{\text{mm}}^{D,W_{\vec{\omega}}}(X,Y)$$

Proof. Let $\pi \in \Pi(Y)$ be the permutation of Y that results from the minimum weight perfect matching of X and Y, i.e.

$$D_{\text{mm}}^{D,W_{\vec{\omega}}}(X,Y) = \sum_{i=1}^{|X|} D(\vec{x}_i,\vec{y}_{\pi(i)}) + \sum_{i=|X|+1}^{|Y|} D(\vec{\omega},\vec{y}_{\pi(i)})$$

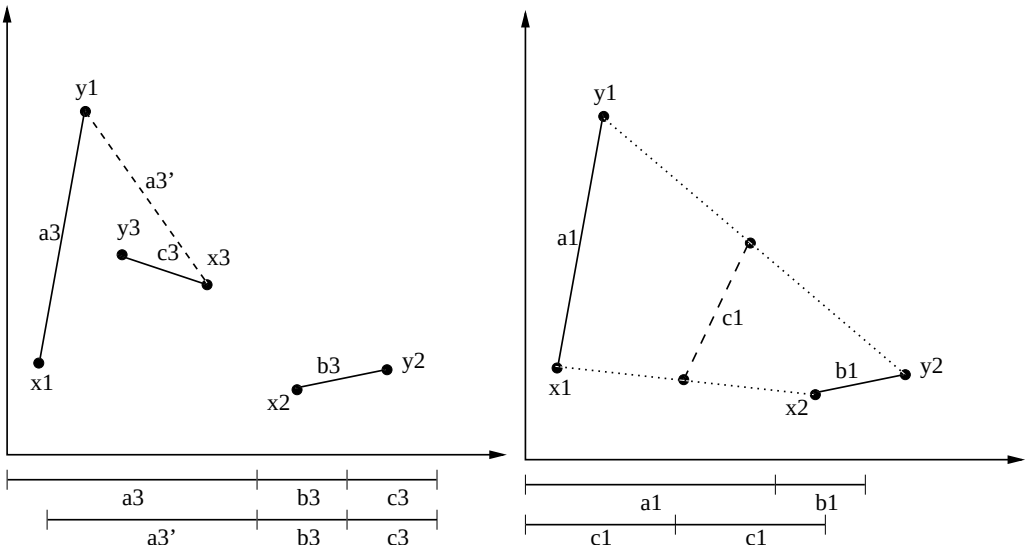
The proof consists of two cases.

Case 1: $D_{\text{cp}}^{D,\vec{\omega}}(X,Y) = \sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_i,\vec{y}_j)$.

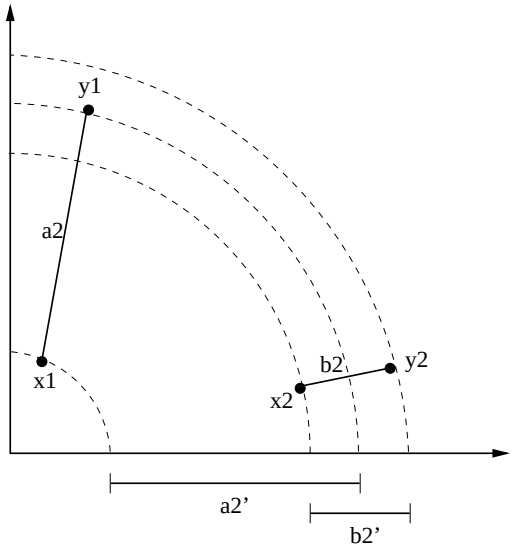
$$\sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_i,\vec{y}_j) =$$
$$\sum_{i=1}^{|X|} \min_{j=1,\ldots|Y|} D(\vec{x}_i,\vec{y}_j) + \sum_{i=|X|+1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{\omega},\vec{y}_j) \leq$$
$$\sum_{i=1}^{|X|} D(\vec{x}_i,\vec{y}_{\pi(i)}) + \sum_{i=|X|+1}^{|Y|} D(\vec{\omega},\vec{y}_{\pi(i)})$$

The inequality holds, if it holds for every pair of i-th addends. This is obviously the case, as we always pick the $\vec{y}_j \in Y$ which minimizes $D(\vec{x}_i,\vec{y}_j)$.

Case 2: $D_{\text{cp}}^{D,\vec{\omega}}(X,Y) = \sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_j,\vec{y}_i)$.



(a) Closest pair. **(b)** Centroid.



(c) Norm vector.

Figure 4.1: Filters for the minimal matching distance.

$$\sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_j, \vec{y}_i) = \sum_{i=1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_j, \vec{y}_{\pi(i)}) =$$
$$\sum_{i=1}^{|X|} \min_{j=1,\ldots|Y|} D(\vec{x}_j, \vec{y}_{\pi(i)}) + \sum_{i=|X|+1}^{|Y|} \min_{j=1,\ldots|Y|} D(\vec{x}_j, \vec{y}_{\pi(i)}) \leq$$
$$\sum_{i=1}^{|X|} D(\vec{x}_i, \vec{y}_{\pi(i)}) + \sum_{i=|X|+1}^{|Y|} D(\vec{\omega}, \vec{y}_{\pi(i)})$$

Again, the inequality holds, if it holds for every pair of i-th addends. This is obviously the case, as we always pick the $\vec{x}_j \in X'$ which minimizes $D(\vec{x}_j, \vec{y}_{\pi(i)})$ (note that $\vec{\omega} \in X'$ if $|X| < |Y|$). □

A 2-dimensional example for the closest pair filter is depicted in Fig. 4.1(a), where $|X| = |Y| = 3$ and

$$a'_3 + b_3 + c_3 = D_{\text{cp}}^{L_2,\vec{0}}(X, Y) \leq D_{\text{mm}}^{L_2,W_{\vec{0}}}(X, Y) = a_3 + b_3 + c_3.$$

As $a'_3 < a_3$, $\vec{x}_3$ is matched to both $\vec{y}_1$ and $\vec{y}_3$ during the filter distance calculation, whereas the minimal matching distance is based on one-to-one matchings.

We adapt the closest pair filter to partial similarity search by adding up just the distances of the s closest pairs of vectors. Thus, the *partial closest pair distance* is defined as follows.

Definition 15 (partial closest pair distance). Let $V \subset \mathbb{R}^d$ and let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function. Let $s \leq |X|$. Then the *partial closest pair distance* $D_{\text{pcp}}^{D,s}(X, Y) : 2^V \times 2^V \to \mathbb{R}$ is defined as follows.

$$D_{\text{pcp}}^{D,s}(X, Y) = \max \left(\min_{\pi \in \Pi(X)} \sum_{i=1}^{s} \min_{j=1,\ldots|Y|} D(\vec{x}_{\pi(i)}, \vec{y}_j), \right.$$
$$\left. \min_{\pi \in \Pi(Y)} \sum_{i=1}^{s} \min_{j=1,\ldots|X|} D(\vec{x}_j, \vec{y}_{\pi(i)}) \right)$$

The partial closest pair distance is a lower bound for the partial minimal matching distance.

Theorem 2. *Let $V \subset \mathbb{R}^d$ and let $X, Y \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $D : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be a distance function. Let*

$s \leq |X|$. *Then the following inequality holds:*

$$D_{\text{pcp}}^{D,s}(X,Y) \leq D_{\text{pmm}}^{D,s}(X,Y)$$

Proof. Analogous to the proof of Theorem 1. □

As the partial closest pair distance can be computed rather efficiently by scanning the matrix of distance values between each pair of vectors in X and Y for the closest pairs and organizing the s closest distances in a heap structure, it is a very beneficial filter for the partial minimal matching distance. The overall runtime complexity is $O(k^2 d)$ for the complete version and $O(k^2 d \log s)$ for the partial version of the clostest pair distance, when an L_p-distance is used between vectors. Although this is more complex than the closest pair approach on norm vectors (cf. Section 4.3.3), it is a more selective filter that saves more of the very expensive calculations of the exact partial minimal matching distance.

4.3.2 Centroid Approach

This filter step is based on the relation between a set of feature vectors and its *extended centroid* [KBK+03].

Definition 16 (extended centroid). Let $V \subset \mathbb{R}^d$ and $\vec{\omega} \in \mathbb{R}^d \setminus V$. Let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\} \in 2^V$ be a vector set where $|X| \leq k$. Then the *extended centroid* $C_{k,\vec{\omega}}(X)$ is defined as follows:

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

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

Theorem 3. *Let $V \subset \mathbb{R}^d$ and $\vec{\omega} \in \mathbb{R}^d \setminus V$. Let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}, Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets where $|X|, |Y| \leq k$ and let $C_{k,\vec{\omega}}(X)$, $C_{k,\vec{\omega}}(Y)$ be their extended centroids. Furthermore, let $W_{\vec{\omega}} : V \to \mathbb{R}$, $W_{\vec{\omega}}(\vec{v}) =$*

$\|\vec{v} - \vec{\omega}\|_p$, *be a weight function for unmatched elements. Then the following inequality holds:*

$$k \, \|C_{k,\vec{\omega}}(X) - C_{k,\vec{\omega}}(Y)\|_p \leq D_{\text{mm}}^{L_p, W_{\vec{\omega}}}(X, Y)$$

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. $|X| = m \geq n = |Y|$.

$$\begin{aligned}
& k \cdot \|C_{k,\vec{\omega}}(X) - C_{k,\vec{\omega}}(Y)\|_p \\
= \; & k \cdot \| \tfrac{\sum_{i=1}^{m} \vec{x}_{\pi(i)} + (k-m)\cdot\vec{\omega}}{k} - \tfrac{\sum_{i=1}^{n} \vec{y}_i + (k-n)\cdot\vec{\omega}}{k} \|_p \\
= \; & \| \textstyle\sum_{i=1}^{m} \vec{x}_{\pi(i)} - \sum_{i=1}^{n} \vec{y}_i - (m-n)\cdot\vec{\omega} \|_p \\
= \; & \| \textstyle\sum_{i=1}^{n} \vec{x}_{\pi(i)} - \sum_{i=1}^{n} \vec{y}_i + \sum_{i=n+1}^{m} \vec{x}_{\pi(i)} - \sum_{i=n+1}^{m} \vec{\omega} \|_p \\
\overset{\text{tri. ineq.}}{\leq} \; & \| \textstyle\sum_{i=1}^{n} (\vec{x}_{\pi(i)} - \vec{y}_i) \|_p + \| \sum_{i=n+1}^{m} (\vec{x}_{\pi(i)} - \vec{\omega}) \|_p \\
\overset{\text{tri. ineq.}}{\leq} \; & \textstyle\sum_{i=1}^{n} \|\vec{x}_{\pi(i)} - \vec{y}_i\|_p + \sum_{i=n+1}^{m} \|\vec{x}_{\pi(i)} - \vec{\omega}\|_p \\
= \; & \textstyle\sum_{i=1}^{n} \|\vec{x}_{\pi(i)} - \vec{y}_i\|_p + \sum_{i=n+1}^{m} w_{\vec{\omega}}(\vec{x}_{\pi(i)}) \\
= \; & D_{mm}^{L_2, W_{\vec{\omega}}}(X, Y)
\end{aligned}$$

□

We have shown that the L_p-distance between the extended centroids multiplied by k is a lower bound 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 $\frac{\varepsilon}{k}$. Often a good choice of $\vec{\omega}$ is $\vec{0}$, since $\vec{0} \notin V$ holds for a lot of applications. Thus, Conditions (1) and (2) for the metric character of the minimal matching distance $D_{\text{mm}}^{L_2, W_{\vec{0}}}$ are satisfied.

A 2-dimensional example for the extended centroid filter is depicted in Fig. 4.1(b), where $|X| = |Y| = 2$ and

$$2c_1 = 2 \, \|C_{k,\vec{0}}(X) - C_{k,\vec{0}}(Y)\|_2 \leq D_{\text{mm}}^{L_2, W_{\vec{0}}}(X, Y) = a_1 + b_1.$$

The centroid approach is not suitable as a filter for the partial minimal matching distance, as the centroid invariably aggregates information of all vectors contained in a vector set.

4.3.3 Norm Vector Approach

Another possible filter for vector set represented data is based on the L_p-norms of all vector elements of a vector set. The idea is as follows: For all vectors $\vec{x}$ in a vector set X, $|X| \leq k$, we compute the L_p-norms $\|\vec{x}\|_p$ and organize these norm values in descending order in a k-dimensional vector. We call this filter the *norm vector filter*.

Definition 17 (norm vector). Let $V \subset \mathbb{R}^d$. Let $X \in 2^V$ be a vector set where $|X| \leq k$. Let $(\|\vec{x}_1\|_p, \ldots \|\vec{x}_{|X|}\|_p)$ be the sequence of the L_p-norm values of the vectors in X in descending order, i.e. for all $i < j \in \{1, \ldots |X|\}$ holds $\|\vec{x}_i\|_p \geq \|\vec{x}_j\|_p$. Then the *norm vector* $V_k(X) = (v_1, \ldots v_k)^t \in \mathbb{R}^k$ is defined as follows:

$$v_i = \begin{cases} \|\vec{x}_i\|_p & \text{for } i = 1, \ldots |X| \\ 0 & \text{for } i = |X|+1, \ldots k \end{cases}$$

Note that if X has a cardinality smaller than k, dimensions $|X|+1$ to k of the norm vector will get filled with 0. We employ the Manhattan distance as a distance function between two norm vectors $V_k(X)$ and $V_k(Y)$. This distance measure fulfills the lower-bounding property with respect to the minimal matching distance, if the L_p-norm is used as the weight function W. Before we show this result in Theorem 4, we derive the following three lemmas.

Lemma 2. *Let $\vec{x}, \vec{y} \in \mathbb{R}^d$ be two d-dimensional feature vectors. Then the difference between the L_p-norms of $\vec{x}$ and $\vec{y}$ underestimates the L_p-distance between $\vec{x}$ and $\vec{y}$:*

$$\big|\|\vec{x}\|_p - \|\vec{y}\|_p\big| \leq \|\vec{x} - \vec{y}\|_p$$

Proof. $\|\vec{x}\|_p = \|\vec{x} - \vec{0}\|_p \overset{\text{tri. ineq.}}{\leq} \|\vec{x} - \vec{y}\|_p + \|\vec{y} - \vec{0}\|_p = \|\vec{x} - \vec{y}\|_p + \|\vec{y}\|_p$ follows $\|\vec{x}\|_p - \|\vec{y}\|_p \leq \|\vec{x} - \vec{y}\|_p$.

$\|\vec{y}\|_p = \|\vec{y} - \vec{0}\|_p \overset{\text{tri. ineq.}}{\leq} \|\vec{x} - \vec{y}\|_p + \|\vec{x} - \vec{0}\|_p = \|\vec{x} - \vec{y}\|_p + \|\vec{x}\|_p$ follows $\|\vec{y}\|_p - \|\vec{x}\|_p \leq \|\vec{x} - \vec{y}\|_p$.

Then $\big|\|\vec{x}\|_p - \|\vec{y}\|_p\big| = \max(\|\vec{x}\|_p - \|\vec{y}\|_p, \|\vec{y}\|_p - \|\vec{x}\|_p) \leq \|\vec{x} - \vec{y}\|_p$. □

Lemma 3. *Let $V \subset \mathbb{R}^d$. Let $X = \{\vec{x}_1, \dots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \dots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Then the following inequality holds:*

$$\sum_{i=1}^{|X|} \big|\|\vec{x}_i\|_p - \|\vec{y}_i\|_p\big| \leq \sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_i\|_p$$

Proof. The proposition holds if $\forall i \in \{1, \dots |X|\} : \big|\|\vec{x}_i\|_p - \|\vec{y}_i\|_p\big| \leq \|\vec{x}_i - \vec{y}_i\|_p$ and this follows directly from Lemma 2. □

Lemma 4. *Let $V \subset \mathbb{R}^d$ and let $X, Y \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Their norm vectors are denoted by $V_k(X)$ and $V_k(Y)$. Let the sequences of the L_p-norm values of the vectors in X and Y in descending order be denoted by $(\|\vec{x}_1\|_p, \dots \|\vec{x}_{|X|}\|_p)$ and $(\|\vec{y}_1\|_p, \dots \|\vec{y}_{|Y|}\|_p)$. Let $\pi \in \Pi(Y)$. Then the following inequality holds:*

$$\|V_k(X) - V_k(Y)\|_1 \leq \sum_{i=1}^{|X|} \big|\|\vec{x}_i\|_p - \|\vec{y}_{\pi(i)}\|_p\big| + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_p$$

Proof. (Sketch) Let $V_k(X) = (x_1, \dots x_k)^t$, $V_k(Y) = (y_1, \dots y_k)^t$.
We first show that the following holds:

$$\|V_k(X) - V_k(Y)\|_1 = \sum_{i=1}^{k} |x_i - y_i| \leq \sum_{i=1}^{k} |x_i - y_{\pi(i)}| \qquad (*)$$

Every given permutation π can be constructed from adjacent permutations $\pi_1, \dots \pi_n$, such that $\pi = \pi_1 \circ \dots \circ \pi_n$ and for each π_l there is some $q \in \{1, \dots |X|\}$, such that $\pi_l(q) = q + 1$, $\pi_l(q + 1) = q$ and $\forall q' \notin \{q, q + 1\} : \pi_l(q') = q'$. Given π_l, we show that $|x_q - y_{\pi_l(q)}| + |x_{q+1} - y_{\pi_l(q+1)}| \geq |x_{q+1} - y_{q+1}| + |x_q - y_q|$. There are in total six cases, because of the ordering within the norm vectors:

1. $x_q \leq x_{q+1} \leq y_{\pi_l(q+1)} \leq y_{\pi_l(q)}$
2. $x_q \leq y_{\pi_l(q+1)} \leq x_{q+1} \leq y_{\pi_l(q)}$
3. $x_q \leq y_{\pi_l(q+1)} \leq y_{\pi_l(q)} \leq x_{q+1}$
4. $y_{\pi_l(q+1)} \leq x_q \leq x_{q+1} \leq y_{\pi_l(q)}$
5. $y_{\pi_l(q+1)} \leq x_q \leq y_{\pi_l(q)} \leq x_{q+1}$
6. $y_{\pi_l(q+1)} \leq y_{\pi_l(q)} \leq x_q \leq x_{q+1}$

We exemplarily show the third case. The proofs of the other five cases are very similar.

$$|x_q - y_{\pi_l(q)}| + |x_{q+1} - y_{\pi_l(q+1)}| = x_{q+1} - y_q + y_{q+1} - x_q =$$
$$(x_{q+1} - y_{q+1}) + (y_{q+1} - y_q) + (y_q - x_q) + (y_{q+1} - y_q) =$$
$$|x_{q+1} - y_{q+1}| + |x_q - y_q| + 2|y_{q+1} - y_q| \geq |x_{q+1} - y_{q+1}| + |x_q - y_q|$$

As for each application of a π_l the sum on the right side of proposition (*) will grow or remain equal, the sum will grow or remain equal when applying π. Thus, proposition (*) holds. Then the following holds:

$$\|V_k(X) - V_k(Y)\|_1 \overset{(*)}{\leq} \sum_{i=1}^{k} |x_i - y_{\pi(i)}| = \sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_p - \|\vec{y}_{\pi(i)}\|_p \right| +$$
$$\sum_{i=|X|+1}^{|Y|} \left| \|0\|_p - \|\vec{y}_{\pi(i)}\|_p \right| + \sum_{i=|Y|+1}^{k} \left| \|0\|_p - \|0\|_p \right| =$$
$$\sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_p - \|\vec{y}_{\pi(i)}\|_p \right| + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_p$$

□

Theorem 4. *Let $V \subset \mathbb{R}^d$ and let $X, Y \in 2^V$ be two vector sets. Their norm vectors are denoted by $V_k(X)$ and $V_k(Y)$. Furthermore, let $W_{\vec{0}} : V \to \mathbb{R}$, $W_{\vec{0}}(\vec{v}) = \|\vec{v}\|_p$, be the L_p-norm used as a weight function for the minimal matching distance. Then the following inequality holds:*

$$\|V_k(X) - V_k(Y)\|_1 \leq D_{\text{mm}}^{L_p, W_{\vec{0}}}(X, Y)$$

Proof. Let the sequences of the L_p-norm values of the vectors in X and Y in descending order be denoted by $(\|\vec{x}_1\|_p, \ldots \|\vec{x}_{|X|}\|_p)$ and $(\|\vec{y}_1\|_p, \ldots \|\vec{y}_{|Y|}\|_p)$. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $\pi \in \Pi(Y)$ be the permutation of Y that results from the minimum weight perfect matching of X and Y. We

combine the results from Lemmas 3 and 4.

$$\|V_k(X) - V_k(Y)\|_1 \overset{\text{Lemma 4}}{\leq}$$

$$\sum_{i=1}^{|X|} \left| \|\vec{x}_i\|_p - \|\vec{y}_{\pi(i)}\|_p \right| + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_p \overset{\text{Lemma 3}}{\leq}$$

$$\sum_{i=1}^{|X|} \|\vec{x}_i - \vec{y}_{\pi(i)}\|_p + \sum_{i=|X|+1}^{|Y|} \|\vec{y}_{\pi(i)}\|_p = D_{\text{mm}}^{L_p, W_{\vec{0}}}(X, Y)$$

□

A 2-dimensional example for the norm vector filter is depicted in Fig. 4.1(c), where $|X| = |Y| = 2$ and

$$a'_2 + b'_2 = \|V_k(X) - V_k(Y)\|_1 \leq D_{\text{mm}}^{L_2, W_{\vec{0}}}(X, Y) = a_2 + b_2.$$

An approach for partial similarity search is to apply a parallel scan through the norm vectors $V_k(X)$ and $V_k(Y)$ and to build a heap structure containing the distances between the closest pairs of norm values found during the parallel scan. Finally, the sum of the top s elements of the heap is reported as the distance measure. This can be done very efficiently in $O(k \log s)$ time using the algorithm in Fig. 4.2. The algorithm corresponds to a closest pair approach on the norm values of the feature vectors, which lower bounds the partial minimal matching distance.

Theorem 5. *Let $V \subset \mathbb{R}^d$ and let $X = \{\vec{x}_1, \ldots \vec{x}_{|X|}\}$, $Y = \{\vec{y}_1, \ldots \vec{y}_{|Y|}\} \in 2^V$ be two vector sets. We assume w.l.o.g. $|X| \leq |Y| \leq k$. Let $s \leq |X|$. Let $\hat{X} = \{\|\vec{x}_1\|_p, \ldots \|\vec{x}_{|X|}\|_p\}$, $\hat{Y} = \{\|\vec{y}_1\|_p, \ldots \|\vec{y}_{|Y|}\|_p\}$ be multisets containing the L_p-norm values of the vectors in X and Y. Then the following inequality holds:*

$$D_{\text{pcp}}^{L_p, s}(\hat{X}, \hat{Y}) \leq D_{\text{pmm}}^{L_p, s}(X, Y)$$

Proof. According to Theorem 2, $D_{\text{pcp}}^{L_p, s}(\hat{X}, \hat{Y}) \leq D_{\text{pmm}}^{L_p, s}(\hat{X}, \hat{Y})$ holds.

```
algorithm PartialNormVectorFilter;
    input: VectorSet X, VectorSet Y, Integer k, Integer s;
    output: Real;
begin
    return max(Distance(X, Y, k, s), Distance(Y, X, k, s));
end;

algorithm Distance;
    input: VectorSet X, VectorSet Y, Integer k, Integer s;
    output: Real;
begin
// initialize
    (x_1, ... x_k) := V_k(X);
    (y_1, ... y_k) := V_k(Y);
    j := 1;
// parallel scan
    for i in 1..k do
        while j < k ∧ |x_i − y_j| ≥ |x_i − y_{j+1}| do
            j := j + 1;
        end while;
        heap.insert(|x_i − y_j|);
    end for;
// add up the distance
    dist := 0;
    for i in 1..s do
        dist := dist + heap.top();
    end for;
    return dist;
end;
```

Figure 4.2: The partial norm vector filter algorithm.

To obtain $D_{\text{pmm}}^{L_p,s}(\hat{X}, \hat{Y}) \leq D_{\text{pmm}}^{L_p,s}(X, Y)$ we have to show that

$$\min_{\pi_1 \in \Pi(X), \pi_2 \in \Pi(Y)} \left(\sum_{i=1}^{s} \left| \|\vec{x}_{\pi_1(i)}\|_p - \|\vec{y}_{\pi_2(i)}\|_p \right| \right) \leq \min_{\pi_1 \in \Pi(X), \pi_2 \in \Pi(Y)} \left(\sum_{i=1}^{s} \|\vec{x}_{\pi_1(i)} - \vec{y}_{\pi_2(i)}\|_p \right)$$

and this follows from Lemma 3. □

Table 4.1: Runtime complexity of the proposed filters.

distance function	complete similarity	partial similarity
exact distance	$O(k^3 + k^2 d)$	$O(\binom{k}{s} sk^2 + k^2 d)$
clostest pair	$O(k^2 d)$	$O(k^2 d \log s)$
centroid	$O(d)$	n/a
norm vector	$O(k)$	$O(k \log s)$

4.3.4 Discussion

As the computation of the minimal matching distance is rather time-consuming, we introduced three different filters. The centroid and the norm vector filtering techniques can be profitably combined. The exact distance computation is only performed if the results of both filter distance computations on the centroids and the norm vectors are small enough. This way, a good deal of the information in the vector sets is incorporated in the filter distance computation. Given d-dimensional data, the centroid filter maps each dimension to a single value, resulting in a d-dimensional vector. On the other hand, the norm vector filter maps each vector to a single value resulting in a k-dimensional vector. Thus, the combined filter contains aggregated information over both the dimensions and the vectors and is therefore suitable for a lot of different data distributions. The time complexity for a combined filter distance evaluation is $O(d + k)$. As the centroid approach is not applicable for partial similarity search, we cannot use the combined filter for this purpose.

In contrast to the other two approaches, which derive a single feature vector for approximating a vector set, the closest pair filter works directly on the vector sets. The resulting distance measure lower bounds the minimal matching distance and can be computed more efficiently. The runtime complexities for partial and complete similarity distance calculations based on the three different filters are summed up in Table 4.1, where we assume vector sets containing k d-dimensional vectors, a partial similarity parameter $s \in \{1, \ldots k\}$, and an L_p-distance between vectors.

4.4 Experimental Evaluation

In this section, we present our experimental results.

4.4.1 Settings

We generated and used two artificial datasets, each containing 100,000 random vector sets. The first dataset consists of vector sets containing 10 2-dimensional vectors each. The other dataset consists of vector sets containing 2 10-dimensional vectors each. The vectors are generated so that all of their components are uniformly distributed in the interval between 0 and 1. All distance measures between vector sets were implemented in Java 1.4 and the experiments were run on a workstation with a Xeon 2.4 GHz processor and 2 GB main memory under Linux.

Furthermore, we used the similarity model presented in [KBK+03], where CAD objects were represented by a vector set consisting of either 3, 5 or 7 vectors in 6D. All experiments were carried out on a dataset containing 5,000 CAD objects from an American aircraft producer. We conducted our experiments on top of the Oracle9i Server using PL/SQL for the computational main memory based programming. We compared our different filters for vector set represented data to a PL/SQL implementation of the M-tree [CPZ97]. For the M-tree based k-nearest neighbor queries the ranking algorithm of [HS95] was used. The experiments were performed on a Pentium III/700 machine with IDE hard drives. The database block cache was set to 500 disk blocks with a block size of 8 KB and was used exclusively by one active session.

The minimal matching distances between sets of feature vectors were computed using an implementation of the Kuhn-Munkres algorithm. Throughout our experiments we used the Euclidean distance as the distance measure between two single vectors. The range queries were based on a sequential scan. The k-nn queries with exact distance calculations were also based on a sequential scan. For the filtered k-nn queries the filter distances between the query object and all vector sets in the database were calculated and sorted in

ascending order. Then the optimal multi-step k-nn search algorithm [SK98] was used. In all tests, we processed 10 different similarity range queries as well as k-nn queries. The presented figures depict the average results from these tests.

4.4.2 Complete Similarity Search

Range queries. In a first experiment, we carried out range queries on the two artificial datasets. Figure 4.3 shows rather good results for the norm vector filter, while the centroid filter performs rather badly. The superiority of the norm vector filter is due to the fact that more information is preserved by approximating a vector set by a 10-dimensional vector in contrast to the 2-dimensional centroid computed by the centroid approach. As expected, the situation is reversed in Fig. 4.4 where each vector set contains 2 10-dimensional vectors. In both tests, the closest pair filter has good to optimal selectivity, but due to its computational complexity the overall runtime is rather high especially for high ε-values.

Using the CAD datasets, we carried out different range queries on a vector set consisting of 5 6-dimensional vectors. Figure 4.5 shows that the selectivity of the closest pair filter is almost optimal, i.e. few unnecessary candidates are produced. Nevertheless, the overall runtime of this filter-step is very high as the runtime complexity of the filter-step is almost as high as the computation of the minimal matching distance itself (cf. Fig. 4.5). Good results were obtained by using the centroid approach. The good performance of the centroid approach can slightly be increased by using the combined filter, i.e. the combination of the norm vector filter and the centroid filter, which can also be efficiently computed and has a slightly higher selectivity. Note that both the selectivity as well as the runtime behavior of the M-tree are outperformed by this combined filter for all ε-values.

k-nn queries. Figure 4.6 shows the average results we obtained for carrying out different k-nn queries on CAD objects represented by vector sets containing 7 vectors. Basically, we made the same observations as for range

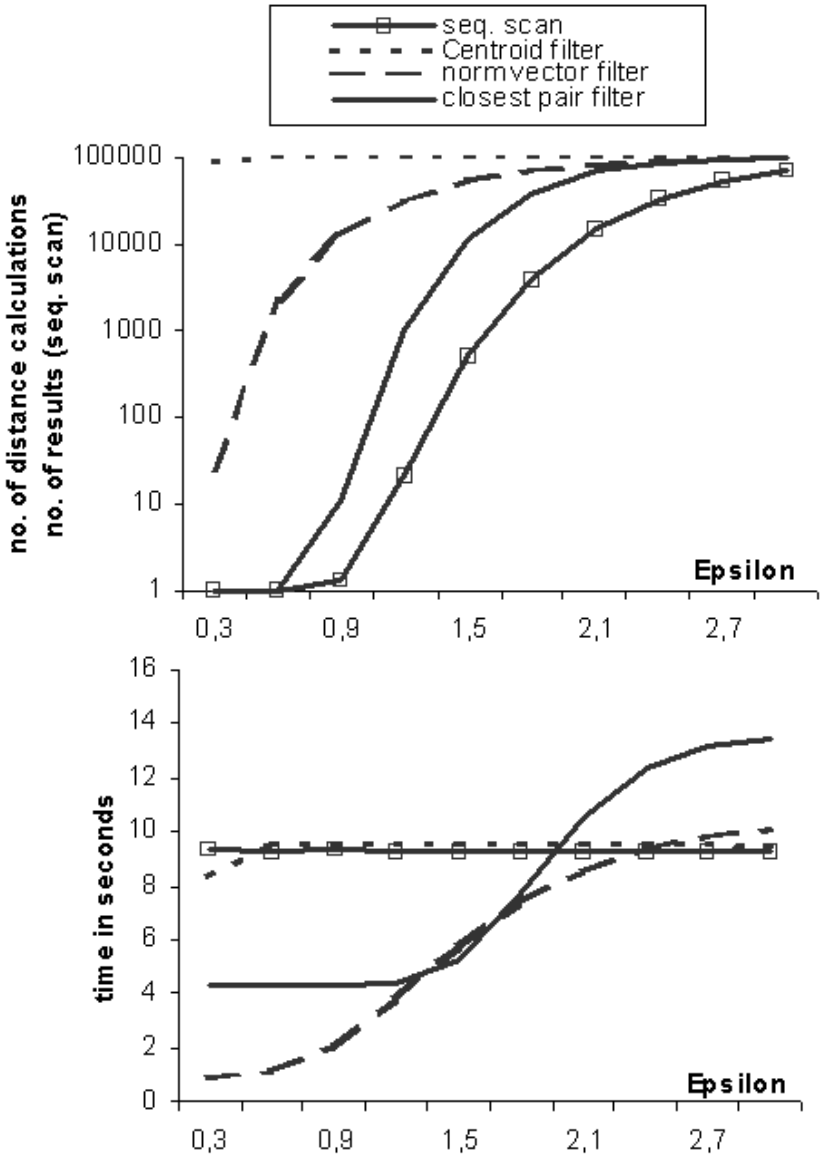


Figure 4.3: Complete range queries, artificial dataset, cardinality 10, dimensionality 2.

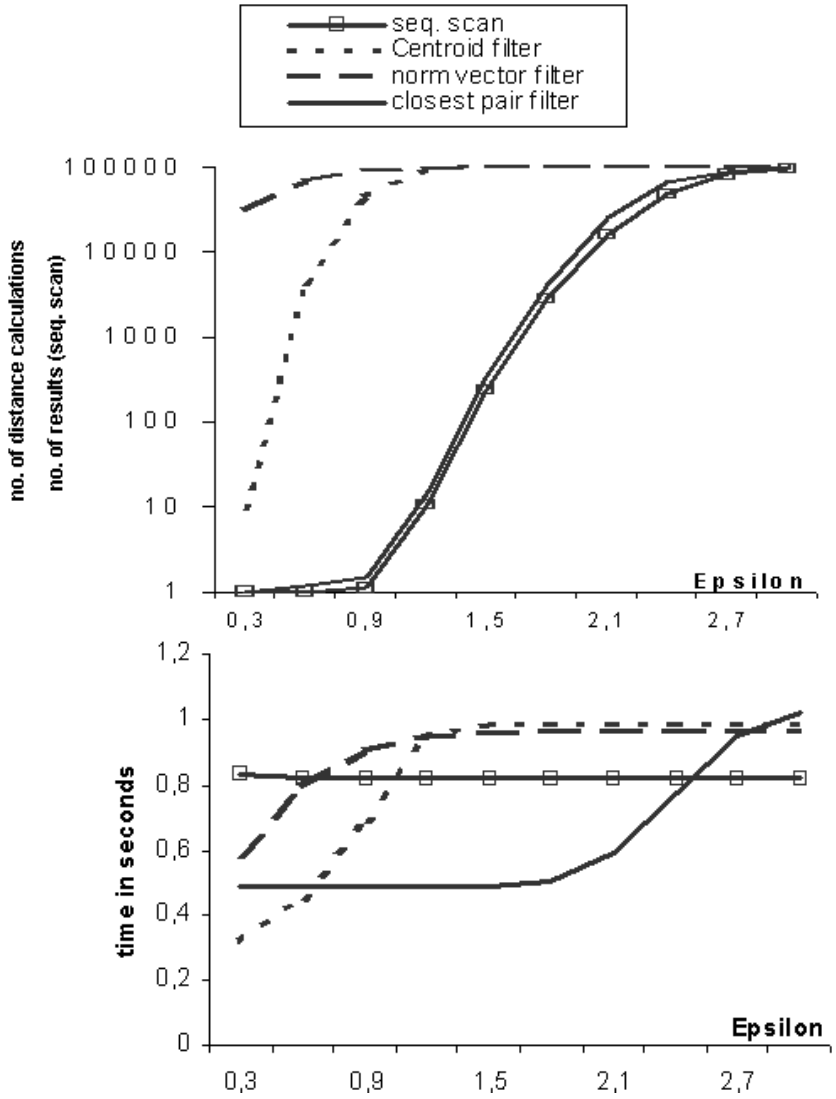


Figure 4.4: Complete range queries, artificial dataset, cardinality 2, dimensionality 10.

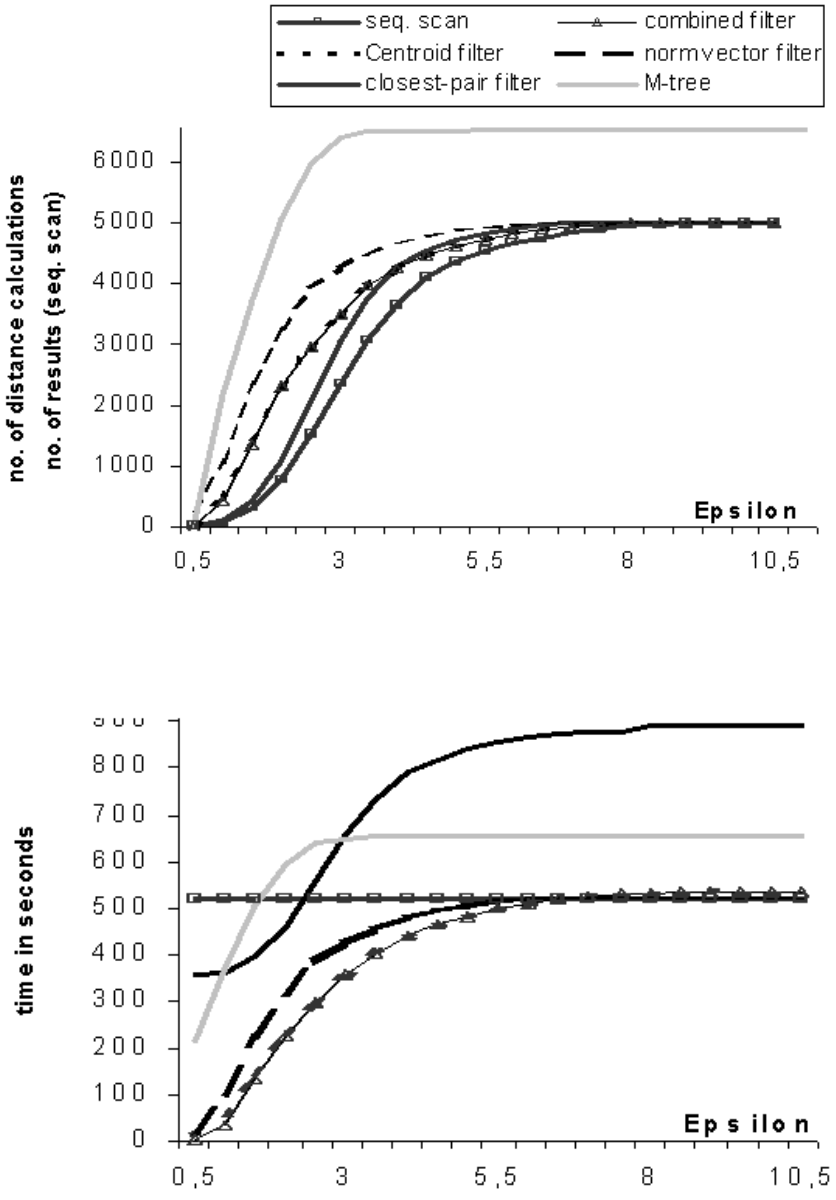


Figure 4.5: Complete range queries, CAD dataset, cardinality 5, dimensionality 6.

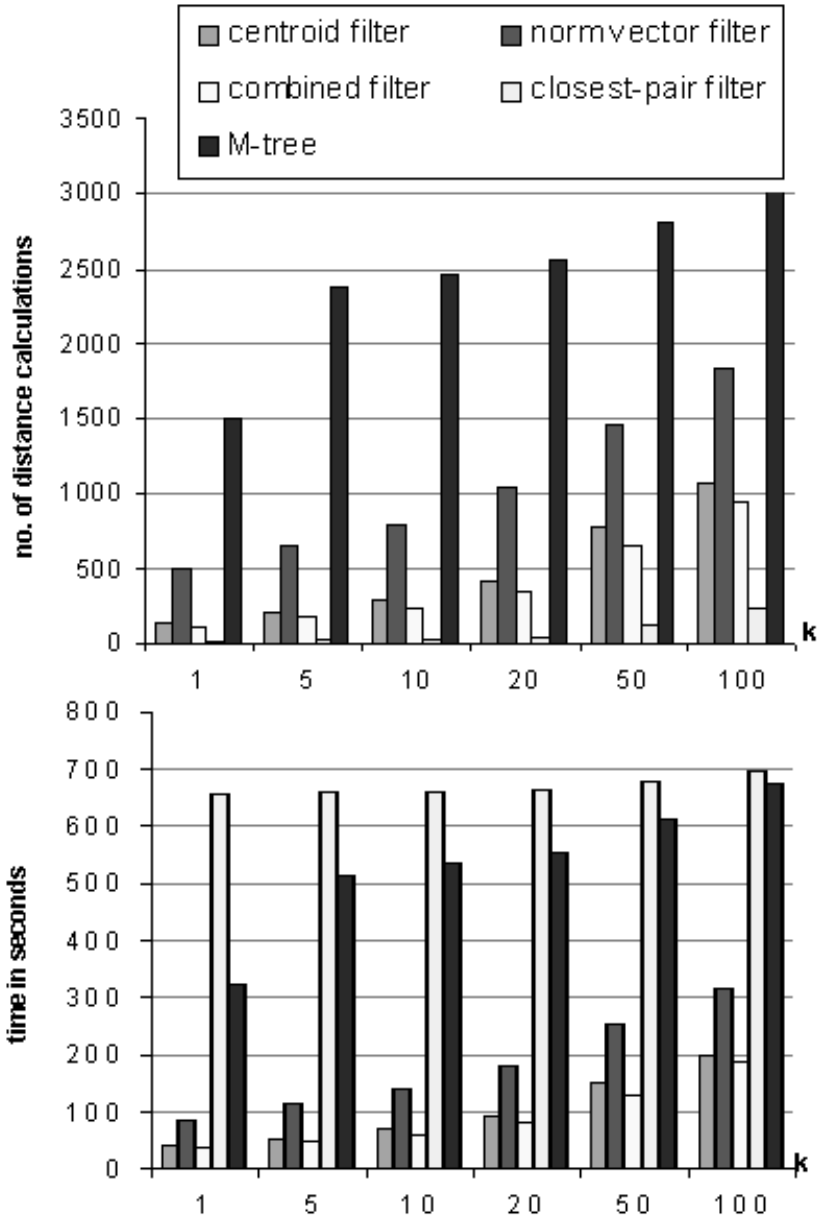


Figure 4.6: Complete k-nn queries, CAD dataset, cardinality 7, dimensionality 6 (the sequential scan took about 1014 sec. for each k).

queries. Although the closest pair filter has a rather good selectivity, it is rather expensive. The best trade off is achieved by using the combination of the norm vector filter and the centroid filter. All filters have a rather good selectivity and accelerate the query process enormously. For instance, for k-nn queries where k is smaller than 20, the combined filter accelerates the query process on the 6-dimensional vector sets by more than one order of magnitude compared to the sequential scan. Again, the selectivity as well as the runtime behavior of the M-tree is clearly outperformed by this combined filter for all values of k, e.g. for k=5 the combined filter outperforms the M-tree by an order of magnitude. We made the same observations for the CAD datasets with 3 and 5 vectors per vector set, except that the absolute runtime is higher for the larger vector sets. The average runtime for 7 vectors is about four times the average runtime for 3 vectors.

4.4.3 Partial Similarity Search

In this section, we tested the norm vector filter and the closest pair filter. Let us note that detecting partial similarity is a very expensive operation. Furthermore, we cannot apply the M-tree as the distance function is not a metric (cf. Definition 13).

Range queries. Figure 4.7 shows the average of 10 range queries for varying ε-values on a vector set of 7 vectors. The partial similarity parameter s was set to 2. Again, the closest pair filter is very selective. As the exact distance function is very expensive, the closest pair filter can be beneficially used for small ε-values. For higher ε-values, the rather high evaluation cost of the closest pair filter carry into weight. On the other hand, the norm vector can safely be used for all values of ε, as there is no noteworthy overhead. For rather small ε-values, it even outperforms the closest pair filter, although the norm vector has a lower selectivity than the closest pair filter. This is because the lower computational cost of the norm vector filter still pays off, compared to the slightly more exact distance computations which have to be carried out.

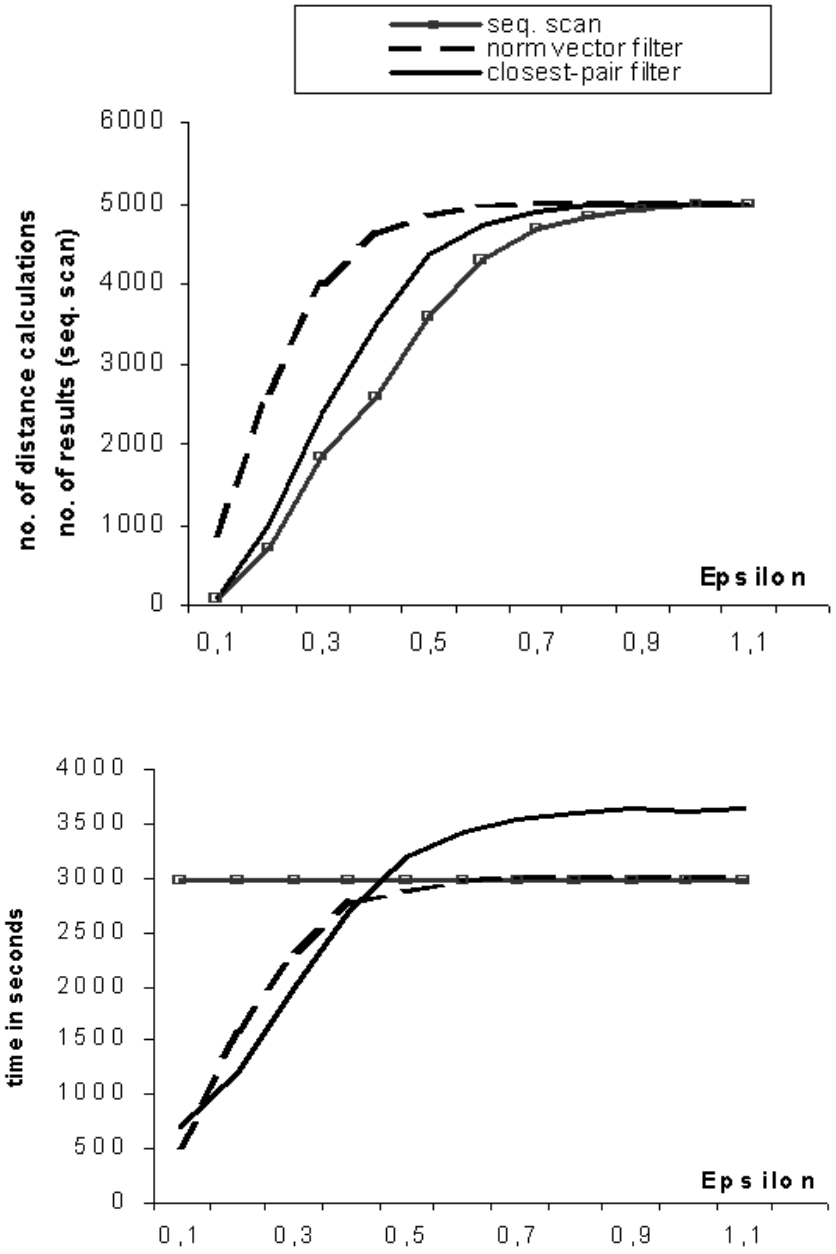


Figure 4.7: Partial range queries for $s = 2$, CAD dataset, cardinality 7, dimensionality 6.

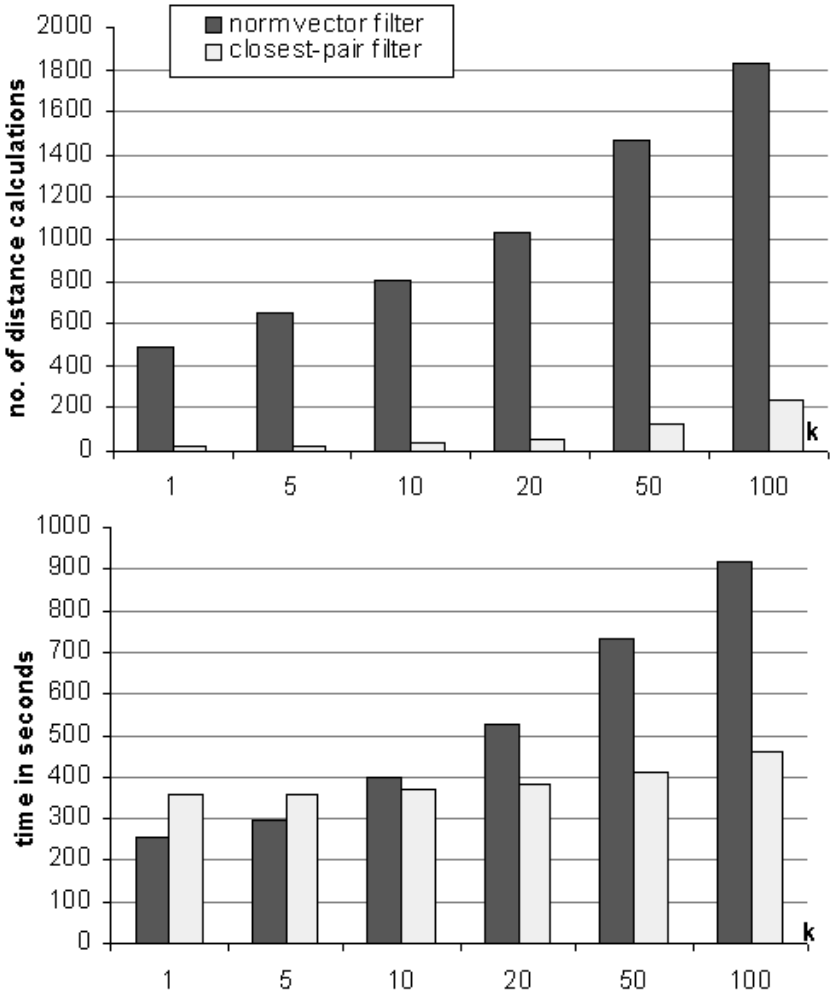


Figure 4.8: Partial k-nn queries for $s = 3$, CAD dataset, cardinality 5, dimensionality 6 (the sequential scan took about 2123 sec. for each k).

k-nn queries. Figure 4.8 shows the average of 10 k-nn queries for vector sets of 5 vectors each having a dimensionality of 6 and a partial similarity parameter $s = 3$. For small values of k, the norm vector filter outperforms the exact distance computation by almost one order of magnitude. For higher values of k, the selectivity of the norm vector filter decreases and thus the overall response time increases. For values of k equal to 100, the norm vector filter still accelerates the query process by 100%. As already mentioned, the closest pair filter is rather expensive. Although it has an excellent selectivity, the norm vector filter is better for rather small values of k. For increasing values of k, the closest pair filter outperforms the norm vector filter because of the much better selectivity and the very expensive exact distance calculations.

4.5 Summary

In this chapter, we motivated the use of vector set data by pointing out the different application areas of this promising representation technique. We introduced a suitable distance function on vector sets, which reflects the intuitive notion of similarity for the presented application ranges. Furthermore, we presented different filtering techniques with different runtime complexities. Our experimental evaluation and our analytical reasoning showed that the closest pair filter is the most selective filter. As this filter is rather expensive, it only pays off for partial similarity queries which are extremely expensive themselves. For complete similarity queries, the combination of the norm vector filter and the centroid filter is the method of choice for a lot of different data distributions, as it can be computed efficiently and the information of each vector and each dimension is taken into consideration. The experimental evaluation on real world datasets demonstrates that the presented filtering techniques accelerate similarity range queries and k-nn queries by up to one order of magnitude compared to metric index structures and the sequential scan.

Chapter 5

Multi-Step Density-Based Clustering

In recent years, the research community spent a lot of attention to the clustering problem resulting in a large variety of different clustering algorithms [JMF99]. One important class of clustering algorithms is density-based clustering which can be used for clustering all kinds of metric data and is not confined to vector spaces. Density-based clustering is rather robust concerning outliers [EKSX96] and is very effective in clustering all sorts of data, e.g. multi-represented objects [KKPS04a]. Furthermore, the reachability plot created by the density-based hierarchical clustering algorithm OPTICS serves as a starting point for an effective data mining tool described in Chapter 7, which helps to visually analyze cluster hierarchies.

Density-based clustering algorithms like DBSCAN and OPTICS, which were introduced in Chapter 3, are based on ε-range queries for each database object. Each range query requires a lot of distance calculations, especially when high ε-values are used. Therefore, these algorithms are only applicable to large collections of complex objects, e.g. trees, point sets, and graphs (cf. Figure 1), if those range queries are supported efficiently. When working with complex objects, the necessary distance calculations are the time-limiting factor. Thus, the ultimate goal is to save as many of these complex distance calculations as possible.

In this chapter, we present an approach which helps to compute density-based clusterings efficiently. The core idea of our approach is to integrate the multi-step query processing paradigm directly into the clustering algorithm rather than using it "only" for accelerating range queries. Our clustering approach itself exploits the information provided by simple distance measures lower-bounding complex and expensive exact distance functions. Expensive exact distance computations are only performed when the information provided by simple distance computations, which are often based on simple object representations, is not enough to compute the exact clustering. Furthermore, we show how our approach can be used for approximated clustering where the result might be slightly different from the one we compute based on the exact information. In order to measure the dissimilarity between the resulting clusterings, we introduce suitable quality measures.

The remainder of this chapter is organized as follows. In Section 5.1, we look at different approaches presented in the literature for efficiently computing these algorithms. We will explain why the presented algorithms are not suitable for expensive distance computations if we are interested in the exact clustering structure. In Section 5.2, we will present our new approach which tries to use lower-bounding distance functions before computing the expensive exact distances. The new approach integrates the multi-step query processing paradigm directly into the clustering algorithms rather than using it independently. As our approach can also be used for generating approximated clusterings, we introduce objective quality measures in Section 5.3 which allow us to assess the quality of approximated clusterings. In Section 5.4, we present a detailed experimental evaluation showing that the presented approach can accelerate the generation of density-based clusterings on complex objects by more than one order of magnitude. We show that for approximated clustering the achieved quality is scalable w.r.t. the overall runtime. Section 5.5 summarizes the chapter.

5.1 Related Work

DBSCAN and OPTICS determine the local densities by performing repeated range queries. In this section, we will sketch different approaches from the literature to accelerate these density-based clustering algorithms and discuss their unsuitability for complex object representations.

5.1.1 Exact Clustering

In the following we will present some approaches leading to exact density-based clusterings.

Multi-Dimensional Index Structures. The most common approach to accelerate each of the required single range queries is to use multi-dimensional index structures. For objects modelled by low-, medium-, or high-dimensional feature vectors there exist several specific R-tree [Gut84] variants. For more detail we refer the interested reader to [GG98].

Metric Index Structures. In contrast to Figure 1.3(a) where the objects are modelled by a high-dimensional feature vector, the objects presented in the example of Figure 1.3(b)–(d) are not modelled by feature vectors. Therefore, we cannot apply the index structures mentioned in the last paragraph. Nevertheless, we can use index structures, such as the M-tree [CPZ97] for efficiently carrying out range queries as long as we have a metric distance function for measuring the similarity between two complex objects. For a detailed survey on metric access methods we refer the reader to [CNBYM01].

Multi-Step Query Processing. The main goal of multi-step query processing is to reduce the number of complex and, therefore, time consuming distance calculations in the query process. In order to guarantee that there occur no false drops, the used filter distances have to fulfill a lower-bounding distance criterion. For any two objects p and q, a lower-bounding distance function d_f in the filter step has to return a value that is not greater than

the exact object distance d_o of p and q, i.e. $d_f(p,q) \leq d_o(p,q)$. With a lower-bounding distance function it is possible to safely filter out all database objects which have a filter distance greater than the current query range because the exact object distance of those objects cannot be less than the query range. Using a multi-step query architecture requires efficient algorithms which actually make use of the filter step. Agrawal, Faloutsos and Swami proposed such an algorithm for range queries [AFS93] which form the foundation of density-based clustering. For efficiency reasons, it is crucial that $d_f(p,q)$ is considerably faster to evaluate than $d_o(p,q)$ and, furthermore, in order to achieve a high selectivity $d_f(p,q)$ should be only marginally smaller than $d_o(p,q)$.

Using Multiple Similarity Queries. In [BBBK00] a schema was presented which transforms query intensive KDD algorithms into a representation using the similarity join as a basic operation without affecting the correctness of the result of the considered algorithm. The approach was applied to accelerate the clustering algorithms DBSCAN and OPTICS by using an R-tree like index structure. In [BEKS00] an approach was introduced for efficiently supporting multiple similarity queries for mining in metric databases. It was shown that many different data mining algorithms can be accelerated by multiplexing different similarity queries.

Summary. Multi-dimensional index structures based on R-tree variants and clustering based on the similarity join are restricted to vector set data. Furthermore, the main problem of all approaches mentioned above is that distance computations can only be avoided for objects located outside the ε-range of the actual query object. In order to create, for instance, a reachability plot without loss of information, the authors in [ABKS99] propose to use a very high ε-value. Therefore, all of the above mentioned approaches lead to $O(|DB|^2)$ exact distance computations for OPTICS.

5.1.2 Approximated Clustering

Other approaches do not aim at producing the exact hierarchical clustering structure, but an approximated one.

Sampling. The simplest approach is to use sampling and apply the expensive data mining algorithms to a subset of the dataspace. Typically, if the sample size is large enough, the result of the data mining method on the sample reflects the exact result well.

Grid-Based Clustering. Another approach is based on grid cells [JMF99] to accelerate query processing. In this case, the data space is partitioned into a number of non-overlapping regions or cells which can be used as a filter step for the range queries. All points in the result set are contained in the cells intersecting the query range. To further improve the performance of the range queries to a constant time complexity, query processing is limited to a constant number of these cells (e.g. the cell covering the query point and the direct neighbor cells) and the refinement step is dropped, thereby trading accuracy for performance.

Distance Mapping. In [WWL+99], 5 different distance mapping algorithms were introduced to map general metric objects to Euclidean or pseudo-Euclidean spaces in such a way that the distances among the objects are approximately preserved. The approximated data mining algorithm is then performed within the Euclidean space based on rather cheap distance functions. If there already exist selective filters which can efficiently be computed, an additional mapping into a feature space is superfluous, i.e. we can carry out the approximated data mining algorithm directly on the filter information.

Data Bubbles. Finally, there exist efficient approximated versions of hierarchical clustering approaches for non-vector data which are based on Data Bubbles [ZS03]. These approaches augment suitable representatives with

additional aggregated information describing the area around the representatives.

Summary. All indicated approximated clustering approaches are able to generate efficiently the corresponding clustering structure. The question at issue is: How much quality do they have to pay for their efficiency gain?

In the following, we will propose an approach which computes exact density-based clusterings trying to confine itself to simple distance computations lower-bounding the exact distances. Further expensive exact distance computations are postponed as long as possible, and are only carried out at that stage of the algorithm where they are compulsory to compute the correct clustering. Furthermore, we will also indicate how to use our algorithm for approximated clustering.

5.2 Accelerated Density-Based Clustering

In this section, we will discuss in detail how we can efficiently compute a flat and a hierarchical density-based clustering. We demonstrate how to integrate the multi-step query processing paradigm into the two density-based clustering algorithms DBSCAN and OPTICS. We present our approach for OPTICS in detail and sketch how a simplified version of this extended OPTICS approach can be used for DBSCAN.

5.2.1 Basic Idea

DBSCAN and OPTICS are both based on performing numerous ε-range queries. None of the approaches discussed in the literature can avoid that we have to compute the exact distance to a given query object q for all objects contained in $N_\varepsilon(q)$. Especially for OPTICS, where ε has to be chosen very high in order to create reachability plots without loss of information, we have to compute $|DB|$ many exact distance computations for each single range query, even when one of the methods discussed in Section 5.1.1 is used. In

the case of DBSCAN, typically, the ε-values are much smaller. Nevertheless, if we apply the traditional multi-step query processing paradigm with non-selective filters, we also have to compute up to $|DB|$ many exact distance computations.

In our approach, the number of exact distance computations does not primarily depend on the size of the database and the chosen ε-value but rather on the value of *MinPts*, which is typically only a small fraction of $|DB|$, e.g. $MinPts = 5$ is a suitable value even for large databases [ABKS99, EKSX96]. Basically, we use *MinPts*-nearest neighbor queries instead of ε-range queries on the exact object representations in order to determine the core-properties of the objects. Further exact complex distance computations are only carried out at that stage of the algorithms where they are compulsory to compute the correct clustering result.

5.2.2 Extended OPTICS

The main idea of our approach is to carry out the range queries based on the lower-bounding filter distances instead of using the expensive exact distances. In order to put our approach into practice, we have to slightly extend the data structure underlying the OPTICS algorithm, i.e. we have to add additional information to the elements stored in the seedlist.

The Extended Seedlist. We do not any longer use a single seedlist as in the original OPTICS algorithm where each list entry consists of a pair (*ObjectId*, *ReachabilityValue*). Instead, we use a list of lists, called *Xseedlist*, as shown in Figure 5.1. The *Xseedlist* consists of an ordered object list OL, quite similar to the original seedlist but without any reachability information. The order of the objects o_i in OL, cf. the horizontal arrow in Figure 5.1, is determined by the first element of each predecessor list $PL(o_i)$ anchored at o_i, cf. the vertical arrows in Figure 5.1.

An entry located at position l of the predecessor list $PL(o_i)$ belonging to object o_i consists of the following information:

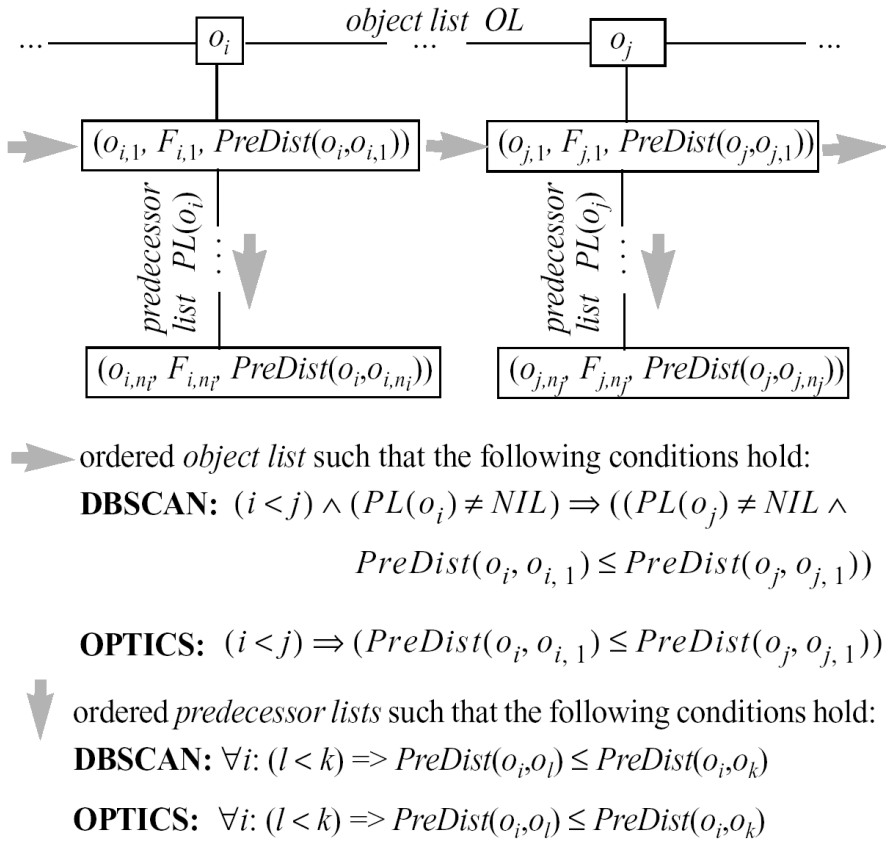


Figure 5.1: The *Xseedlist* data structure.

- **Predecessor ID.** A processed object $o_{i,l}$ which was already added to the reachability plot which is computed from left to right.
- **Predecessor Flag.** A flag $F_{i,l}$ indicating whether we already computed the exact object distance $d_o(o_i, o_{i,l})$ between o_i and $o_{i,l}$, or whether we only computed the distance $d_f(o_i, o_{i,l})$ of these two objects based on the lower-bounding filter information.
- **Predecessor Distance.** $PreDist(o_i, o_{i,l})$ is equal to

  $$\max(CoreDist(o_{i,l}), d_o(o_i, o_{i,l})),$$

  if we already computed the exact object distance $d_o(o_i, o_{i,l})$, else it is equal to

  $$\max(CoreDist(o_{i,l}), d_f(o_i, o_{i,l})).$$

Throughout our new algorithm, the conditions depicted in Figure 5.1 belonging to this extended OPTICS algorithm are maintained. In the following, we will describe the extended OPTICS algorithm trying to minimize the number of exact distance computations.

Algorithm. The extended OPTICS algorithm exploiting the filter information is depicted in Figure 5.2. The algorithm always takes the first element o_1 from OL. If it is at the first position due to a filter computation, we compute the exact distance $d_o(o_1, o_{1,1})$ and reorganize the *Xseedlist*. The reorganization might displace $o_{1,1}$ from the first position of $PL(o_1)$. Furthermore, object o_1 might be removed from the first position of OL. On the other hand, if the filter flag $F_{1,1}$ indicates that an exact distance computation was already carried out, we add object o_1 to the reachability plot with a reachability-distance equal to $PreDist(o_1, o_{1,1})$. Furthermore, we carry out the procedure update-Xseedlist(o_1).

Update-Xseedlist. This is the core function of our extended OPTICS algorithm. First, we carry out a range query around the query object $q := o_1$ based on the filter information, yielding the result set $N_\varepsilon^{\text{filter}}(q)$. Then we compute the core-distance of q by computing the *MinPts*-nearest neighbors of q as follows:

- If $|N_\varepsilon^{\text{filter}}(q)| < MinPts$, we set the core-distance of q to INFINITY and we are finished. Otherwise, we initialize a list $SortList_\varepsilon(q)$ containing tuples $(obj, flag, dist)$ which are organized in ascending order according to *dist*. For all objects $o \in N_\varepsilon^{\text{filter}}(q)$, we insert an entry $(o, \text{Filter}, d_f(o, q))$ into $SortList_\varepsilon(q)$.
- We walk through $SortList_\varepsilon(q)$ starting at the first element. We set

$$\begin{aligned} SortList_\varepsilon(q)[1].dist &= d_o(SortList_\varepsilon(q)[1].obj, q), \\ SortList_\varepsilon(q)[1].flag &= \text{Exact}, \end{aligned}$$

  and reorder $SortList_\varepsilon(q)$. This step is repeated until the first *MinPts* elements of $SortList_\varepsilon(q)$ are at their final position due to an exact

distance computation. The core-distance of q is equal to the distance

$$dist_{MinPts} = SortList_{\varepsilon}(q)[MinPts].dist,$$

if $dist_{MinPts} \leq \varepsilon$ holds, else it is set to INFINITY.

A tuple $(obj_j, flag_j, dist_j) \in SortList_{\varepsilon}(q)$ is transferred into an *Xseedlist* entry, if q is a core object and $dist_j \leq \varepsilon$ holds. If there exists no entry for obj_j in OL, $(obj_j, \langle(q, flag_j, \max(dist_j, CoreDist(q)))\rangle)$ is inserted into OL, else $(q, flag_j, \max(dist_j, CoreDist(q)))$ is inserted into $PL(obj_j)$. Note that in both cases the ordering of Figure 5.1 has to be maintained.

Lemma 5. *The result of the extended OPTICS algorithm is equivalent to the result of the original one.*

Proof. First, the extended OPTICS algorithm computes the correct core-distances by applying a *MinPts*-nearest neighbor search algorithm. Second, in each cycle the extended and the original OPTICS algorithm add the object o_1 having the minimum reachability-distance, w.r.t. all objects reported in the foregoing steps, to the cluster ordering. For the extended OPTICS algorithm this is true, as we have computed $d_o(o_1, o_{1,1})$ before adding it to the cluster ordering, due to the ordering conditions of Figure 5.1, and due to the lower-bounding filter property. □

Note that this approach carries out exact distance computations only for those objects which are very close to the current query object q according to the filter information, whereas the traditional multi-step query approach would compute exact distance computations for all objects $o \in N_{\varepsilon}^{\text{filter}}(q)$. As ε has to be chosen very high in order to create reachability plots without loss of information [ABKS99], the traditional approach has to compute $|DB|$ many exact distance computations, even when one of the approaches discussed in Section 5.1.1 is used. On the other hand, the number of exact distance computations in our approach does not depend on the size of the database but rather on the value of *MinPts*, which is only a small fraction of the cardinality of the database. Note that our approach only has to compute $|DB| \cdot MinPts$, i.e. $O(|DB|)$, exact distance computations if we assume an

```
algorithm ExtendedOPTICS
begin
   repeat
      if the Xseedlist is empty
         if all points are marked "done", terminate;
         choose "not-done" point q;
         add (q, empty_list) to the Xseedlist;
      end if;
      (o1, list) = first entry in the Xseedlist;
      if list[1].PredecessorFlag == Filter
         compute do(o1, list[1].PredecessorID);                    (*)
         update list[1].PredecessorDistance;
         list[1].PredecessorFlag = Exact;
         reorganize Xseedlist according to the conditions of Fig. 5.1;
      else
         remove (o1, list) from Xseedlist;
         mark o1 as "done";
         output (o1, list[1].PredecessorDistance);
         update-Xseedlist(o1);
      end if;
   end repeat;
end;
```

Figure 5.2: The extended OPTICS algorithm.

optimal filter, in contrast to the $O(|DB|^2)$ distance computations carried out by the original OPTICS run. Only when necessary, we carry out further exact distance computations (cf. line (*) in Figure 5.2).

5.2.3 Extended DBSCAN

Our extended DBSCAN algorithm is a simplified version of the extended OPTICS algorithm also using the *Xseedlist* as its main data structure. We carry out an ε-range query on the lower-bounding filter distances for an arbitrary database object q which has not yet been processed. Due to the lower-bounding properties of the filters, $N_\varepsilon(q) \subseteq N_\varepsilon^{\text{filter}}(q)$ holds. Therefore, if $|N_\varepsilon^{\text{filter}}(q)| < MinPts$, q is certainly no core point. Otherwise, we test whether q is a core point as follows.

We organize all elements $o \in N_\varepsilon^{\text{filter}}(q)$ in ascending order according to

their filter distance $d_f(o,q)$ yielding a sorted list. We walk through this sorted list, and compute for each visited object o the exact distance $d_o(o,q)$ until for *MinPts* elements $d_o(o,q) \leq \varepsilon$ holds or until we reach the end. If we reached the end, we certainly know that q is no core point. Otherwise q is a core object initiating a new cluster C.

If our current object q is a core object, some of the objects $o \in N_\varepsilon^{\text{filter}}(q)$ are inserted into the *Xseedlist* (cf. Figure 5.1). All objects for which we have already computed $d_o(o,q)$, and for which $d_o(o,q) \leq \varepsilon$ holds, certainly belong to the same cluster as the core-object q. At the beginning of OL, we add the entry (o, NIL), where $PL(o) = \text{NIL}$ indicates that o certainly belongs to the same cluster as q. Objects o for which $d_o(o,q) > e$ holds are discarded. All objects $o \in N_\varepsilon^{\text{filter}}(q)$ for which we did not yet compute $d_o(o,q)$ are handled as follows:

- If there exists no entry belonging to o in OL, $(o, \langle(q, \text{Filter}, d_f(o,q))\rangle)$ is inserted into OL and the ordering conditions of Figure 5.1 are reestablished.
- If there already exists an entry for o in OL and, furthermore, $PL(o) = \text{NIL}$ holds, nothing is done.
- If there already exists an entry for o in OL and, furthermore, $PL(o) \neq \text{NIL}$ holds, $(q, \text{Filter}, d_f(o,q))$ is inserted into $PL(o)$ and the ordering conditions of Figure 5.1 are reestablished.

DBSCAN expands a cluster C as follows. We take the first element o_1 from OL and, if $PL(o_1) = \text{NIL}$ holds, we add o_1 to C, delete o_1 from OL, carry out a range query around o_1, and try to expand the cluster C. If $PL(o_1) \neq \text{NIL}$ holds, we compute $d_o(o_1, o_{1,1})$. If $d_o(o_1, o_{1,1}) \leq \varepsilon$, we proceed as in the case where $PL(o_1) = \text{NIL}$ holds. If $d_o(o_1, o_{1,1}) > \varepsilon$ holds and length of $PL(o_1) > 1$, we delete $(o_{1,1}, F_{1,1}, \mathit{PreDist}(o_1, o_{1,1}))$ from $PL(o_1)$. If $d_o(o_1, o_{1,1}) > \varepsilon$ holds and length of $PL(o_1) = 1$, we delete o_1 from OL. Iteratively, we try to expand the current cluster by examining the first entry of $PL(o_1)$ until OL is empty.

Lemma 6. *The result of the extended DBSCAN algorithm is equivalent to the result of the original one.*

Proof. First, the determination whether an object o is a core object is correct as $o' \in N_\varepsilon(o) \Rightarrow o' \in N_\varepsilon^{\text{filter}}(o)$ holds due to the lower-bounding filter property. We test as many elements $o' \in N_\varepsilon^{\text{filter}}(o)$ as necessary to decide whether $|N_\varepsilon(o)| \geq \mathit{MinPts}$ holds. Second, similar to the proof of Lemma 5, we can guarantee that an object o is only added to the current cluster if $d_o(o,p) \leq \varepsilon$ holds for an object p which has already been singled out as a core object of the current cluster. □

5.2.4 Length-Limitation of the Predecessor Lists

In this section, we introduce two approaches for limiting the size of the predecessor lists to a constant l_{max} trying to keep the main memory footprint as small as possible. The first approach computes additional exact distances to reduce the length of the object reachability lists, while still computing the exact clustering. On the other hand, the second approach dispenses with additional exact distance computations leading to an approximated clustering.

Exact Clustering. In the case of OPTICS, for each object o_i in OL, we store all potential predecessor objects $o_{i,p}$ along with $\mathit{PreDist}(o_i, o_{i,p})$ in $PL(o_i)$. Due to the lower-bounding property of d_f, we can delete all entries in $PL(o_i)$ which are located at positions $l' > l$, if we have already computed the exact distance between o_i and the predecessor object $o_{i,l}$ located at position l. So each exact distance computation might possibly lead to several delete operations in the corresponding predecessor list. In order to limit the main memory footprint, we introduce a parameter l_{max} which restricts the allowed number of elements stored in a predecessor list. If more than l_{max} elements are contained in the list, we compute the exact distance for the predecessor $o_{i,1}$ located at the first position. Such an exact distance computation between o_i and $o_{i,1}$ usually causes $o_{i,1}$ to be moved upward in the list. All elements located behind its new position l are deleted. So if $l \leq l_{max}$

holds, the predecessor list is limited to at most l_{max} entries. Otherwise, we repeat the above procedure.

For DBSCAN, if the predecessor list of o_i is not NIL, we can limit its length by starting to compute $d_o(o_i, o_{i,1})$, i.e. the exact distance between o_i and the first element of $PL(o_i)$. If $d_o(o_i, o_{i,1}) \leq \varepsilon$ holds, we set $PL(o_i) = \text{NIL}$ indicating that o_i certainly belongs to the current cluster. Otherwise, we delete $(o_{i,1}, F_{i,1}, \mathit{PreDist}(o_i, o_{i,1}))$ and if the length of $PL(o_i)$ is still larger than l_{max}, we iteratively repeat this limitation procedure.

Lemma 7. *The above length limitation approach does not change the result of the extended DBSCAN and OPTICS algorithms.*

Proof. The presented DBSCAN algorithm guarantees that no entries $(o_{i,l}, F_{i,l}, \mathit{PreDist}(o_i, o_{i,l}))$ are deleted which are necessary for determining whether an object is directly density-reachable (cf. Definition 6) from a core object of the current cluster. For OPTICS we do not delete any entries which are necessary for computing the minimum reachability-distance w.r.t. all already processed objects. □

Approximated Clustering. In our approximated approach, we artificially limit the length of the predecessor lists by discarding all elements which are located at a position higher than l_{max} without computing any additional exact distances. This approach might not produce the same result as the original OPTICS and DBSCAN algorithms as the filter distances do not necessarily have to coincide with the exact distances. Note that if we have a very exact filter, cutting off the predecessor lists will not worsen the quality heavily (cf. Section 5.4.3). Nevertheless, we need to know how much quality we have to pay for the achieved efficiency gain.

5.3 Similarity Measures for Clusterings

The similarity measures introduced in this section are suitable for generally measuring the quality between partitioning and hierarchical approximated

clusterings w.r.t. a given reference clustering. Both partitioning and hierarchical clustering algorithms rely on the notion of a cluster.

Definition 18 (cluster). A *cluster* C is a non-empty subset of objects from a database DB, i.e. $C \subseteq DB$ and $C \neq \emptyset$.

Definition 19 (partitioning clustering). Let DB be a database of arbitrary objects. Furthermore, let $C_1, \ldots C_n$ be pairwise disjoint clusters of DB, i.e. $\forall i, j \in \{1, \ldots n\} : i \neq j \Rightarrow C_i \cap C_j = \emptyset$. Then we call $CL_p = \{C_1, \ldots C_n\}$ a *partitioning clustering* of DB.

Note that due to the handling of noise, we do not demand from a partitioning clustering $CL_p = \{C_1, \ldots C_n\}$ that $C_1 \cup \ldots \cup C_n = DB$ holds. In contrast to the partitioning structure computed by DBSCAN, OPTICS computes a hierarchical clustering order which can be transformed into a tree structure by means of suitable cluster recognition algorithms [ABKS99, BKKP04, SQL+03].

Definition 20 (hierarchical clustering). Let DB be a database of arbitrary objects. A *hierarchical clustering* is a tree t_{root} where each subtree t represents a cluster C_t, i.e. $t = (C_t, (t_1, \ldots t_n))$, and the n subtrees t_i of t represent non-overlapping subsets C_{t_i}, i.e. $\forall i, j \in \{1, \ldots n\} : i \neq j \Rightarrow C_{t_i} \cap C_{t_j} = \emptyset \wedge C_{t_1} \cup \ldots \cup C_{t_n} \subseteq C_t$. Furthermore, the root node t_{root} represents the complete database, i.e. $C_{t_{root}} = DB$.

Again, we do not demand from the n subtrees t_i of $t = (C_t, (t_1, \ldots t_n))$ that $C_{t_1} \cup \ldots \cup C_{t_n} = C_t$ holds (cf. Figure 3.4 where $A_1 \cup A_2 \neq A$).

5.3.1 Similarity Measure for Clusters

As outlined in the last section, both partitioning and hierarchical clusterings consist of flat clusters. In order to compare flat clusters to each other we need a suitable distance measure between sets of objects. One possible approach is to use distance measures as used for constructing distance-based hierarchical clusterings, e.g. the distance measures used by *single-link*, *average-link*

or *complete-link* [JMF99]. Although these distance measures are used for the construction of hierarchical clusterings, these measures are not suitable when it comes to evaluating the quality of flat clusters. The similarity of two clusters w.r.t. quality solely depends on the number of identical objects contained in both clusters which is reflected by the *symmetric set difference*.

Definition 21 (symmetric set difference). Let C_1 and C_2 be two clusters of a database DB. Then the *symmetric set difference* $d_\Delta : 2^{DB} \times 2^{DB} \to [0..1]$ and the *normalized symmetric set difference* $d_\Delta^{norm} : 2^{DB} \times 2^{DB} \to [0..1]$ are defined as follows:

$$\begin{aligned} d_\Delta(C_1, C_2) &= |C_1 \cup C_2| - |C_1 \cap C_2|, \\ d_\Delta^{norm}(C_1, C_2) &= \frac{|C_1 \cup C_2| - |C_1 \cap C_2|}{|C_1 \cup C_2|}. \end{aligned}$$

Note that $(2^{DB}, d_\Delta)$ and $(2^{DB}, d_\Delta^{norm})$ are metric spaces.

5.3.2 Similarity Measure for Partitioning Clusterings

In this section, we will introduce a suitable distance measure between sets of clusters. Several approaches for comparing two sets S and T to each other exist in the literature. In [EM97] the authors survey the following distance functions: the *Hausdorff distance*, the *sum of minimal distances*, the *(fair-)surjection distance* and the *link distance*. All of these approaches rely on the possibility to match several elements in one set to just one element in the compared set which is questionable when comparing the quality of an approximated clustering to a reference clustering.

A distance measure on sets of clusters that demonstrates to be suitable for defining similarity between two partitioning clusterings is the minimal matching distance defined in Definition 12, which is based on the minimal weight perfect matching of sets. This graph problem can be applied here by building a complete bipartite graph $G = (Cl, Cl', E)$ between two clusterings Cl and Cl'. The weight of each edge $(C_i, C'_j) \in Cl \times Cl'$ in this graph G is defined by the distance $d_\Delta(C_i, C'_j)$ introduced in the last section between the two clusters C_i and C'_j. In this scenario, a perfect matching is a subset

$M \subseteq Cl \times Cl'$ that connects each cluster $C_i \in Cl$ to exactly one cluster $C'_j \in Cl'$ and vice versa. A minimal weight perfect matching is a matching having 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 clusters when defining a distance measure between clusterings.

Note that the symmetric set difference d_Δ is a metric and can be used as the underlying distance function D for the minimal matching distance. Furthermore, the unnormalized symmetric set difference allows us to define a meaningful weight function W based on a dummy cluster $\emptyset$ since the empty set is not included as an element in a clustering (cf. Definition 19). We propose to use the following weight function $w_\emptyset(C) = d_\Delta(C, \emptyset)$ where each unmatched cluster C is penalized with a value equal to its cardinality $|C|$. Thus the metric character of the minimal matching distance is satisfied. Furthermore, large clusters which cannot be matched are penalized more than small clusters which is a desired property for an intuitive quality measure.

Definition 22 (clustering distance). Let $Cl = \{C_1, \dots C_{|Cl|}\}$ and $Cl' = \{C'_1, \dots C'_{|Cl'|}\}$ be two clusterings. We assume w.l.o.g. $|Cl| \leq |Cl'|$. Furthermore, let π be a mapping that assigns $C' \in Cl'$ a unique number $i \in \{1, \dots |Cl'|\}$, denoted by $\pi(Cl') = (C'_1, \dots C'_{|Cl'|})$. The family of all possible permutations of Cl' is called $\Pi(Cl')$. Then the *clustering distance* $D_{mm}^{d_\Delta, w_\emptyset}$ is defined as follows:

$$D_{mm}^{d_\Delta, w_\emptyset}(Cl, Cl') = \min_{\pi \in \Pi(Cl')} \left(\sum_{i=1}^{|Cl|} d_\Delta(C_i, C'_{\pi(i)}) + \sum_{i=|Cl|+1}^{|Cl'|} w_\emptyset(C'_{\pi(i)}) \right).$$

Based on Definition 22, we can define our final quality criterion. We compare the costs for transforming an approximated clustering $Cl^{\approx}$ into a reference clustering Cl^{ref} to the costs piling up when transforming $Cl^{\approx}$ first into $\emptyset$, i.e. a clustering consisting of no clusters, and then transforming $\emptyset$ into Cl^{ref}.

Definition 23 (quality measure Q_{APC}). Let $Cl^{\approx}$ be an approximated partitioning clustering and Cl^{ref} the corresponding reference clustering. Then,

the *approximated partitioning clustering quality* $Q_{\text{APC}}(Cl^{\approx}, Cl^{ref})$ is equal to 100% if $Cl^{\approx} = Cl^{ref} = \emptyset$, else it is defined as

$$\left(1 - \frac{d_{mm}^{D_\Delta, w_\emptyset}(Cl^{\approx}, Cl^{ref})}{D_{mm}^{d_\Delta, w_\emptyset}(Cl^{\approx}, \emptyset) + D_{mm}^{d_\Delta, w_\emptyset}(\emptyset, Cl^{ref})}\right) \cdot 100\%.$$

Note that our quality measure Q_{APC} is between 0% and 100%. If $Cl^{\approx}$ and Cl^{ref} are identical, $Q_{\text{APC}}(Cl^{\approx}, Cl^{ref}) = 100\%$ holds. On the other hand, if the clusterings are not identical and the clusters from $Cl^{\approx}$ and Cl^{ref} have no objects in common, i.e. $\forall C^{\approx} \in Cl^{\approx}, C^{ref} \in Cl^{ref} : C^{\approx} \cap C^{ref} = \emptyset$, $Q_{\text{APC}}(Cl^{\approx}, Cl^{ref})$ is equal to 0%.

5.3.3 Similarity Measure for Hierarchical Clusterings

In this section, we present a quality measure for approximated hierarchical clusterings. To the best of our knowledge, the only quality measure for an approximated hierarchical clustering was introduced in [ZS03]. A simple heuristic was applied to find the "best" cut-line, i.e. the most meaningful ε_{cut}-value (cf. Figure 3.4), for a reachability plot resulting from an approximated OPTICS run. The number of clusters found w.r.t. ε_{cut} was compared to the maximum number of clusters found in the reachability plot resulting from an exact clustering. This quality measure has two major drawbacks. First, it does not reflect the *hierarchical* clustering structure, but compares two flat clusterings to each other. Second, the actual elements building up a cluster are not accounted for. Only the number of clusters is used for computing the quality. In the following, we will present a quality measure for hierarchical clusterings which overcomes the two mentioned shortcomings.

As already outlined, a hierarchical clustering can be represented by a tree (cf. Definition 20). In order to define a meaningful quality measure for approximated hierarchical clusterings, we need a suitable distance measure for describing the similarity between two trees $t^{\approx}$ and t^{ref}. Note that each node of the trees reflects a flat cluster, and the complete trees represent the entire hierarchical clusterings.

A common and successfully applied approach to measure the similarity

between two trees is the degree-2 edit distance [ZWS96]. It minimizes the number of edit operations necessary to transform one tree into the other using three basic operations, namely the insertion and deletion of a tree node and the change of a node label. Using these operations, we can define the degree-2 edit distance between two trees.

Definition 24 (cost of an edit sequence). An edit operation e is the insertion, deletion or relabeling of a node in a tree t. Each edit operation e is assigned a non-negative cost $c(e)$. The cost $c(S)$ of a sequence of edit operations $S = \langle e_1, \ldots e_m \rangle$ is defined as the sum of the cost of each edit operation, i.e. $c(S) = c(e_1) + \ldots + c(e_m)$.

Definition 25 (degree-2 edit distance). The degree-2 edit distance is based on degree-2 edit sequences which consist only of insertions or deletions of nodes n with $degree(n) \leq 2$, or of relabelings. Then, the *degree-2 edit distance* ED_2 between two trees t and t' is the minimum cost of all degree-2 edit sequences that transform t into t' or vice versa, i.e. $ED_2(t, t') = \min\{c(S) \mid S$ is a degree-2 edit sequence transforming t into $t'\}$.

It is important to note that the degree-2 edit distance is well defined. Two trees can always be transformed into each other using only degree-2 edit operations. This is true because it is possible to construct any tree using only degree-2 edit operations. As the same is true for the deletion of an entire tree, it is always possible to delete t completely and then build t' from scratch resulting in a distance value for this pair of trees. In [ZWS96] the authors presented an algorithm which computes the degree-2 edit distance in $O(|t| \cdot |t'| \cdot D)$ time, where D denotes the maximum fanout of the trees, and $|t|$ and $|t'|$ the number of tree nodes.

We propose to set the cost $c(e)$ for each insert and delete operation e to 1. Furthermore, we propose to use the *normalized symmetric set difference* d_Δ^{norm} as introduced in Definition 21 to weight the relabeling cost. Using the normalized version allows us to define a well-balanced trade-off between the relabeling cost and the other edit operations, i.e. the insert and delete operations. Based on these costs, we can define our final quality criterion. We compare the costs for transforming an approximated hierarchical clustering

$Cl^{\approx}$ modelled by a tree $t^{\approx}$ into a reference clustering Cl^{ref} modelled by a tree t^{ref}, to the costs piling up when transforming $t^{\approx}$ first into an "empty" tree t^{nil} and then transforming t^{nil} into t^{ref}.

Definition 26 (quality measure Q_{AHC}). Let t^{ref} be a tree representing a hierarchical reference clustering Cl^{ref}, and t^{nil} a tree consisting of no nodes at all, representing an empty clustering. Furthermore, let $t^{\approx}$ be a tree representing an approximated clustering $Cl^{\approx}$. Then, the *approximated hierarchical clustering quality* $Q_{\text{AHC}}(Cl^{\approx}, Cl^{ref})$ is equal to

$$\left(1 - \frac{ED_2(t^{\approx}, t^{ref})}{ED_2(t^{\approx}, t^{nil}) + ED_2(t^{nil}, t^{ref})}\right) \cdot 100\%.$$

As the degree-2 edit distance is a metric [ZWS96], the approximated hierarchical clustering quality Q_{AHC} is between 0% and 100%.

5.4 Experimental Evaluation

In this section, we present a detailed experimental evaluation which demonstrates the characteristics and benefits of our new approach.

5.4.1 Settings

Test Data Sets. As test data, we used real-world CAD data represented by 81-dimensional feature vectors [KKM+03] and vector sets where each element consists of 7 6D vectors [KBK+03]. Furthermore, we used graphs [KKSS04] to represent real-world image data. If not otherwise stated, we used 1,000 complex objects from each data set, and we employed the filter and exact object distance functions proposed in [KBK+03, KKM+03, KKSS04]. The used distance functions can be characterized as follows:

- The exact distance computations on the graphs are very expensive. On the other hand, the used filter is rather selective and can efficiently be computed [KKSS04].

- The exact distance computations on the feature vectors and vector sets are also very expensive as normalization aspects for the CAD objects are taken into account. We compute 48 times the distance between two 81-dimensional feature vectors, and between two vector sets, in order to determine a normalized distance between two CAD objects [KBK+03, KKM+03]. As a filter for the feature vectors we use their Euclidean norms [FJ03] which is not very selective, but can be computed very efficiently. The filter used for the vector sets is more selective than the filter for the feature vectors, but also computationally more expensive [KBK+03].

Implementation. The original OPTICS and DBSCAN algorithms, along with their extensions introduced in this chapter and the used filter and exact object distances were implemented in Java 1.4. The experiments were run on a workstation with a Xeon 2.4 GHz processor and 2 GB main memory under Linux.

Parameter Setting. As suggested in [ABKS99], for an OPTICS run we used a maximum ε parameter in order to create reachability plots containing the complete hierarchical clustering information. For DBSCAN, we chose an ε parameter, based on the reachability plots (cf. the ε_{cut}-values in Figure 3.4), yielding as many flat clusters as possible. Furthermore, if not otherwise stated, the *MinPts* parameter is set to 5, and the length of the predecessor lists is not limited.

Comparison Partners. As a comparison partner for extended OPTICS, we chose the full table scan based on the exact distances, because any other approach would include an unnecessary overhead and is not able to reduce the number of the required $|DB|^2$ exact distance computations. Furthermore, we compared our extended DBSCAN algorithm to the original DBSCAN algorithm based on a full table scan on the exact object distances, and we compared it to a version of DBSCAN which is based on ε-range queries efficiently carried out according to the multi-step query processing paradigm

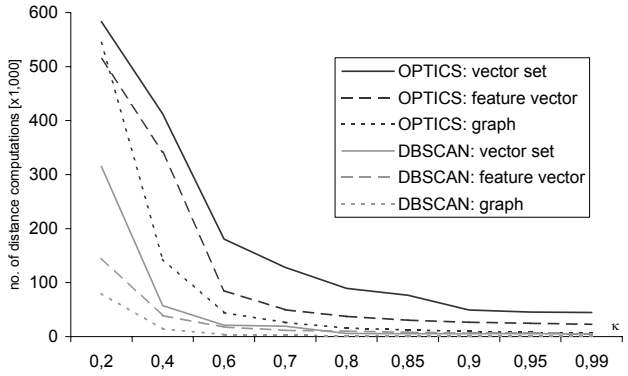
[AFS93]. According to all our tests, this second comparison partner outperforms a DBSCAN algorithm using ε-range queries based on an M-tree [CPZ97] and the DBSCAN algorithm according to [BEKS00].

5.4.2 Exact Clustering Experiments

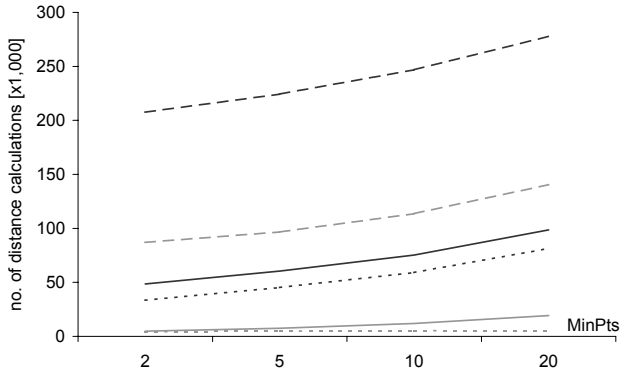
In this section, we first investigate the dependency of our approach on the filter quality, the *MinPts* parameter, and the maximum allowed length of the predecessor lists. For these tests, we concentrate on the discussion of the overall number of distance computations. Furthermore, we investigate the influence of the ε-value in the case of DBSCAN, and, finally, we present the absolute runtimes, in order to show that the required overhead of our approach is negligible compared to the saved exact distance computations.

Dependency on the Filter Quality. In order to demonstrate the dependency of our approach on the quality of the filters, in a first experiment we utilized artificial filter distances d_f lower bounding the exact object distances d_o, i.e. $d_f(o_1, o_2) = \kappa \cdot d_o(o_1, o_2)$ where κ is between 0 and 1. Figure 5.3(a) depicts the number of distance computations n_{dist} w.r.t. κ. In the case of DBSCAN, even rather bad filters, i.e. small values of κ, help to reduce the number of required distance computations considerably, indicating a possible high speed-up compared to both comparison partners of DBSCAN. For good filters, i.e. values of κ close to 1, n_{dist} is very small for DBSCAN and OPTICS indicating a possible high speed-up compared to a full table scan based on the exact distances d_o.

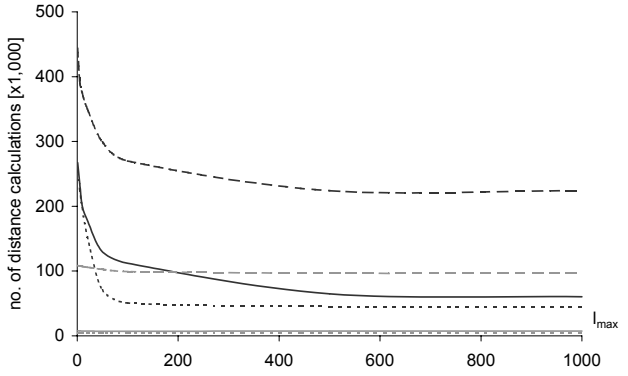
Dependency on the *MinPts* Parameter. Figure 5.3(b) demonstrates the dependency of our approach for a varying *MinPts* parameter while using the filters introduced in [FJ03, KBK$^+$03, KKM$^+$03]. As our approach is based on *MinPts*-nearest neighbor queries, obviously the efficiency of our approach increases with a decreasing *MinPts* parameter. Note that even for rather high *MinPts*-values around $10 = 1\% \cdot |DB|$, our approach saves up to one order of magnitude of exact distance computations compared to a full table



(a) Dependency on the filter quality $d_f(o_1, o_2) = \kappa \cdot d_o(o_1, o_2)$.



(b) Dependency on the *MinPts* parameter.



(c) Dependency on the maximum allowed length of the predecessor lists.

Figure 5.3: Distance calculations for exact clusterings.

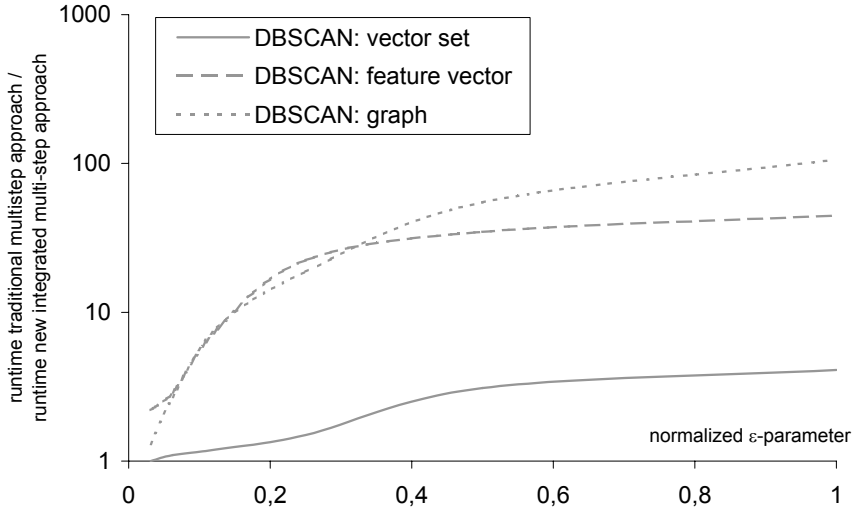


Figure 5.4: Speed-up dependent on the ε parameter.

scan based on d_o, if selective filters are used, e.g. the filters for the vector sets and the graphs. Furthermore, even for the filter of rather low selectivity used by the feature vectors, our approach needs only 1/9 of the maximum number of distance computations in the case of DBSCAN and about 1/4 in the case of OPTICS.

Dependency on the Maximum Allowed Length of the Predecessor Lists. Figure 5.3(c) depicts how the number of distance computations n_{dist} depends on the available main memory, i.e. the maximum allowed length l_{max} of the predecessor lists. Obviously, the higher the value for l_{max}, the less exact distance computations are required. The figure shows that for OPTICS we have an exponential decrease of n_{dist} w.r.t. l_{max}, and for DBSCAN n_{dist} is almost constant w.r.t. changing l_{max} parameters, indicating that small values of l_{max} are sufficient to reach the best possible runtimes.

Dependency on the ε parameter. Figure 5.4 shows how the speed-up for DBSCAN between our integrated multi-step query processing approach and the traditional multi-step query processing approach depends on the chosen ε parameter. The higher the chosen ε parameter, the more our new approach

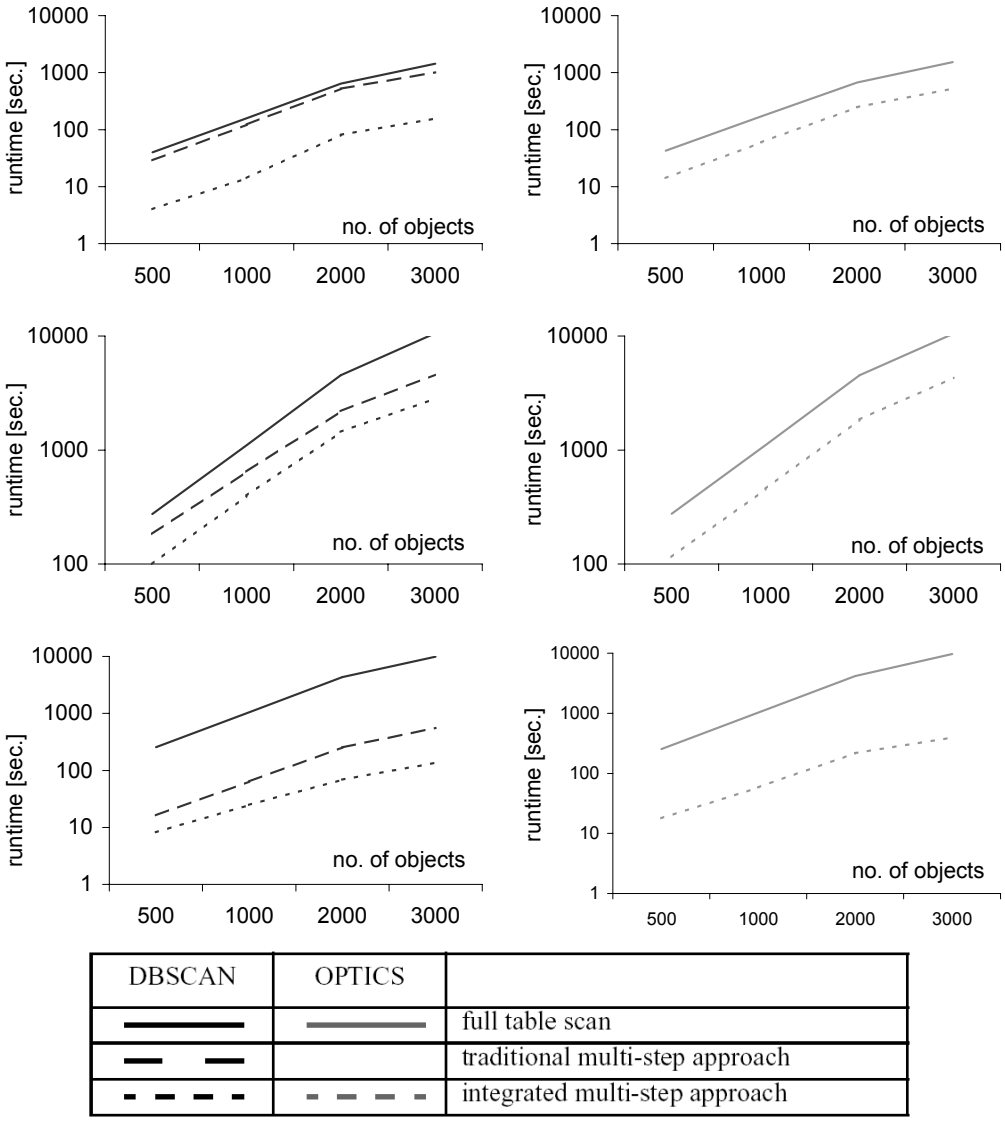


Figure 5.5: Absolute runtimes w.r.t. varying database sizes.

outperforms the traditional one which has to compute the exact distances between o and q for all $o \in N_\varepsilon^{\text{filter}}(q)$. In contrast, our approach confines itself to *MinPts*-nearest neighbor queries on the exact distances and computes further distances only if compulsory to compute the exact clustering result.

Absolute Runtimes. Figure 5.5 presents the absolute runtimes of the new extended DBSCAN and OPTICS algorithms which integrate the multi-step query processing paradigm compared to the full table scan on the exact object representations. Furthermore, we also compare our extended DBSCAN to a DBSCAN variant using ε-range queries based on the traditional hmulti-step query processing paradigm. Note, that this comparison partner would induce an unnecessary overhead in the case of OPTICS where we have to use very high ε parameters in order to detect the complete hierarchical clustering order. In all experiments, our approach was always the most efficient one. For instance, for DBSCAN on the feature vectors, our approach outperforms both comparison partners by an order of magnitude indicating that rather bad filters are already useful for our new extended DBSCAN algorithm. Note that the traditional multi-step query processing approach does not benefit much from non-selective filters even when small ε-values are used. In the case of OPTICS, the performance of our approach improves with increasing filter quality. For instance, for the graphs we achieve a speed-up factor of more than 30 indicating the suitability of our extended OPTICS algorithm.

5.4.3 Approximated Clustering Experiments

In this section, we carry out experiments where we just cut off the predecessor lists $PL(o)$ after the l_{max}-th element without computing any additional exact distance computations between o and the discarded potential predecessor objects. Note that this approach might lead to an information loss. Figure 5.6 shows that the maximum number of needed distance calculations only marginally increases for higher l_{max}-values for the graphs and the vector sets indicating that we can cut off the object reachability lists at small l_{max}-values without a considerable information loss. On the other hand, for the feature

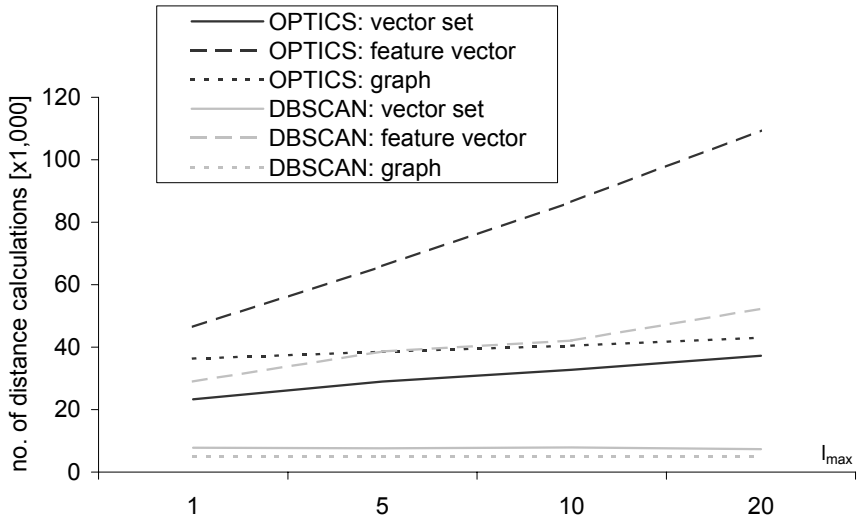


Figure 5.6: Distance calculations for approximated clusterings.

vectors we have to compute more exact distance computations the higher the l_{max}-value is. The additionally needed exact distance computations (cf. line (*) in Figure 5.2) are due to the rather low filter selectivity of the used filter.

Next, we examine the quality of our approximated clustering algorithms by using the quality measures introduced in Section 5.3. For extracting the hierarchical tree structure, we used the cluster recognition algorithm presented in [BKKP04]. Figure 5.7 depicts the quality measures Q_{APC} for DBSCAN and Q_{AHC} for OPTICS for our three test data sets w.r.t. varying l_{max} values. Our quality measures indicate a very high quality for the graphs and the vector sets over the full range of the investigated l_{max} values. On the other hand, when using the feature vectors both quality measures Q_{APC} (for DBSCAN) and Q_{AHC} (for OPTICS) increase with increasing l_{max} values. These tests not only indicate that we can cut off the predecessor lists at small values of l_{max} without considerably worsening the clustering quality when using selective filters. The tests also demonstrate the suitability of our quality measures Q_{APC} and Q_{AHC} which indicate low quality when filters of low selectivity are combined with small l_{max} values.

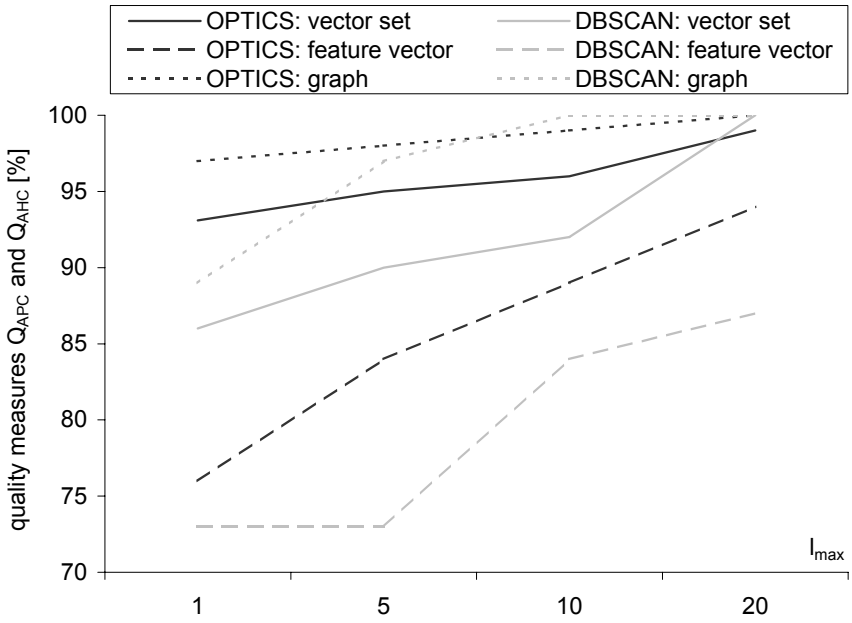


Figure 5.7: Quality measures for approximated clusterings.

5.5 Summary

In many different application areas, density-based clustering is an effective approach for mining complex data. Unfortunately, the runtime of these data mining algorithms is rather high, as the distance functions between complex object representations are often very expensive. In this chapter, we showed how to integrate the well-known multi-step query processing paradigm directly into the two density-based clustering algorithms DBSCAN and OPTICS. We replaced the expensive exact ε-range queries by *MinPts*-nearest neighbor queries which themselves are based on ε-range queries on lower-bounding filter distances. Further exact complex distance computations are only carried out at that stage of the algorithms where they are compulsory to compute the correct clustering result. Furthermore, we showed how we can use the presented approach for approximated clustering. In order to evaluate the trade-off between the achieved efficiency gain and the quality loss, we introduced suitable quality measures for comparing the partitioning and hierarchical approximated clusterings to the exact ones. In a broad experimental

evaluation based on real-world test data sets we demonstrated that our new approach leads to a significant speed-up compared to a full table scan on the exact object representations as well as compared to an approach, where the ε-range queries are accelerated by means of the traditional multi-step query processing concept. Furthermore, we showed that for approximated clusterings we can reduce the number of required distance computations even further. Finally, we pointed out that the resulting approximated clustering quality heavily depends on the filter quality demonstrating the suitability of our introduced quality measures.

Chapter 6

Parallel Density-Based Clustering of Complex Objects

Density-based clustering algorithms like DBSCAN are based on ε-range queries for each database object. Thereby, each range query requires a lot of distance calculations. When working with complex objects, e.g. trees, point sets, and graphs, often complex time-consuming distance functions are used to measure similarity accurately. As these distance calculations are the time-limiting factor of the clustering algorithm, the ultimate goal is to save as many as possible of these complex distance calculations.

In Chapter 5 an approach was presented for the efficient density-based clustering of complex objects. The core idea of this approach is to integrate the multi-step query processing paradigm directly into the clustering algorithm rather than using it "only" for accelerating range queries. In this chapter, we present a sophisticated parallelization of this approach. Similar to the area of join processing where there is an increasing interest in algorithms which do not assume the existence of any index structure, we propose an approach for parallel DBSCAN which does not rely on the pre-clustering of index structures.

First, the data is partitioned according to the clustering result carried out on cheaply computable distance functions. The resulting approximated clustering conservatively approximates the exact clustering. The objects of

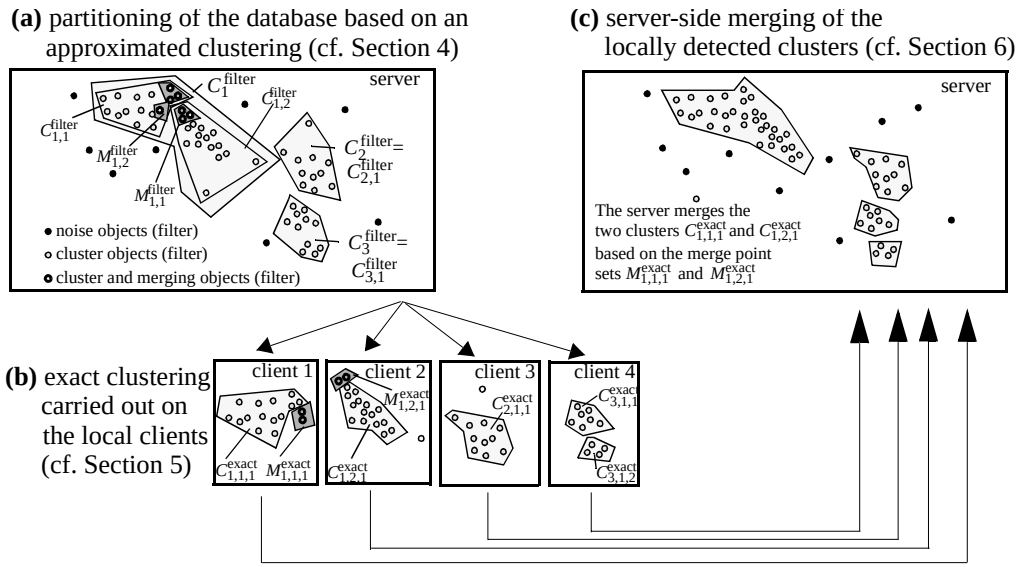


Figure 6.1: Basic idea of parallel density-based clustering.

the conservative cluster approximations are then distributed onto the available slaves in such a way that each slave has to cluster the same amount of objects, and that the objects to be clustered are close to each other. Note that already at this early stage, we can detect some noise objects which do not have to be transmitted to the local clients. In addition to the objects to be clustered by a client, we send some filter merge points to this client. These filter merge points are also determined based on approximated distance functions (cf. Figure 6.1(a)).

Second, each client carries out the clustering independently of all the other clients. No further communication is necessary throughout this second step. The presented local clustering approach also takes advantage of the approximating lower-bounding distance functions. The detected clusters and the detected exact merge point sets are then transmitted to the server (cf. Figure 6.1(b)).

Finally, the server determines the correct clustering result by merging the locally detected clusters. This final merging step is based on the exact merge points detected by the clients. Based on these merge points, cluster connectivity graphs are created. In these graphs, the nodes represent the

locally detected clusters. Two local clusters are connected by an edge if a merge point of one cluster is a core object in the other cluster (cf. Figure 6.1(c)).

The remainder of this chapter is organized as follows. In Section 6.1, we shortly sketch the work from the literature related to our approach. In Sections 6.2, 6.3 and 6.4, we explain the server-side partitioning algorithm, the client-side clustering algorithm, and the server-side merging of the results from the clients, respectively. In Section 6.5, we present a detailed experimental evaluation based on real world test data sets. We close the chapter in Section 6.6 with a short summary.

6.1 Related Work

Complex Object Representations. High-dimensional feature vectors [KKM+03], vector sets [KBK+03], trees and graphs [KS03] are helpful complex object representations, which model real world objects accurately. The similarity between these complex object representations is often measured by means of expensive distance function, e.g. the edit distance. For a more detailed survey on this topic, we refer the interested reader to [Kai04].

Clustering. Given a set of objects with a distance function on them, an interesting data mining question is, whether these objects naturally form groups (called clusters) and what these groups look like. Data mining algorithms that try to answer this question are called clustering algorithms. For a detailed overview on clustering, we refer the interested reader to [JMF99].

Density-Based Clustering. Density based clustering algorithms apply a local cluster criterion to detect clusters. Clusters are regarded as regions in the data space in which the objects are dense, and which are separated by regions of low object density (noise). One of the most prominent representatives of this clustering paradigm is DBSCAN [EKSX96].

Parallel Density-Based Clustering. In [XJK99] a parallel version of DBSCAN has been introduced. The algorithm starts with the complete data set residing on one central sever and then distributes the data among the different clients. The underlying data structure is the dR*-tree, a modification of the R*-tree [BKSS90]. The directory of the R*-tree is replicated on all available computers to enable efficient access to the distributed data. This distributed R*-tree is called the dR*-tree, which has the following structural differences from a traditional centralized R*-tree: the data pages are distributed on different computers, the indices are replicated on all computers, and the pointer to a data page consists of a computer identifier and a page ID. In order to distribute the different data pages onto the different slaves, the centers of the leaf pages are ordered by their Hilbert values. Then each client receives an equal number of data pages having adjacent Hilbert values. The different slaves communicate via message-passing and cluster their data separately. Finally, the server has to merge the different clustering results. This approach suffers from several drawbacks. First, the clients have to communicate to each other after the partitioning of the data took place. Second, the approach is only applicable to feature vector represented objects but not generally to metric objects. Third, the existence of an index structure is presumed. In the remainder of this chapter, we will present a more general approach for parallelizing DBSCAN which overcomes all of these shortcomings.

6.2 Server-Side Data Partitioning

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 *MinPts* objects, i.e. the cardinality of the neighborhood has to exceed a given threshold. A flat density-based *cluster* is defined as a set of density-connected objects which is maximal w.r.t. density-reachability. Thus a cluster contains not only core objects but also border objects that do not satisfy the core object condition. The *noise* is the set of objects not contained in any cluster. While the partitioning density-based clustering algorithm

DBSCAN can only identify a flat clustering, the newer algorithm OPTICS computes an ordering of the points augmented by the reachability-distance introduced in Definition 9. The reachability-distance basically denotes the smallest distance of the current object q to any core object which belongs to the current cluster and which has already been processed. The clusters detected by DBSCAN can also be found in the OPTICS ordering when using the same parametrization, i.e. the same ε and *MinPts* values. For an initial clustering with OPTICS based on the lower-bounding filter distances the following two lemmas hold.

Lemma 8. *Let* $C_1^{\text{exact}}, \ldots, C_n^{\text{exact}}$ *be the clusters detected by OPTICS based on the exact distances, and let* $C_1^{\text{filter}}, \ldots, C_m^{\text{filter}}$ *be the clusters detected by OPTICS based on the lower-bounding filter distances. Then the following statement holds:*

$$\forall i \in \{1, \ldots, n\} \exists j \in \{1, \ldots, m\} : C_i^{\text{exact}} \subseteq C_j^{\text{filter}}.$$

Proof. Let $N_\varepsilon^{\text{filter}}(o)$ denote the ε-neighborhood of o according to the filter distances, and let $N_\varepsilon^{\text{exact}}(o)$ denote the ε-neighborhood according to the exact distances. Due to the lower-bounding filter property $N_\varepsilon^{\text{exact}}(o) \subseteq N_\varepsilon^{\text{filter}}(o)$ holds. Therefore, each object o which is a core object based on the exact distances is also a core object based on the lower-bounding filter distances. Furthermore, each object p which is directly density-reachable from o according to the exact distances is also directly density-reachable according to the filter functions. Induction on this property shows that if p is density-reachable from o based on the exact distances, it also holds for the filter distances. Therefore, all objects which are in one cluster according to the exact distances are also in one cluster according to the approximated distances. □

Lemma 9. *Let* $\text{noise}^{\text{exact}}$ *denote the noise objects detected by OPTICS based on the exact distances and let* $\text{noise}^{\text{filter}}$ *denote the noise objects detected by OPTICS based on the lower-bounding filter distances. Then the following statement holds:*

$$\text{noise}^{\text{filter}} \subseteq \text{noise}^{\text{exact}}.$$

Proof. An object p is a noise object if it is not included in the ε-neighborhood of any core object. Again, let $N_\varepsilon^{\text{filter}}(o)$ and $N_\varepsilon^{\text{exact}}(o)$ denote the ε-neighborhood of o according to the filter distances and the exact distances, respectively. Due to the lower-bounding filter property $N_\varepsilon^{\text{exact}}(o) \subseteq N_\varepsilon^{\text{filter}}(o)$ holds. Therefore, if $p \notin N_\varepsilon^{\text{filter}}(o)$, it cannot be included in $N_\varepsilon^{\text{exact}}(o)$, proving the lemma. □

Lemmas 8 and 9 are both helpful to partition the data onto the different slaves. Lemma 8 shows that exact clusters are conservatively approximated by the clusters resulting from a clustering on the lower-bounding distance functions. On the other hand, Lemma 9 shows that exact noise is progressively approximated by the set of noise objects resulting from an approximated clustering. For this reason, noise objects according to the filter distances do not have to be transmitted to the slaves, as we already know that they are also noise objects according to the exact distances. All other N objects have to be refined by the P available slave processors. Let $C_1^{\text{filter}}, \ldots, C_m^{\text{filter}}$ be the approximated clusters resulting from an initial clustering with OPTICS. In this approach, we assign $P_{\text{slave}} = \sum_{i=1}^{m} |C_i^{\text{filter}}|/P$ objects to each of the P slaves. We do this partitioning online while carrying out the OPTICS algorithm. At each time during the clustering algorithm, OPTICS knows the slave j having received the smallest number L_j of objects up to now, i.e. the client j has the highest free capacity $C_j = P_{\text{slave}} - L_j$. OPTICS stops the current clustering at two different event points: In the first case, a cluster C_i^{filter} of cardinality $|C_i^{\text{filter}}| \leq C_j$ was completely determined. This cluster is sent to the slave j. In the second case, OPTICS determined C_j more points belonging to the current cluster C_i^{filter}. These points are grouped together to a filter cluster $C_{i,j}^{\text{filter}}$. Then, we transmit the cluster $C_{i,j}^{\text{filter}}$ along with the filter merge points $M_{i,j}^{\text{filter}}$ to the slave j. The set $M_{i,j}^{\text{filter}}$ can be determined throughout the clustering of the set $C_{i,j}^{\text{filter}}$ and can be defined as follows.

Definition 27 (filter merge points). Let C_i^{filter} be a cluster which is split during an OPTICS run into n clusters $C_{i,1}^{\text{filter}}, \ldots, C_{i,n}^{\text{filter}}$. Then, the *filter merge points* $M_{i,j}^{\text{filter}}$ for a partial filter cluster $C_{i,j}^{\text{filter}}$ are defined as follows: $M_{i,j}^{\text{filter}} = \{q \in C_i^{\text{filter}} - C_{i,j}^{\text{filter}} \mid \exists p \in C_{i,j}^{\text{filter}} : q \text{ is directly density-reachable from } p\}$.

The filter merge points $M_{i,j}^{\text{filter}}$ are necessary in order to decide whether objects $o \in C_{i,j}^{\text{filter}}$ are core objects. Furthermore, a subset $M_{i,j}^{\text{exact}} \subseteq M_{i,j}^{\text{filter}}$ is used to merge exact clusters in the final merge step (cf. Section 6.4).

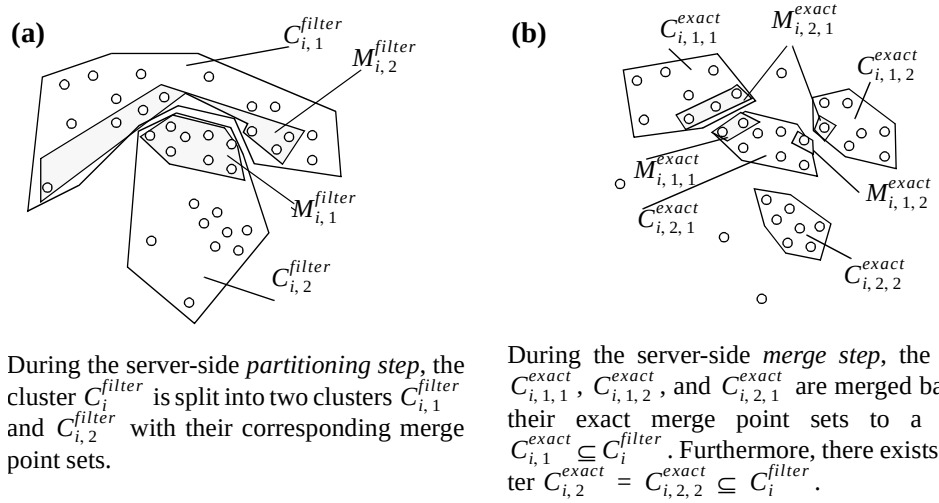
6.3 Client-Side Clustering

Each of the filter clusters $C_{i,j}^{\text{filter}}$ is clustered independently on the exact distances by the assigned slave j. For clustering these filter clusters, we adapt the approach presented in Chapter 5, so that it can also handle the additional merge points $M_{i,j}^{\text{filter}}$. The main idea of the client-side clustering approach is to carry out the range queries based on the lower-bounding filter distances instead of using the expensive exact distances. Thereto, we do not use the simple seedlist of the original DBSCAN algorithm, but we use the extended data structure *Xseedlist* introduced in Chapter 5. The *Xseedlist* consists of an ordered *object list OL*. Each entry $(o, T, PL) \in OL$ contains a flag T indicating whether $o \in C_{i,j}^{\text{filter}}$ ($T = \text{C}$) or $o \in M_{i,j}^{\text{filter}}$ ($T = \text{M}$). Each entry of the *predecessor list PL* consists of the following information: a *predecessor* o_p of o, which is a core object already added to the current cluster, and the *predecessor distance*, which is equal to the filter distance $d_f(o, o_p)$ between the two objects.

The result of the extended DBSCAN algorithm is a set of exact clusters $C_{i,j,l}^{\text{exact}} \subseteq C_{i,j}^{\text{filter}}$ along with their additional exact merge points $M_{i,j,l}^{\text{exact}} \subseteq M_{i,j}^{\text{filter}}$. To expand a cluster $C_{i,j,l}^{\text{exact}}$ we take the first element (o, T, PL) from OL and set o_p to the nearest predecessor object in PL.

Let us first assume that $T = \text{C}$ holds. If $PL = \text{NIL}$ holds, we add o to $C_{i,j,l}^{\text{exact}}$, delete o from OL, carry out a range query around o, and try to expand the cluster $C_{i,j,l}^{\text{exact}}$. If $PL \neq \text{NIL}$ holds, we compute $d_o(o, o_p)$. If $d_o(o, o_p) \leq \varepsilon$, we proceed as in the case where $PL = \text{NIL}$ holds. If $d_o(o, o_p) > \varepsilon$ and length of $PL > 1$ hold, we delete the first entry from PL. If $d_o(o, o_p) > \varepsilon$ and length of $PL = 1$ hold, we delete o from OL. Iteratively, we try to expand the current cluster $C_{i,j,l}^{\text{exact}}$ by examining the first entry of OL until OL is empty.

Let us now assume that $T = \text{M}$ holds. If $PL = \text{NIL}$ holds, we add o



During the server-side *partitioning step*, the cluster C_i^{filter} is split into two clusters $C_{i,1}^{filter}$ and $C_{i,2}^{filter}$ with their corresponding merge point sets.

During the server-side *merge step*, the cluster $C_{i,1,1}^{exact}$, $C_{i,1,2}^{exact}$, and $C_{i,2,1}^{exact}$ are merged based on their exact merge point sets to a cluster $C_{i,1}^{exact} \subseteq C_i^{filter}$. Furthermore, there exists a cluster $C_{i,2}^{exact} = C_{i,2,2}^{exact} \subseteq C_i^{filter}$.

Figure 6.2: Server-side partitioning step (a) and merge step (b).

to $M_{i,j,l}^{\text{exact}}$, delete o from OL, and try to expand the exact merge point set $M_{i,j,l}^{\text{exact}}$. If $PL \neq \text{NIL}$ holds, we compute $d_o(o, o_p)$. If $d_o(o, o_p) \leq \varepsilon$, we proceed as in the case where $PL = \text{NIL}$ holds. If $d_o(o, o_p) > \varepsilon$ and length of $PL > 1$ hold, we delete the first entry from PL. If $d_o(o, o_p) > \varepsilon$ and length of $PL = 1$ hold, we delete o from OL. Iteratively, we try to expand the current exact merge point set $M_{i,j,l}^{\text{exact}}$ by examining the first entry of OL until OL is empty.

6.4 Server-Side Merging

Obviously, we only have to carry out the merge process for those clusters C_i^{filter} which were split in several clusters $C_{i,j}^{\text{filter}}$. The client detects that each of these clusters $C_{i,j}^{\text{filter}}$ contains t clusters $C_{i,j,1}^{\text{exact}}, \ldots, C_{i,j,t}^{\text{exact}}$. Note that t can also be equal to 0, i.e. no exact cluster is contained in the cluster $C_{i,j}^{\text{filter}}$. For each of the t exact clusters $C_{i,j,l}^{\text{exact}}$ there also exists a corresponding set of exact merge points $M_{i,j,l}^{\text{exact}} \subseteq M_{i,j}^{\text{filter}}$ (cf. Figure 6.2) defined as follows.

Definition 28 (exact merge points). Let $C_{i,j}^{\text{filter}}$ be a cluster to be refined on the slave with the corresponding merge point set $M_{i,j}^{\text{filter}}$. Let $C_{i,j,l}^{\text{exact}} \subseteq C_{i,j}^{\text{filter}}$ be an exact cluster determined during the client-side refinement clustering. Then, we determine the set $M_{i,j,l}^{\text{exact}} \subseteq M_{i,j}^{\text{filter}}$ of *exact merge points* where

$M_{i,j,l}^{\text{exact}} = \{q \in M_{i,j}^{\text{filter}} \mid \exists p \in C_{i,j,l}^{\text{exact}} : q \text{ is directly density-reachable from } p\}$.

Based on these exact merge point sets and the exact clusters, we can define a "cluster connectivity graph".

Definition 29 (cluster connectivity graph). Let C_i^{filter} be a cluster which was refined on one of the s different slaves. Let $C_{i,j,l}^{\text{exact}} \subseteq C_{i,j}^{\text{filter}} \subseteq C_i^{\text{filter}}$ be an exact cluster determined by slave j along with the corresponding merge point sets $M_{i,j,l}^{\text{exact}} \subseteq M_{i,j}^{\text{filter}}$. Then a graph $G_i = (V_i, E_i)$ is called a *cluster connectivity graph* for C_i^{filter} iff the following statements hold:

- $V_i = \{C_{i,1,1}^{\text{exact}}, \ldots, C_{i,1,n_1}^{\text{exact}}, \ldots, C_{i,s,1}^{\text{exact}}, \ldots, C_{i,s,n_s}^{\text{exact}}\}$.
- $E_i = \{(C_{i,j,l}^{\text{exact}}, C_{i,j',l'}^{\text{exact}}) \mid \exists p \in M_{i,j,l}^{\text{exact}} : p \in C_{i,j',l'}^{\text{exact}} \wedge p \text{ is a core point}\}$.

Note that two clusters $C_{i,j,l}^{\text{exact}}$ and $C_{i,j',l'}^{\text{exact}}$ from the same slave $j = j'$ are never connected by an edge. Such a connection of the two clusters would already have taken place throughout the refinement clustering on the slave j. Based on the connectivity graphs G_i for the approximated clusterings C_i^{filter}, we can determine the *database connectivity graph*.

Definition 30 (database connectivity graph). Let C_i^{filter} be one of n approximated clusters along with the corresponding cluster connectivity graph $G_i = (V_i, E_i)$. Then we call $G = (\bigcup_{i=1}^{n} V_i, \bigcup_{i=1}^{n} E_i)$ the *database connectivity graph*.

The database connectivity graph is nothing else but the union of the connectivity graphs of the approximated clusters. Based on the above definition, we state the central lemma of this chapter.

Lemma 10. *Let G be the database connectivity graph. Then the determination of all maximal connected subgraphs of G is equivalent to a DBSCAN clustering carried out on the exact distances.*

Proof. For each object the client-side clustering determines correctly, whether it is a core object, a border object, or a noise object. Note, that we assign a border object which is directly density-reachable from core objects of different

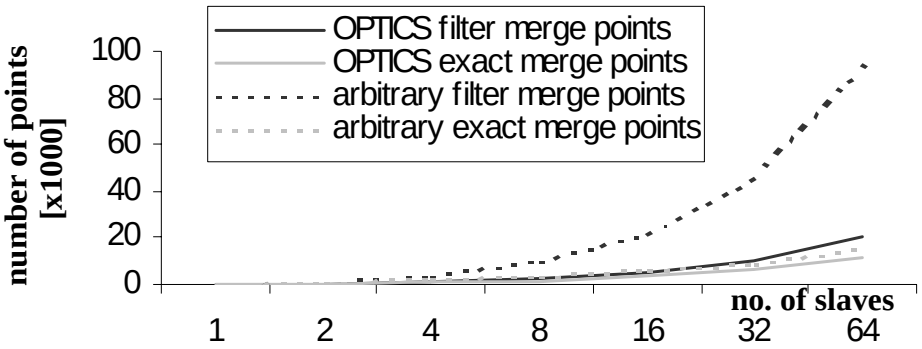
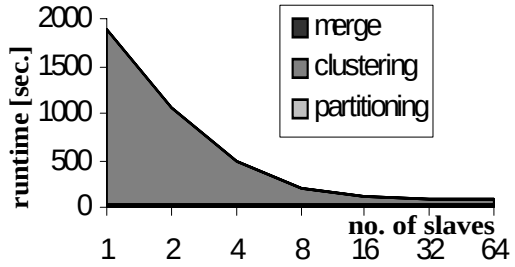


Figure 6.3: Number of merge points w.r.t. a varying number of slaves for the graph dataset.

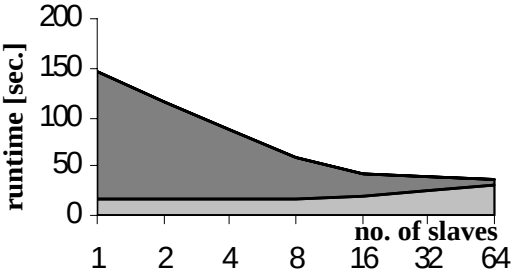
clusters redundantly to all of these clusters. Therefore, the only remaining issue is to show that two core objects which are directly density-reachable to each other are in the same maximal connected subgraph. By induction, according to the definition of density-reachability, two clusters then contain the same core objects. Obviously, two core objects o_1 and o_2 are directly density-reachable if they are either in the same exact cluster $C_{i,j,l}^{\text{exact}}$ or if $o_1 \in C_{i,j,l}^{\text{exact}}$ and $o_2 \in M_{i,j,l}$exact resulting in an edge of the database connectivity graph. Therefore, depth-first traversals through all of the connectivity graphs G_i corresponding to a filter cluster C_i^{filter} create the correct clustering result where each subgraph corresponds to one cluster. □

6.5 Experimental Evaluation

In this section, we present a detailed experimental evaluation based on real-world data sets. We used CAD data represented by 81-dimensional feature vectors [KKM+03] and vector sets where each element consists of 7 6D vectors [KBK+03]. Furthermore, we used graphs [KS03] to represent image data. The used distance functions can be characterized as follows: (i) The exact distance computations on the graphs are very expensive. On the other hand, the filter is rather selective and can efficiently be computed. (ii) The exact distance computations on the feature vectors and vector sets are also very expensive as normalization aspects for the CAD objects are taken into



(a) Vector sets.



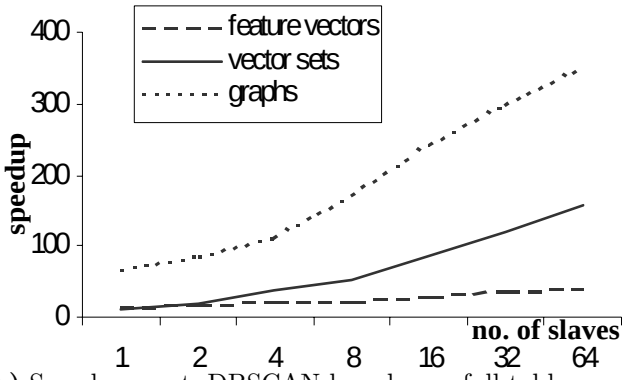
(b) Graphs.

Figure 6.4: Absolute runtimes w.r.t. a varying number of slaves.

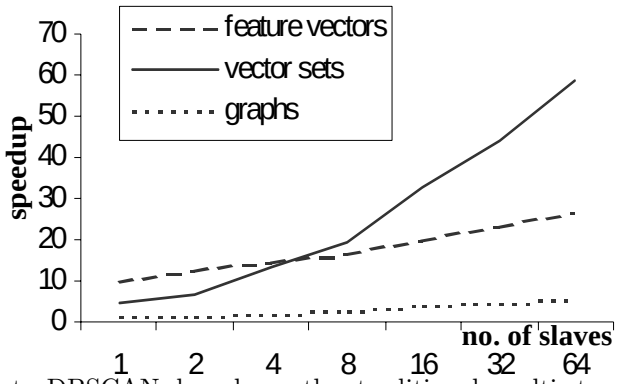
account [KBK+03, KKM+03]. As a filter for the feature vectors we use their Euclidean norms [FJ03] which is not very selective, but can be computed very efficiently. The filter used for the vector sets is more selective than the filter for the feature vectors, but also computationally more expensive. If not otherwise stated, we used 3,000 complex objects from each data set.

The original OPTICS and DBSCAN algorithms, their extensions introduced in this chapter, and the used filter and exact distances functions were implemented in Java 1.4. The experiments were run on a workstation with a Xeon 2.4 GHz processor and 2 GB main memory. All experiments were run sequentially on one computer. Thereby, the overall time for the client-side clustering is determined by the slowest slave. If not otherwise stated, we chose an ε-parameter yielding as many flat clusters as possible, and the *MinPts*-parameter was set to 5.

Characteristics of the partitioning step. Figure 6.3 compares the number of merge points for different split techniques applied to filter clusters. As explained in Section 6.2, we split a filter cluster during the partitioning step



(a) Speedup w.r.t. DBSCAN based on a full table scan.



(b) Speedup w.r.t. DBSCAN based on the traditional multi-step query processing paradigm.

Figure 6.5: Overall speedup w.r.t. a varying number of slaves.

along the ordering produced by OPTICS. Note that OPTICS always walks through a cluster by visiting the densest areas first. Figure 6.3 shows that this kind of split strategy yields considerably less merge points than a split strategy which arbitrarily groups objects from a filter cluster together. Thus, the figure proves the good clustering properties of our metric space filling curve OPTICS.

Dependency on the Number of Slaves. Figure 6.4 shows the absolute runtimes of our parallel DBSCAN approach dependent on the number of available slaves for the vector sets and for the graph dataset. The figure shows the accumulated times after the partitioning, client-side clustering, and the merge step. The partitioning times also include simulated communication

times for the transfer of the objects to the slaves in a 100 Mbit LAN. No communication costs arise from the client-side clustering step, as each client already received all needed filter merge points. A growing number of slaves leads to a significant speedup of the client-side clustering. A lower bound of the achievable total runtime is given by the time needed for the initial partitioning step. It is worth to note the time needed for the final merging step is negligible even for a high number of slaves. Although the number of exact merge points grows with an increasing number of slaves (cf. Figure 6.3), the merge step remains cheap.

Speedup. Finally, Figure 6.5 depicts the speedup achieved by our new parallel DBSCAN approach based on a server-side partitioning with OPTICS. We compared this approach to a DBSCAN approach based on a full table scan and compared to a DBSCAN approach based on the traditional multi-step query processing paradigm. The figure shows that for the feature vectors we achieve a speedup of one order of magnitude already when only one slave is available. In the case of the graph dataset we have a speedup of 67 compared to DBSCAN based on a full table scan. These results demonstrate the suitability of the client-side clustering approach. For the vector sets the benefits of using several slaves can clearly be seen. For instance, our approach achieves a speedup of 4 for one slave and a speedup of 20 for eight slaves compared to DBSCAN based on traditional multi-step range queries.

6.6 Summary

In this chapter, we applied the novel concept of using efficiently computable lower-bounding distance functions for the parallelization of data mining algorithms to the density-based clustering algorithm DBSCAN. For partitioning the data, we used the hierarchical clustering algorithm OPTICS as a kind of space filling curve for general metric objects, which provides the foundation for a fair and suitable partitioning strategy. We showed how the local clients can carry out their clustering efficiently by integrating the multi-step query processing paradigm directly into the clustering algorithm. Based on the con-

cept of merge points, we constructed a global cluster connectivity graph from which the final clustering result can easily be derived. In the experimental evaluation, we demonstrated that our new approach is able to efficiently cluster metric objects. We showed that if several slaves are available, the benefits achieved by the full computational power of the slaves easily outweigh the additional costs of partitioning and merging by the master.

Part III

Advanced Similarity Search Applications

Chapter 7

Visual Density-Based Data Analysis

Hierarchical clustering was shown to be effective for evaluating similarity models [KKM+03, KBK+03]. Especially, the reachability plot generated by OPTICS is suitable for assessing the quality of a similarity model. Furthermore, visually analyzing cluster hierarchies helps the user, e.g. an engineer, to find and group similar objects. Solid cluster extraction and meaningful cluster representatives form the foundation for providing the user with significant and quick information.

In this chapter, we introduce algorithms for automatically detecting hierarchical clusters along with their corresponding representatives. In order to evaluate our ideas, we developed a prototype called *BOSS* (*B*rowsing *O*PTICS Plots for *S*imilarity *S*earch). BOSS is based on techniques related to *visual data mining*. It helps to visually analyze cluster hierarchies by providing meaningful cluster representatives. Furthermore, we developed a second prototype called *VICO* (*Vi*sualizing *C*onnected *O*bject Orderings). It allows the user to interactively compare different views on the same set of data objects. The idea of VICO is to compare the position of data objects or even complete clusters in a set of data spaces by highlighting them in various OPTICS plots.

The remainder of the chapter is organized as follows. In Section 7.1 we

present the application areas of hierachical clustering along with the corresponding requirements in the industrial and in the scientific community which motivated the development of BOSS. In Sections 7.2 and 7.3, we introduce suitable algorithms for cluster recognition and cluster representatives, respectively. The prototype application BOSS for browsing cluster hierarchies is described in Section 7.4. VICO, the protorype for comparing data spaces, is presented in Section 7.5. In Section 7.6 we evaluate our new cluster recognition and representation algorithms. The chapter concludes in Section 7.7 with a short summary.

7.1 Application Ranges

In this section, we outline the application ranges which led to the development of our interactive browsing tool, called BOSS. In order to understand the connection between BOSS and the application requirements we first point out the major concepts of the hierarchical clustering algorithm OPTICS and its output, the so-called reachability plots, which served as a starting point for BOSS. The technical aspects related to BOSS are described later in Section 7.6.

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 *MinPts* of objects. Using the density-based hierarchical clustering algorithm OPTICS yields several advantages due to the following reasons.

- OPTICS is – in contrast to most other algorithms – relatively insensitive to its two input parameters, ε and *MinPts*. The authors in [ABKS99] state that the input parameters just have to be large enough to produce good results.
- OPTICS is a hierarchical clustering method which yields more information about the cluster structure than a method that computes a flat partitioning of the data (e.g. k-means [McQ67]).
- There exists a very efficient variant of the OPTICS algorithm which

is based on a sophisticated data compression technique called "Data Bubbles" [BKKS01], where we have to trade only very little quality of the clustering result for a great increase in performance.

- There exists an efficient incremental version of the OPTICS algorithm [KKG03].

BOSS was designed for three different purposes: visual data mining, similarity search and evaluation of similarity models. For the first two applications, the choice of the representative objects of a cluster is the key step. It helps the user to get a meaningful and quick overview over a large existing data set. Furthermore, BOSS helps scientists to evaluate new similarity models.

7.1.1 Visual Data Mining

As defined in [Ank00], visual data mining is a step in the KDD process that utilizes visualization as a communication channel between the computer and the user to produce novel and interpretable patterns. Based on the balance and sequence of the automatic and the interactive (visual) part of the KDD process, three classes of visual data mining can be identified.

- Visualization of the data mining result:
  An algorithm extracts patterns from the data. These patterns are visualized to make them interpretable. Based on the visualization, the user may want to return to the data mining algorithm and run it again with different input parameters (cf. Figure 7.1(a)).

- Visualization of an intermediate result:
  An algorithm performs an analysis of the data not producing the final patterns but an intermediate result which can be visualized. Then the user retrieves the interesting patterns in the visualization of the intermediate result (cf. Figure 7.1(b)).

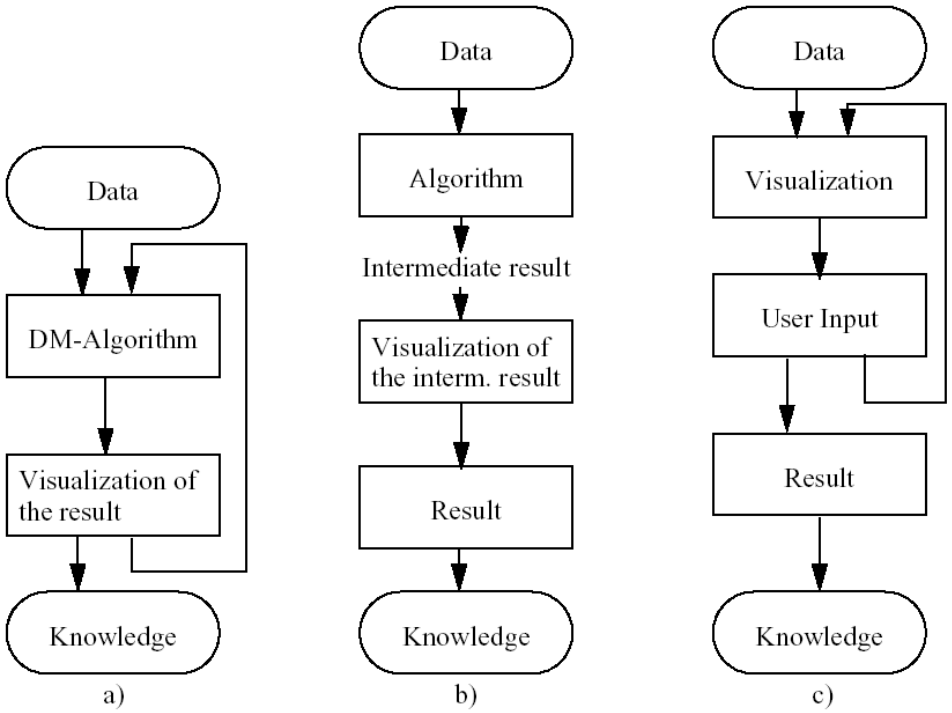


Figure 7.1: Different approaches to visual data mining.

- Visualization of the data:
  Data is visualized immediately without running a sophisticated algorithm before. Patterns are obtained by the user by exploring the visualized data (cf. Figure 7.1(c)).

The approach presented in this chapter belongs to the second class. A hierarchical clustering algorithm is applied to the data, which extracts the clustering structure as an intermediate result. There is no meaning associated with the generated clusters. However, our approach allows the user to visually analyze the contents of the clusters. The clustering algorithm used in the algorithmic part is independent from an application. It performs the core part of the data mining process and its result serves as a multi-purpose basis for further analysis directed by the user. This way the user may obtain novel information which was not even known to exist in the data set. This is in contrast to similarity search where the user is restricted to find similar parts respective to a query object and a predetermined similarity measure.

7.1.2 Similarity Search

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 automobile and aircraft market. These demands can only be met if the engineers have an overview of already existing CAD parts. It would be desirable to have 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 would support the user in the time-consuming task to find similar parts. From the industrial user's point of view, this browsing tool should meet the following two requirements:

- The hierarchical clustering structure of the dataset is revealed at a glance. The reachability plot is an intuitive visualization of the clustering hierarchy which helps to assign each object to its corresponding cluster or to noise. Furthermore, the hierarchical representation of the clusters using 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 density threshold ε_{cut} (cf. Figure 7.2).

- The user is not only interested in the shape and the number of the clusters, but also in the specific objects building up a cluster. As for large clusters it is rather difficult to depict all objects, representatives of each cluster should be displayed. To follow up a first idea, these representatives could be simply constructed by superimposing all parts belonging to the regarded cluster (cf. Figure 7.3). We can browse through the hierarchy of the representatives in the same way as through the OPTICS plots.

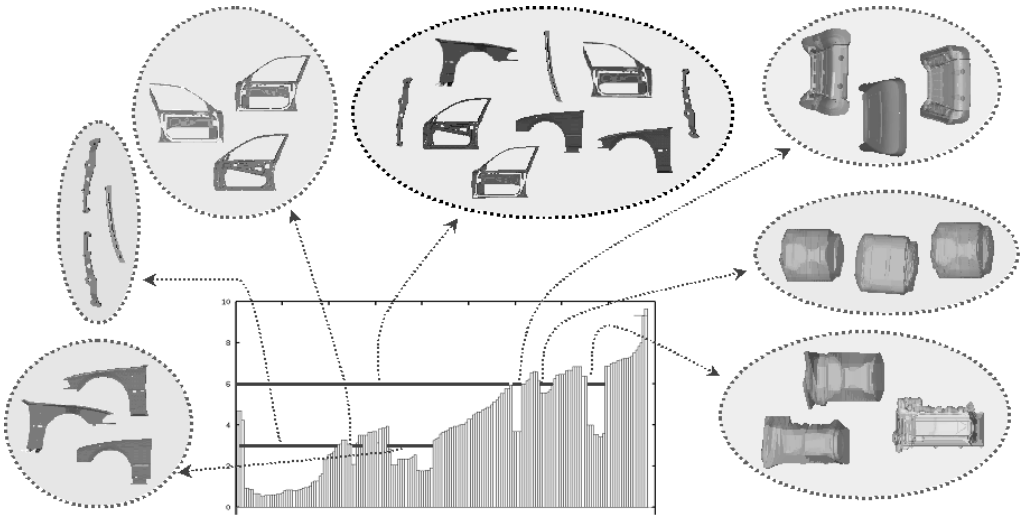


Figure 7.2: Browsing reachability plots with different density thresholds.

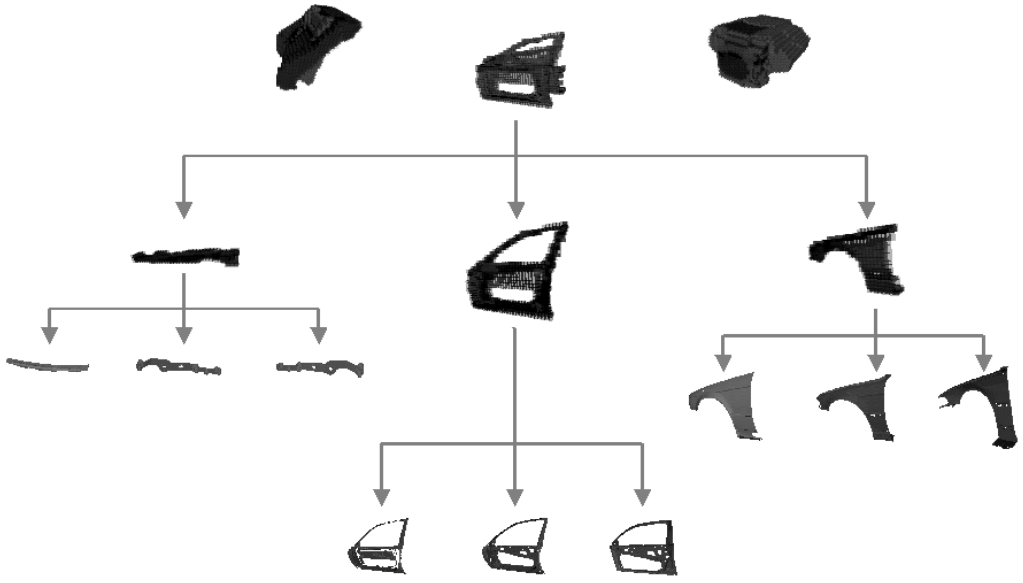


Figure 7.3: Hierarchically ordered representatives.

This way, the cost of developing and producing new parts could be reduced by maximizing the reuse of existing parts, because the user can browse 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.

7.1.3 Evaluation of Similarity Models

In general, similarity models can be evaluated by computing k-nearest neighbour queries. As shown in [KKM+03], this evaluation approach is subjective and error-prone because the quality measure of the similarity model depends on the results of a 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.

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. It is more objective since each object of the data set is taken into account to evaluate the data models.

7.2 Cluster Recognition

In this section, we address the first task of automatically extracting clusters from the reachability plots. After a brief discussion of related work in that area, we propose a new approach for hierarchical cluster recognition based on reachability plots called *Gradient Clustering*.

7.2.1 Related Work

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

The first approach proposed in [ABKS99] is called ξ-Clustering and is based on the steepness of the valleys in the reachability plot. The steepness is defined by means of an input parameter ξ. The method suffers from the fact that this input parameter is difficult to understand and hard to determine. Rather small variations of the value ξ often lead to drastic changes of the resulting clustering hierarchy. As a consequence, this method is unsuitable for our purpose of automatic cluster extraction.

The second approach was proposed by Sander et al. [SQL+03]. The authors describe an algorithm called Tree Clustering that automatically extracts a hierarchical clustering from a reachability plot and computes a cluster tree. It is based on the idea that *significant* local maxima in the reachability plot separate clusters. Two parameters are introduced to decide whether a local maximum is significant: The first parameter specifies the minimum cluster size, i.e. how many objects must be located between two significant local maxima. The second parameter specifies the ratio between the reachability of a significant local maximum m and the average reachabilities of the regions to the left and to the right of m. The authors in [SQL+03] propose to set the minimum cluster size to 0.5% of the data set size and the second parameter to 0.75. They empirically show, that this default setting approximately represents the requirements of a typical user.

Although the second method is rather suitable for automatic cluster extraction from reachability plots, it has one major drawback. Many real-world data sets consist of narrowing clusters, i.e. clusters each consisting of exactly one smaller subcluster (cf. Figure 7.4).

Since the Tree Clustering algorithm runs through a list of all local maxima (sorted in descending order of reachability) and decides at each local maxi-

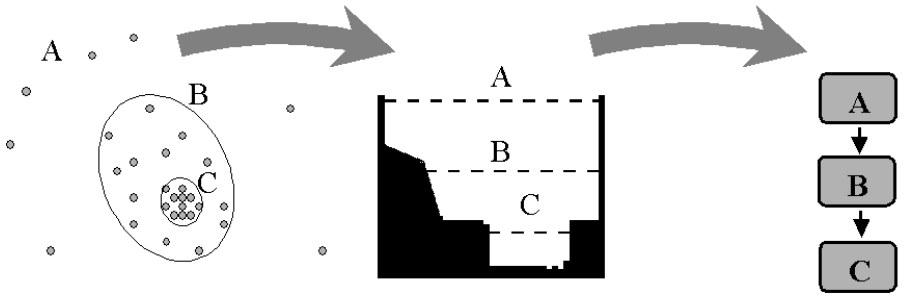


Figure 7.4: Sample narrowing clusters: data space (left), reachability plot (center), cluster hierarchy (right).

mum m, whether m is significant to split the objects to the left of m and to the right of m into two clusters, the algorithm cannot detect such narrowing clusters. These clusters cannot be split by a significant maximum. Figure 7.4 illustrates this fact. The narrowing cluster A consists of one cluster B which is itself narrowing consisting of one cluster C (the clusters are indicated by dashed lines). The Cluster Tree algorithm will only find cluster A since there are no local maxima to split clusters B and C. The ξ-Clustering will detect only one of the clusters A, B or C depending on the ξ parameter but also fails to detect the cluster hierarchy.

A new cluster recognition algorithm should meet the following requirements:

- It should detect all kinds of subclusters, including narrowing subclusters.
- It should create a clustering structure which is close to the one which an experienced user would manually extract from a given reachability plot.
- It should allow an easy integration into the OPTICS algorithm. We do not want to apply an additional cluster recognition step after the OPTICS run is completed. In contrast, the hierarchical clustering structure should be created on the fly during the OPTICS run without

causing any noteworthy additional cost.

- It should allow an easy integration into the incremental version of OPTICS presented in [KKG03], as most of the discussed application ranges benefit from such a incremental version.

7.2.2 Gradient Clustering

In this section, we introduce our new *Gradient Clustering* algorithm which fulfills all of the above mentioned requirements. The idea behind our new cluster extraction algorithm is based on the concept of *inflexion points*. During the OPTICS run, we decide for each point added to the result set, i.e. the reachability plot, whether it is an inflexion point or not. If it is an inflexion point we might be at the start or at the end of a new subcluster. We store the possible starting points of the subclusters in a list, called *StartPts*. This stack consists of pairs $(o.P, o.R)$. The Gradient Clustering algorithm can easily be intergrated into OPTICS and is described in full detail after a formal introduction of the new concept of inflexion points.

In the following, we assume that CO is a cluster ordering as defined in Definition 10. We call two objects $o_1, o_2 \in CO$ *adjacent* in CO if $o_2.P = o_1.P + 1$. Let us recall, that $o.R$ is the reachability of $o \in CO$ assigned by OPTICS while generating CO. For any two objects $o_1, o_2 \in CO$ adjacent in the cluster ordering, we can determine the gradient of the reachability values $o_1.R$ and $o_2.R$. The gradient can easily be modelled as a 2D vector where the y-axis measures the reachability values ($o_1.R$ and $o_2.R$) in the ordering, and the x-axis represent the ordering of the objects. If we assume that each object in the ordering is seperated by width w, the gradient of o_1 and o_2 is the vector

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

An example for a gradient vector of two objects x and y adjacent in a cluster ordering is depicted in Figure 7.5.

Intuitively, an inflexion point should be an object in the cluster ordering where the gradient of the reachabilities changes significantly. This significant

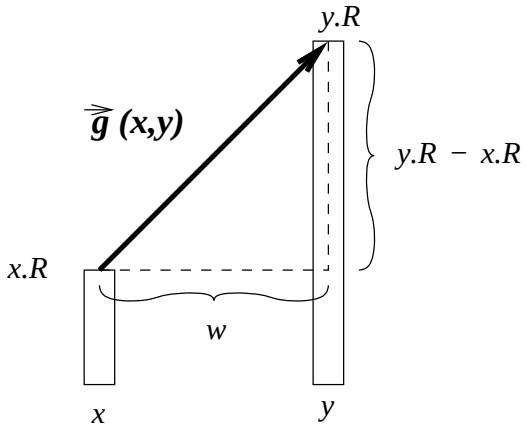


Figure 7.5: Gradient vector $\vec{g}(x, y)$ of two objects x and y adjacent in the cluster ordering.

change indicates a starting or an end point of a cluster.

Let $x, y, z \in CO$ be adjacent, i.e. $x.P + 1 = y.P = z.P - 1$. We can now measure the differences between the gradient vector $\vec{g}(x, y)$ and $\vec{g}(y, z)$ be computing the cosinus function of the angle between these two vectors. The cosinus of this angle is equal to -1 if the angle is 180°, i.e. the vectors have the same direction. On the other hand, if the gradient vectors differ a lot, the angle between them will be clearly smaller than 180° and thus the cosinus will be significantly greater than -1. This observation motivates the concepts of inflection index and inflexion points:

Definition 31 (inflexion index). Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO. The *inflexion index* of y, denoted by $II(y)$, is defined as the cosinus of the angle between the gradient vector of x, y ($\vec{g}(x, y)$) and the gradient vector of z, y ($\vec{g}(z, y)$), formally:

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

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

Definition 32 (inflexion point). Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO and let $t \in \mathbb{R}$. Object y is an *inflexion point* iff

$$II(y) > t.$$

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

Definition 33 (gradient determinant). Let CO be a cluster ordering and $x, y, z \in CO$ be objects adjacent in CO. The *gradient determinant* of the gradients $\vec{g}(x, y)$ and $\vec{g}(z, y)$ is defined as

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

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

The sign of $gd(y)$ indicates whether $y \in CO$ is a starting point or end point of a cluster. In fact, we can distinguish the following two cases which are visualized in Figure 7.6:

- $II(y) > t$ and $gd(y) > 0$:
  Object y is either a starting point of a cluster (e.g. object a in Figure 7.6) or the first object outside of a cluster (e.g. object z in Figure 7.6).
- $II(y) > t$ and $gd(y) < 0$:
  Object y is either an end point of a cluster (e.g. object n in Figure 7.6) or the second object inside a cluster (e.g. object b in Figure 7.6).

Let us note that a local maximum $m \in CO$ which is the cluster seperation point in [SQL+03] is a special form of the first case (i.e. $II(m) > t$ and $gd(m) > 0$).

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

Obviously, the gradient clustering algorithm is able to extract narrowing clusters. First experimental comparisons with the methods in [SQL+03] and [ABKS99] are presented in Section 7.6.

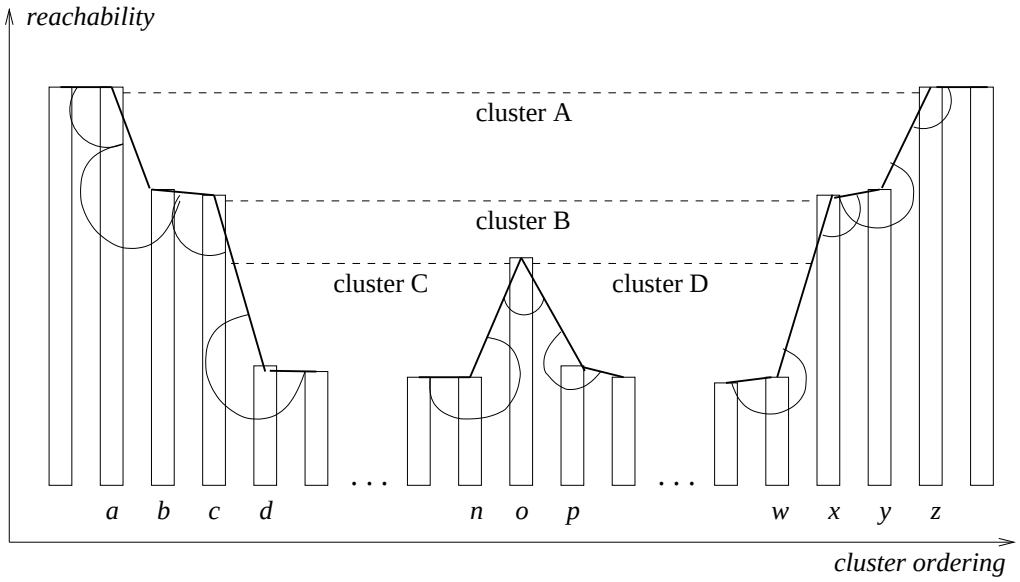


Figure 7.6: Inflexion points measuring the angle between the gradient vectors of objects adjacent in the ordering.

The pseudo code of the Gradient Clustering algorithm is depicted in Figure 7.7, which works like this. Initially, the first object of the cluster ordering CO is pushed to the stack of starting points *StartPts*. Whenever a new starting point is found, it is pushed to the stack. If the current object is an end point, a new cluster is created containing all objects between the starting point on top of the stack and the current end point. Starting points are removed from the stack if their reachablity is lower than the reachability of the current object. Clusters are created as described above for all removed starting points as well as for the starting point which remains in the stack. The input parameter *MinPts* determines the minimum cluster size and the parameter t was discussed above. Finally the parameter w influences the gradient vectors and proportionally depends on the reachability values of the objects in CO.

```
algorithm Gradient_Clustering(ClusterOrdering CO, Integer MinPts, Real t)

   StartPts := emptyStack;
   SetOfClusters := emptySet;
   CurrCluster := emptySet;

   o := CO.getFirst();                          // first object is a starting point
   StartPts.push(o);

   WHILE o.hasNext() DO                         // for all remaining objects
      o := o.next;

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

   RETURN SetOfClusters;
END. // Gradient_Clustering
```

Figure 7.7: Pseudo code of the Gradient Clustering algorithm.

7.3 Cluster Representatives

In this section, we present three different approaches to determine representative objects for clusters computed by OPTICS. A simple approach could be to superimpose all objects of a cluster to build the representative as it is depicted in Figure 7.3. However, this approach has the huge drawback that the representatives on a higher level of the cluster hierarchy become rather unclear. Therefore, we choose real objects of the data set as cluster representatives.

In the following, CO denotes the cluster ordering from which we want to extract clusters. A cluster $C \subseteq CO$ will be represented by a set of k objects of the cluster, denoted as $REP(C)$. The number of representatives k can be a user defined number or a number which depends on the size and data distribution of the cluster C.

7.3.1 The Extended Medoid Approach

Many partitioning clustering algorithms are known to use medoids as cluster representatives. The medoid of a cluster C is the closest object to the mean of all objects in C. The mean of C is also called centroid. For $k > 1$ we could choose the k closest objects to the centroid of C as representatives. The choice of medoids as cluster representative is somehow questionable. Obviously, if C is not of convex shape, the medoid is not really meaningful. An extension of this approach coping with the problems of clusters with non-convex shape is the computation of k medoids by applying a k-medoid clustering algorithm to the objects in C. The clustering using a k-medoid algorithm is rather efficient due to the expectation that the clusters are much smaller than the whole data set. This approach can also be easily extended to cluster hierarchies. At any level we can apply the k-medoid clustering algorithm to the merged set of objects from the child clusters or – due to performance reasons – merge the medoids of child clusters and apply k-medoid clustering on this merged set of medoids.

7.3.2 Minimizing the Core-Distance

The second approach to choose representative objects of hierarchical clusters uses the density-based clustering notion of OPTICS. The core-distance $o.C = \mathit{CoreDist}(o)$ of an object $o \in CO$ (cf. Definition 8) indicates the density of the surrounding region. The smaller the core-distance of o, the denser the region surrounding o. This observation led us to the choice of the object having the minimum core-distance as representative of the respective cluster. Formally, $\mathit{REP}(C)$ can be computed as

$$\mathit{REP}(C) := \{o \in C \mid \forall x \in C : o.C \leq x.C\}.$$

We choose the k objects with the minimum core-distances of the cluster as representatives. The straightforward extension for cluster hierarchies is to choose the k objects from the merged child clusters having the minimum core-distances.

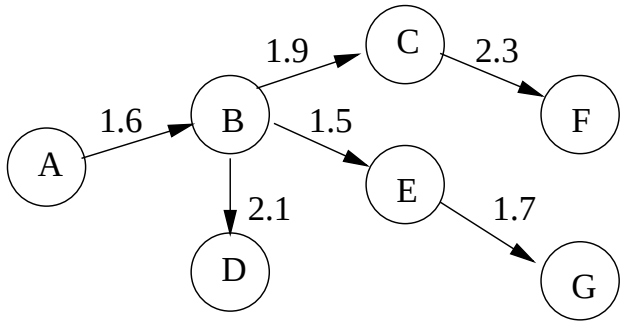
7.3.3 Maximizing the Successors

The third approach to choose representative objects of hierarchical clusters also uses the density-based clustering notion of OPTICS but in a more sophisticated way. In fact, it makes use of the density-connected relationships underlying the OPTICS algorithm.

As mentioned above, the result of OPTICS is an ordering of the database minimizing the reachability relation. At each step of the ordering, the object o having the minimum reachability w.r.t. the already processed objects occurring before o in the ordering is choosen. Thus, if the reachability of object o is not ∞, it is determined by $\mathit{ReachDist}(p, o)$ where p is an unique object located before o in the cluster ordering. We call p the *predecessor* of o, formally:

Definition 34 (predecessor). Let CO be a cluster ordering. For each entry $o \in CO$ the *predecessor* is defined as

$$\mathit{Pre}(o) = \begin{cases} p & \text{if} \quad o.R = \mathit{ReachDist}(p, o) \\ \text{UNDEFINED} & \text{if} \quad p.R = \infty. \end{cases}$$



$SIR_C(A) = 0.385$ $SIR_C(D) = 0$

$SIR_C(B) = 1.067$ $SIR_C(C) = 0.303$

Figure 7.8: Sample successor graph for a cluster of seven objects.

Intuitively, $Pre(o)$ is the object in CO from which o has been reached. Let us note, that an object and its predecessor need not to be adjacent in the cluster ordering.

Definition 35 (set of successors). Let DB be a database of objects. For each object $o \in DB$ in a cluster ordering computed by OPTICS, the *set of successors* is defined as $S(o) := \{s \in DB \mid Pre(s) = o\}$.

Let us note, that objects may have no predecessor, e.g. each object having a reachability of ∞ does not have a predecessor, including the first object in the ordering. On the other hand, some objects may have more than one successor. In that case, some other objects have no successors. Again, an object and its successors need not to be adjacent in the ordering.

We can model this succsessor relationship within each cluster as a directed *successor graph* where the nodes are the objects of one cluster and a directed edge from object o to s represents the relationship $s \in S(o)$. Each edge (x, y) can further be labeled by $ReachDist(x, y)$ $(= y.R)$. A sample successor graph is illustrated in Figure 7.8. Some of the corresponding *SIR*-values are depicted.

For the purpose of computing representatives of a cluster, the objects having many successors are interesting. Roughly speaking, these objects are

responsible for the most density-connections within a cluster. The reachability values of these "connections" further indicate the distance between the objects. For example, for the objects in the cluster visualized in Figure 7.8, object B is responsible for the most density-connections since its node in the successor graph has the most outgoing edges.

Our third strategy selects the representatives of clusters by maximizing the number of successors and minimizing the according reachabilities. For this purpose, we compute for each object o of a cluster C, the Sum of the Invers Reachability distances of the successors of o within C, denoted by $SIR_C(o)$:

$$SIR_C(o) := \begin{cases} 0 & \text{if} \quad S(o) = \emptyset \\ \sum\limits_{s \in S(o) \cap C} \frac{1}{1 + ReachDist(o,s)} & \text{otherwise.} \end{cases}$$

We add 1 to $ReachDist(o, s)$ in the denominator to weight the impact of the number of successors over the significance of the reachability values. Based on $SIR_C(o)$, the representatives can be computed as follows:

$$REP(C) := \{o \in C \mid \forall x \in C : SIR_C(o) \geq SIR_C(x)\}.$$

In Figure 7.8, the SIR-values of some objects of the depicted successor graph for a cluster of seven objects are computed. Since D has no successors, $SIR_C(D)$ is zero. In fact object B has the highest SIR-value indicating the central role of B in the cluster: B has three successors with relatively low reachability distance values. Our third strategy would select object B as representative for the cluster.

Let us note, that there is no additional overhead to compute the reachability distances $ReachDist(o, S(o))$ for each $o \in CO$ since these values have been computed by OPTICS during the generation of CO and $ReachDist(o, S(o)) = S(o).R$.

If we want to select k representatives for C we simply have to choose the k objects with the maximum SIR_C-values.

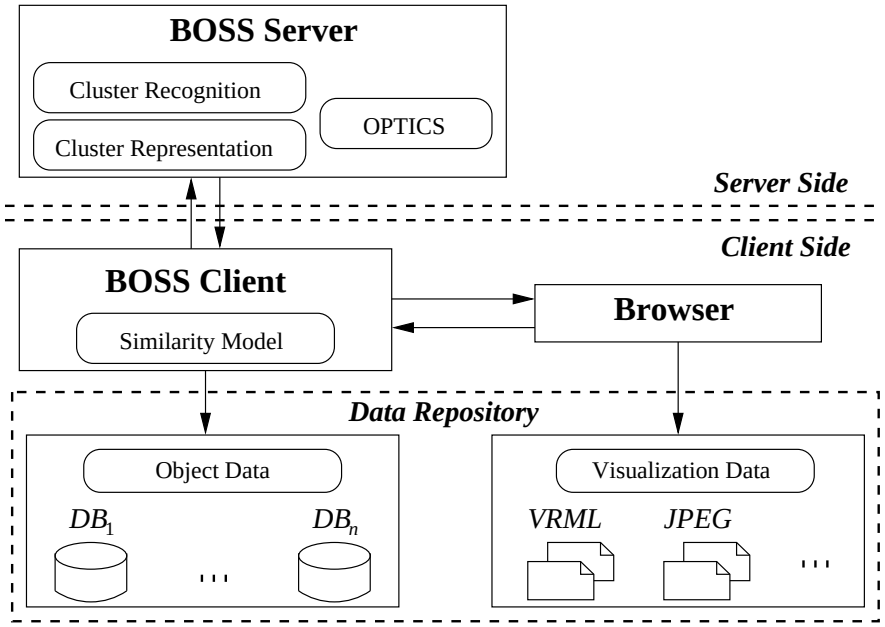


Figure 7.9: The distributed architecture of BOSS

7.4 Browsing Cluster Hierarchies

The development of the industrial prototype BOSS is a first step towards developing a comprehensive, scalable and distributed computing solution designed to make the efficiency of OPTICS and the analytical capabilities of BOSS available to a broader audience. BOSS is a client/server system allowing users to provide their own data locally, along with an appropriate similarity model (cf. Figure 7.9).

The data provided by the user will be comprised of the objects to be clustered, as well as a data set to visualize these objects, e.g. VRML files for CAD data or JPEG images for multimedia data. Since this data resides on the user's local computer and is not transmitted to the server, heavy network traffic can be avoided. In order for BOSS to be able to interpret this data, the user must supply his own similarity model with which the reachability data can be calculated.

The independence of the data processing and the data specification enables maximum flexibility. Further flexibility is introduced through the sup-

port of external visual representation. As long as the user is capable of displaying the visualization data in a browser, e.g. by means of a suitable plug-in, the browser will then load web pages generated by BOSS displaying the appropriate data. Thus, multimedia data such as images or VRML files can easily be displayed (cf. Figure 7.10). By externalizing the visualization procedure, we can resort to approved software components, which have been specifically developed for displaying objects which are of the same type as the objects within our clusters.

7.5 Visualizing Connected Object Orderings

In modern databases, complex objects like multimedia data, proteins or text objects can be modeled in a variety of representations and can be compared by a variety of distance or similarity functions. Thus, it quite often occurs that we have multiple views on the same set of data objects and do not have any intuition about how the different views on data objects agree or disagree about the similarity of objects. VICO is a tool for comparing these different views on the same set of data objects. Our system is heavily based on OPTICS. The idea of VICO is to select data objects or even complete clusters in one OPTICS plot and additionally highlight the same objects in all other displayed views on the data. A cluster order can be considered as an image of the data distribution in one representation. VICO has the following three main applications: First, if more than one distance function for a given data set is available, it allows direct comparisons of the distance functions. Second, in a multi-represented setting, where multiple feature transformations for an object are available, the relationships between the given data representations can be examined by comparing the clusterings resulting w.r.t. these representations. Third, the connection between multi-instance objects and their single instances can be examined by comparing the clustering of multi-instance objects to the clusterings w.r.t. single instances.

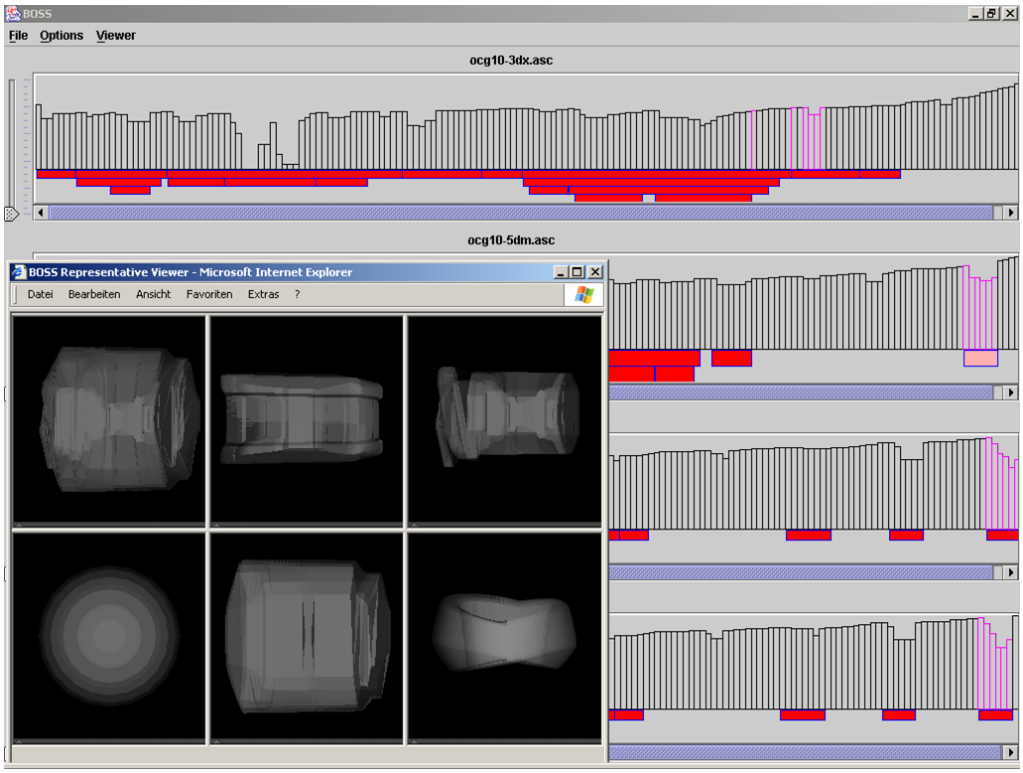


Figure 7.10: BOSS displaying contents of OPTICS clusters.

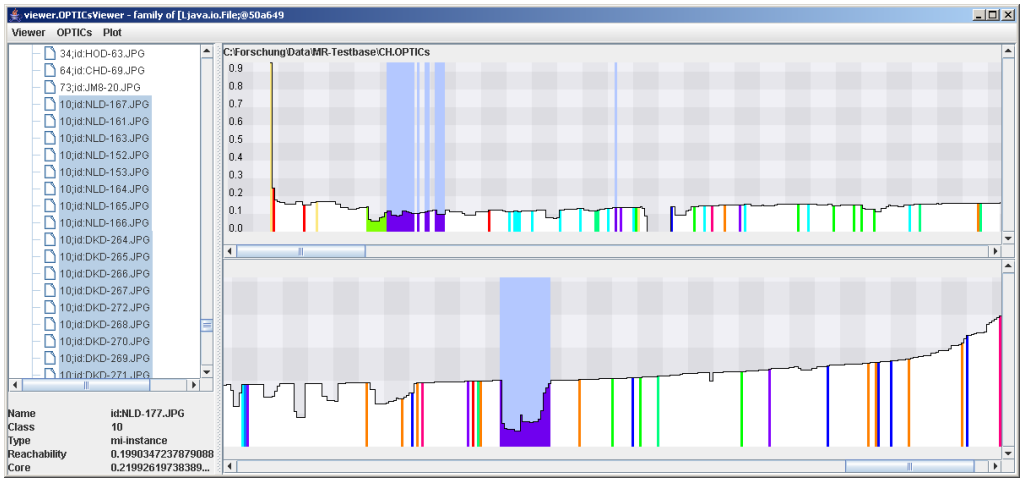


Figure 7.11: VICO displaying OPTICS plots of multi-represented data.

7.5.1 Analysis of Complex Data Spaces

The main purpose of VICO is to compare different feature spaces that describe the same set of data. For this comparison, VICO relies on the interactive visual exploration of reachability plots. Therefore, VICO displays any available view on a set of data objects as adjacent reachability plots and allows comparions between the local neighborhoods of each object. Fig. 7.11 displays the main window of VICO. The left side of the window contains a so-called tree control that contains a subtree for each view of the data set. In each subtree, the keys are ordered w.r.t. the cluster order of the corresponding view. The tree control allows a user to directly search for individual data objects. In addition to the object keys displayed in the tree control, VICO displays the reachability plot of each view of the data set.

Since valleys in the reachability plot represent clusters in the underlying representation, the user gets an instant impression of the richness of the cluster structure in each representation. However, to explore the relationships between the representations, we need to find out whether objects that are clustered in one representation are also similar in the other representation. To achieve this type of comparison, VICO allows the user to select any data object in any reachability plot or the tree control. By selecting a set of objects in one view, the objects are highlighted in any other view as well. For example, if the user looks at the reachability plot in one representation and selects a cluster within this plot, the corresponding object keys are highlighted in the tree control and identify the objects that are contained in the cluster. Let us note that it is possible to visualize the selected objects as well, as long as there is a viewable object representation. In addition to the information about which objects are clustered together, the set of objects is highlighted in the reachability plots of the other representations as well. Thus, we can easily decide whether the objects in one representation are placed within a cluster in another representation as well or if they are spread among different clusters or are part of the noise. If there exist contradicting reachability plots for the same set of data objects, it is interesting to know which of these representations is closer to the desired notion of similarity. Thus, VICO allows the user to label data objects w.r.t. some class value.

The different class values for the objects are displayed by different colors in the reachability plot. Thus, a reachability plot of a data space that matches the user's notion of similarity should display clusters containing objects of the same color. Fig. 7.11 displays a comparison of two feature spaces for an image data set. Each image is labelled with w.r.t. the displayed motive.

Another feature of VICO is the ability to handle multi-instance objects. In a multi-instance representation, one data object is given by a set of separated feature objects. An example are CAD parts that can be decomposed to a set of spatial primitives, which can be represented by a single feature vector. This way, the complete CAD part is represented by a set of feature vectors, which can be compared by a variety of distance functions. To find out which instances are responsible for clusters of multi-instance objects, VICO allows us to cluster the instances without considering the multi-instance object they belong to. Comparing this instance plot to the plot derived on the complete multi-instance objects allows us to analyze which instance clusters are typical for the clusters on the complete multi-instance object. Thus, for multi-instance settings, VICO highlights all instances belonging to some selected multi-instance object.

7.5.2 Architecture and Implementation

VICO is implemented in Java 1.5 and thus, runs on any platform supporting the current version of the Java Runtime Environment. VICO includes an integrated version of OPTICS allowing the user to load and cluster data sets described in a variety of file formats like CSV and ARFF files. For this version of OPTICS there are several distance measures already implemented like the Euclidian, Manhattan or Cosine distance. Furthermore, VICO already implements various distance functions for multi-instance objects, e.g. the Hausdorff distance, the Sum of Minimum Distances and the Minimal Matching Distance. The system is based on an extensible architecture, so that additional components like new distance functions can be integrated easily by implementing Java interfaces.

7.6 Experimental Evaluation

We evaluated both the effectiveness and efficiency of our approaches using two real-world test data sets. The first one contains approximately 200 CAD objects from a German car manufacturer, and the second one is a sample of the Protein Databank [BWF+00] containing approximately 5000 protein structures. We tested on a workstation featuring a 1.7 GHz CPU and 2 GB RAM. In the following, three cluster recognition algorithms as well as three approaches for generating cluster representatives are evaluated.

7.6.1 Cluster Recognition

Automatic cluster recognition is clearly very desirable when analyzing large sets of data. In this section, we will first discuss the quality of our three cluster recognition algorithms. For this evaluation we use the Car and the Protein dataset. Secondly, we discuss the efficiency by using the Car and the Plane data set.

Effectivity

Both the Car and the Protein data set exhibit the commonly seen quality of unpronounced but nevertheless to the observer clearly visible clusters. The corresponding reachability plots of the two data sets are depicted in Figure 7.12.

Figure 7.12(c) shows that the Tree Clustering algorithm does not find any clusters at all in the Car data set, with the suggested default ratio parameter of 75% [SQL+03]. In order to detect clusters in the CAR data set, we had to adjust the ratio parameter to 95%. In this case Tree Clustering detected some clusters but missed out on some other important clusters and did not detect any cluster hierarchies at all. If we have rather high reachability values, e.g. values between 5 and 7 as in Figure 7.12 for the Car data set, the ratio parameter for the Tree Clustering algorithm should be higher than for smaller values. In the case of the Protein data set we detected three clusters

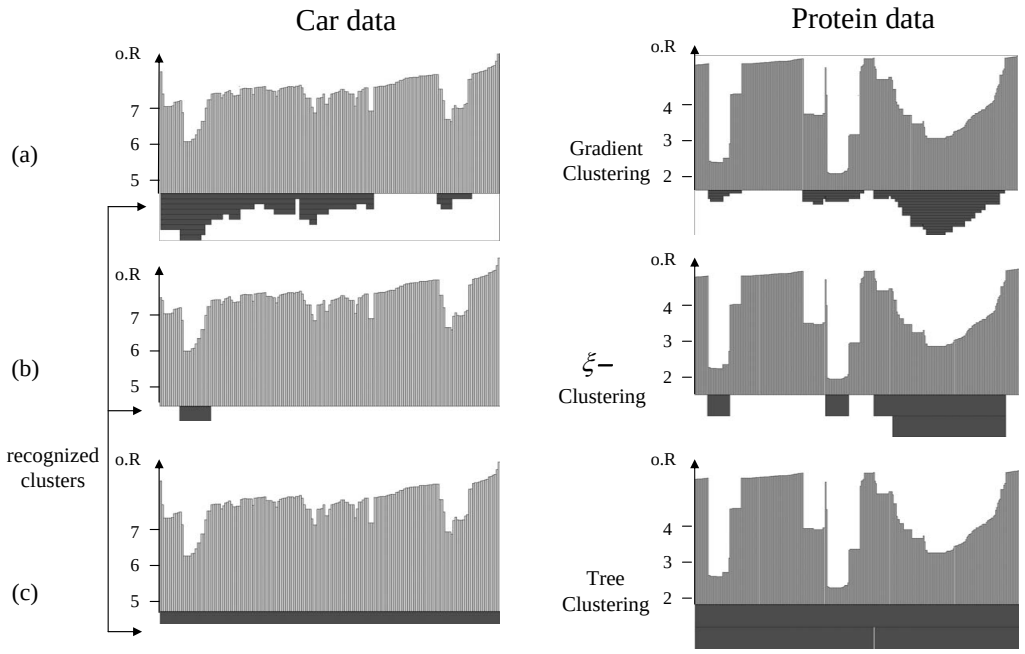


Figure 7.12: Sample clusters of car parts.

with the default parameter setting, but again missed out on some important clusters. Generally, in cases where a reachability graph consists of rather high reachability values or does not present spikes at all, but clusters are formed by smooth troughs in the waveform, this cluster recognition algorithm is unsuitable. Furthermore, it is inherently unable to detect narrowing clusters where a cluster has one subcluster of increased density (cf. Figure 7.4).

On the other hand, the ξ-Clustering approach successfully recognizes some clusters while also missing out on significant subclusters (cf. Figure 7.12(b)). This algorithm has some trouble recognizing cluster structures with a significant differential of "steepness". For instance, in Figure 7.4 it does not detect the narrowing cluster B inside of cluster A because it tries to create steep down-areas containing as many points as possible. Thus, it will merge the two steep edges if their steepness exceeds the threshold ξ. On the other, it is able to detect cluster C within A.

Finally, we look at our new Gradient Clustering algorithm. Figure 7.12(a) shows that the recognized cluster structure is close to the intuitive one, which

Table 7.1: CPU time for cluster recognition.

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

an experienced user would manually derive. Clusters which are clearly distinguishable and contain more than *MinPts* elements are detected by this algorithm. Not only does it detect a lot of clusters, but it also detects a lot of meaningful cluster hierarchies, consisting of narrowing subclusters.

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

Efficiency

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

The overall runtimes for the three different cluster recognition algorithms are depicted in Table 7.1. Our new Gradient Clustering algorithm does not only produce the most meaningful results, but also in sufficiently short time. This is due to its runtime complexity of $O(n)$.

7.6.2 Cluster Representation

After a cluster recognition algorithm has analyzed the data, algorithms for cluster representation can help to get a quick visual overview of the data. With the help of representatives, large sets of objects may be characterized

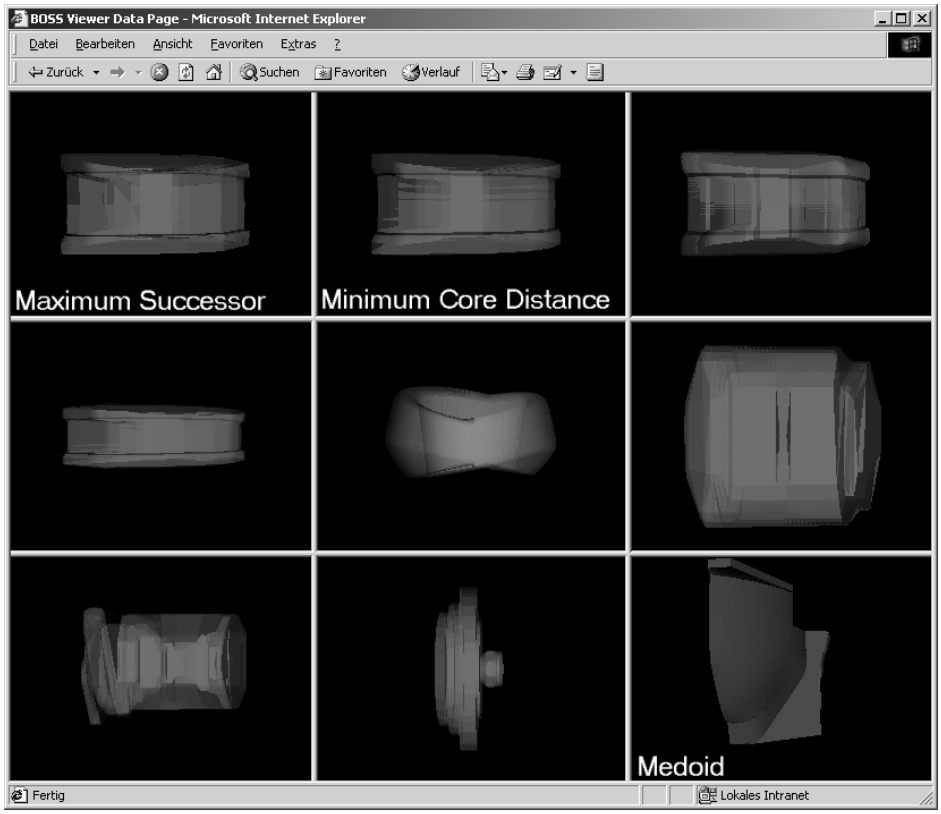


Figure 7.13: A cluster of CAD objects with representative objects.

through a single object of the data set. We extract sample clusters from both data sets in order to evaluate the different approaches for cluster representatives. In our first tests, we set the number of representatives to $k = 1$.

The objects of one cluster from the car data set are displayed in Figure 7.13 and the objects of one cluster from the protein data set are displayed in Figure 7.14. The annotated objects are the representatives computed by the respective algorithms. Both the *Maximum Successor* and the *Minimum Core Distance* approaches give good results. Despite the slight inhomogeneity of the clusters, both representatives sum up the majority of the elements within both clusters. This cannot be said of the representatives computed by the commonly used medoid method, which selects objects from the trailing end of the cluster. These two clusters and their corresponding representatives are no isolated cases, but reflect our general observations. Nevertheless, there have

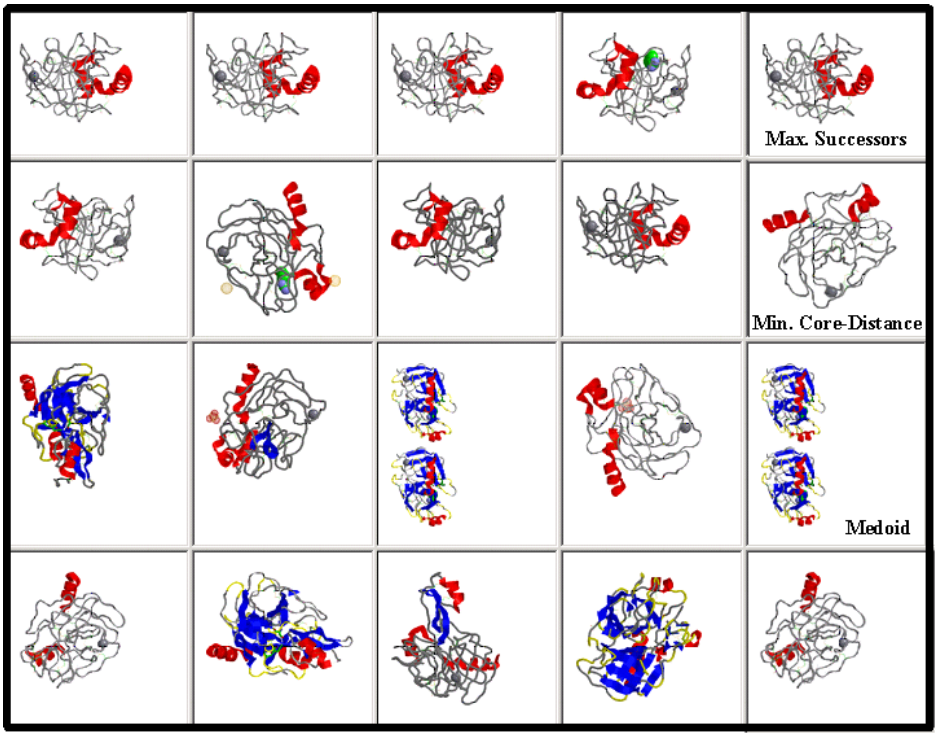


Figure 7.14: A cluster of proteins with representative objects.

been some rare cases where the medoid approach yielded the more intuitive representative than the other two approaches.

If we allow a higher number of representatives, for instance $k = 3$, it might be better to display the representatives of all three approaches to reflect the content of the cluster, instead of displaying the three best representatives of one single approach. If we want to confine ourselves to only one representative per cluster, the best possible choice is to use the representative of the Maximum Successor approach.

7.6.3 Discussion

The results of our experiments show, that our new approaches for the automatic cluster extraction and for the determination of representative objects outperform existing methods. It theoretically and empirically turned out, that the Gradient Clustering algorithm seems to be more practical than pre-

vious work for automatic cluster extraction from hierarchical cluster representations. We also empirically showed that our approaches for the determination of cluster representatives is in general more suitable than the simple (extended) medoid approach.

7.7 Summary

In this chapter, we proposed hierarchical clustering combined with automatic cluster recognition and selection of representatives as a promising visualization technique. Its areas of application include visual data mining, similarity search and evaluation of similarity models. We surveyed three approaches for automatic extraction of clusters. The first method, ξ-Clustering, fails to detect some clusters present in the clustering structure and suffers from the sensitivity concerning the choice of its input parameter. Tree Clustering is by design unsuitable in the presence of narrowing clusters. To overcome these shortcomings, we proposed a new method, called Gradient Clustering. The experimental evaluation showed that this algorithm is able to extract narrowing clusters. Furthermore, it can easily be integrated into the hierarchical clustering algorithm. Thus, it produces no noteworthy overhead. The cluster hierarchies produced by Gradient Clustering are similar to the clustering structures which an experienced user would manually extract. Furthermore, we presented three different approaches to determine representative objects for clusters. The commonly known medoid approach is shown to be questionable for real-world data, while the approaches minimizing the core-distance and maximizing the successors both deliver good results.

Chapter 8

Hierarchical Music Genre Classification

The progress of computer hardware and software technology in recent years made it possible to manage large collections of digital music on an average desktop computer. Thus, modern computer systems are able to compress a piece of music to a few megabytes in very fast time. Easy to use software that automates this process is available. Often, this software stores meta information, such as artist, album or title, along with the audio file. However, the amount and quality of the available meta information in publicly accessible online databases, e.g. freedb.org, is often limited. This meta data is especially useful when searching for a specific piece of music in a large collection. To organize and structure a collection, additional information such as the genre would be very useful. Unfortunately, the genre information stored in online databases is often incorrect or does not meet the user's expectations.

In this chapter, a content-based hierarchical genre classification framework for digitized audio is presented as sketched in Figure 8.1. It is often problematic to assign a piece of music to exactly one class in a natural way. Genre assignment is a somewhat fuzzy concept and depends on the taste of the user. Therefore, our approach allows multi-assignment of one song to several classes. The classification is based on feature vectors obtained from three acoustic realms namely *timbre*, *rhythm* and *pitch*. Thus, each song

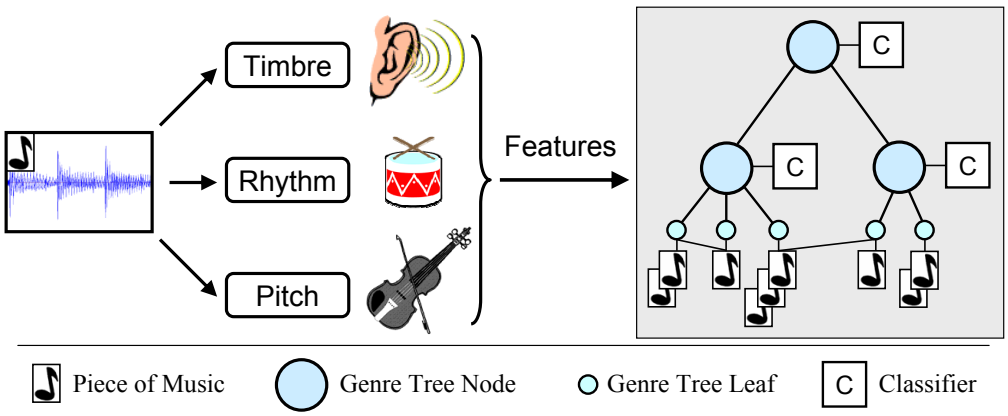


Figure 8.1: Architecture of the genre classification framework.

is described by multiple representations, each of them containing a set of feature vectors, so called *multiple instances*.

Our main contributions are:

1. A novel semi-supervised, hierarchical *instance reduction* (IR) technique which enables us to use only a small number of relevant features for each classifier.

2. An effective and efficient framework for *hierarchical genre classification* (HGC) of music pieces in a *multi-representation* (MR) and *multi-instance* (MI) setting. Let us note that our framework can also be used for *genre classification* (GC) in flat class systems.

3. A powerful prototype implementing the proposed framework. The tool features a graphical user interface to enable the user to easily analyze large collections of digitized music.

8.1 Related Work

Feature extraction. Timbre features are derived from the frequency domain and were mainly developed for the purpose of speech recognition. The extraction of the timbral texture is performed by computing the short time

fourier transform. We use the Mel-frequency cepstral coefficients (MFCCs), spectral flux and spectral rolloff as timbral representations [TC02]. Rhythmic content features are useful for describing the beat frequency and beat strength of a piece of music. In our framework, we use features derived from beat histograms [TC02] as the description of the rhythmic content. Pitch extraction tries to model the human perception by simulating the behavior of the cochlea. Similar to the rhythmic content features, we derive pitch features from pitch histograms which were generated by a multipitch analysis model [TK00].

Genre classification. The general idea of hierarchical classification is that a classifier located on an inner node of the genre tree solves only a small classification problem and therefore achieves more effective results more efficiently than a classifier that works on a large number of flat organized classes. There exist only a few approaches for automatic genre classification of audio data. In [CVJK04], music pieces are classified into either rock or classic using k-nearest neighbor and MLP classifiers. Zhang [Zha03] proposes a method for a hierarchical genre classification which follows a fixed schema and where is only limited support for user-created genre folders. Moreover, the above mentioned hierarchical classification methods do not take full advantage of MI and MR music objects. In contrast, our approach handles such rich object representations as well as an arbitrary genre hierarchy, and supports multi-assignment of songs to classes.

Hierarchical Classification. The use of class hierarchies to improve large scale classification problems has predominantly been applied in text classification. Several approaches have been introduced picking up this idea. The authors of [KKPS04b] investigated multiple representations of objects in the context of hierarchical classification and proposed a so called *object adjusted weighting* for linear combination of MR objects.

Support Vector Machines. In recent years, *support vector machines* (SVMs) [CV95] have received much attention offering superior performance in various applications. For example, [WLCS04] presents a fusion technique for multimodal objects. Basic SVMs distinguish between two classes by calculating the maximum margin hyperplane between the training examples

of both given classes. To employ SVMs for distinguishing more than two classes, several approaches were introduced [PCST99]. In order to handle sets of feature vectors in SVMs so called kernel functions were introduced [GFKS02]. A weakness of MI kernels is the need to calculate distances between all instances, i.e. $O(n^2)$ single distance calculations are required in order to compare two MI objects with n instances. Thus, MI kernels seem to be unsuitable for solving large scale classification problems in music collections.

Instance Reduction Techniques. As mentioned above, a piece of music is usually described by a set of feature vectors and is an MI object. The number of instances can vary from tens to hundreds per second, i.e. a song is represented by 10,000 to 50,000 feature vectors. In order to handle such MI objects two classes of IR techniques can be distinguished, namely higher-order and first order. Higher-order IR techniques use optimization algorithms on feature vectors. They describe an MI object as a mix of statistical distributions or cluster representatives. In [GGM02], a higher-order instance technique is presented which is based on Gaussian distributions. The authors use methods such as Expectation Maximization for parameter estimation. The authors of [CSL99] propose an IR approach that computes the optimal representatives by minimizing the Hausdorff distance between the original object and its representation. If the Euclidian metric is used as a distance function on the feature vectors, the k-means method can be applied for summarization of multimedia content [ZRHM98]. In case of general metric spaces, the k-medoid method can be applied for summarization. A randomized first order IR technique, called signature, is proposed in [CZ02]. A multimedia sequence in the database is described by selecting a number of its instances closest to a set of random vectors. The authors in [CZ02] also propose a specialized distance function on the derived first order summarization vectors. Both first and higher-order techniques reduce the MI object to a small set of feature vectors. Thus, using the reduced representations of the MI object requires the application of kernel functions for SVMs. In context of large databases, the use of kernel functions seems impracticable for efficient classification.

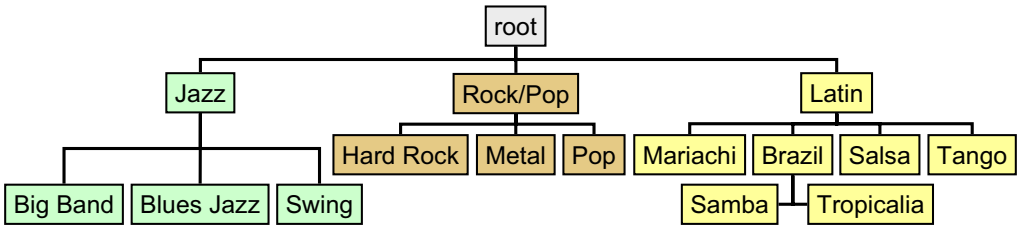


Figure 8.2: An example genre hierarchy.

8.2 Efficient Hierarchical Genre Classification

In this section, we describe our approach for classifying large collections of music pieces in a genre taxonomy (cf. Figure 8.2). Since a music piece is described by a set of feature vectors, we first describe a novel hierarchical semi-supervised technique for instance reduction. The reduced descriptions are used afterwards for hierarchical classification of music pieces with SVMs. Furthermore, we use object adjusted weighting in order to take advantage from multiple representations.

Hierarchical Instance Reduction. Let DB be a set of music objects. We argue that an MI object $X = \{x_1, \ldots, x_n\} \in DB$ can be described by a vector $X_{reduced}$ containing minimal distances to a given set of so called *support objects* $S = \{s_1, \ldots, s_m\}$ where $m \ll n$. Formally,

$$X_{reduced} = (\min_{x_i \in X} dist(x_i, s_1), \ldots, \min_{x_i \in X} dist(x_i, s_m)).$$

The set S can either be calculated by a random selection of m instances from DB, or it is possible to choose each $s_i \in S$ as a centroid of a clustering that can be calculated on a small sample of instances from DB. An example for the instance reduction is illustrated in Figure 8.3.

The number of elements in $X_{reduced}$ may still be too large for solving the classification problem efficiently. Thus, we propose to exploit the hierarchical organization of classes and to select only a small subset $S_N \subseteq S$ for each inner node N of the genre taxonomy. The elements of S_N should be selected so that the subclasses C_N of N can be distinguished in the best possible way. Therefore, the subset of support objects is individual for each inner node N.

To calculate S_N we suggest to apply a semi-supervised method based on

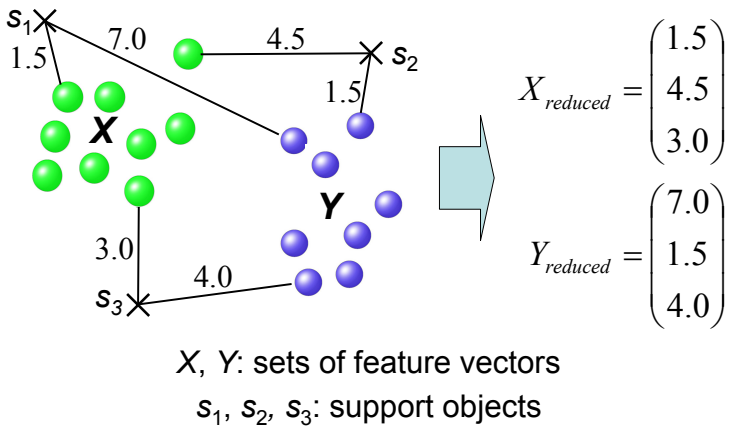


Figure 8.3: Instance reduction with help of support objects.

the *information gain criterion*. Let $T(C_N)$ be a set of all training objects belonging to C_N. The domains $D(s_i)$ are discretized by using the method described in [FI92]. After discretization the information gain criterion for each attribute can by calculated by

$$InfoGain(s_i, T(C_N)) = H(T(C_N)) - \sum_{t \in T(C_N)} \frac{|t|}{|T(C_N)|} \cdot H(t),$$

where $H(t)$ denotes the entropy. Finally, S_N is determined as the smallest set that contains k elements and for which the following condition holds:

$$\forall s_j \in S_N \forall a \in S : InfoGain(a, T(C_N)) \leq InfoGain(s_j, T(C_N)).$$

After that, S_N is used for training and classification on the node N.

Hierarchical Genre Classification by Using Multiple Representations. A *two layer classification process* (2LCP) handles the hierarchical classification problem on each inner node N of the genre taxonomy. This process acts as a guidepost for the hierarchical classification. We train SVMs in the first layer of the 2LCP that distinguishes only single classes C_{single} in each representation. Since standard SVMs are able to make only binary decisions we apply the so-called one-versus-one (OvO) approach (cf. Figure 8.4) in order to make a classification decision for more than two classes. We argue that for our application the OvO approach is best suitable because the voting vectors Φ_i provided by this method are a meaningful intermediate

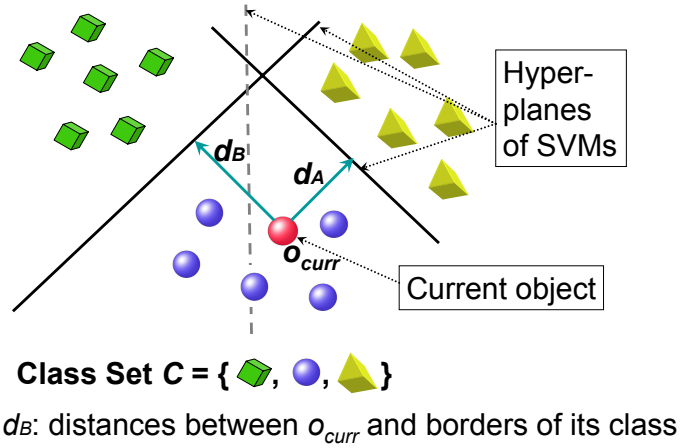


Figure 8.4: Border distance based derivation of weights for a multi-represented object.

description that is useful for solving the multi-assignment problem in the second layer of our 2LCP. In order to perform the multi-assignment we take advantage of the class properties in our application domain. We limit the possible class combinations to a subset $C_{combi} \subset 2^{C_{single}}$ because there exist several combinations that do not make sense, e.g. a piece of music belonging to the class 'salsa' is very implausible to be also in the class 'metal'. For this purpose, we only take those $c \in 2^{C_{single}}$ into account, which occur in the training set.

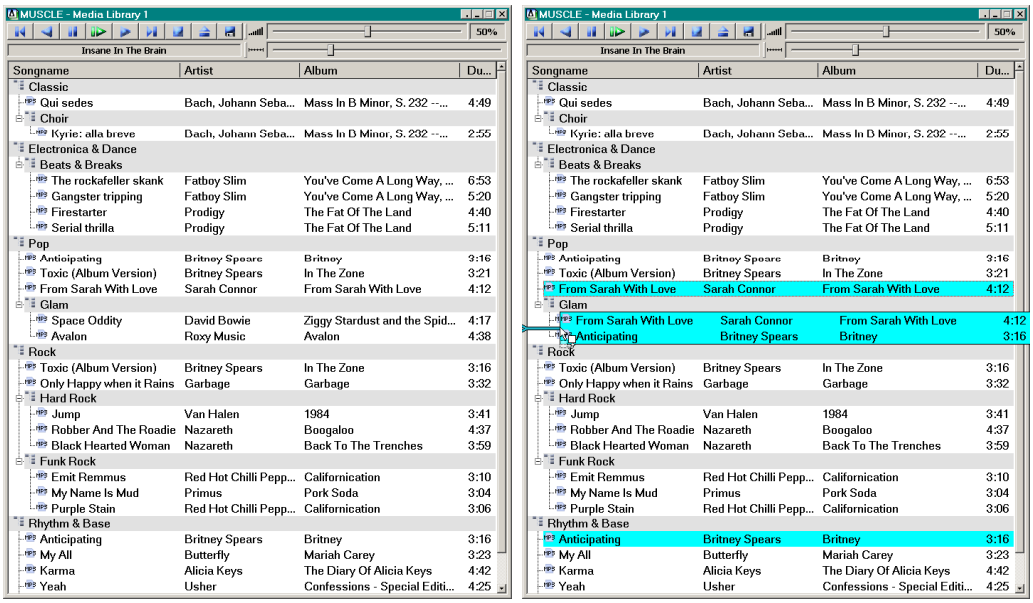
The SVM classifier in the second layer of the 2LPC uses an aggregation of the voting vectors Φ_i from the first layer of the 2LPC as input to assign an object to a class $c \in C_N = C_{single} \cup C_{combi}$. The second task that is handled by the classifier in the second layer is the aggregation of multiple representations. The voting vectors $\Phi_1, \ldots, \Phi_k$ provided by the first layer SVMs for each representation $R_1, \ldots, R_k \in R$ are aggregated by using a weighted linear combination $V = \sum_{i=1}^{k} \omega_i \Phi_i$. Then V is used as the input for the classifier in the second layer. The weights ω_i in the combination are calculated by using object adjusted weighting. The intuition behind the object adjusted weighting is that the current object o_{curr} used in training or to be classified needs to have a sufficient distance from any of the other classes. More formally, let c_j be the class of o_{curr} determined by majority

vote in Φ_i, then $\omega_i = \min_{c_i \in C_{single} \wedge c_i \neq c_j} dist(o_{curr}, HyperPlane(c_j, c_i))$, where $HyperPlane(c_j, c_i)$ denotes the maximum margin hyperplane separating the classes c_j and c_i. Figure 8.4 depicts an example of weight calculation where the weight ω should be set to d_A.

8.3 Practical Music Classification with User Feedback

To provide a powerful tool for the analysis of digitized audio, we integrated our approach to hierarchical genre classification into a graphical prototype called *MUSCLE* (*Mus*ic *Cl*assification *E*ngine with User Feedback). The installation archive of MUSCLE contains a default genre taxonomy including the necessary training data in the form of feature vectors for each song. Using aggregated information such as feature vectors makes it possible to share the training data without having to distribute the underlying music data. Classes and training data in the genre taxonomy can be deleted, moved or added by the user. When the user commits the changes of the class hierarchy or of the corresponding training data, MUSCLE trains the affected classifiers. Note that usually only a small subset of the entire classifier hierarchy has to be trained because a modification at a node requires a partial adaptation of the node and all parent nodes only. It is also possible to start the training automatically after each modification or to run the training in the background. When the user is satisfied with the training setup, a folder to automatically classify all contained songs can be selected. MUSCLE is implemented in C/C++ and runs on the Windows platform.

Fig. 8.5 illustrates MUSCLE's user interface. In the main window the playlist containing the classification result in form of a genre tree is displayed. An example for a multiple assignment of the song 'Anticipating' to the classes 'pop' and 'rhythm & base' can be seen in Fig. 8.5(a). In case the user wants to manually adjust the genre assignment of a song, entries can be rearranged using drag & drop as shown in Fig. 8.5(b).



(a) Multi-Assignment of Songs **(b)** User Feedback

Figure 8.5: The MUSCLE User Interface

8.4 Experimental Evaluation

We implemented the classification framework in Java 1.5 and performed all experiments on a Pentium IV workstation equipped with 2 GByte main memory. The genre hierarchy depicted in Figure 8.2 was used in all following experiments. A music collection consisting of almost 500 songs was the basis for the classification experiments, which results in approximately 30 songs per class. Depending on the representation, we extracted 30 to 200 features per second. We performed 10-fold cross-validation for evaluating the classification accuracy. In the following, we present the results of our experiments with particular emphasis to efficiency and effectiveness.

Effectiveness. In the first experiment, we compared the quality of GC on multiple, and HGC on single and multiple representations. Figure 8.6 depicts the experimental results. When working with multiple representations, our HGC approach (70.03%) achieves higher classification accuracy than using a single representation only. Furthermore, the classification accuracy of HGC is comparable to that of the flat GC approach (72.01%).

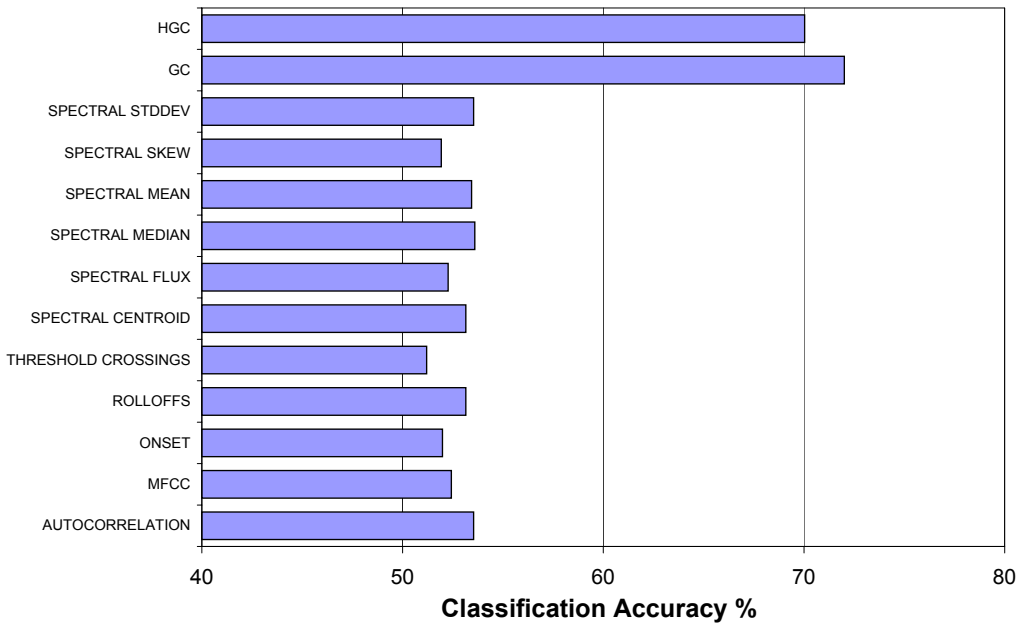


Figure 8.6: Classification accuracy on single- and multi-representations.

In the next experiment, we investigated how the classification accuracy of our approach is influenced by the number and the choice of the support objects. For choosing S_N, we either randomly picked the support objects or applied our strategy described in Section 8.2. The experimental results are depicted in Figure 8.7 and show that our approach always outperforms the random selection. For both approaches, the accuracy increases with an increasing number of support objects. However, especially for a low number of support objects, the random approach achieves a lower accuracy compared to our method. For a high number of support objects, both approaches yield a similar classification accuracy.

Efficiency. In a last experiment, we examined the runtime performance of GC and HCG for a varying number of support objects. As depicted in Figure 8.8, the runtime increases with an increasing number of support objects. The higher the number of support objects, the larger the runtime difference. Altogether, our approach achieves a good trade-off between the quality of the result and the required runtime when using 300 support objects.

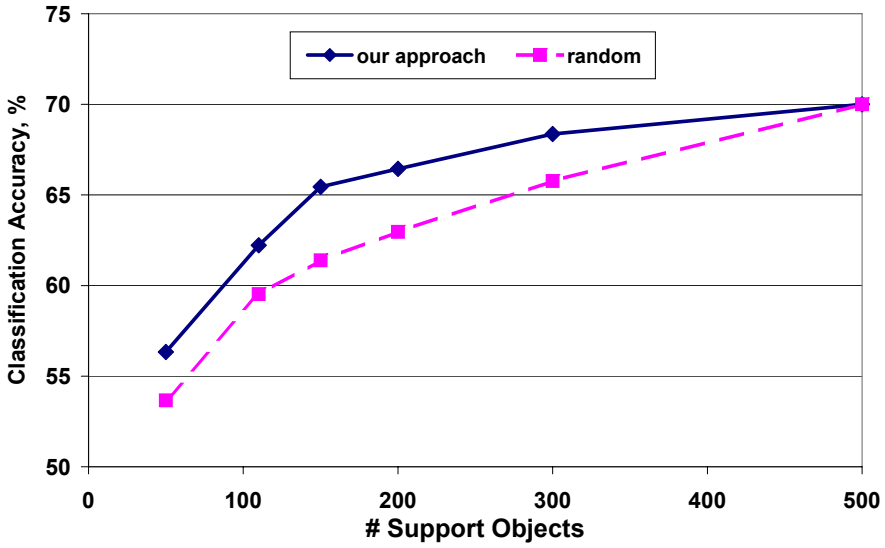


Figure 8.7: Classification accuracy on single- and multi-representations.

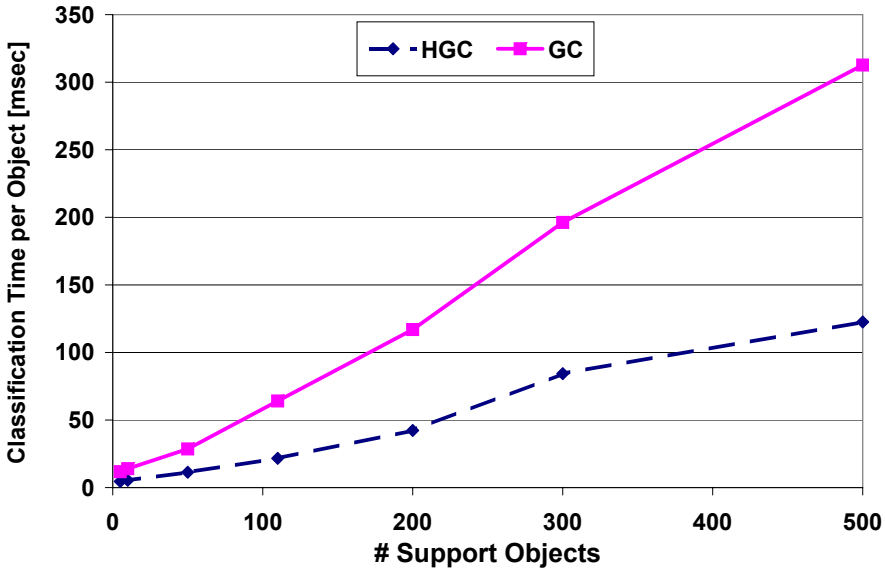


Figure 8.8: Classification time per object.

8.5 Summary

In this chapter, we introduced a framework for hierarchical music classification using multiple representations consisting of multiple instances. We showed that our hierarchical classification can compete with a flat class system in terms of effectiveness and greatly surpasses it in terms of efficiency.

Part IV

Conclusions

Chapter 9

Summary and Outlook

This chapter concludes the thesis by a summary of the theoretical and practical results. After a description of the main contributions in Section 9.1, we give an outlook on the potentials and future work in the area of similarity search in complex data in Section 9.2.

In this work, we presented our research on efficient and effective similarity search in large databases of complex objects. We started with an analysis of important challenges for modern database systems. One of those challenges is the necessity to support complex, internally structured data, which is founded in the growing importance of database systems as knowledge bases. Sets of feature vectors, trees and graphs are natural models for such complex data objects and, therefore, were the main topic of this thesis.

Another important challenge for modern database systems is the growing demand for new methods to extract knowledge stored in databases. This task is usually called knowledge discovery in databases. Many knowledge discovery problems, like clustering, outlier detection or classification, are based on some notion of similarity. This makes similarity search in databases an important basic technology.

Finally, the size of databases in science and industry is rapidly growing and the growth rate is often higher than the increase in computing power. Consequently, the efficiency of search methods gains more and more importance.

9.1 Summary of Contributions

Part I illustrates the topic and the background of this work. After a very general introduction to KDD, data mining, and clustering, we investigate the major aspects of similarity search in large databases of complex data. The density-based clustering notion underlying the algorithms DBSCAN and OPTICS is reviewed in more detail.

Part II motivates the use of vector sets for similarity modeling. For this purpose, a metric distance function is defined, which is suitable for various application ranges, but time-consuming to compute. Therefore, a filter refinement technology is suggested to efficiently process range queries and k-nearest neighbor queries, two basic query types within the field of similarity search. Several filter distances are presented, which approximate the exact object distance and can be computed efficiently. Moreover, a multi-step query processing approach is described, which can be directly integrated into the well-known density-based clustering algorithms DBSCAN and OPTICS. In addition, an extended parallel version of DBSCAN is presented which is also based on the aforementioned multi-step approach.

In Part III, new application ranges for density-based hierarchical clustering using OPTICS are discussed. Prototype systems for density-based data analysis are introduced, which have been developed for these new application areas and are based on the aforementioned similarity models and accelerated clustering algorithms for complex objects. The prototypes BOSS and VICO facilitate interactive semi-automatic cluster analysis and allows visual search for similar objects in multimedia databases. This way, the user can navigate through large datasets comfortably and can compare different views on complex data spaces, which occur in the presence of different distance functions defined in a data space as well as in a multi-represented or multi-instance setting. Finally, a novel framework for efficient and effective hierarchical genre classification for large music collections is introduced along with a prototype called MUSCLE which implements the framework.

9.2 Future Work

At the end of this thesis, let us emphasize the potentials of the proposed methods for future research.

In our opinion, other data mining algorithms besides DBSCAN and OPTICS, which have to deal with complex object representations, would likewise benefit from a direct integration of the multi-step query processing paradigm. For example an extended version of k-mediod clustering incorporating filter technology would be a valuable addition.

In many real-world databases, the data objects are distributed over several sites. A parallel and/or distributed version of OPTICS may be required since a centralized clustering could be impossible due to network bandwidth constraints. This would be the first step towards a data analysis system for a distributed database environment.

To improve the proposed data analysis tools, a quality measure for the representatives displayed in the browsable hierarchy is needed. Such a quality measure could be based on the concept of local outlier detection. Having such a quality measure at hand, we could compare the generated representatives and present a ranked list of representatives to the user.

The proposed framework for hierarchical genre classification for music collections could be extended to handle more types of multimedia data, as for example video data.

List of Figures

List of Tables

List of Definitions

References

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

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami. Efficient Similarity Search in Sequence Databases. In *Proc. 4th International Conference on Foundations of Data Organization and Algorithms (FODO'93), Chicago, IL, USA*, pages 69–84, 1993.

[AKKS99a] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl. 3D Shape Histograms for Similarity Search and Classification in Spatial Databases. In *Proc. 6th International Symposium on Large Spatial Databases (SSD'99), Hong Kong, China*, pages 207–226, 1999.

[AKKS99b] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl. Nearest Neighbor Classification in 3D Protein Databases. In *Proc. 7th International Conference on Intelligent Systems for Molecular Biology (ISMB'99), Heidelberg, Germany*, pages 34–43, 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. 21st International*

Conference on Very Large Databases (VLDB'95), Zurich, Switzerland, pages 490–501, 1995.

[Ank00] M. Ankerst. *Visual Data Mining*. PhD thesis, Institute for Computer Science, University of Munich, 2000.

[BBBK00] C. Böhm, B. Braunmüller, M. Breunig, and H.-P. Kriegel. High Performance Clustering Based on the Similarity Join. In *Proc. 9th International Conference on Information and Knowledge Management (CIKM'00), Washington, DC, USA*, pages 298–313, 2000.

[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. 16th International Conference on Data Engineering (ICDE'00), San Diego, CA, USA*, pages 577–588, 2000.

[BEKS00] B. Braunmüller, M. Ester, H.-P. Kriegel, and J. Sander. Efficiently Supporting Multiple Similarity Queries for Mining in Metric Databases. In *Proc. 16th International Conference on Data Engineering (ICDE'00), San Diego, CA, USA*, pages 256–267, 2000.

[BK97] S. Berchtold and H.-P. Kriegel. S3: Similarity Search in CAD Database Systems. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'97), Tucson, AZ, USA*, 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. 22nd International Conference on Very Large Databases (VLDB'96), Mumbai, India*, pages 28–39, 1996.

[BKK+04] S. Brecheisen, H.-P. Kriegel, P. Kröger, M. Pfeifle, M. Pötke, and M. Viermetz. BOSS: Browsing OPTICS-Plots for Similarity Search. In *Proc. 20th International Conference on Data Engineering (ICDE'04), Boston, MA, USA*, page 858, 2004.

[BKK+06] S. Brecheisen, H.-P. Kriegel, P. Kunath, A. Pryakhin, and F. Vorberger. MUSCLE: Music Classification Engine with User Feedback. In *Proc. 11th International Conf on Extending Database Technology (EDBT'06), Munich, Germany*, pages 1164–1167, 2006.

[BKKP04] S. Brecheisen, H.-P. Kriegel, P. Kröger, and M. Pfeifle. Visually Mining Through Cluster Hierarchies. In *Proc. SIAM International Conference on Data Mining (SDM'04), Lake Buena Vista, FL, USA*, pages 400–412, 2004.

[BKKP06] S. Brecheisen, H.-P. Kriegel, P. Kunath, and A. Pryakhin. Hierarchical Genre Classification for Large Music Collections. In *Proc. IEEE International Conference on Multimedia & Expo (ICME'06), Toronto, Ontario, Canada*, pages 1385–1388, 2006.

[BKKS01] M. M. Breunig, H.-P. Kriegel, P. Kröger, and J. Sander. Data Bubbles: Quality Preserving Performance Boosting for Hierarchical Clustering. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'01), Santa Barbara, CA, USA*, pages 79–90, 2001.

[BKP04] S. Brecheisen, H.-P. Kriegel, and M. Pfeifle. Efficient Density-Based Clustering of Complex Objects. In *Proc. 4th IEEE International Conference on Data Mining (ICDM'04), Brighton, UK*, pages 43–50, 2004.

[BKP05] S. Brecheisen, H.-P. Kriegel, and M. Pfeifle. Efficient Similarity Search on Vector Sets. In *Proc. 11. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW'05), Karlsruhe, Germany*, pages 425–443, 2005.

[BKP06a] S. Brecheisen, H.-P. Kriegel, and M. Pfeifle. Multi-Step Density-Based Clustering. *Knowledge and Information Systems (KAIS)*, 9(3):284–308, 2006.

[BKP06b] S. Brecheisen, H.-P. Kriegel, and M. Pfeifle. Parallel Density-Based Clustering of Complex Objects. In *Proc. 10th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'06), Singapore*, pages 179–188, 2006.

[BKSG06] S. Brecheisen, H.-P. Kriegel, M. Schubert, and M. Gruber. VICO: Visualizing Connected Object Orderings. In *Proc. 11th International Conf on Extending Database Technology (EDBT'06), Munich, Germany*, pages 1151–1154, 2006.

[BKSS90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'90), Atlantic City, NJ, USA*, pages 322–331, 1990.

[BÖ97] T. Bozkaya and Z. M. Özsoyoglu. Distance-Based Indexing for High-Dimensional Metric Spaces. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'97), Tucson, AZ, USA*, pages 357–368, 1997.

[Bri95] S. Brin. Near Neighbor Search in Large Metric Spaces. In *Proc. 21st International Conference on Very Large Databases (VLDB'95), Zurich, Switzerland*, pages 574–584, 1995.

[BWF+00] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[CNBYM01] E. Chávez, G. Navarro, R. Beaza-Yates, and J. Marroquín. Searching in Metric Spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

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

[CSL99] H. S. Chang, S. Sull, and S. U. Lee. Efficient Video Indexing Scheme for Content-Based Retrieval. *IEEE Transactions on Circuits and Systems for Video Technology (TCSVT)*, 9(8):1269–1279, 1999.

[CV95] C. Cortes and V. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.

[CVJK04] C. H. L. Costa, J. D. Valle Jr., and A. L. Koerich. Automatic classification of audio data. In *Proc. IEEE International Conference on Systems, Man, and Cybernetics (SMC'04)*, pages 562–567, 2004.

[CZ02] S. S. Cheung and A. Zakhor. Efficient video similarity measurement with video signature. In *Proc. IEEE International Conference on Image Processing (ICIP'02), Rochester, NY, USA*, pages 621–624, 2002.

[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 International Conference on Knowledge Discovery and Data Mining (KDD'96), Portland, OR, USA*, pages 291–316, 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 (JIIS)*, 3(3/4):231–262, 1994.

[FI92] U. M. Fayyad and K. B. Irani. On the Handling of Continuous-Valued Attributes in Decision Tree Generation. *Machine Learning*, 8(1):87–102, 1992.

[FJ03] M. J. Fonseca and J. A. Jorge. Indexing High-Dimensional Data for Content-Based Retrieval in Large Databases. In

Proc. 8th International Conference on Database Systems for Advanced Applications (DASFAA'03), Kyoto, Japan, pages 267–274, 2003.

[FPSS96] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge Discovery and Data Mining: Towards a Unifying Framework. In *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96), Portland, OR, USA*, pages 82–88, 1996.

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

[GFKS02] T. Gärtner, P. A. Flach, A. Kowalczyk, and A. J. Smola. Multi-Instance Kernels. In *Proc. 19th International Conference on Machine Learning (ICML'02), Sydney, Australia*, pages 179–186, 2002.

[GG98] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[GGM02] H. Greenspan, J. Goldberger, and A. Mayer. A Probabilistic Framework for Spatio-Temporal Video Representation & Indexing. In *Proc. 7th European Conference on Computer Vision, Copenhagen, Denmark*, pages 461–475, 2002.

[Gut84] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'84), Boston, MA, USA*, pages 47–57, 1984.

[HCH99] B. Huet, A. Cross, and E. Hancock. Graph Matching for Shape Retrieval. *Advances in Neural Information Processing Systems*, 11(2):866–902, 1999.

[HS95] G. R. Hjaltason and H. Samet. Ranking in Spatial Databases. In *Proc. 4th International Symposium on Large Spatial Databases (SSD'95), Portland, ME, USA*, pages 83–95, 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 Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 17(7):729–736, 1995.

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

[JB92] H. V. Jagadish and A. M. Bruckstein. On sequential shape descriptions. *Pattern Recognition*, 25(2):165–172, 1992.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: A Review. *ACM Computing Surveys*, 31(3):265–323, 1999.

[Kai04] K. Kailing. *New Techniques for Clustering Complex Objects*. PhD thesis, Institute for Computer Science, University of Munich, 2004.

[KBK+03] H.-P. Kriegel, S. Brecheisen, P. Kröger, M. Pfeifle, and M. Schubert. Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD'03), San Diego, CA, USA*, pages 587–598, 2003.

[KKG03] H.-P. Kriegel, P. Kröger, and I. Gotlibovich. Incremental OPTICS: Efficient Computation of Updates in a Hierarchical Cluster Ordering. In *Proc. 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'03), Prague, Czech Republic*, pages 224–233, 2003.

[KKM+03] H.-P. Kriegel, P. Kröger, Z. Mashael, M. Pfeifle, M. Pötke, and T. Seidl. Effective Similarity Search on Voxelized CAD Objects. In *Proc. 8th International Conference on Database Systems for Advanced Applications (DASFAA'03), Kyoto, Japan*, pages 27–36, 2003.

[KKPS04a] K. Kailing, H.-P. Kriegel, A. Pryakhin, and M. Schubert. Clustering Multi-Represented Objects with Noise. In *Proc. 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04), Sydney, Australia*, pages 394–403, 2004.

[KKPS04b] H.-P. Kriegel, P. Kröger, A. Pryakhin, and M. Schubert. Using Support Vector Machines for Classifying Large Sets of Multi-Represented Objects. In *Proc. SIAM International Conference on Data Mining (SDM'04), Lake Buena Vista, FL, USA*, pages 102–114, 2004.

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

[KKSS04] K. Kailing, H.-P. Kriegel, S. Schönauer, and T. Seidl. Efficient Similarity Search for Hierarchical Data in Large Databases. In *Proc. 9th International Conference on Extending Database Technology (EDBT'04), Heraklion, Greece*, pages 676–693, 2004.

[KKV90] E. Kubicka, G. Kubicki, and I. Vakalis. Using Graph Distance in Object Recognition. In *Proc. 18th ACM Computer Science Conference (CSC'90), Washington, DC, USA*, pages 43–48, 1990.

[KS86] H.-P. Kriegel and B. Seeger. Multidimensional Order Preserving Linear Hashing with Partial Expansions. In *Proc. International Conference on Database Theory (ICDT'86), Rome, Italy*, pages 203–220, 1986.

[KS03] H.-P. Kriegel and S. Schönauer. Similarity Search in Structured Data. In *Proc. 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK'03), Prague, Czech Republic*, pages 309–319, 2003.

[KSF+98] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast and Effective Retrieval of Medical Tumor Shapes. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 10(6):889–904, 1998.

[Kuh55] H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[McQ67] J. McQueen. Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley, CA, USA*, pages 281–297, 1967.

[Mun57] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the SIAM*, 5:32–38, 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 *Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187. SPIE, 1993.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.

[NJ02] A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *Proc. 5th International Workshop on the Web and Databases (WebDB'02), Madison, Wisconsin, USA*, pages 61–66, 2002.

[PCST99] J. Platt, N. Cristianini, and J. Shawe-Taylor. Large Margin DAGs for Multiclass Classification. In *Advances in Neural Information Processing Systems 12 (NIPS'99), Denver, CO, USA*, pages 547–553, 1999.

[RB01] J. Ramon and M. Bruynooghe. A polynomial time computable metric between point sets. *Acta Informatica*, 37(10):765–780, 2001.

[Sei97] T. Seidl. *Adaptable Similarity Search in 3-D Spatial Database Systems.* PhD thesis, Institute for Computer Science, University of Munich, 1997.

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

[SKK01] T. B. Sebastian, P. N. Klein, and B. B. Kimia. Recognition of Shapes by Editing Shock Graphs. In *Proc. 8th International Conference on Computer Vision (ICCV'01), Vancouver, BC, Canada*, pages 755–762, 2001.

[SQL+03] J. Sander, X. Qin, Z. Lu, N. Niu, and A. Kovarsky. Automatic Extraction of Clusters from Hierarchical Clustering Representations. In *Proc. 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003), Seoul, Korea*, pages 75–87, 2003.

[TC02] G. Tzanetakis and P. Cook. Musical Genre Classification of Audio Signals. *IEEE Transactions on Speech and Audio Processing (TASLP)*, 10(5):293–302, 2002.

[TJTSF00] C. Traina Jr., A. Traina, B. Seeger, and C. Faloutsos. Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes. In *Proc. 7th International Conference on Extending Database Technology (EDBT'00), Konstanz, Germany*, pages 51–65, 2000.

[TK00] T. Tolonen and M. Karjalainen. A Computationally Efficient Multipitch Analysis Model. *IEEE Transactions on Speech and Audio Processing (TASLP)*, 8(6):708–716, 2000.

[Uhl91] J. Uhlmann. Satisfying general Proximity/Similarity Queries with Metric Trees. *Information Processing Letters (IPL)*, 40(4):175–179, 1991.

[WFKvdM97] L. Wiskott, J.-M. Fellous, N. Krüger, and C. von der Malsburg. Face Recognition by Elastic Bunch Graph Matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 19(7):775–779, 1997.

[WLCS04] Y. Wu, C.-Y. Lin, E. Chang, and J. R. Smith. Multimodal information fusion for video concept detection. In *Proc. IEEE International Conference on Image Processing (ICIP'04), Singapore*, pages 2391–2394, 2004.

[WWL+99] J. T. L. Wang, X. Wang, K. I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. Evaluating a Class of Distance-Mapping Algorithms for Data Mining and Clustering. In *Proc. 5th International Conference on Knowledge Discovery and Data Mining (KDD'99), San Diego, CA, USA*, pages 307–311, 1999.

[WZCS02] J. T. L. Wang, K. Zhang, G. Chang, and D. Shasha. Finding Approximate Patterns in Undirected Acyclic Graphs. *Pattern Recognition*, 35(2):473–483, 2002.

[XJK99] X. Xu, J. Jäger, and H.-P. Kriegel. A Fast Parallel Clustering Algorithm for Large Spatial Databases. *Data Mining and Knowledge Discovery*, 3(3):263–290, 1999.

[Yia93] P. Yianilos. Data Structures and Algorithms for Nearest Neighbor Serch in General Metric Spaces. In *Proc. 4th ACM-SIAM Symposion on Discrete Algorithms (SODA'93), Austin, TX, USA*, pages 311–321, 1993.

[Zha03] T. Zhang. Semi-Automatic Approach for Music Classification. In *Internet Multimedia Management Systems IV*, volume 5242, pages 81–91. SPIE, 2003.

[ZRHM98] Y. Zhuang, Y. Rui, T. S. Huang, and S. Mehrotra. Adaptive Key Frame Extraction Using Unsupervised Clustering. In *Proc. IEEE International Conference on Image Processing (ICIP'98), Chicago, IL, USA*, pages 866–870, 1998.

[ZS03] J. Zhou and S. Sander. Data Bubbles for Non-Vector Data: Speeding-up Hierarchical Clustering in Arbitrary Metric Spaces. In *Proc. 29th International Conference on Very Large Databases (VLDB'03), Berlin, Germany*, pages 452–463, 2003.

[ZSS92] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters (IPL)*, 42:133–139, 1992.

[ZWS96] K. Zhang, J. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs. *International Journal of Foundations of Computer Science (IJFCS)*, 7(1):43–57, 1996.

Curriculum Vitae

Stefan Brecheisen was born on December 19, 1975 in Munich, Germany. He attended primary school from 1982 to 1986, and high school from 1986 to 1995. From October 1995 to October 1996, he served in the mandatory civil service at the old people's home *Altenheim St. Elisabeth* in Munich, Germany.

He entered the *Ludwig-Maximilians-Universität München* (LMU) in November 1996, studying Computer Science with a minor in Mathematics. His diploma thesis is titled "Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects" and was supervised by Prof. Hans-Peter Kriegel.

In February 2003, Stefan Brecheisen started working at the LMU as a teaching and research assistant in the group of Prof. Hans-Peter Kriegel, the chair of the teaching and research unit for database and information systems at the Department "Institute for Informatics". His research interests include similarity search and data mining in large standard, spatial, and multimedia databases.