New Techniques for Clustering Complex Objects

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

von
Karin Kailing

Tag der Einreichung: 18.08.2004
Tag der mündlichen Prüfung: 15.11.2004

Berichterstatter:
Prof. Dr. Hans-Peter Kriegel, Ludwig-Maximilians-Universität München
Prof. Dr. Johannes Gehrke, Cornell University (USA)

Acknowledgment

A lot of people supported and encouraged me while I was working at my thesis. I am very grateful for all the help I got during this time. Unfortunately, I can only mention some of them here, but my dearest thanks goes, of course, to all of them.

First of all, I want to express my warmest thanks to my supervisor Professor Dr. Hans-Peter Kriegel who convinced me to come to his group and then made sure that I never regretted this step. Without the productive and inspiring working atmosphere he created, this work could never have been done. I also want to thank Professor Johannes Gehrke who readily agreed to act as the second referee to this thesis.

This work was inspired by many fruitful discussions and cooperations with my colleagues. Without them this work could never have grown. I'm very grateful for all the support I got during the past years and of course, not to forget, for all the fun we had. A special thanks goes to my colleagues Dr. Peer Kröger and Dr. Stefan Schönauer who were always willing to answer all my questions.

The background help of Susanne Grienberger, who managed much of the administrative burden, was another reason that working in this group was so comfortable.

Furthermore I want to thank Franz Krojer who was always on the spot when I had trouble with technical things. He did a marvellous job in keeping our computers running and additionally gave me a lot of new insights into astrophysics.

I also want to thank all the students whose work helped to prove and establish my ideas. To name just a few of them: Alex Bosshammer, Adham

Haddad, Heribert Mühlberger, Claudia Plant and Thomas Müller.

Last but not least, I like to express my deepest thanks to my husband, my family and my friends who always believed in me, sometimes more than I did.

Karin Kailing

Munich, August 2004.

Abstract

The tremendous amount of data produced nowadays in various application domains such as molecular biology or geography can only be fully exploited by efficient and effective data mining tools. One of the primary data mining tasks is clustering, which is the task of partitioning points of a data set into distinct groups (clusters) such that two points from one cluster are similar to each other whereas two points from distinct clusters are not.

Due to modern database technology, e.g. object relational databases, a huge amount of complex objects from scientific, engineering or multimedia applications is stored in database systems. Modelling such complex data often results in very high-dimensional vector data ("feature vectors"). In the context of clustering, this causes a lot of fundamental problems, commonly subsumed under the term "Curse of Dimensionality". As a result, traditional clustering algorithms often fail to generate meaningful results, because in such high-dimensional feature spaces data does not cluster anymore. But usually, there are clusters embedded in lower dimensional subspaces, i.e. meaningful clusters can be found if only a certain subset of features is regarded for clustering. The subset of features may even be different for varying clusters.

In this thesis, we present original extensions and enhancements of the density-based clustering notion to cope with high-dimensional data. In particular, we propose an algorithm called SUBCLU (density-connected *Sub*space *Clu*stering) that extends DBSCAN (*D*ensity-*B*ased *S*patial *C*lustering of *A*pplications with *N*oise) to the problem of subspace clustering. SUBCLU efficiently computes all clusters of arbitrary shape and size that would have been found if DBSCAN were applied to all possible subspaces

of the feature space. Two subspace selection techniques called RIS (*R*anking *I*nteresting *S*ubspaces) and SURFING (*SU*bspaces *R*elevant *F*or cluster*ING*) are proposed. They do not compute the subspace clusters directly, but generate a list of subspaces ranked by their clustering characteristics. A hierarchical clustering algorithm can be applied to these interesting subspaces in order to compute a hierarchical (subspace) clustering. In addition, we propose the algorithm 4C (*C*omputing *C*orrelation *C*onnected *C*lusters) that extends the concepts of DBSCAN to compute density-based correlation clusters. 4C searches for groups of objects which exhibit an arbitrary but uniform correlation.

Often, the traditional approach of modelling data as high-dimensional feature vectors is no longer able to capture the intuitive notion of similarity between complex objects. Thus, objects like chemical compounds, CAD drawings, XML data or color images are often modelled by using more complex representations like graphs or trees. If a metric distance function like the edit distance for graphs and trees is used as similarity measure, traditional clustering approaches like density-based clustering are applicable to those data. However, we face the problem that a single distance calculation can be very expensive. As clustering performs a lot of distance calculations, approaches like filter and refinement and metric indices get important. The second part of this thesis deals with special approaches for clustering in application domains with complex similarity models. We show, how appropriate filters can be used to enhance the performance of query processing and, thus, clustering of hierarchical objects. Furthermore, we describe how the two paradigms of filtering and metric indexing can be combined. As complex objects can often be represented by using different similarity models, a new clustering approach is presented that is able to cluster objects that provide several different complex representations.

Abstract (in German)

Innovative Technologien und neueste Methoden zur Datengewinnung in verschiedenen Teilbereichen der Wissenschaft, wie z.B. Biowissenschaften, Medizin, Astronomie, Geographie, erzeugen eine wahre Flut an Rohdaten. Um dieser Flut Herr werden zu können, werden dringend effiziente und effektive Methoden zur automatischen Datenanalyse und zur Wissensextraktion (Data Mining) benötigt. Ein äußerst wichtiges Teilproblem des Data Mining ist das Clustering. Beim Clustering sollen die Objekte einer Datenbank in (a priori unbekannte) Gruppen, auch Cluster genannt, eingeteilt werden, so dass zwei Objekte aus einem gleichen Cluster möglichst ähnlich zueinander und zwei Objekte aus unterschiedlichen Clustern möglichst unähnlich zueinander sind.

Dank moderner Datenbanktechnologien, z.B. objekt-relationale Datenbanken, lassen sich heute beliebig komplexe Objekte in großen Mengen verwalten. Die Modellierung solch komplexer Objekte führt häufig zu sehr hochdimensionalen Merkmalsvektoren. Dies verursacht im Kontext Clustering eine Menge von grundsätzlichen Problemen, die mit dem Ausdruck "Curse of Dimensionality" (Fluch der Dimensionalität) zusammengefasst werden. Die Folge ist, dass die Objekte in hochdimensionalen Räumen stark streuen, und nicht mehr sinnvoll zu clustern sind. Dennoch gibt es meistens Cluster in verschiedenen Teilräumen niedrigerer Dimensionalität (Unterräume), d.h. die Daten clustern, wenn man gewisse (teilweise unterschiedliche) Attribute ausblendet.

Im ersten Teil dieser Arbeit werden spezielle Verfahren vorgestellt, die auf das Problem des Clusterings hochdimensionaler Daten zugeschnitten sind. Es wird ein dichtebasiertes Unterraum-Clusteringverfahren präsentiert,

das in der Lage ist, Cluster beliebiger Form und Größe in Teilräumen zu finden. Zusätzlich werden zwei Verfahren vorgestellt, die die Unterraumcluster nicht direkt berechnen, sondern eine Liste geeigneter Unterräume erzeugen. In diesen interessanten Unterräumen kann dann ein hierarchischer Clustering-Algorithmus eingesetzt werden, um ein hierarchisches Unterraum-Clustering zu erzeugen. Da hochdimensionale Daten sehr häufig auch starken Korrelationen unterliegen, wird außerdem ein Clusteringverfahren eingeführt, das gezielt nach Gruppen von Objekten mit einer einheitlichen Korrelation sucht.

Oft reicht die traditionelle Modellierung der Daten als Merkmalsvektoren nicht mehr aus, um die intuitive Ähnlichkeit der Objekte adäquat auszudrücken. Daher verwendet man für komplexe Daten wie Moleküle, CAD-Zeichnungen, XML-Daten, Websites oder Farbbilder häufig komplexere Modellierungsformen wie zum Beispiel Graph- oder Baumdarstellungen, um diese Daten geeignet zu repräsentieren. Lässt sich auf dieser Repräsentation eine metrische Abstandsfunktion finden, so können für "punktartige" Objekte konzipierte Clustering-Algorithmen weiter genutzt werden. Einzelne Distanzberechnungen können in diesem Fall jedoch sehr teuer sein und müssen daher geeignet unterstützt werden. Da beim Clustering meist sehr viele Distanzberechnungen nötig sind, spielen Ansätze wie Filterverfeinerungstechniken und metrische Indizes hier eine große Rolle. Im Rahmen der Arbeit wurden spezielle Verfahren für komplex modellierte Objekte entwickelt. Es wird gezeigt, wie die Anfragebearbeitung auf hierarchischen Objekten mit Hilfe geeigneter Filter beschleunigt werden kann. Außerdem beschäftigt sich dieser Teil damit, wie die beiden Ansätze Filterverfeinerung und metrische Indizes kombiniert werden können. Da für ein komplexes Objekt häufig mehrere unterschiedliche Repräsentationen vorliegen, wird zusätzliche ein Verfahren vorgestellt, das in der Lage ist, Objekte zu clustern, für die beliebig viele verschiedene komplexe Repräsentationen existieren.

Contents

Part I

Preliminaries

Chapter 1

Introduction

Due to the enormous amount of data in various application domains, the requirements of database systems have changed. Techniques to analyze the given information and find so far hidden knowledge are mandatory to draw maximum benefit from the collected data. Knowledge Discovery in Databases (KDD) is an interdisciplinary field, aimed at extracting valuable knowledge from large databases. At the core of the KDD process is the Data Mining step which embraces many data mining methods. One of them is clustering, the central topic of this thesis. In this chapter, the KDD process is introduced and discussed in detail. Afterwards we describe the data mining step in more detail, and review the most important and influential methods of data mining. As this thesis focuses on clustering complex objects, we give a short overview over the special requirements for clustering complex objects. The chapter concludes with an outline of this thesis.

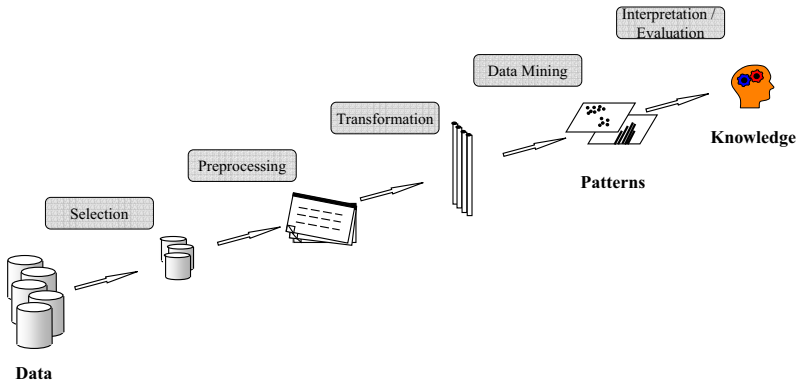


Figure 1.1: An overview of the steps comprising the KDD process.

1.1 Knowledge Discovery in Databases, Data Mining, Clustering

Modern methods in several application domains such as molecular biology, astronomy, geography, etc. produce a tremendous amount of data. Since all this data can no longer be managed without the help of automated analysis tools, there is an ever increasing need for efficient and effective data mining methods to make use of the information contained implicitly in that data. ***Knowledge Discovery in Databases*** *is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data* [FPSS96]. The KDD process (see Figure 1.1 for an illustration) is an interactive process and consists of an iterative sequence of the following steps:

- **Selection:** Creating a target data set by selecting a subset of the data or focusing on a subset of attributes.
- **Preprocessing and Transformation:** Finding useful features to represent the data, e.g. using dimensionality reduction or transformation methods to reduce the number of attributes or to find invariant representations for the data.
- **Data Mining:** Searching for patterns of interest in the particular representation of the data: classification rules or trees, association

rules, regression, clustering, etc.

- **Interpretation and Evaluation:** Applying visualization and knowledge representation techniques to the extracted patterns. It is possible that the user has to return to previous steps in the KDD process if the results are unsatisfactory.

The core step of the KDD process is the application of a data mining algorithm. Hence, the notions "KDD" and "Data Mining" are often used in the same way. Actually, most of the research done in the field of knowledge discovery is about data mining algorithms. The following broad definition of data mining can be found in [FPSS96]:

Data Mining *is a step in the KDD process consisting of applying data analysis and discovery algorithms that, under acceptable computational efficiency limitations, produce a particular enumeration of patterns over the data.*

The different data mining algorithms that have been proposed in the literature can be classified according to the following primary data mining methods ([HK00]):

- **Association Analysis:** Discovering association rules, showing attribute-value conditions that occur frequently together in a given data set.
- **Classification and Prediction:** Learning a function that maps (classifies) a data item into one of several predefined classes.
- **Clustering:** Identifying a set of categories or clusters to describe the data.
- **Characterization and Discrimination:** Finding a compact description for a subset of the data or comparing a particular subset of the data with comparative subsets.
- **Outlier Detection:** Finding outliers, i.e. data objects that do not correspond to the general behavior or model of the data.
- **Evolution Analysis:** Describing and Modelling regularities or trends for objects whose behavior changes over time.

One of the primary data mining tasks is clustering (sometimes also called segmentation) which is intended to help a user discovering and understanding the natural structure or grouping in a data set. In particular, clustering is the task of partitioning objects of a data set into distinct groups (clusters) such that two objects from one cluster are similar to each other, whereas two objects from distinct clusters are not.

Clustering has been studied extensively in statistics for many years (see e.g. [Eve81]). A similar approach in machine learning is called unsupervised learning (see e.g. [Fis87, CKS+88]). These well-known techniques are usually very inefficient on large databases and also assume that all objects to be clustered can be kept in main memory at the same time. Thus, clustering has recently received a lot of attention in the database community (e.g. [NH94, ZRM96, GRS98, SEKX98, AGGR98, AY00]).

Existing clustering algorithms can broadly be classified into *hierarchical* and *partitioning clustering* algorithms [JD88]. Partitioning clustering algorithms construct a flat (single level) partition of a database DB of n objects into a set of k clusters. Hierarchical algorithms decompose a database DB of n objects into several levels of nested partitionings (clustering), generally represented by a tree that iteratively splits DB into smaller subsets. In such a hierarchy, each node of the tree represents a cluster of DB.

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, GM93, FBF+94, FRM94, ALSS95, KSF+96, BBB+97, BK97, KKS98, Kei99, AKKS99, PM99, SKK01, NJ02, KBK+03, KKM+03]. As clustering relies on 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 a non 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

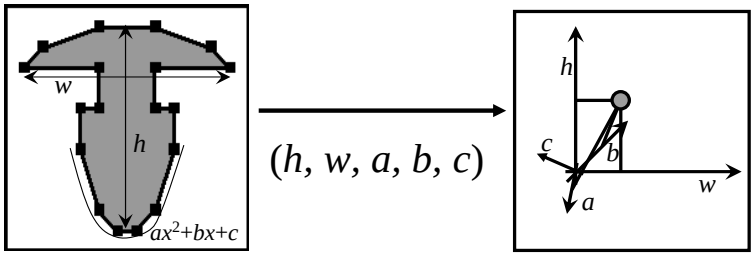


Figure 1.2: The idea of feature transformation.

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 Complex Objects Represented 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 distance 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 [AKKS99]. The problem we address in this thesis is that feature vectors often get very high-dimensional which leads to several problems for clustering algorithms.

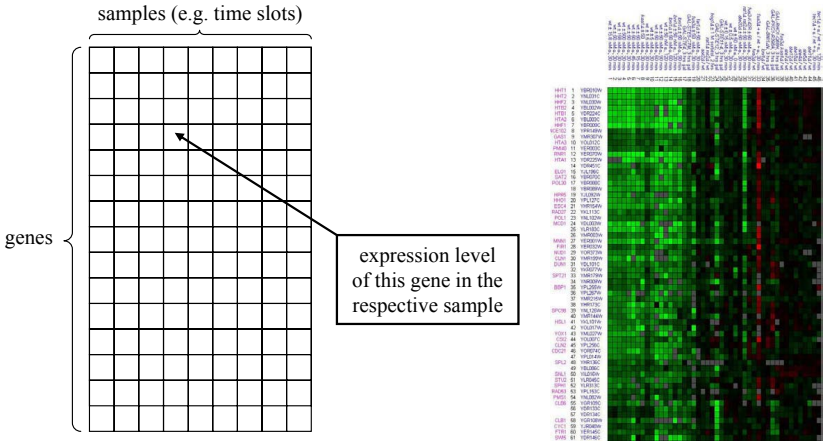


Figure 1.3: Gene expression data matrix: schematic view (left), visualization of a sample raw data excerpt (right).

Often, traditional clustering approaches which cluster the data, taking all features into account, produce no meaningful results. Nevertheless, interesting clusters can be found if not all features of the feature vectors are taken into account, i.e. the data sets often contain interesting clusters which are hidden in various subspaces of the original feature space. Additionally, the data may cluster differently if different subspaces are examined.

Gene expression data is a prominent example for that phenomenon. Microarray chip technologies enable a user to measure the expression level of thousands of genes simultaneously. Roughly speaking, the expression level of a gene is a measurement for the frequency the gene is expressed. Usually, gene expression data appear as a matrix where the rows represent genes, and the columns represent samples (e.g. different experiments, time slots, test persons). The value of the i-th feature of a particular gene is the expression level of this gene in the i-th sample (see Figure 1.3 for an illustration).

It is interesting from a biological point of view to cluster both the rows (genes) and the columns (samples) of the matrix, depending on the research scope. Clustering the genes is the method of choice if one searches for co-expressed genes, i.e. genes, whose expression levels are similar. Co-expression usually indicates that the genes are functionally related. If one searches for homogeneous groups in the set of samples, the problem is to

cluster the samples. For example in cancer diagnostics, the samples may represent test persons. The ultimate goal of the clustering is then to distinguish between healthy and sick patients.

When clustering the genes to detect co-expressed genes, one has to cope with the problem that usually the co-expression of the genes can only be detected in subsets of the samples. In other words, different subsets of the attributes (samples) are responsible for different co-expressions of the genes. When clustering the samples to identify e.g. homogeneous groups of patients, this situation is even worse. As various phenotypes, e.g. hair color, gender, cancer, etc., are hidden in varying subsets of the genes, the samples could usually be clustered differently according to these phenotypes, i.e. in varying subspaces.

1.2.2 Complex Objects Represented as Arbitrary Metric Data

Sometimes the similarity between complex objects can not be captured by a feature transformation. In this case, the use of more complex similarity models like the edit distance for graphs or trees are necessary. The remainder of this section presents three metric similarity models for complex objects and points out the challenges for clustering in such application domains. 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 their 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 1.4 left for an illustration). The cover sequence model introduced in [Jag91, JB92] is extended by generating several representations

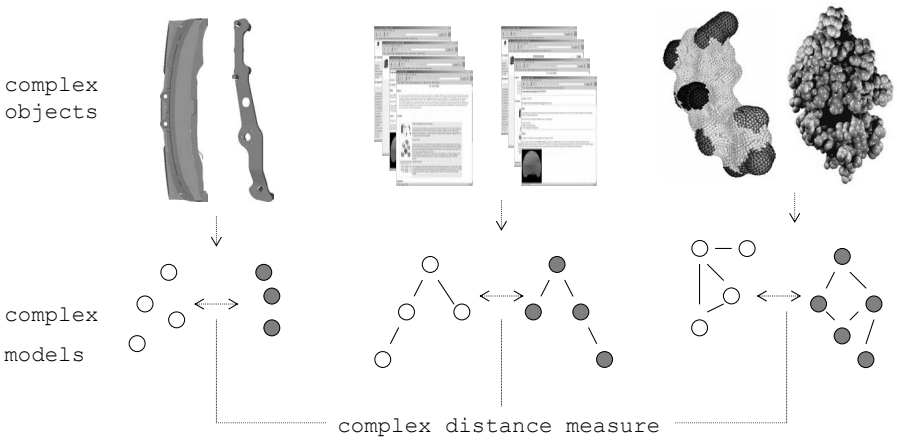


Figure 1.4: Examples of complex metric data.

for each object, resulting in a set of feature vectors for each object. In the experimental evaluation the authors show 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 1.4 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 1.4 right for an illustration). As graphs are a very general object model, graph similarity has been studied in many fields. Similarity 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.

Challenges for Clustering. Clustering relies on computing the distance between objects and, thus, the complexity of the above mentioned similarity models has a severe influence on the efficiency of the clustering algorithms. Especially for density-based clustering, range queries must be supported efficiently to reduce the runtime of clustering. Another challenge for clustering is that often there are multiple representation forms for each object. Proteins, for example, are characterized by an amino acid sequence, a secondary and a tertiary structure. Therefore, the clustering algorithm should be able to take the information of more than one representation into account.

1.3 Basic Notations

Let DB be a data set of n objects. Except for the third part of the thesis we assume the following:

DB is a database of d-dimensional feature vectors ($DB \subseteq \mathbb{R}^d$). All feature vectors have normalized values, i.e. all values fall into $[0, attrRange]$ for a fixed $attrRange \in \mathbb{R}^+$. We call those feature vectors points. Let $\mathcal{A} = \{a_1, \ldots, a_d\}$ be the set of all attributes a_i of DB. Any subset $S \subseteq \mathcal{A}$ is called a subspace. The cardinality of S ($|S|$) is called the dimensionality of S. The projection of a point p into a subspace $S \subseteq \mathcal{A}$ is denoted by $\pi_S(p)$. The distance function is denoted by $dist$. We assume that $dist$ is one of the L_p-norms. The ε-neighborhood of a point p is defined by $\mathcal{N}_\varepsilon(p) = \{x \in DB \mid dist(p, x) \leq \varepsilon\}$. The ε-neighborhood of a point in a subspace $S \subseteq \mathcal{A}$ is denoted by $\mathcal{N}_\varepsilon^S(p) := \{x \in DB \mid dist(\pi_S(p), \pi_S(x)) \leq \varepsilon\}$.

The value of the i-th attribute ($1 \leq i \leq d$) of P is denoted by p_i (i.e. $P = (p_1, \ldots, p_d)^{\mathbf{T}}$).

1.4 Outline of the Thesis

The starting point of this thesis is the density-based clustering approach, in particular the concepts of density-connected clusters underlying the algorithms DBSCAN (*D*ensity-*B*ased *S*patial *C*lustering of *A*pplications with *N*oise) [EKSX96]. We propose new techniques to cope with the challenges clustering complex objects as described in Section 1.2 imply. The thesis is organized as follows:

Part I deals with the preliminaries.

Chapter 1 should give the reader a short introduction to the broader context of this thesis.

Chapter 2 provides a brief overview over existing clustering algorithms and introduces the density-based clustering notion underlying DBSCAN. As mentioned above, the concepts described in this chapter form the basis of the techniques proposed in this thesis.

Part II presents new techniques to cope with the challenges of clustering in high-dimensional feature spaces.

Chapter 3 extends the density-based clustering concepts to the problem of subspace clustering. Based on monotonicity properties for density-based clusters, we present SUBCLU (density-connected *SUB*space *CLU*stering), a density-based subspace clustering algorithm. A broad experimental evaluation of SUBCLU shows its superior accuracy compared to existing subspace clustering algorithms.

Chapter 4 presents an extension of SUBCLU called RIS (*R*anking *I*nteresting *S*ubspaces for Clustering) which does not directly compute the subspace clusters. Instead RIS ranks all interesting subspaces according to a certain quality criterion. Afterwards the hierarchical clustering algorithm OPTICS (*O*rdering *P*oints *T*o *I*dentify the *C*lustering *S*tructure) is applied to the top-ranked subspaces.

Chapter 5 further explores the idea of subspace selection for clustering. An algorithm SURFING (*SU*bspaces *R*elevant *F*or cluster*ING*) is presented which is able to rank the subspaces of a data set according to their hierarchical clustering structure. A comparative evaluation of SUFRING, RIS and SUBCLU reveals that SURFING in combination with OPTICS is able to gain significantly more information than SUBCLU and RIS in combination with OPTICS.

Chapter 6 proposes a novel correlation clustering algorithm called 4C (*C*omputing *C*orrelation *C*onnected *C*lusters) which is based on a combination of density-based clustering and principal component analysis. First, the concept of correlation-connected clusters is formalized. Then, we present the details of the 4C algorithm. An experimental evaluation compares 4C to several competing clustering algorithms, showing its superior performance.

Part III deals with the problems complex similarity measures present to clustering.

Chapter 7 uses images as a motivating example for the new challenges complex objects present to clustering algorithms.

Chapter 8 shows, how the information of different representations can be integrated into the clustering process of complex objects. The density-based clustering notion is extended to handle multi-represented objects. The evaluation shows that this approach yields more accurate results than using only one single representation.

Chapter 9 presents filters for tree-structured data and shows that they successfully reduce the runtimes of queries on hierarchical data, like images or web sites. This is extremely important for clustering algorithms like DBSCAN which rely on computing range queries for each database object.

Chapter 10 shows how the combination of filtering and metric indexing can further enhance the performance of range query processing.

Part IV concludes this thesis.

Chapter 11 summarizes and discusses the major contributions of the thesis. It concludes with pointing out some future research directions.

Chapter 2

Density-Based Clustering

Many clustering algorithms have been proposed. This thesis is especially based on the density-based clustering approach which turned out to be one of the most effective and also efficient ones. The chapter starts with a short overview of recently proposed clustering algorithms. After that, a detailed introduction to the density-based clustering notion is given. In particular, we introduce the notion of density-connected sets underlying the algorithm DBSCAN (*D*ensity-*B*ased *S*patial *C*lustering of *A*pplications with *N*oise) [EKSX96]. The chapter concludes with two extensions to DBSCAN, namely OPTICS (*O*rdering *P*oints *T*o *I*dentify the *C*lustering *S*tructure) [ABKS99] and BOSS (*B*rowsing *O*PTICS-Plots for *S*imilarity *Search*) [BKKP04], and points out the advantages of density-based clustering.

2.1 General Clustering Approaches

In the past decade, many algorithmic solutions for the problem of clustering have been proposed. The choice of the clustering algorithm depends both on the type of data and the particular application. In this section, we will provide a brief overview of existing clustering approaches together with a short list of reference methods. According to [HK00] they can broadly be classified into the following categories:

2.1.1 Partitioning Algorithms

Partitioning clustering algorithms compute a "flat" partition of the data into a given number of clusters, i.e. a unique assignment of each data object to a cluster. The number of clusters k is often a user specified parameter. Typically, partitioning algorithms start with an initial partitioning of the database into k clusters which may be user-defined or randomly generated. The initial partitioning is then iteratively optimized by moving objects from one group to another. The general criterion of a good partitioning is that objects in the same cluster are "close", whereas objects of different clusters are "far apart". If the clustering quality does not decrease after an iteration, i.e. converges, the clustering algorithms terminate. Most applications adopt one of the following two heuristic methods. The k-means algorithm represents each cluster by the mean value of all objects in the cluster, whereas the k-medoids algorithm represents each cluster by one of the objects located near the center of the cluster.

Partitioning algorithms usually converge to local minima and therefore may miss the "best" clustering in terms of cluster quality. In addition, these algorithms tend to produce clusters of spherical shapes and are rather sensitive to noise, since all objects of the database are assigned to a cluster. A further drawback of these algorithms is the sensitivity to the input parameter k, because the number of clusters is usually not known before. Sample algorithms are k-means [McQ67], PAM [KR90], and CLARANS [NH94].

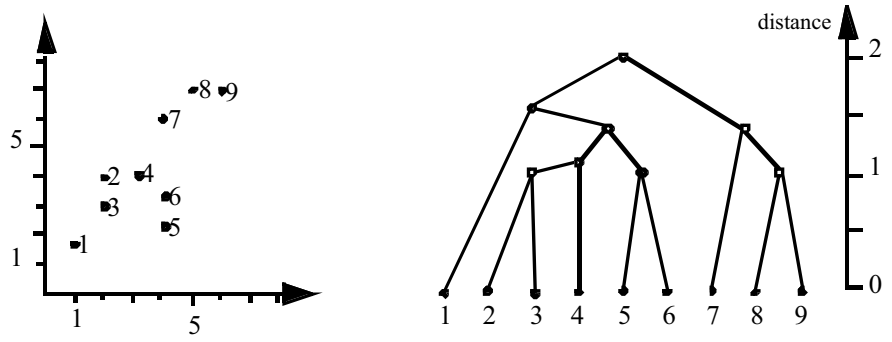


Figure 2.1: A dendrogram (right) for a sample data set (left).

2.1.2 Hierarchical Algorithms

Hierarchical clustering algorithms compute a hierarchical decomposition of the data objects instead of a unique assignment of data objects to clusters. The hierarchical clustering structure is usually visualized by using a tree representation, a so-called *dendrogram* (cf. Figure 2.1). Each leaf of a dendrogram corresponds to one data object, whereas the root represents the entire database. Each node in the dendrogram represents a cluster containing all objects of the leaf nodes below this node. Each level of the dendrogram represents a clustering of the database. A bottom-up algorithmic scheme to construct a dendrogram starts with placing each object in the database into a unique cluster (leaf nodes) and then merges in each step the pair of clusters having the minimal distance until all data objects are contained in one cluster. In [Bou96] several definitions of the distance between two clusters (e.g. single link [Sib73]) are discussed. It is shown that each approach yields the same result in terms of clustering quality.

2.1.3 Density-Based Methods

Density-based methods search for regions of high point density that are separated by regions of lower point density. These algorithms usually need two input parameters; one specifying a volume and a second one specifying a minimum number of points. Using these two parameters, a density threshold is specified. Sets of objects must exceed this density threshold to be detected as clusters. The pioneering density-based approach is DBSCAN [EKSX96]

which is founded on the notion of density-connected sets. Since this clustering notion is the basis of this thesis, we will present the concepts underlying DBSCAN in more detail in Section 2.2. The combination of density-based clustering and hierarchical concepts is presented in [ABKS99]. There, the algorithm OPTICS is proposed to compute a density-based hierarchical decomposition of the data. In particular, OPTICS computes the clustering of DBSCAN for a broad range of parameter settings in a single scan over the data.

2.1.4 Grid-Based Methods

Grid-based methods divide the data space into a finite number of cells that form a grid structure. The clustering is then performed on the grid structure. The main advantage of this approach is its fast processing time which is typically independent from the number of data objects. However, grid-based approaches heavily depend on the resolution and the positioning of the grid. STING [WYM97] is a typical example of a grid-based method. Other techniques are CLIQUE [AGGR98], WaveCluster [SCZ98] or Opti-Grid [HK99].

2.1.5 Model-Based Methods

Model-based clustering methods attempt to optimize the fit between the given data and some mathematical model. Such methods are often based on the assumption that each object is drawn from one of k underlying probability distributions. Often, objects are assigned to one of the k clusters using a maximum likelihood decision. An example of such a model-based algorithm is the EM-algorithm [DLR77].

2.2 Foundations of Density-Based Clustering

The density-based notion is a common approach for clustering, used by various algorithms such as DBSCAN [EKSX96], DBCLASD [XEKS98], DENCLUE [HK98], and OPTICS [ABKS99]. All these methods search for regions of high density in a feature space that are separated by regions of lower den-

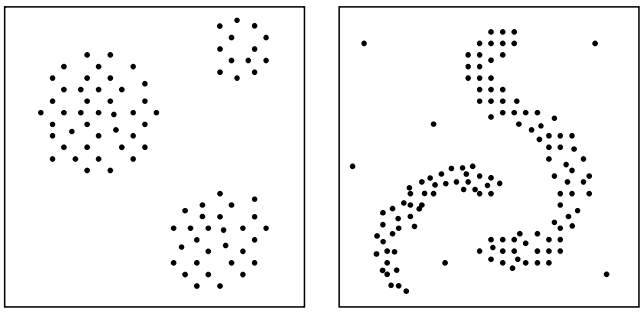


Figure 2.2: Convex (left) and arbitrarily (right) shaped clusters.

sity. The approaches presented in this thesis are particularly based on the formal definitions of density-connected clusters underlying the algorithm DBSCAN [EKSX96]. As illustrated in Figure 2.2, the density-connected clustering notion is capable of finding clusters of arbitrary shapes. In the following, we introduce the concepts necessary to find all density-connected clusters of a given data set.

Definition 2.1 (ε-neighborhood)
Let $\varepsilon \in \mathbb{R}_0^+$, $o \in DB$. The ε-neighborhood *of o, denoted by $\mathcal{N}_\varepsilon(o)$, is defined by*

$$\mathcal{N}_\varepsilon(o) = \{x \in DB \mid dist(o,x) \leq \varepsilon\}.$$

Based on the two input parameters (ε and k), dense regions can be defined by means of core objects:

Definition 2.2 (core object)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $o \in DB$ is called core object*, denoted by* $\textsc{Core}_{\varepsilon,k}(o)$ *if its ε-neighborhood contains at least k objects, formally:*

$$\textsc{Core}_{\varepsilon,k}(o) \Leftrightarrow |\mathcal{N}_\varepsilon(o)| \geq k.$$

Clusters contain core objects, located inside a cluster, and border objects, located at the border of the cluster (see Figure 2.3(a) for an illustration). In addition, a cluster should form a dense region and thus, all objects within a cluster should be "connected". Using the concept of connectivity, any core object o can be used to expand a cluster. To find all density-connected objects of o the following concepts are used.

Definition 2.3 (direct density-reachability)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $p \in DB$ is directly density-reachable *from $q \in DB$ if q is a core object and p is an element of $\mathcal{N}_\varepsilon(q)$, formally:*

$$\text{DirReach}_{\varepsilon,k}(q,p) \Leftrightarrow \text{Core}_{\varepsilon,k}(q) \ \wedge \ p \in \mathcal{N}_\varepsilon(q).$$

The concept of direct density-reachability is depicted in Figure 2.3(b). As we want to be independent of the order of processing, we can only use direct density-reachability for core objects. For border objects, this relation is not symmetric.

To find all density-connected objects, we can now build the transitive closure of direct density-reachability.

Definition 2.4 (density-reachability)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $p \in DB$ is density-reachable *from $q \in DB$ if there is a sequence of objects $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i, formally:*

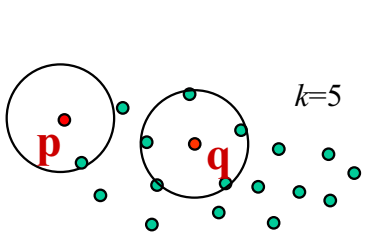
$$\begin{aligned}
&\text{Reach}_{\varepsilon,k}(q,p) \Leftrightarrow \\
&\exists p_1, \ldots, p_n \in DB : p_1 = q \ \wedge \ p_n = p \ \wedge \\
&\forall i \in \{1, \ldots, n-1\} : \text{DirReach}_{\varepsilon,k}(p_i, p_{i+1}).
\end{aligned}$$

Density-reachability is illustrated in Figure 2.3(c). Density-reachability is still not symmetric in general. Thus, we finally introduce the concept of density-connectivity.

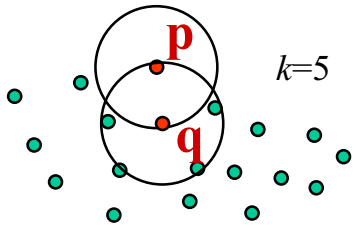
Definition 2.5 (density-connectivity)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $p \in DB$ is density-connected *to an object $q \in DB$ if there is an object o such that both p and q are density-reachable from o, formally:*

$$\begin{aligned}
&\text{Connect}_{\varepsilon,k}(q,p) \Leftrightarrow \\
&\exists o \in DB : \text{Reach}_{\varepsilon,k}(o,q) \ \wedge \ \text{Reach}_{\varepsilon,k}(o,p).
\end{aligned}$$

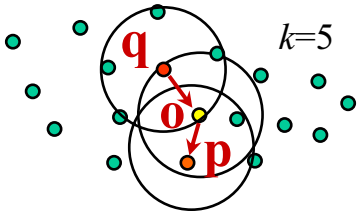
Density-connectivity is a symmetric relation. Thus, searching for all density-connected points is independent from the order of processing. The concept is visualized in Figure 2.3(d).



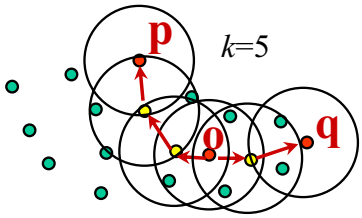
(a) q is a core object and p a border object.



(b) p is direct density-reachable from q; q is not direct density-reachable from p.



(c) p is density-reachable from q.



(d) p and q are density-connected.

Figure 2.3: Concepts of DBSCAN.

Definition 2.6 (density-connected set)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. A non-empty subset $C \subseteq DB$ is called a density-connected set *if all objects in C are density-connected in S, formally:*

$$\textsc{ConSet}_{\varepsilon,}^{k}(C) \Leftrightarrow \forall o, q \in C : \textsc{Connect}_{\varepsilon,}^{k}(o, q).$$

Finally, a density-connected cluster is defined as a set of density-connected objects which is maximal w.r.t. density-reachability.

Definition 2.7 (density-connected cluster)
Let $\varepsilon \in \mathbb{R}_0^+$ and $k \in \mathbb{N}$. A non-empty subset $\mathcal{C} \subseteq DB$ is called a density-connected cluster *w.r.t. ε and k if all objects in $\mathcal{C}$ are density-connected and $\mathcal{C}$ is maximal w.r.t. density-reachability, formally:*

$\textsc{ConCluster}_{\varepsilon,k}(\mathcal{C}) \Leftrightarrow$

(1) *Connectivity:* $\forall o, q \in \mathcal{C} : \textsc{Connect}_{\varepsilon,k}(o, q)$.

(2) *Maximality:* $\forall p, q \in DB : q \in \mathcal{C} \wedge \textsc{Reach}_{\varepsilon,k}(q, p) \Rightarrow p \in \mathcal{C}$.

```
DBSCAN(SetOfObjects DB, Real ε, Integer k)
  // each point in DB is marked as unclassified
  generate new clusterID cid;
  for each p ∈ DB do
    if p.clusterID = UNCLASSIFIED then
      if ExpandCluster(DB, p, cid, ε, k) then
        cid := cid + 1;
      end if
    end if
  end for
```

Figure 2.4: The algorithm DBSCAN.

Using these concepts, DBSCAN is able to detect arbitrarily shaped clusters by one single pass over the data. To do so, DBSCAN uses the fact that a density-connected cluster can be detected by finding one of its core objects o and computing all objects which are density-reachable from o. The pseudo code of DBSCAN is depicted in Figure 2.4.

The method ExpandCluster which computes the density-connected cluster starting from a given core point, is shown in Figure 2.5.

The correctness of DBSCAN can be formally proven (cf. lemmata 1 and 2 in [EKSX96], proofs in [SEKX98]). Although DBSCAN is not in a strong sense deterministic (the run of the algorithm depends on the order in which the points are stored), both the run-time as well as the result (number of detected clusters and association of core objects to clusters) are determinate. Note that according to the definitions above border objects may be border objects to more than one density-connected cluster. In this version a border object is added to the first cluster where it is a border object. Dependent on the application domain other solutions are possible. The worst case time complexity of DBSCAN is $O(n \log n)$, assuming an efficient index and $O(n^2)$ if no index exists.

```
ExpandCluster(SetOfObjects DB, Object start, Integer cid, Real ε, Integer k) → boolean
  SetOfObjects seeds := N_ε(start);
  if |seeds| < k then
    start.clusterID := NOISE;
    return false;
  end if
  for each o ∈ seeds do
    o.clusterID := cid;
  end for
  remove start from seeds;
  while seeds ≠ ∅ do
    o := first point in seeds;
    neighbors := N_ε()(o);
    if |neighbors| ≥ k then
      for each p ∈ neighbors do
        if p.clusterID ∈ {UNCLASSIFIED, NOISE} then
          if p.clusterID = UNCLASSIFIED then
            insert p into seeds;
          end if
        p.clusterID := cid;
        end if
      end for
    end if
    remove o from seeds;
  end while
  return true;
```

Figure 2.5: The method ExpandCluster.

2.3 Extensions of Density-Based Clustering

Hierarchical Density-Based Clustering

DBSCAN computes a flat density-based decomposition of a database w.r.t. a global density parameter, specified by ε and k. However, there may be clusters of different density and/or nested clusters in the database (see Figure 2.6 for an illustration). In this case, the globally chosen density threshold determines which clusters will be found and DBSCAN is not able to detect all the clustering information contained in such data.

To overcome this problem, the density-connected clustering notion is

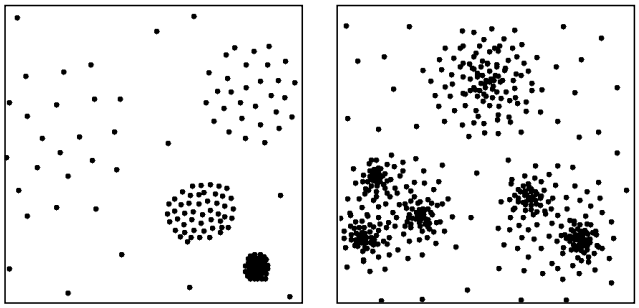


Figure 2.6: Clusters with different density (left) and nested clusters (right).

extended by hierarchical concepts [ABKS99]. Based on these concepts, the algorithm OPTICS is presented. The key idea is that (for a constant k-value) density-based clusters w.r.t. a higher density (i.e. a lower value for ε) are completely contained in density-based clusters w.r.t. a lower density (i.e. a higher value for ε). Figure 2.7 illustrates this observation: C_1 and C_2 are density-based clusters w.r.t. $eps_1 < eps_2$ and C is a density-based cluster w.r.t. eps_2, completely containing C_1 and C_2.

The algorithm OPTICS works like an extended DBSCAN algorithm, computing the density-connected clusters w.r.t. all parameters ε_i that are smaller than a generic value ε. In contrast to DBSCAN, OPTICS does not assign cluster memberships, but stores the order in which the data objects are processed and the information which would be used by an extended DBSCAN algorithm to assign cluster memberships. This information consists of only two values for each object, the *core distance* and the *reachability distance*. The core distance of a point q is the smallest threshold $\hat{\varepsilon} \leq \varepsilon$ such

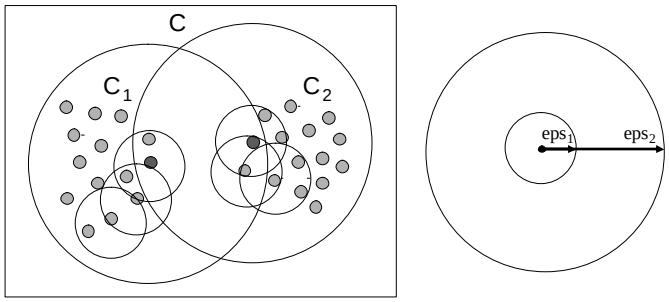


Figure 2.7: Nested clusters of different density.

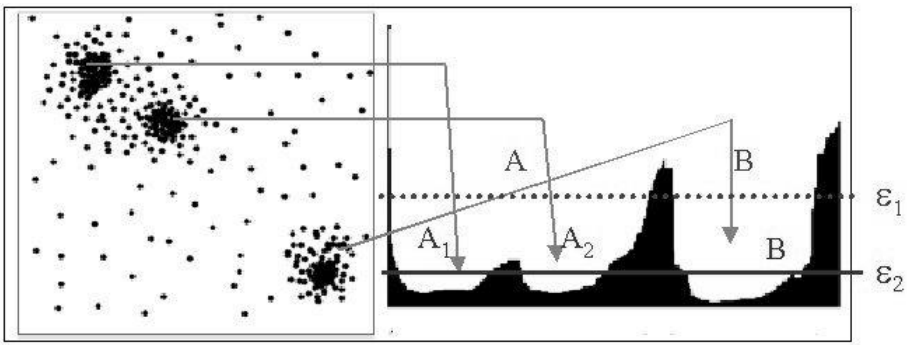


Figure 2.8: Reachability plot (right) computed by OPTICS for a sample 2D data set (left).

that q is a core point w.r.t. $\hat{\varepsilon}$ and k. The reachability distance of a point p w.r.t. another point q is the smallest threshold $\hat{\varepsilon} \leq \varepsilon$ such that p is directly density-reachable from q.

A great advantage of OPTICS is that the resulting cluster ordering can be visualized very intuitively and clearly by means of a so-called *reachability plot*. A reachability plot is a two-dimensional visualization of a cluster ordering, where the points are plotted according to the sequence specified in the cluster ordering along the x-axis, and for each point, the reachability distance along the y-axis. Figure 2.8 (right) depicts the reachability plot based on the cluster ordering computed by OPTICS for the sample two-dimensional data set in Figure 2.8 (left). Intuitively, clusters are "valleys" or "dents" in the plot, because sets of consecutive points with a lower reachability value are packed more densely. In particular, to manually obtain a density-based clustering w.r.t. any $\varepsilon' \leq \varepsilon$ by visual analysis, one simply has to cut the reachability plot at y-level ε' (i.e. parallel to the x-axis). The consecutive valleys in the plot below this cutting line contain the respective clusters. An example is presented in Figure 2.8 (right): 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. This illustrates, how the hierarchical cluster structure of a database is revealed at a glance and can be easily explored by visual inspection.

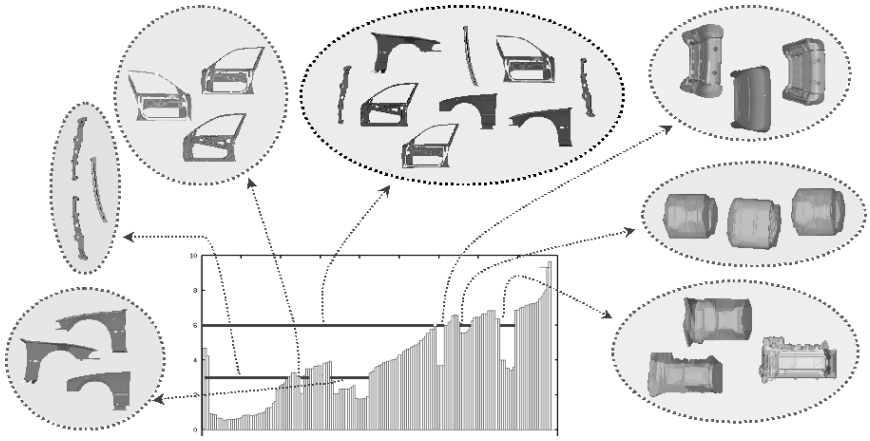


Figure 2.9: Browsing through cluster hierarchies.

Visually Mining through Cluster Hierarchies

In [BKKP04] the authors show how visualizing the hierarchical clustering structure of a database of objects can aid the user in his time consuming task to find similar objects. Based on reachability plots produced by OPTICS, approaches which automatically extract the significant clusters in a hierarchical cluster representation along with suitable cluster representatives are proposed. These techniques can be used as a basis for visual data mining. The resulting interactive browsing tool is called BOSS (*B*rowsing *O*PTICS-Plots for *S*imilarity *Search*), which utilizes automatic cluster recognition and extraction of cluster representatives in order to provide the user with significant and quick information (see Figure 2.9 for an illustration). The effectiveness and efficiency of this approach is for example shown for CAD objects from a German car manufacturer.

2.4 Advantages of Density-Based Clustering

As this thesis focuses on extensions to the density-connected clustering notion, we summarize here the most important advantages of the density-based clustering. In particular, density-based clustering algorithms provide the following advantages:

- They are able to find clusters of arbitrary size and shape.

- They can be used for all kinds of metric data spaces and are not confined to vector spaces.
- They are robust concerning outliers.
- They have proved to be very efficient and effective in clustering all sorts of data.
- Parallel [EKS+98, KKG03] and distributed [JKP04] versions enhance the efficiency.
- OPTICS is – in contrast to most other algorithms – relatively insensitive to its two input parameters ε and k. The authors in [ABKS99] state that the input parameters just have to be large enough to produce good results.

Part II

Clustering High-Dimensional Vector Data

Chapter 3

Density-Based Subspace Clustering

We start this chapter with a review of current approaches for clustering high-dimensional data. Such data sets often contain interesting clusters which are hidden in various subspaces of the original feature space. Therefore, the concept of subspace clustering has recently been addressed, which aims at automatically identifying subspaces of the feature space in which clusters exist. We introduce SUBCLU (density-connected *Sub*space *Clu*stering), an effective and efficient approach to the subspace clustering problem. SUBCLU is based on the concept of density-connectivity as described in Section 2.2. In contrast to existing grid-based approaches, it is able to detect arbitrarily shaped and positioned clusters in subspaces. The monotonicity of density-connectivity is used to efficiently prune subspaces in the process of generating all clusters in a bottom-up way. Parts of the material presented in this chapter have been published in [KKK04].

3.1 Introduction

Traditional clustering algorithms often fail to detect meaningful clusters because most real-world data sets are characterized by a high-dimensional, inherently sparse data space. Nevertheless, the data sets often contain interesting clusters which are hidden in various subspaces of the original feature space.

A common approach to cope with the "curse of dimensionality" for data mining tasks, such as clustering, are methods to reduce the dimensionality of the data space. In general, dimensionality reduction methods map the whole feature space onto a lower-dimensional subspace of relevant attributes in which clusters can be found. The feature selection is usually based on attribute transformations by creating functions of attributes. Examples of such functions are: principal component analysis (PCA), also called Karhunen-Loève transformation (KLT), used in multivariate statistics, e.g. [Jol86]; methods based on singular value decomposition (SVD) used in information retrieval, e.g. [BDL95], and in statistics, e.g. [Fuk90]; other transformations, for example based on wavelets [KCMP01] or low frequency Fourier harmonics in conjunction with Parseval´s theorem [AFS93].

However, dimensionality reduction methods have major drawbacks: First, the transformed attributes often have no intuitive meaning any more and thus the resulting clusters are hard to interpret. Second, in some cases, dimensionality reduction does not yield the desired results (e.g. [AGGR98] present an example where PCA/KLT does not reduce the dimensionality). Third, using dimensionality reduction techniques, the data is clustered only in a particular subspace. The information of points clustered differently in varying subspaces is lost.

A second approach for coping with clustering high-dimensional data is projected clustering which aims at computing k pairs $(C_i, S_i)_{(0 \leq i \leq k)}$ where C_i is a set of points representing the i-th cluster, S_i is a set of attributes spanning the subspace in which C_i exists (i.e. optimizes a given clustering criterion), and k is a user defined integer. Representative algorithms include PROCLUS [AP99] and ORCLUS [AY00] which are both related to k-means. While the projected clustering approach is more flexible than dimensionality

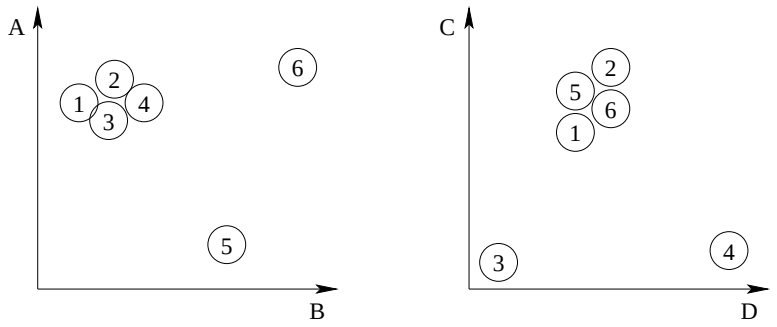


Figure 3.1: Drawback of the projected clustering approach.

reduction, it also suffers from the fact that the information of points which are clustered differently in varying subspaces is lost. Figure 3.1 illustrates this problem, using a feature space of four attributes A,B,C, and D. In the subspace AB the points 1 and 2 cluster together with points 3 and 4, whereas in the subspace CD they cluster with points 5 and 6. Either the information of the cluster in subspace AB or in subspace CD will be lost.

In recent years, the task of subspace clustering was introduced to overcome these problems. Subspace clustering is the task of automatically detecting clusters in subspaces of the original feature space. In this chapter, we introduce a density-connected approach to subspace clustering, overcoming the problems of existing approaches mentioned beneath. SUBCLU (density-connected *Sub*space *Clu*stering) is an effective answer to the problem of subspace clustering.

The remainder of the chapter is organized as follows. In Section 3.2, we review current subspace clustering algorithms and point out our contributions to subspace clustering. The application of the density-connected clustering notion to subspace clustering is presented in Section 3.3. Section 3.4 describes our algorithm SUBCLU in full detail. A broad experimental evaluation of SUBCLU based on artificial as well as on real-world data sets is presented in Section 3.5. Section 3.6 summarizes the chapter.

3.2 Related Work and Contributions

3.2.1 Discussion of Recent Approaches for Subspace Clustering

The pioneering approach to subspace clustering is CLIQUE (*CL*ustering *I*n *QUE*st) [AGGR98]. CLIQUE is a grid-based algorithm, using an *apriori*-like method to recursively navigate through the set of possible subspaces in a bottom-up way. The data space is first partitioned by an axis-parallel grid into equi-sized blocks of width ξ, called *units*. Only units whose densities exceed a threshold τ are retained. Both ξ and τ are the input parameters of CLIQUE. The bottom-up approach of finding such dense units starts with one-dimensional dense units. The recursive step from $(k-1)$-dimensional dense units to k-dimensional dense units takes $(k-1)$ dimensional dense units as candidates, and generates the k-dimensional units by self-joining all candidates having the first $(k-2)$-dimensions in common. All generated candidates which are not dense are eliminated. For efficiency reasons, a pruning criterion, called *coverage*, is introduced to eliminate dense units lying in less "interesting" subspaces as soon as possible. For deciding whether a subspaces is interesting or not, the Minimum Description Length principle is used. Naturally, this pruning bears the risk of missing some information. After generating all "interesting" dense units, clusters are found as a maximal set of connected dense units. For each k-dimensional subspace, CLIQUE takes all dense units of this subspace and computes disjoint sets of connected k-dimensional units. These sets are in a second step used to generate minimal cluster descriptions. This is done by covering each set of connected dense units with maximal regions and then determining the minimal cover.

A slight modification of CLIQUE is the algorithm ENCLUS (*EN*tropy-based *CLUS*tering) [CFZ99]. The major difference is the criterion used for subspace selection. The criterion of ENCLUS is based on an entropy computation of a discrete random variable. The entropy of any subspace S is high when the points are uniformly distributed in S, it is lower the more closely the points in S are packed. Subspaces with an entropy below an input threshold ω are considered as good for clustering. A monotonicity criterion

is presented which can be be used for a similar bottom-up algorithm as in CLIQUE [CFZ99].

A more significant modification of CLIQUE is presented in [GNC99, NGC01], introducing the algorithm called MAFIA (*M*erging of *A*daptive *F*inite *I*nterv*A*ls). MAFIA uses adaptive, variable-sized grids in each dimension. A dedicated technique based on histograms which aims at merging grid cells is used to reduce the number of bins compared to CLIQUE. An input parameter α is used as a so-called *cluster dominance factor* to select bins which are α-times more densely populated (relative to their volume) than the average. The algorithm starts to produce such one-dimensional dense units as candidates and proceeds recursively to higher dimensions. In contrast to CLIQUE, MAFIA uses any two k-dimensional dense units to construct a new $(k+1)$-dimensional candidate as soon as they share an arbitrary $(k-1)$-face (not only the first $(k-1)$ dimensions). As a consequence, the number of generated candidates is much larger compared to CLIQUE. Neighboring dense units are merged to form clusters. Redundant clusters, i.e. clusters that are true subsets of higher dimensional clusters, are removed.

A big drawback of all these methods is caused by the use of grids. In general, grid-based approaches heavily depend on the positioning of the grids. Figure 3.2(a) illustrates this problem for CLIQUE: Each grid cell by itself is not dense if $\tau > 4$, and thus, the cluster C is not found. On the other hand if $\tau = 4$, the cell with four points in the lower right corner just above the x-axis is reported as a cluster. Clusters may also be missed if they are inadequately oriented or shaped.

Another recent approach called DOC (*D*ensity-based *O*ptimal projective *C*lustering) [PJAM02] proposes a mathematical formulation for the notion of an optimal subspace cluster, regarding the density of points in subspaces. DOC is not grid-based but as the density of subspaces is measured using hypercubes of fixed width w, it has similar problems drafted in Figure 3.2(b). If a cluster is bigger than the hypercube, some points may be missed. Furthermore, the distribution inside the hypercube is not considered, and thus it need not necessarily contain only points of one cluster.

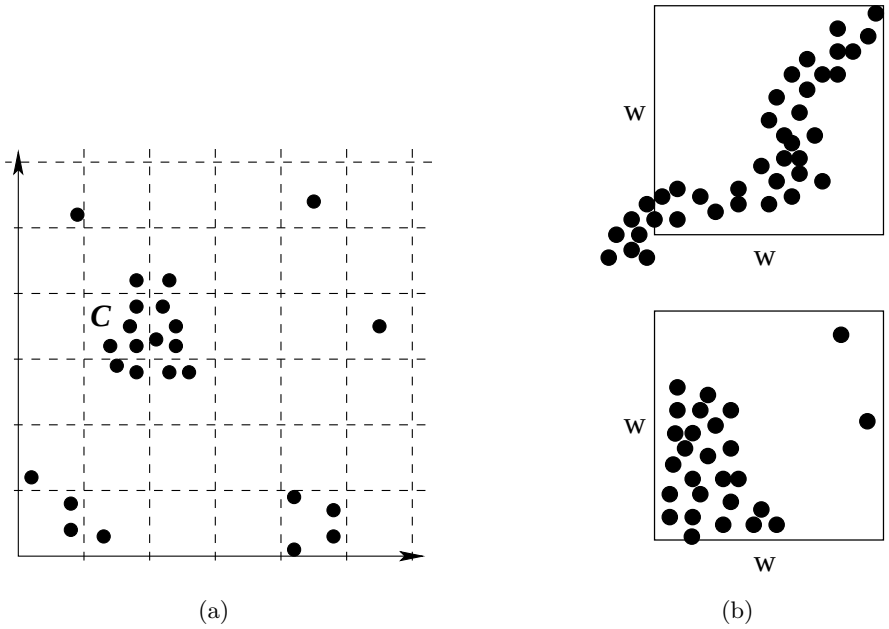


Figure 3.2: Drawbacks of existing subspace clustering algorithms.

3.2.2 Contributions

In this chapter, we propose a new approach which eliminates the problems mentioned above and enables the user to gain all the clustering information contained in high-dimensional data. Instead of using grids, we adopt the notion of density-connectivity to the subspace clustering problem. This has the following advantages:

- Our algorithm SUBCLU is able to detect arbitrarily shaped and positioned clusters in subspaces.

- In contrast to CLIQUE and its successors, the underlying cluster notion is well defined.

- Since SUBCLU does not use any pruning heuristics like CLIQUE, it provides for each subspace the same clusters as if DBSCAN is applied to this subspace.

3.3 Density-Connected Subspace Clustering

3.3.1 Clusters as Density-Connected Sets

Our approach SUBCLU is based on the formal definitions of density-connected clusters underlying the algorithm DBSCAN. The original formal definition of the clustering notion for the entire feature space were presented and discussed in Section 2.2. In the following, we adopt these definitions for the problem of subspace clustering.

Definition 3.1 (ε-neighborhood in a subspace)
Let $\varepsilon \in \mathbb{R}_0^+$, $S \subseteq \mathcal{A}$ and $o \in DB$. The ε-neighborhood *of o in S, denoted by $\mathcal{N}_\varepsilon^S(o)$, is defined by*

$$\mathcal{N}_\varepsilon^S(o) = \{x \in DB \mid dist(\pi_S(o), \pi_S(x)) \leq \varepsilon\}.$$

Definition 3.2 (core point in a subspace)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$, and $S \subseteq \mathcal{A}$. A point $o \in DB$ is called core point *in S, denoted by $\textsc{Core}_{\varepsilon,k}^S(o)$ if its ε-neighborhood in S contains at least k points, formally:*

$$\textsc{Core}_{\varepsilon,k}^S(o) \Leftrightarrow |\mathcal{N}_\varepsilon^S(o)| \geq k.$$

Definition 3.3 (direct density-reachability in a subspace)
Let $\varepsilon \in \mathbb{R}$, $k \in \mathbb{N}$, and $S \subseteq \mathcal{A}$. A point $p \in DB$ is directly density-reachable *from $q \in DB$ in S if q is a core point in S and p is an element of $\mathcal{N}_\varepsilon^S(q)$, formally:*

$$\textsc{DirReach}_{\varepsilon,k}^S(q,p) \Leftrightarrow \textsc{Core}_{\varepsilon,k}^S(q) \ \wedge \ p \in \mathcal{N}_\varepsilon^S(q).$$

Definition 3.4 (density-reachability in a subspace)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$, and $S \subseteq \mathcal{A}$. A point $p \in DB$ is density-reachable *from $q \in DB$ in S if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is directly density-reachable from p_i, formally:*

$$\begin{aligned}
&\textsc{Reach}_{\varepsilon,k}^S(q,p) \Leftrightarrow \\
&\exists p_1, \ldots, p_n \in DB : p_1 = q \ \wedge \ p_n = p \ \wedge \\
&\forall i \in \{1 \ldots n-1\} : \textsc{DirReach}_{\varepsilon,k}^S(p_i, p_{i+1}).
\end{aligned}$$

Definition 3.5 (density-connectivity in a subspace)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$, and $S \subseteq \mathcal{A}$. A point $p \in DB$ is density-connected *to a point $q \in DB$ in S if there is a point o such that both p and q are density-reachable from o, formally:*

$$\begin{aligned}&\textsc{Connect}^S_{\varepsilon,k}(q,p) \Leftrightarrow \\ &\exists o \in DB \,:\, \textsc{Reach}^S_{\varepsilon,k}(o,q) \,\wedge\, \textsc{Reach}^S_{\varepsilon,k}(o,p).\end{aligned}$$

Definition 3.6 (density-connected set in a subspace)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$, and $S \subseteq \mathcal{A}$. A non-empty subset $C \subseteq DB$ is called a density-connected set in *S if all points in C are density-connected in S, formally:*

$$\textsc{ConSet}^S_{\varepsilon,k}(C) \Leftrightarrow \forall o,q \in C \,:\, \textsc{Connect}^S_{\varepsilon,k}(o,q).$$

Finally, a density-connected cluster is defined as a set of density-connected points which is maximal w.r.t. density-reachability [EKSX96]. This definition can easily be adopted to clusters in a particular subspace.

3.3.2 Monotonicity of Density-Connected Sets

A straightforward approach would be to run DBSCAN in all possible subspaces to detect all density-connected clusters. The problem is that the number of subspaces is 2^d. A more effective strategy would be to use the clustering information of previous subspaces in the process of generating all clusters and drop all subspaces that cannot contain any density-connected clusters.

Unfortunately, density-connected clusters are not monotonic, i.e. if $C \subseteq DB$ is a density-connected cluster in subspace $S \subseteq \mathcal{A}$, it need not be a density-connected cluster in any $T \subseteq S$. The reason for this is that in T the density-connected cluster C may not be maximal w.r.t. density-reachability. There may be additional points which are not in C but are density-reachable in T from a point in C.

However, density-connected sets are monotonic. In fact, if $C \subseteq DB$ is a density-connected set in a subspace $S \subseteq \mathcal{A}$ then C is also a density-connected set in any subspace $T \subseteq S$.

Lemma 3.1 (monotonicity)
Let $\varepsilon \in \mathbb{R}_0^+$, $k \in \mathbb{N}$, $o, q \in DB$, $C \subseteq DB$, where $C \neq \emptyset$ and $S \subseteq \mathcal{A}$. Then the following monotonicity properties hold:

$$\forall T \subseteq S:$$

$$(1) \quad \textsc{Core}_{\varepsilon,k}^S(o) \Rightarrow \textsc{Core}_{\varepsilon,k}^T(o)$$

$$(2) \quad \textsc{DirReach}_{\varepsilon,k}^S(o,q) \Rightarrow \textsc{DirReach}_{\varepsilon,k}^T(o,q)$$

$$(3) \quad \textsc{Reach}_{\varepsilon,k}^S(o,q) \Rightarrow \textsc{Reach}_{\varepsilon,k}^T(o,q)$$

$$(4) \quad \textsc{Connect}_{\varepsilon,k}^S(o,q) \Rightarrow \textsc{Connect}_{\varepsilon,k}^T(o,q)$$

$$(5) \quad \textsc{ConSet}_{\varepsilon,k}^S(o,q) \Rightarrow \textsc{ConSet}_{\varepsilon,k}^T(o,q)$$

Proof.

(1)
$$\begin{aligned}
\textsc{Core}_{\varepsilon,k}^S(o) &\Leftrightarrow |\mathcal{N}_\varepsilon^S(o)| \geq k \\
&\Leftrightarrow |\{x \mid dist(\pi_S(o), \pi_S(x)) \leq \varepsilon\}| \geq k \\
&\Leftrightarrow |\{x \mid \sqrt[p]{\sum_{a_i \in S} (\pi_{a_i}(o) - \pi_{a_i}(x))^p} \leq \varepsilon\}| \geq k \\
&\overset{(T \subseteq S)}{\Rightarrow} |\{x \mid \sqrt[p]{\sum_{a_i \in T} (\pi_{a_i}(o) - \pi_{a_i}(x))^p} \leq \varepsilon\}| \geq k \\
&\Leftrightarrow |\{x \mid dist(\pi_T(o), \pi_T(x)) \leq \varepsilon\}| \geq k \\
&\Leftrightarrow |\mathcal{N}_\varepsilon^T(o)| \geq k \\
&\Leftrightarrow \textsc{Core}_{\varepsilon,k}^T(o)
\end{aligned}$$

(2)
$$\begin{aligned}
\textsc{DirReach}_{\varepsilon,k}^S(o,q) &\Leftrightarrow \textsc{Core}_{\varepsilon,k}^S(o) \wedge q \in \mathcal{N}_\varepsilon^S(o) \\
&\Leftrightarrow \textsc{Core}_{\varepsilon,k}^S(o) \wedge dist(\pi_S(o), \pi_S(q)) \leq \varepsilon \\
&\Leftrightarrow \textsc{Core}_{\varepsilon,k}^S(o) \wedge \sqrt[p]{\sum_{a_i \in S} (\pi_{a_i}(o) - \pi_{a_i}(q))^p} \leq \varepsilon \\
&\overset{(T \subseteq S)}{\Longrightarrow} {}^{(1)} \textsc{Core}_{\varepsilon,k}^T(o) \wedge \sqrt[p]{\sum_{a_i \in T} (\pi_{a_i}(o) - \pi_{a_i}(q))^p} \leq \varepsilon \\
&\Leftrightarrow \textsc{Core}_{\varepsilon,k}^T(o) \wedge dist(\pi_T(o), \pi_T(q)) \leq \varepsilon \\
&\Leftrightarrow \textsc{Core}_{\varepsilon,k}^T(o) \wedge q \in \mathcal{N}_\varepsilon^T(o)
\end{aligned}$$

$$\Leftrightarrow \textsc{DirReach}^T_{\varepsilon,k}(o,q)$$

(3) $\textsc{Reach}^S_{\varepsilon,k}(o,q)$

$$\begin{aligned}
&\Leftrightarrow \exists p_1,\ldots,p_n \in DB : p_1 = o \wedge p_n = q \\
&\quad \wedge \forall i \in \{1 \ldots n-1\} : \textsc{DirReach}^S_{\varepsilon,k}(p_i, p_{i+1}) \\
&\overset{(T \subseteq S)}{\Longrightarrow}{}^{(2)} \exists p_1,\ldots,p_n \in DB : p_1 = o \wedge p_n = q \\
&\quad \wedge \forall i \in \{1 \ldots n-1\} : \textsc{DirReach}^T_{\varepsilon,k}(p_i, p_{i+1}) \\
&\Leftrightarrow \textsc{Reach}^T_{\varepsilon,k}(o,q)
\end{aligned}$$

(4) $\textsc{Connect}^S_{\varepsilon,k}(o,q)$

$$\begin{aligned}
&\Leftrightarrow \exists x \in DB : \textsc{Reach}^S_{\varepsilon,k}(x,o) \wedge \textsc{Reach}^S_{\varepsilon,k}(x,q) \\
&\overset{(T \subseteq S)}{\Longrightarrow}{}^{(3)} \exists x \in DB : \textsc{Reach}^T_{\varepsilon,k}(x,o) \wedge \textsc{Reach}^T_{\varepsilon,k}(x,q) \\
&\Leftrightarrow \textsc{Connect}^T_{\varepsilon,k}(o,q)
\end{aligned}$$

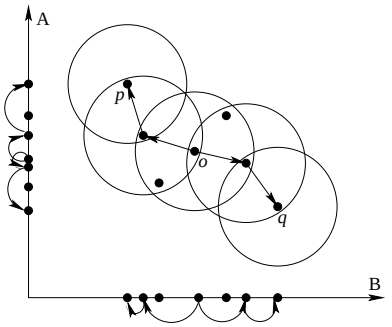
(5) $\textsc{ConSet}^S_{\varepsilon,k}(C)$

$$\begin{aligned}
&\Leftrightarrow \forall o,q \in C : \textsc{Connect}^S_{\varepsilon,k}(o,q) \\
&\overset{(T \subseteq S)}{\Longrightarrow}{}^{(4)} \forall o,q \in C : \textsc{Connect}^T_{\varepsilon,k}(o,q) \\
&\Leftrightarrow \textsc{ConSet}^T_{\varepsilon,k}(C)
\end{aligned}$$

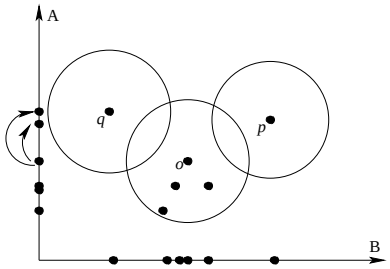
$\diamond$

The monotonicity of density-connectivity is illustrated in Figure 3.3. In Figure 3.3(a), p and q are density-connected via o in the subspace spanned by attributes A and B. Thus, p and q are also density-connected via o in each subspace A and B of AB. The inverse conclusion is depicted in Figure 3.3(b): p and q are not density-connected in subspace B. Thus, they are also not density-connected in the superspace AB although they are density-connected in subspace A via o.

The inversion of Lemma 3.1(5) is the key idea for an efficient bottom-up algorithm to detect the density-connected sets in all subspaces of high-dimensional data. We do not have to examine any subspace S if at least one $T_i \subset S$ contains no cluster, i.e. no density-connected set. On the other hand, we have to test each subspace S if all $T_i \subset S$ contain clusters whether those clusters are maintained.



(a) p and q are density-connected via o in AB.



(b) p and q are not density-connected in AB.

Figure 3.3: Monotonicity of density-connectivity (the circles indicate the ε-neighborhoods, $k = 4$).

3.4 The Algorithm SUBCLU

SUBCLU is based on a bottom-up, greedy algorithm to detect the density-connected clusters in all subspaces of high-dimensional data. The algorithm is presented in Figure 3.4. The following data structures are used:

- DB^S denotes the database DB projected onto the subspace S.
- $\mathcal{C}^S$ denotes the set of all density-connected clusters of DB in the subspace S w.r.t. ε and k, and can be computed by the method DBSCAN, i.e. $\mathcal{C}^S := DBSCAN(DB^S, \varepsilon, k)$. Note that we assume here that the noise set is not included in $\mathcal{C}^S$.
- $\mathcal{S}_l$ denotes the set of all l-dimensional subspaces, containing at least one cluster, i.e. $\mathcal{S}_l := \{S \subseteq A \,|\, |S| = l \text{ and } \mathcal{C}^S \neq \emptyset\}$.
- $\mathcal{C}_l$ denotes the set of sets of all clusters in l-dimensional subspaces, i.e. $\mathcal{C}_l := \{\mathcal{C}^S \,|\, S \subseteq A \text{ and } |S| = l\}$.

We begin with generating all one-dimensional clusters by applying DBSCAN to each one-dimensional subspace (STEP 1 in Figure 3.4).

For each detected cluster, we have to check whether this cluster is (or parts of it are) still existent in higher dimensional subspaces. Due to Lemma 3.1, no other cluster can exist in higher dimensional subspaces. Thus, we

```
SUBCLU(SetOfPoints DB, Real ε, Integer k)

  // STEP 1 Generate all 1D clusters
  S_1 := ∅   // set of 1D subspaces containing clusters
  C_1 := ∅   // set of all sets of clusters in 1D subspaces
  for each a_i ∈ A do
      C^{a_i} := DBSCAN(DB^{a_i}, ε, k)   // set of all clusters in subspace a_i;
      if C^{a_i} ≠ ∅ then   // at least one cluster in subspace {a_i} found
          S_1 := S_1 ∪ {a_i};
          C_1 := C_1 ∪ C^{a_i};
      end if
  end for

  // STEP 2 Generate (l + 1)-D clusters from l-D clusters
  l := 1;
  while C_l ≠ ∅

      // STEP 2.1 Generate (l + 1)-D candidate subspaces
      CandS_{l+1} := GenerateCandidateSubspaces(S_l);

      // STEP 2.2 Test candidates and generate (l + 1)-D clusters
      for each cand ∈ CandS_{l+1} do
          // Search l-dim subspace of cand with minimal number of points in the clusters
          bestSubspace := ArgMin_{s∈S_l ∧ s⊆cand} Σ_{C_i∈C^s} |C_i|
          C^{cand} := ∅;
          for each cluster cl ∈ C^{bestSubspace} do
              C^{cand} = C^{cand} ∪ DBSCAN(cl^{cand}, ε, k);
              if C^{cand} ≠ ∅ then
                  S_{l+1} := S_{l+1} ∪ cand;
                  C_{l+1} := C_{l+1} ∪ C^{cand};
              end if
          end for
      end for
      l := l + 1
  end while
```

Figure 3.4: The algorithm SUBCLU.

```
GenerateCandidates(SetOfSubspaces S_l)
    // STEP 2.1.1 Generate (l+1)-D candidate subspaces
    CandS_{l+1} := ∅;
    for each s_1 ∈ S_l do
        for each s_2 ∈ S_l do
            if s_1.attr_1 = s_2.attr_1 ∧ ... ∧ s_1.attr_{l-1} = s_2.attr_{l-1} ∧ s_1.attr_l < s_2.attr_l
            then insert {s_1.attr_1, ..., s_1.attr_l, s_2.attr_l} into CandS_{l+1};
        end for
    end for
    // STEP 2.1.2 Prune irrelevant candidates subspaces
    for each cand ∈ CandS_{l+1} do
        for each s ⊂ cand with |s| = l do
            if s ∉ S_l then delete cand from CandS_{l+1};
            end if
        end for
    end for
```

Figure 3.5: The procedure GenerateCandidates.

search for each l-dimensional subspace $S \in \mathcal{S}_l$ all other l-dimensional subspaces $T \in \mathcal{S}_l$ having $(l-1)$ attributes in common and join them to generate $(l+1)$-dimensional candidate subspaces (STEP 2.1.1 of the procedure GenerateCandidates in Figure 3.5). The set of $(l+1)$-dimensional candidate subspaces is denoted by $CandS_{l+1}$.

For each candidate subspace $S \in CandS_{l+1}$, $\mathcal{S}_l$ must contain each l-dimensional subspace $T \subset S$, $|T| = l$. (cf. Lemma 3.1). Consequently, we can prune all candidates having at least one l-dimensional subspace not included in $\mathcal{S}_l$ (STEP 2.1.2 of procedure GenerateCandidates in Figure 3.5). This reduces the number of $(l+1)$-dimensional candidate subspaces.

In the last step (STEP 2.2 in Figure 3.4), we generate the $(l+1)$-dimensional clusters and the corresponding $(l+1)$-dimensional subspaces, containing these clusters. To do so, we use the l-dimensional subclusters and the list of $(l+1)$-dimensional candidate subspaces. For each candidate subspace $cand \in CandS_{l+1}$, we take one l-dimensional subspace $T \subset cand$ and simply call the procedure $DBSCAN(cl^{cand}, \varepsilon, k)$ for each cluster cl in T ($cl \in \mathcal{C}^T$) to generate $\mathcal{C}^{cand}$. To minimize the cost of the runs of DBSCAN in $cand$, we choose that subspace $bestSubspace \subset cand$ from $\mathcal{S}_l$ in which a

minimum number of points is in the cluster, i.e.

$$bestSubspace := \underset{s \in \mathcal{S}_l \wedge s \subseteq cand}{\text{ArgMin}} \sum_{C_i \in \mathcal{C}^s} |C_i|.$$

This minimize the number of necessary range queries during the runs of DBSCAN in S. If $\mathcal{C}^S \neq \emptyset$, we add it to $\mathcal{C}_{l+1}$ and add S to $\mathcal{S}_{l+1}$.

Steps 2.1 to 2.3 are recursively executed as long as the set of l-dimensional subspaces containing clusters is not empty.

The most time consuming part of our algorithm is the execution of all the partial range queries on arbitrary subspaces of the data space. As DBSCAN is applied to different subspaces, an index structure for the full-dimensional data space is not applicable. Therefore, we apply the approach of inverted files. Our algorithm provides an efficient index support for range queries on each single attribute in logarithmic time. For range queries on more than one attribute, we apply the range query to each separate attribute (index structure) and generate the intersection of all intermediate results to obtain the final result.

3.5 Performance Evaluation

We tested SUBCLU using several synthetic data sets and a real-world gene expression data set. All experiments were run on a workstation with a 1.7 GHz processor and 2 GB RAM.

3.5.1 Data Sets

We tested SUBCLU using synthetic as well as real-world gene expression data sets.

Synthetic Data Sets. The synthetic data sets were generated by a data generator. It permits to control the size and structure of the generated data sets through parameters such as number and dimensionality of subspace clusters, dimensionality of the feature space and density parameters for the whole data set as well as for each cluster. In a subspace that contains a cluster, the average density of data points in that cluster is much larger than the density of points not belonging to the cluster in this subspace. In

addition, it is ensured that none of the synthetically generated data sets contain cluster in the full-dimensional space.

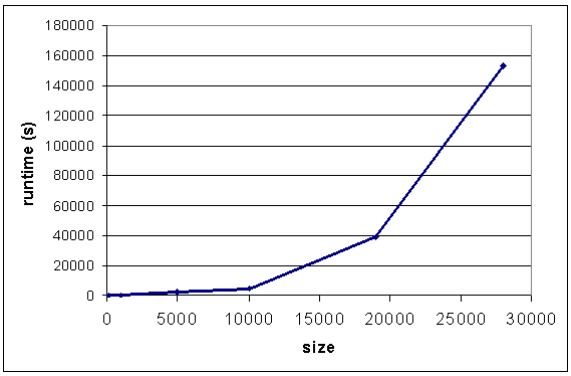
Spellman Data Set. The gene expression data set [SSZ+98] studies the yeast mitotic cell cycle. We used only the data set of the CDC15 mutant. The expression level of 6,000 genes was measured at 24 different time slots. Since some genes have missing expression values and the handling of missing values in gene expression analysis is a non-trivial task, we eliminated those genes from our test data set. The resulting data set contains around 4,000 genes expressed at 24 different time slots.

3.5.2 Efficiency

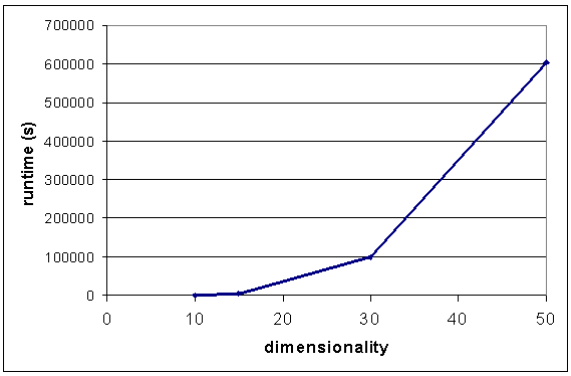
We evaluated the efficiency of SUBCLU using several synthetic data sets. All tests were run with $k = 8$ and $\varepsilon = 2.0$.

The scalability of SUBCLU w.r.t. the size of the data set, the dimensionality of the data set and the dimensionality of the hidden subspace clusters are depicted in Figure 3.5.2. In all three cases, SUBCLU grows with an at least quadratic factor. The reason for this scalability w.r.t. the size of the data set is that SUBCLU performs multiple range queries in arbitrary subspaces. As mentioned above, we can only support these queries, using inverted files, since there is no index structure that can support partial range queries in average case logarithmic time. The scalability w.r.t. the dimensionality of the data set and w.r.t. the hidden subspaces can be explained by the *apriori*-like bottom-up greedy algorithm used to navigate through the space of all possible subspaces.

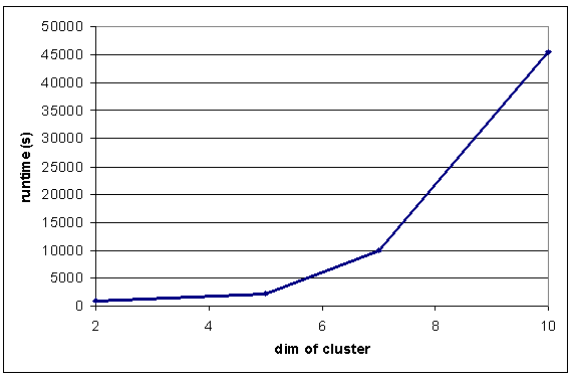
Let us note that we also implemented a parallel version of SUBCLU to improve its scalability. All relevant subspaces of a certain dimensionality can be processed parallel. Thus, the most time consuming part of SUBCLU, the runs of DBSCAN in step 2.2 of our algorithm can be executed parallel. Using a two processor workstation, the parallel version was on average by a factor of 1.86 faster. Of course, the runtimes can be further decreased if more than two processors are available.



(a) Scalability w.r.t. the size of the data set.



(b) Scalability w.r.t. the dimensionality of the data set.



(c) Scalability w.r.t. the maximum dimensionality of the hidden subspace clusters.

Figure 3.6: Scalability of SUBCLU.

Data set	d	dim. of clusters	N	# input clusters	true clusters found by	
					SUBCLU	CLIQUE
DS01	10	4	18999	1	1	1
DS02	10	4	27704	1	1	1
DS03	15	5,5	3802	3	3	1
DS04	15	3,5,7	4325	3	2	1
DS05	15	5,5,5	4057	3	3	1
DS06	15	4,4,6,7,10	2671	6	5	2

Table 3.1: Comparative evaluation of SUBCLU and CLIQUE: summary of the results on synthetic data sets.

3.5.3 Accuracy

To evaluate the effectivity of SUBCLU, we compared it to CLIQUE [AGGR98]. Since CLIQUE is a product of IBM and its code is not easy to obtain, we re-implemented CLIQUE according to [AGGR98]. In all accuracy experiments, we run CLIQUE with a broad range of parameter settings and took only the best results.

We applied SUBCLU and CLIQUE to several synthetic data sets which we generated as described above. In each data set, several clusters are hidden in subspaces of varying dimensionality. The results are depicted in Table 3.1. In almost all cases, SUBCLU computed the artificial clusters whereas CLIQUE had difficulties in detecting all patterns properly.

We also applied SUBCLU to the Spellman data set in order to find co-expressed genes. SUBCLU found many interesting clusters in several subspaces of this data set. The most interesting clusters were found in the subspaces spanned by time slots 90, 110, 130, and 190 as well as time slots 190, 270, and 290. The functional relationships of the genes in the resulting clusters were investigated by using the public yeast genome database at the Stanford University (Saccharomyces Genome Database, SGD: `http://www.yeastgenome.org/`).

The contents of four sample clusters in two different subspaces are depicted in Table 3.2. The first cluster (in subspace spanned by time slots 90, 110, 130, 190) contains several genes which are known to play a role dur-

Gene Name	**Function**
Cluster 1 (subspace 90, 110, 130, 190)	
RPC40	subunit of RNA pol I and III, builds complex with CDC60
CDC60	tRNA synthesase, builds complex with RPC40
FRS1	tRNA synthesase
DOM34	protein synthesis, mitotic cell cycle
CKA1	mitotic cell cycle control
CPA1	control of translation
MIP6	RNA binding activity, mitotic cell cycle
Cluster 2 (subspace 90, 110, 130, 190)	
STE12	transcription factor (regulation of cell cycle)
CDC27	regulation of cell cycle, possible STE12-site
EMP47	Golgi membrane protein, possible STE12-site
XBP1	Transcription factor
Cluster 3 (subspace 90, 110, 130, 190)	
CDC25	starting control factor for mitosis
MYO3	control/regulation factor for mitosis
NUD1	control/regulation factor for mitosis
Cluster 4 (subspace 190, 270, 290)	
RPT6	protein catabolism; builds complex with RPN10
RPN10	protein catabolism; builds complex with RPT6
UBC1	protein catabolism; subunit of 26S protease
UBC4	protein catabolism; subunit of 26S protease
MRPL17	component of mitochondrial large ribosomal subunit
MRPL31	component of mitochondrial large ribosomal subunit
MRPL32	component of mitochondrial large ribosomal subunit
MRPL33	component of mitochondrial large ribosomal subunit
SNF7	direct interaction with VPS2
VPS4	mitochondrial protein; direct interaction with SNF7

Table 3.2: Contents of four sample clusters in different subspaces.

ing the cell cycle, e.g. DOM34, CKA1, CPA1, and MIP6. In addition, the products of two genes in that cluster are part of a common protein complex. The second cluster contains the gene STE12, identified by [SSZ+98] as an important transcription factor for the regulation of the mitotic cell cycle. In addition, the genes CDC27 and EMP47 which have possible STE12-sites and are most likely co-expressed with STE12 are in that cluster. The third cluster consists of the genes CDC25 (starting point for mitosis), MYO3 and NUD1 (known for an active role during mitosis) as well as various other transcription factors, e.g. CHA4, ELP3, required during the cell cycle. The fourth cluster contains several mitochondrion related genes which have similar functions. For example, the genes MRPL17, MRPL31, MRPL32, and MRPL33 are four mitochondrial large ribosomal subunits, the genes UBC1 and UBC4 are subunits of a certain protease, and the genes SNF7 and VPS4 are direct interaction partners. This indicates a higher mitochondrial activity at these time slots which might be explained by a higher demand of biological energy during the cell cycle (the energy metabolism is located in mitochondrion).

Let us note that the described four clusters are only a representative glance at the results SUBCLU yields when applied to the gene expression data set. Each cluster contains additional genes with yet unknown function. We also detected few clusters with no significant functional relationship among the grouped genes. However, most of the resulting clusters contained functional related genes, indicating that the detected co-expression is biological meaningful. Since most clusters also contain genes which do not have any annotated function yet, the results of SUBCLU might propose a biologically interesting prediction for these unknown genes.

We also applied CLIQUE to the gene expression data set. We again tested a broad range of parameter settings and compared SUBCLU to the best results of CLIQUE. Since the parameter ξ of CLIQUE (width of grid cells) affects the runtime of CLIQUE heavily, we were forced to run CLIQUE with rather low values for ξ. As a consequence, CLIQUE was not able to find any reasonable clusters in the gene expression data set. Let us note that a more efficient implementation of CLIQUE would enable a better parameter setting, i.e. higher values for ξ, and would thus also detect some

of the clusters computed by SUBCLU. On the other hand, in real-world data sets, such as gene expression data, it is most likely that the clusters are not axis-parallel hypercubes. Thus, SUBCLU is much more suitable than CLIQUE, due to the fact that the density-connected clustering notion underlying SUBCLU is able to detect arbitrarily shaped (subspace) clusters.

3.6 Summary

In this chapter, we presented SUBCLU, a density-based subspace clustering algorithm for detecting clusters in high-dimensional data. Built on an adaption of the density-connected notion of clusters underlying the algorithm DBSCAN, we developed an efficient greedy algorithm to compute all density-connected sets hidden in subspaces of high-dimensional data. A comparison with CLIQUE empirically showed that SUBCLU outperforms state-of-the-art subspace clustering algorithms concerning the quality. An application of SUBCLU to real-world gene expression data yields biologically interesting and meaningful results, and thus demonstrates the very usefulness of SUBCLU.

Chapter 4

Density-Based Subspace Ranking

A drawback of subspace clustering algorithms like SUBCLU and CLIQUE is the use of a global density threshold. As there is no obvious solution for an efficient hierarchical extension of SUBCLU or CLIQUE, we propose another solution. In this chapter we present a pre-processing step for traditional clustering algorithms which detects all interesting subspaces of high-dimensional data containing clusters. Afterwards any clustering algorithm can be applied, especially a hierarchical clustering algorithm like OPTICS. We define a quality criterion for the interestingness of a subspace and propose an efficient algorithm called RIS (*R*anking *I*nteresting *S*ubspaces) to detect all such subspaces. A broad evaluation based on synthetic and real-world data sets empirically shows that RIS is suitable to find all relevant subspaces in large, high-dimensional, sparse data and to rank them accordingly. The basic ideas contained in this chapter have been published in [KKKW03].

4.1 Introduction

The drawback of subspace clustering algorithms like SUBCLU and CLIQUE is the use of a global density threshold. As we have seen in Section 2.3, using the hierarchical density-based algorithm OPTICS [ABKS99] clusters of different density can be found in a single run of the algorithm. However, the global density threshold used for SUBCLU leads to the fact that similar to DBSCAN, SUBCLU is not able to detect clusters of different density in one single run of the algorithm. The problem is that we need the global density parameters to maintain the monotonicity which is needed for the efficiency. In this chapter, we propose a first approach to overcome this problem. We present a preprocessing step which selects all interesting subspaces, using again a density-connected clustering notion. Thus, we are able to detect all subspaces containing clusters of arbitrary size and shape. The remainder of this chapter is organized as follows. After shortly reviewing some related work in Section 4.2, we define the "interestingness" of subspaces in Section 4.3 and provide a quality criterion to rank the subspaces according to their interestingness. Afterwards any clustering algorithm, especially the hierarchical density-based algorithm OPTICS, can be applied to these subspaces. In Section 4.4, we present an efficient algorithm called RIS (*R*anking *I*nteresting *S*ubspaces) for computing all those subspaces. A broad experimental evaluation of RIS in combination with OPTICS based on artificial as well as on gene expression data is presented in Section 4.5. Section 4.6 summarizes the chapter.

4.2 Related Work

In [DCSL02] a quality criterion for subspaces based on the entropy of point-to-point distances is introduced. However, there is no algorithm presented to compute the interesting subspaces. The authors propose to use a forward search strategy which most likely will miss interesting subspaces or an exhaustive search strategy which is obviously not efficient in higher dimensional spaces. An experimental comparison with this technique can be found in Section 5.4. For related work on subspace clustering refer to Section 3.2.

4.3 Ranking Interesting Subspaces

4.3.1 Interestingness of a Subspace

Our approach to rate the interestingness of subspaces is again based on a density-based notion of clusters. We use the core object property to decide about the interestingness of a subspace. Obviously, if a subspace contains no core object, it contains no dense region (cluster) and therefore contains no relevant information for a density-based clustering algorithm.

Observation 1 *The number of core objects of a data set DB (w.r.t. ε and k) is proportional to the number of different clusters in DB and/or the size of the clusters in DB and/or the density of clusters in DB.*

This observation can be used to rate the interestingness of subspaces. However, summing up all the core objects for each subspace delivers not enough information. Even if two subspaces contain the same number of core objects, the quality may differ. This is due to the fact that dense regions contain objects which are no core objects but lie within the ε-neighborhood of a core object and are thus, an essential part of the dense region. Therefore, it is not only interesting how many core objects a subspace contains, but also how many objects lie within the ε-neighborhood of these core objects. In the following, the variable $count[S]$ denotes the sum of all points lying in the ε-neighborhood of all core objects in the subspace S. The number of core objects in S is denoted by $core[S]$. If we measure the interestingness of a subspace S according to its $count[S]$ value and rank all subspaces according to this quality value, two problems are not addressed. The first problem is that naturally with each dimension the number of expected objects in the ε-neighborhood of an object decreases and thus, this naive quality value favors lower dimensional subspaces over higher dimensional ones. To overcome this problem, we introduce a scaling coefficient $count[T_{uniform}]$ that takes the dimensionality of the subspace S into account. We compute the value $count[T_{uniform}]$ assuming that T has the same number of objects and the same dimensionality d as S and all objects in T are uniformly distributed. For that purpose, we compute the volume of a d-dimensional

ε-neighborhood, denoted by Vol_ε^d and the number of objects lying in Vol_ε^d, assuming uniform distribution.

Definition 4.1 (quality of a subspace)
The quality of a subspace S, measuring the interestingness of S is defined by:

$$\text{QUALITY}(S) = \frac{count[S]}{n \cdot \frac{Vol_\varepsilon^{dim[S]} \cdot n}{attrRange^{\dim[S]}}}.$$

If *dist* is the L_∞-norm, Vol_ε^d is a hypercube and can be computed by $Vol_\varepsilon^d = (2\varepsilon)^d$, or if *dist* is the Euclidian distance (L_2-norm), Vol_ε^d is a hypersphere and can be computed as given below:

$$Vol_\varepsilon^d = \frac{\sqrt{\pi^d}}{\Gamma(d/2+1)} \cdot \varepsilon^d$$

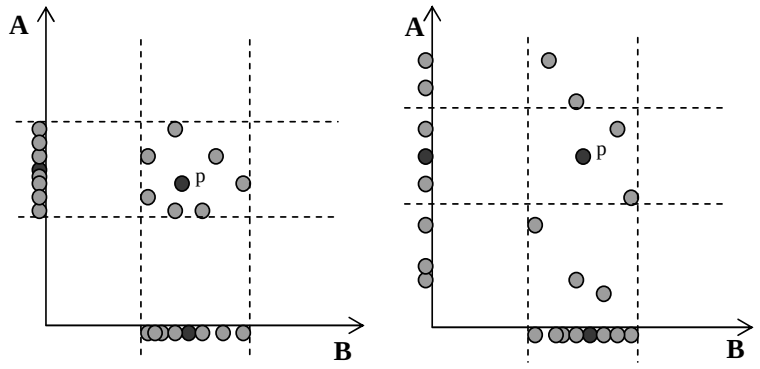
where $\Gamma(x+1) = x \cdot \Gamma(x), \quad \Gamma(1) = 1 \quad$ and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$.

The second problem is the phenomenon that in high-dimensional spaces more and more points are located on the boundary of the data space. The ε-neighborhoods of these objects are smaller because they exceed the borders of the data space. In [BBKK97] the authors show that the average volume of the intersection of the data space and a hypersphere with radius ε can be expressed as the integral of a piecewise defined function that is integrated over all possible positions of the ε-neighborhood, i.e the core objects. For our implementation, we choose a less complex heuristic to eliminate this effect that is based on periodical extensions of the data space (cf. Section 4.4.2 for details).

For two arbitrary subspaces $U, V \in \mathbb{R}^d$, our quality criterion has two complementary effects which are summarized in the following observation:

Observation 2 *Let $U \supset V$. Then the following inequalities hold:*

1. *$core[U] \leq core[V]$ and $count[U] \leq count[V]$.*
2. *If $core[U] = core[V]$ and $count[U] = count[V]$ then* $\text{QUALITY}(U) >$ $\text{QUALITY}(V)$.



(a) p core object in space AB → p core object in subspace A and in subspace B.

(b) p no core object in subspace A → p no core object in space AB.

Figure 4.1: Visualization of Lemma 3.1 (1) for $k = 5$ (2D feature space).

The first observation states that, while navigating through the subspaces in a bottom-up way, at a certain point the core objects loose their core object property due to the addition of irrelevant features and thus the quality decreases. On the other hand, as long as we add relevant features (features preserving the clustering structure) the quality increases.

4.3.2 General Idea of Finding Interesting Subspaces

A straightforward approach would be to examine all possible subspaces, e.g. bottom-up. The problem is that the number of subspaces is 2^d. Basically all subspaces that do not contain any core object can be dropped since they cannot contain any clusters. Furthermore, the core object condition is decreasing strictly monotonic as we have seen in Lemma 3.1(1).

If an object o is a core object in S, then it is also a core object in any subspace $T \subseteq S$ w.r.t. the same ε and k. This is visualized in Figure 4.1(a). The reverse conclusion is illustrated in Figure 4.1(b) and states: If an object o is not a core object in T, then o is also not a core object in any superspace $S \supset T$.

How this property helps to eliminate a lot of subspaces in the process of generating all relevant subspaces in a bottom-up process will be presented in the next sections.

```
RIS(SetOfPoints DB, Real ε, Integer k)
    Subspaces := emptySet;
    for i from 1 to DB.size() do
        Point := DB.get(i);
        RelevantSubspaces := GenerateSubspaces(Point,DB);
        Subspaces.add(RelevantSubspaces);
    end for
    Subspaces.prune();
    Subspaces.sort();
```

Figure 4.2: The algorithm RIS.

4.4 Implementation of RIS

4.4.1 Algorithm

Given a set of objects DB and density parameters ε and k, RIS finds all interesting subspaces and presents them to the user sorted according to their quality. The pseudocode of the algorithm RIS is given in Figure 4.2. For each object $o \in DB$, RIS computes a set of relevant subspaces, i.e. all subspaces in which the core object condition holds for o. This step will be described in detail in Section 4.4.2. Let us note that the algorithm can also be applied to a sample of DB, e.g. for performance reasons (cf. Section 4.5.2). For each detected subspace, statistical data are accumulated and this information is merged for all objects. The detected subspaces are then pruned according to certain criteria. In Section 4.4.3, these criteria will be discussed. Finally, the subspaces are sorted for a more comprehensible user presentation. The clustering in these subspaces can then be done by any clustering algorithm.

4.4.2 Efficient Generation of Subspaces

For a given object $o \in DB$, the method `GenerateSubspaces` finds all subspaces S in which the core object condition holds w.r.t. ε and k. Formally, it computes the following set: $K_o := \{T \subseteq \mathcal{A} \mid \quad \textsc{Core}^T_{\varepsilon,k}(o)\}$.

For the L_∞-norm as distance function, the problem of finding the set K_o is equivalent to the problem of determining all frequent item sets in the context of mining association rules [AS94] and thus can be computed rather

efficiently:

For each $x \in DB$ a transaction $T_x \subseteq \mathcal{A}$ is defined such that

$$a_i \in T_x \Leftrightarrow |\, \pi_{a_i}(x) - \pi_{a_i}(o)\,| \leq \varepsilon \quad \text{for all } i \in \{1, \ldots, d\}.$$

Lemma 4.1

$K_o = \{T \subseteq \mathcal{A} \mid \mathrm{Supp}_{DB}(T) \geq \frac{k}{|DB|}\}$ *where* $\mathrm{Supp}_{DB}(T) = \frac{|\{x \in DB \mid T \subseteq T_x\}|}{|DB|}$

Proof.

$T \subseteq \mathcal{A} \wedge |N_\varepsilon^T(o)| \geq k$

$\Leftrightarrow T \subseteq \mathcal{A} \wedge |\{x \in DB \mid dist_{L_\infty}(\pi_T(o), \pi_T(x)) \leq \varepsilon\}| \geq k$

$\Leftrightarrow T \subseteq \mathcal{A} \wedge$

$\quad |\{x \in DB \mid \forall i \in \{1, \ldots, d\} : a_i \in T \Rightarrow |\pi_{a_i}(o) - \pi_{a_i}(x)| \leq \varepsilon\}| \geq k$

$\Leftrightarrow T \subseteq \mathcal{A} \wedge |\{x \in DB \mid T \subseteq T_x\}| \geq k \Leftrightarrow T \subseteq \mathcal{A} \wedge \mathrm{Supp}_{DB}(T) \geq \frac{k}{|DB|}$ ◇

The method `GenerateSubspaces` extends the familiar *Apriori* [AS94] algorithm in accumulating the statistical information for measuring the subspace quality, using the monotonicity of the core object condition (cf. Lemma 3.1). As mentioned before, we are extending the data space periodically to ensure that all ε-neighborhoods have the same size. This can be done very easily by changing the way the transactions are defined. Instead of only checking if $|\pi_{a_i}(x) - \pi_{a_i}(o)| \leq \varepsilon$, we have to check if $|\pi_{a_i}(x) - \pi_{a_i}(o)| \leq \varepsilon$ or $|\pi_{a_i}(x) - \pi_{a_i}(o)| \geq attrRange - \varepsilon$.

Let us note that the use of L_∞-norm is no serious constraint. Obviously, all objects that lie within the ε-neighborhood of an object according to the L_2 norm (Euclidean distance) also lie within the ε-neighborhood according to the L_∞ norm (cf. Figure 4.3). It follows that using the L_∞ norm, we will find at least those core objects (and thus those subspaces containing clusters) which we would find when using the L_2 norm. The only difference is that, using the L_∞ norm, we may find additional core objects. However, the additional subspaces which would not have been found when using the L_2 norm have low quality values, anyway. In other words, using the L_∞ norm, we obtain compared to any other L_p norm, $p < \infty$, false positives but no false drops.

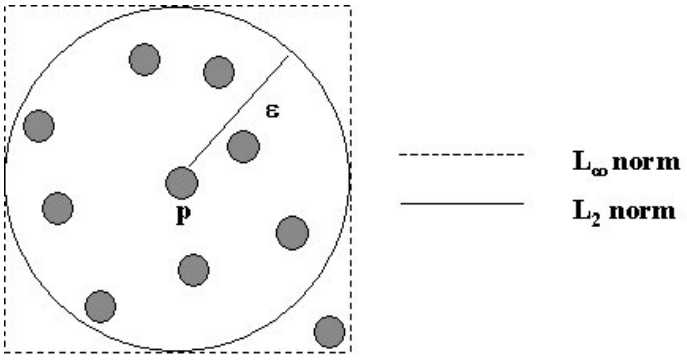


Figure 4.3: ε-neighborhood of a sample core object p (e.g. for $k = 5$) using L_∞ and L_2 norm.

4.4.3 Pruning of Subspaces

To reduce the number of computed subspaces we perform the following two pruning steps.

Upward Pruning. If the quality value of a $(k-1)$-dimensional subspace is high, then adding a noise attribute to this subspace may still result in a relatively high quality value. As, in this case, we are not interested in the resulting k-dimensional subspace, we can perform the following heuristic upward pruning. Let S be a k-dimensional attribute space and $S_{k-1} := \{T \,|\, T \subset S \wedge \dim[T] = k-1\}$ be the set of all $(k-1)$-dimensional subspaces of S. Let $\overline{count}$ be the mean *count* value of all $T \in S_{k-1}$ and $\overline{s}$ be the standard deviation. Let $maxdiff := \max_{T \in S_{k-1}} (\,|\, count[T] - \overline{count}|\,)$ be the maximum deviation of the *count* values of all $T \in S_{k-1}$ from the mean *count* value. Then, the so-called *bias-value* can be computed as follows: $bias = \frac{\overline{s}}{maxdiff}$. If this bias-value falls below a certain threshold, we prune the k-dimensional subspace S. Our experimental evaluations indicate that 0.56 is a good value for this bias-criterion.

Downward Pruning. As we are only interested in the subspaces with the highest quality, we can perform the following downward pruning step to eliminate redundant subspaces: If there is a $(k+1)$-dimensional subspace S with higher quality than the k-dimensional subspace T ($S \supset T$), we delete T.

4.4.4 Determination of Density Parameters

A heuristic method, which we experimentally verified to be sufficient, suggests $k \approx \ln(n)$ where n is the size of the database. Then, ε must be chosen depending on the value of k. In [EKSX96] a simple heuristics is presented to determine the ε of the "thinnest" cluster in the database (for a given k). But as we do not know beforehand in which subspaces clusters will be found, we cannot determine ε to find a single subspace with one particular clustering. Quite the contrary, we want to choose the parameters such that RIS detects subspaces which might have clusters of different density and different dimensionality.

However, we can determine an upper bound for ε for a given value of k. If we take uniform distribution as worst case, the ε-neighborhood of an object should not contain more than $k-1$ objects in the full-dimensional space. Otherwise, all objects are core objects. In case of the L_∞-norm, an upper bound for ε can be computed as follows:

$$n \cdot \frac{Vol_\varepsilon^d}{attrRange^{dim}} < k \quad \overset{L_\infty}{\Longrightarrow} \quad \varepsilon < \frac{attrRange}{2} \cdot \sqrt[dim]{\frac{k}{n}}$$

where $dim = d$. If we have any knowledge about the dimensionality of the subspaces we want to find, we can further decrease the upper bound by setting dim to the highest dimension of such a subspace.

This upper bound is very rough. Nevertheless, it provides a good indication for the choice of ε. Indeed, it empirically turned out that $upperbound/4$ is a reasonable choice for ε. Experiments on synthetic data sets show that our suggested criteria for the choice of the density parameters are sufficient to detect all subspaces containing clusters.

4.5 Performance Evaluation

We tested RIS, using several synthetic as well as a real-world data set. For a description of the data used, refer to Section 3.5.1. The experiments were run on a 1.7 GHz workstation with 2 GB RAM.

A subsequent clustering of the data sets in the detected subspaces was performed for each experiment, using the hierarchical density-based cluster-

ing algorithm OPTICS (cf. Section 2.3) to validate the interestingness of the subspaces computed by RIS.

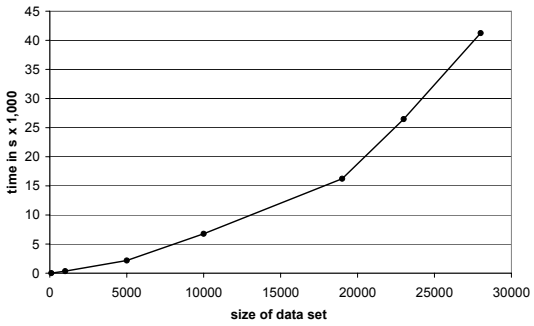
4.5.1 Efficiency Evaluation

The results of the efficiency evaluation are depicted in Figure 4.5.1. This evaluation is based on several synthetic data sets. The experiments were run with $k = \ln(n)$ and ε was chosen as suggested in Section 4.4.4. All runtimes are in seconds.

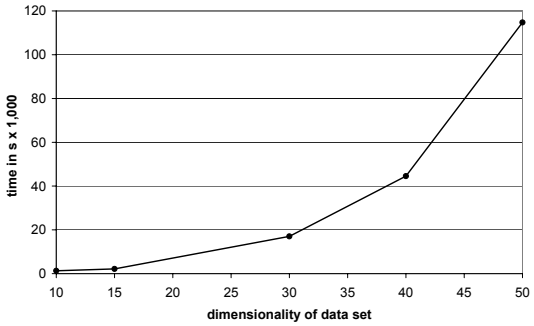
RIS scales well w.r.t. the dimensionality of the subspaces containing clusters. With increasing dimensionality of these subspaces, the runtime of RIS grows with a linear factor. On the other hand, the scalability of RIS w.r.t. the size n and the dimensionality d of the input data set is not linear. With increasing n and d, the runtime of RIS grows with an at least quadratic factor for rather large n and d, respectively. The reason for this scalability w.r.t. the size n is that RIS performs multiple range queries without any index support. There is again no index structure that efficiently supports range queries in arbitrary subspaces. The observed scalability with respect to d can be explained by the *apriori*-like navigation through the search space of all subspaces.

4.5.2 Speed-up for Large Data Sets

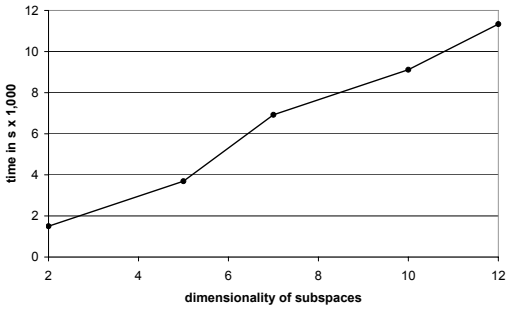
Since the runtime of RIS is rather high, especially for large data sets, we applied random sampling to accelerate our algorithm. In this case, the loop in our algorithm (cf. Section 4.4.1 is executed only for a random sample of the database objects. Figure 4.5 shows that for a large data set of $n = 750,000$ data points, sampling yields a rather good speed-up. The data set contained two overlapping four-dimensional subspace clusters, containing approximately 400,000 and 350,000 points. Even using only 100 sample points, RIS had no problem to detect the subspaces of these two clusters. For all sample sizes, these subspaces had by far the highest quality values. Further experiments empirically show that random sampling can be successfully applied to RIS in order to speed up the runtime of this algorithm, paying a minimum loss of quality.



(a) Scalability w.r.t. the size of the data set (d=10).



(b) Scalability w.r.t. the dimensionality of the data set (n=4,000).



(c) Scalability w.r.t. the dimensionality of the detected subspaces (d=15, n=4,000).

Figure 4.4: Efficiency evaluation of RIS.

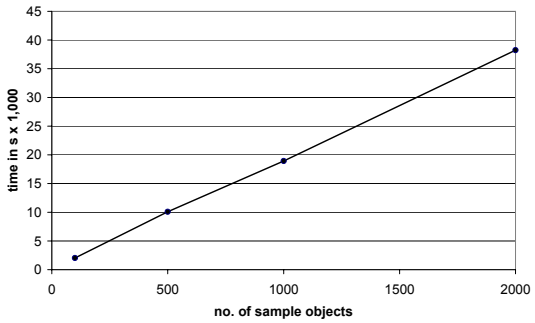


Figure 4.5: Scalability of RIS w.r.t. the size of the random sample (d=10, n=750,000).

4.5.3 Effectiveness Evaluation

Synthetic Data Sets. We evaluated the effectiveness of RIS, using several synthetic data sets of varying dimensionality. The data sets contained between two and five overlapping clusters in varying subspaces. In all experiments, RIS detected the correct subspaces in which clusters exist and assigned the highest quality values to them. All higher dimensional subspaces which were generated were removed by the upward pruning procedure.

Gene Expression Data. We also applied RIS to the Spellman data set (cf. 3.5.1). The two top-ranked subspaces were the subspace spanned by the time slots 90, 110, 130, and 190 and the subspace spanned by the time slots 190, 270, and 290. Both subspaces played also a central role in the evaluation of the algorithm SUBCLU (cf. Section 3.5.3). A clustering using OPTICS in these two top-ranked subspaces provided several clusters and in fact more information than SUBCLU yielded. This is due to the use of a hierarchical clustering algorithm in the detected subspaces. For example, the genes MRPL17, MRPL31, MRPL32, and MRPL33 (four mitochondrial large ribosomal subunits) were clustered together with other mitochondrial proteins SNF7 and VPS4 (which are direct interaction partners) by SUBCLU. However, several other genes that code for mitochondrial proteins, e.g. MEF1, PHB1, CYC1, MGE1, ATP12, could be added to this cluster because of the information OPTICS yielded in this subspace. Figure 4.6 illustrates the part of the cluster ordering generated by OPTICS in the par-

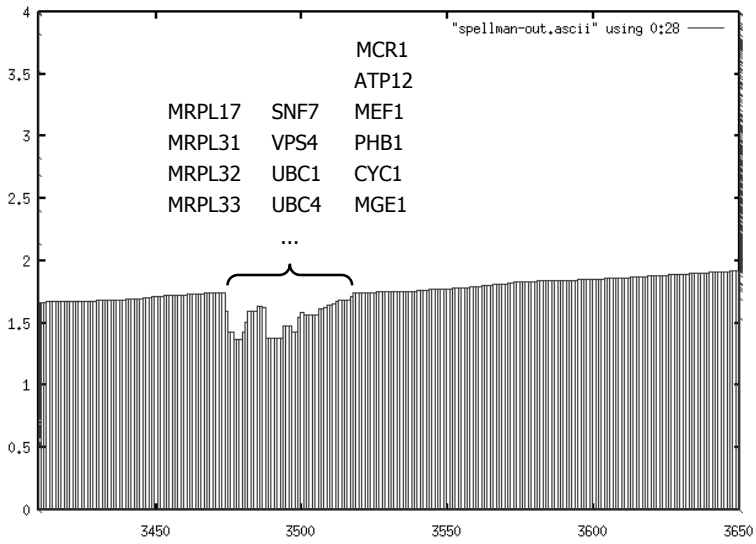


Figure 4.6: Part of the reachability plot generated by OPTICS for the subspace which was ranked second by RIS.

ticular subspace. It can be seen that the additional genes are less dense than the core part of the cluster. To detect the entire nested cluster, the global parameter setting for the SUBCLU run in Section 3.5.3 was too strict, i.e. the ε-value was to small. However, running SUBCLU with a higher ε-value adds also other non-related genes, i.e. noise points to the cluster.

Additionally, RIS combined with OPTICS found some clusters which were not detected by SUBCLU. An excerpt of such a cluster is depicted in Table 4.1. This cluster was again found in the subspace spanned by the time slots 90, 110, 130, and 190 and contains several transcription related genes that directly interact with each other. It was not detected by SUBCLU because it does not fit the density threshold used for the SUBCLU run. However, it yields a significant valley in the reachability plot generated by OPTICS for that subspace. The functional relationship of the contained genes is biologically meaningful and important.

In summary, RIS detects several subspaces containing several biologically relevant co-expressions. All significant clusters SUBCLU has found were reproduced by the combined application of RIS and OPTICS. Furthermore, the application of the hierarchical algorithm OPTICS yielded new infor-

Gene Name	Function
RRP3	RNA splicing, builds complex with NPL3
NPL3	RNA splicing, builds complex with RRP3
TFA1	transcription elongation factor
SPT5	part of transcription elongation factor complex (TEFC)
CDC73	part of TEFC, builds complex with CKB1
CKB1	cell cycle transition gene, builds complex with CDC73

Table 4.1: A cluster missed by SUBCLU but detected by RIS/OPTICS.

mation such as extended nested clusters and additional clusters showing different densities. By outperforming SUBCLU, the combined application of RIS and OPTICS also yields superior accuracy than CLIQUE.

4.6 Summary

In this chapter, we introduced a preprocessing step for clustering high-dimensional data. Based on a quality criterion for the interestingness of a subspace, we presented an efficient algorithm called RIS to compute all interesting subspaces containing dense regions of arbitrary shape and size. Furthermore, the well-established technique of random sampling can be applied to RIS in order to speed up the runtime of the algorithm significantly with a minimum loss of quality. The effectiveness evaluation shows that a combination of RIS and OPTICS can be successfully applied to high-dimensional real-world data, e.g. gene expression data in order to find co-regulated genes.

Chapter 5

Advanced Subspace Selection for Clustering

The previous chapter showed that the combination of the subspace selection technique RIS and the hierarchical clustering algorithm OPTICS is superior to subspace clustering algorithms which are based on a global density threshold. The problem that still remains is that RIS itself is again based on a global density threshold. In this chapter, we present a feature selection technique called SURFING (*SU*bspaces *R*elevant *F*or cluster*ING*) that finds all subspaces interesting for clustering and is independent from any global density threshold. The sorting is based on a quality criterion, using the k-nearest neighbor distances of the points to measure the hierarchical clustering structure of a subspace. A broad evaluation based on synthetic and real-world data sets demonstrates that SURFING is suitable to find all relevant subspaces in large, high-dimensional, sparse data sets and produces better results than comparative methods.

5.1 Introduction

Recent density-based approaches to subspace clustering or comparable subspace selection methods (RIS) use a global density threshold for the definition of clusters due to efficiency reasons. However, the application of one global density threshold to subspaces of different dimensionality as well as to all clusters in one subspace is rather unacceptable. The data space naturally increases exponentially with each dimension that is added to a subspace. The clusters in the same subspace may exceed different density parameters or exhibit a nested hierarchical clustering structure. Therefore, for subspace clustering, it would be highly desirable to adapt the density threshold to the dimensionality of the subspaces or even better to rely on a hierarchical clustering notion that is independent from a globally fixed threshold.

In this chapter, we introduce SURFING (*SU*bspaces *R*elevant *F*or cluster*ING*), a feature selection method for clustering which does not rely on a global density parameter. Our approach explores all subspaces exhibiting an *interesting* hierarchical clustering structure and ranks them according to a quality criterion based on the k-nearest neighbor distances of the points. SURFING does not demand that the user specifies parameters that are hard to anticipate such as the number of clusters, the (average) dimensionality of subspace clusters or a global density threshold.

The remainder of this chapter is organized as follows. A quality criterion for ranking the interestingness of subspaces is developed in Section 5.2. In Section 5.3 we present our algorithm SURFING to rank all subspaces that are relevant for clustering. A thorough experimental evaluation of the performance of SURFING including a comparison to comparative subspace clustering methods is presented in Section 5.4. Section 5.5 concludes the chapter.

5.2 Subspaces Relevant for Clustering

5.2.1 General Idea

The main idea of SURFING is to measure the "interestingness" of a subspace w.r.t. its hierarchical clustering structure, independent from its dimensionality. Like most previous approaches to subspace clustering, we base our measurement on a density-based clustering notion. Since we do not want to rely on a global density parameter, we developed a quality criterion for relevant subspaces built on the k-nearest neighbor distances (k-nn distances) of the points in DB.

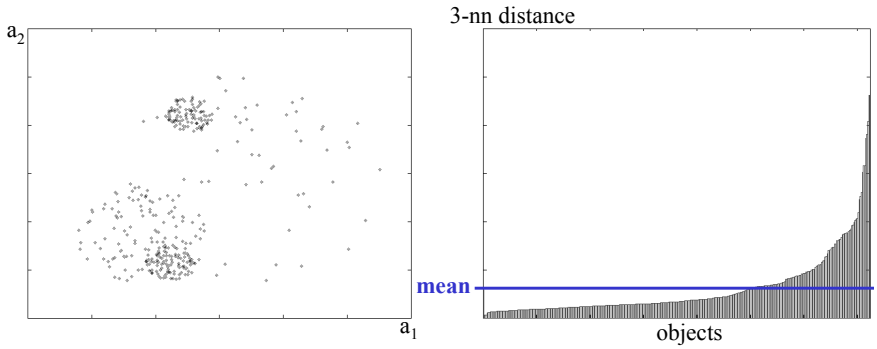
Definition 5.1 (k-nn distance in a subspace)
Let $k \in \mathbb{N}$ ($k \leq N$) and $S \subseteq \mathcal{A}$. For a point $o \in DB$, the set of k-nearest neighbors of o in a subspace S, *denoted by $NN_k^S(o)$, is the smallest set that contains (at least) k points from the database and for which the following condition holds:*

$$\forall p \in NN_k^S(o), q \in DB - NN_k^S(o) :$$
$$dist(\pi_S(o), \pi_S(p)) < dist(\pi_S(o), \pi_S(q)).$$

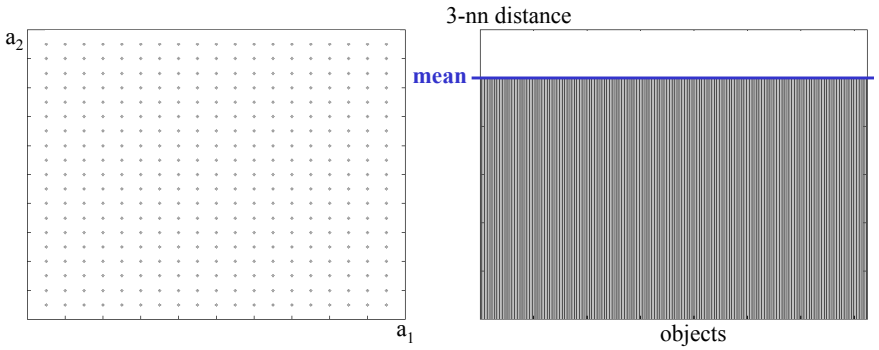
The k-nn distance *of a point $o \in DB$ in a subspace S, denoted by $nn\text{-}Dist_k^S(o)$, is the distance between o and its k-nearest neighbor, formally:*

$$nn\text{-}Dist_k^S(o) = \max\{dist(\pi_S(o), \pi_S(p)) \,|\, p \in NN_k^S(o)\}.$$

The k-nn distance of a point o indicates how densely the data space is populated around o in S. The smaller the value of $nn\text{-}Dist_k^S(o)$, the more dense the points are packed around o, and *vice versa*. If a subspace contains a recognizable hierarchical clustering structure, i.e. clusters with different densities and noise points, the k-nn distances of points should differ significantly. On the other hand, if all points are uniformly distributed, the k-nn distances can be assumed to be almost equal. Figure 5.1 illustrates these considerations using a sample 2D subspace $S = \{a_1, a_2\}$ and $k = 3$. In Figure 5.1(a), the data exhibits a complex hierarchical clustering structure in S. The corresponding 3-nn distances (sorted in ascending order) differ



(a) Hierarchical clustering structure in a 2D subspace(left); corresponding sorted 3-nn graph (right).



(b) Uniform distribution in a 2D subspace (left); corresponding sorted 3-nn graph (right).

Figure 5.1: Usefulness of the k-nn distance to rate the interestingness of subspaces.

significantly among the points. In Figure 5.1(b), the data are uniformly distributed in S. The corresponding 3-nn distances are equal for all points.

Consequently, we are interested in subspaces where the k-nn distances of the points differ significantly from each other, because the hierarchical clustering structure in such subspaces will be considerably clearer than in subspaces where the k-nn distances are rather similar to each other.

5.2.2 A Quality Criterion for Subspaces

As mentioned above, we want to measure how much the k-nn distances in S differ from each other. To achieve comparability between subspaces of different dimensionality, we scale all k-nn distances in a subspace S into the range $[0, 1]$. Thus, we assume that $nn\text{-}Dist_k^S(o) \in [0, 1]$ for all $o \in DB$ throughout the rest of the chapter.

Two well-known statistical measures for our purpose are the mean value μ_S of all k-nn distances in subspace S, i.e.

$$\mu_S := \frac{\sum_{o \in DB} nn\text{-}Dist_k^S(o)}{N}$$

and its variance. However, the variance is not appropriate for our purpose because it measures the squared differences of each k-nn distance to μ_S and thus, high differences are weighted stronger than low differences. For our quality criterion, we want to measure the non-weighted differences of each k-nn distance to μ_S. Since the sum of the differences of all points *above* μ_S is equal to the sum of the differences of all points *below* μ_S, we only take half of the sum of all differences to the mean value, denoted by $diff_{\mu_S}$, which can be computed by

$$diff_{\mu_S} = \frac{1}{2} \sum_{o \in DB} (\mid \mu_S - nn\text{-}Dist_k^S(o) \mid).$$

In fact, $diff_{\mu_S}$ is already a good measure for rating the interestingness of a subspace. We can further scale this value by μ_S times the number of points having a smaller k-nn distance in S than μ_S, i.e. the points contained in the following set:

$$Below_S := \{o \in DB \mid nn\text{-}Dist_k^S(o) < \mu_S\}.$$

Obviously, if $Below_S$ is empty, the subspace contains uniformly distributed noise.

Definition 5.2 (quality of a subspace)
Let $S \subseteq \mathcal{A}$. The quality of S, denoted by $quality(S)$, is defined as follows:

$$quality(S) = \begin{cases} 0 & \text{if } Below_S = \emptyset \\ \frac{diff_{\mu_S}}{|Below_S| \cdot \mu_S} & \text{else.} \end{cases}$$

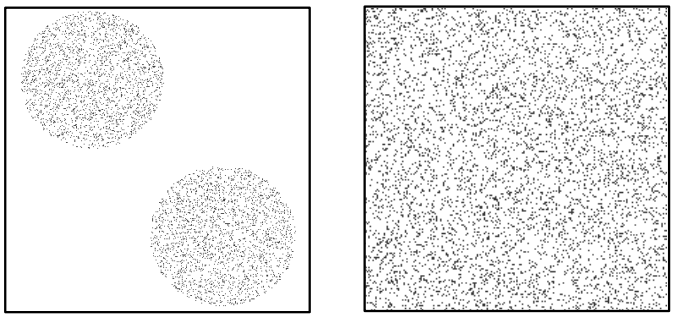
The quality values are in the range between 0 and 1. A subspace where all points are uniformly distributed, e.g. as depicted in Figure 5.1(b), has a quality value of approximately 0, indicating a less interesting clustering structure. On the other hand, the clearer the hierarchical clustering structure in a subspace S is, the higher is the value of $quality(S)$. For example, the sample 2D subspace in which the data is highly structured as depicted in Figure 5.1(a) will have a significantly higher quality value. Let us note that in the synthetic case where all points in $Below_S$ have a k-nn distance of 0 and all other points have a k-nn distance of $2 \cdot \mu_S$, the quality value $quality(S)$ is 1.

In almost all cases we can detect the relevant subspaces with this quality criterion, but there are two artificial cases rarely found in natural data sets which nevertheless cannot be ignored.

First, there might be a subspace containing some clusters, each of the same density and without noise, e.g. data set A in Figure 5.2. If the number of data points in the clusters exceeds k, such subspaces cannot be distinguished from subspaces containing uniformly distributed data points spread over the whole attribute range, e.g. data set B in Figure 5.2, because in both cases the k-nn distances of the points will marginally differ from the mean value.

Second, subspaces containing data of one Gaussian distribution spread over the whole attribute range are not really interesting. However, the k-nn distances of the points will scatter significantly around the mean value. Thus, such subspaces cannot be distinguished from subspaces containing two or more Gaussian clusters without noise.

To overcome these two artificial cases, we can temporarily insert some randomly generated points before computing the quality value of a subspace.



data set A data set B

% of additionally inserted points	quality of data set	
	A	B
0	0.13	0.15
0.1	0.15	0.15
0.2	0.19	0.15
0.5	0.31	0.15
1	0.38	0.15
5	0.57	0.15
10	0.57	0.15

Figure 5.2: Benefit of inserted points.

In cases of uniform or Gaussian distribution over the whole attribute range, the insertion of a few randomly generated additional points does not significantly affect the quality value. The k-nn distances of these points are similar to the k-nn distances of all the other data points. However, if there are dense and empty areas in a subspace, the insertion of some additional points very likely increases the quality value, because these additional points have large k-nn distances compared to those of the other points. The table in Figure 5.2 shows the quality value of the 2D data set A depicted in Figure 5.2 w.r.t. the percentage of temporarily inserted random points. The data set B in Figure 5.2 has no visible cluster structure and therefore the temporarily inserted points do not affect the quality value. For example, 0.2 % additionally inserted points means that for $n = 5,000$ 10 random points have been temporarily inserted before calculating the quality value.

Thus, inserting randomly generated points is a proper strategy to dis-

tinguish (good) subspaces containing several uniformly distributed clusters of equal density or several Gaussian clusters without noise from (bad) subspaces containing only one uniform or Gaussian distribution. In fact, it empirically turned out that 1% of additional points is sufficient to achieve the desired results. Let us note that this strategy is only required if the subspaces contain a clear clustering structure without noise. In most real-world data sets, the subspaces do not show a clear cluster structure and often have much more than 10% noise. In addition, the number of noise points is usually growing with increasing dimensionality. In such data sets, inserting additional points is not required. Since our quality criterion is very sensible to areas of different density, it is suitable to detect relevant subspaces in data sets with high percentages of noise, e.g. in gene expression data sets or in synthetic data sets containing up to 90% noise.

5.3 Algorithm

The pseudocode of the algorithm SURFING is given in Figure 5.3. Since lower dimensional subspaces are more likely to contain an interesting clustering, SURFING generates all relevant subspaces in a bottom-up way, i.e. it starts with all one-dimensional subspaces $\mathcal{S}_1$ and discards as many irrelevant subspaces as early as possible. Therefore, we need a criterion to decide whether it is interesting to generate and examine a certain subspace or not. Our above described quality measure can only be used to decide about the interestingness of an already given subspace. An important information we have gathered while proceeding to dimension l is the quality of all $(l-1)$-dimensional subspaces. We can use this information to rate all l-dimensional candidate subspaces $\mathcal{S}_l$. We use the lowest quality value of any $(l-1)$-dimensional subspace as threshold. If the quality values of the $(l-1)$-dimensional subspaces do not differ enough (it turned out empirically that a difference of at least $1/3$ is a reasonable reference difference), we take half of the best quality value instead. Using this quality threshold, we can divide all l-dimensional subspaces into three different categories:

SURFING(SetOfPoints DB, Integer k)

// 1-dimensional subspaces
  $\mathcal{S}_1 := \{\{a_1\}, \ldots, \{a_d\}\}$;
  compute quality of all subspaces $S \in \mathcal{S}_1$;
  $S_l := S \in \mathcal{S}_1$ with lowest quality;
  $S_h := S \in \mathcal{S}_1$ with highest quality;
  if $quality(S_l) > \frac{2}{3} \cdot quality(S_h)$ **then**
    $\tau := \frac{quality(S_h)}{2}$;
  else
    $\tau := quality(S_l)$;
    $\mathcal{S}_1 = \mathcal{S}_1 - \{S_l\}$;
  end if

// k-dimensional-subspaces
  $k := 2$;
  create $\mathcal{S}_2$ from $\mathcal{S}_1$;
  while not $\mathcal{S}_k = \emptyset$ **do**
    compute quality of all subspaces S in $\mathcal{S}_k$;
    $Interesting := \{S \in \mathcal{S}_k | quality(S) \uparrow\}$;
    $Neutral := \{S \in \mathcal{S}_k | quality(s) \downarrow \wedge quality(S) > \tau\}$;
    $Irrelevant := \{S \in \mathcal{S}_k | quality(S) \leq \tau\}$;
    $S_l := S \in \mathcal{S}_k$ with lowest quality;
    $S_h := S \in \mathcal{S}_k - Interesting$ with highest quality;
    if $quality(S_l) > \frac{2}{3} \cdot quality(S_h)$ **then**
      $\tau := \frac{quality(S_h)}{2}$;
    else
      $\tau := quality(s_l)$;
    end if
    if not all subspaces irrelevant **then**
      $\mathcal{S}_k := \mathcal{S}_k - Irrelevant$;
    end if
    create $\mathcal{S}_{k+1}$ from $\mathcal{S}_k$;
    $k := k + 1$;
  end while

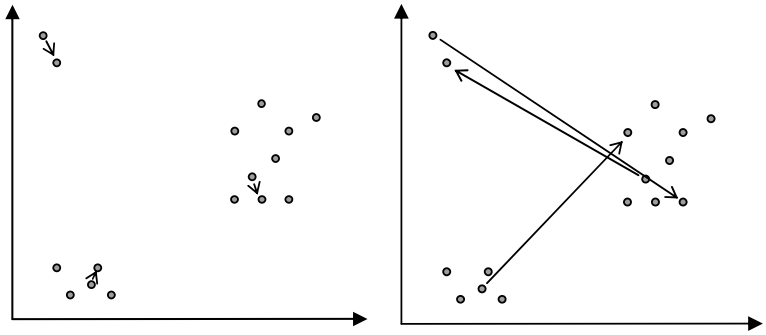
Figure 5.3: The algorithm SURFING.

Interesting Subspace: the quality value increases or stays the same w.r.t. its $(l-1)$-dimensional subspaces.
Neutral Subspaces: the quality decreases w.r.t. its $(l-1)$-dimensional subspaces, but lies above the threshold and thus might indicate a higher dimensional interesting subspace.
Irrelevant Subspaces: the quality decreases w.r.t. its $(l-1)$-dimensional subspace and lies below the threshold.

We use this classification to discard all irrelevant l-dimensional subspaces from further consideration. We know that these subspaces are not interesting itself and, as our quality value is comparable over different dimensions, we further know that no superspace of such a subspace will obtain a high quality value compared to interesting subspaces of dimensionality l. Even if through adding a "good" dimension, the quality value would slightly increase, it will not be getting better than already existing ones.

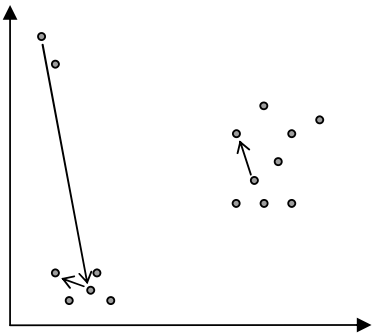
However, before we discard an irrelevant subspace S of dimensionality l, we have to test whether its clustering structure exhibits one of the artificial cases mentioned in the previous section. For that purpose, if the quality of S is lower than the quality of a subspace containing an l-dimensional Gaussian distribution, we insert 1% random points and recompute the quality of S. Otherwise, the clustering structure of S cannot get better through the insertion of additional points. In case of a clean cluster structure without noise in S, the quality value improves significantly after the insertion. At least it will be better than the quality of the l-dimensional Gaussian distribution and, in this case, S is not discarded.

If, due to the threshold, there are only irrelevant l-dimensional subspaces, we do not use the threshold but keep all l-dimensional subspaces. In this case, the information we have so far is not enough to decide about the interestingness.



(a) Parameter $k = 1$ (too small).

(b) Parameter $k = 10$ (too big).



(c) Parameter $k = 5$ (adequate).

Figure 5.4: Influence of parameter k.

Finally, the remaining l-dimensional subspaces in $\mathcal{S}_l$ are joined if they share any $(l-1)$-dimensions to generate the set of $(l+1)$-dimensional candidate subspaces $\mathcal{S}_{l+1}$. SURFING terminates if the resulting candidate set is empty.

SURFING needs only one input parameter k. The choice of k is rather simple. If k is too small, the k-nn distances are not meaningful since points within dense regions might have similar k-nn distance values as points in sparse regions. This is illustrated in Figure 5.4(a) where the arrows denote the distance of a point to its k-th nearest neighbor. If k is too high, the same phenomenon may occur (Figure 5.4(b)). Obviously, k must somehow correspond to the minimum cluster size, i.e. the minimal number of points regarded as a cluster (Figure 5.4(c)).

5.4 Evaluation

We tested SURFING on several synthetic and real-world data sets and evaluated its accuracy in comparison to CLIQUE, RIS and the subspace selection proposed in [DCSL02], in the following called Entropy. All experiments were run on a 2.8 GHz workstation with 512 MB RAM. Again, we combined SURFING, RIS and Entropy with the hierarchical clustering algorithm OPTICS [ABKS99] to compute the hierarchical clustering structure in the detected subspaces.

Synthetic Data and Gene Expression Data.
We used again the data generator and the Spellman data set described in Section 3.5.1.

Metabolome Data. In addition, we tested SURFING on high-dimensional metabolic data, provided from the newborn screening program in Bavaria, Germany. Our experimental data sets were generated from modern tandem mass spectrometry. In particular, we focused on a dimensionality of 14 metabolites in order to mine single and promising combinations of key markers in the abnormal metabolism of phenylketonuria (PKU), a severe amino acid disorder. The resulting database contains 319 cases designated as PKU and 1,322 control individuals expressed as 14 amino acids and intermediate metabolic products, i.e. Ala, Arg, ArgSuc, Cit, Glu, Gly, Met, Orn, Phe, Pyrglt, Ser, Tyr, Val and Xle. The task is to extract a subset of metabolites that corresponds well to the abnormal metabolism of PKU.

5.4.1 Efficiency

The runtimes of SURFING applied to the synthetic data sets are summarized in Table 5.1. In all experiments we set $k = 10$.

For each subspace, SURFING needs $O(N^2)$ time to compute for each of the N points in DB, the k-nn distance. Again, there is no index structure which could support the partial k-nn queries in arbitrary subspaces in logarithmic time. If SURFING analyzes m different subspaces, the overall runtime complexity is $O(m \cdot N^2)$. Of course, in the worst case m can be 2^d, but in practice we are only examining a very small percentage of all possi-

data set	d	cluster dim.	N	# subspaces m	# subspaces %	time (s)
02	10	4	4936	107	10.45	351
03	10	4	18999	52	5.08	2069
04	10	4	27704	52	5.08	4401
05	15	2	4045	119	0.36	194
06	15	5	3802	391	1.19	807
07	15	3,5,7	4325	285	0.87	715
08	15	5	4057	197	0.60	391
09	15	7	3967	1046	3.19	3031
10	15	12	3907	4124	12.59	15321
11	10	5	3700	231	22.56	442
12	20	5	3700	572	0.05	1130
13	30	5	3700	1077	0.0001	2049
14	40	5	3700	1682	$1.5 \cdot 10^{-7}$	3145
15	50	5	3700	2387	$2.1 \cdot 10^{-10}$	4255
16	15	4,6,7,10	2671	912	2.8	4479

Table 5.1: Results on synthetic data sets.

ble subspaces. Indeed, our experiments show that the heuristic generation of subspace candidates used by SURFING ensures a small value for m (cf. Table 5.1). For most complex data sets, SURFING computes less than 5% of the total number of possible subspaces. In most cases, this ratio is even significantly less than 1%. For data set 10 in Table 5.1 where the cluster is hidden in a 12-dimensional subspace of a 15-dimensional feature space, SURFING only computes 12.5% of the possible subspaces. Finally, for both real-world data sets, SURFING computes even significantly less than 0.1% of the possible subspaces (not shown in Table 5.1). The worst ever observed percentage was around 20%. This empirically demonstrates that SURFING is a highly efficient solution for the complex subspace selection problem.

5.4.2 Effectivity

Results on Synthetic Data. We applied SURFING to several synthetic data sets (cf. Table 5.1). In all but one case, SURFING detected the correct subspaces containing the relevant clusters and ranked them first. Even for

data set 16, SURFING was able to detect 4 out of 5 subspaces containing clusters, although the clustering structure of the subspaces containing clusters was rather weak, e.g. one of the 4-dimensional subspaces contained a cluster with only 20 points, having an average k-nn distance of 2.5 (the average k-nn distance for all points in all dimensions was 15.0). SURFING only missed a 10-dimensional subspace which contained a cluster with 17 points, having an average k-nn distance of 9.0.

Results on Gene Expression Data. We tested SURFING on the gene expression data set and retrieved a hierarchical clustering by applying OPTICS [ABKS99] to the top-ranked subspaces. We found many biologically interesting and significant clusters in several subspaces. The functional relationships of the genes in the resulting clusters were validated by using the public Saccharomyces Genome Database[1]. Some excerpts from sample clusters in varying subspaces found by SURFING applied to the gene expression data are depicted in Table 5.2. Cluster 1 contains several cell cycle genes. In addition, the two gene products are part of a common protein complex. Cluster 2 contains the gene STE12, an important regulatory factor for the mitotic cell cycle [SSZ+98] and the genes CDC27 and EMP47 which are most likely co-expressed with STE12. Cluster 3 consists of the genes CDC25 (starting point for mitosis), MYO3 and NUD1 (known for an active role during mitosis) and various other transcription factors required during the cell cycle. Cluster 4 contains several genes related to the protein catabolism. Cluster 5 contains several structural parts of the ribosomes and related genes. Let us note that MPI6 is clustered differently in varying subspaces (cf. Cluster 1 and Cluster 5). Cluster 6 contains the genes that code for proteins participating in a common pathway.

Results on Metabolome Data. Applying SURFING to metabolic data, we identified 13 subspaces considering quality values > 0.8. In detail, we extracted 5 one-dimensional spaces (the metabolites ArgSuc, Phe, Glu, Cit and Arg), 6 two-dimensional spaces (e.g. Phe-ArgSuc, Phe-Glu) and 3 three-dimensional spaces (e.g. Phe-Glu-ArgSuc). Alterations of our best ranked single metabolites correspond well to the abnormal metabolism of PKU

[1] http://www.yeastgenome.org/

Gene Name	**Function**
Cluster 1 (subspace 90, 110, 130, 190)	
RPC40	builds complex with CDC60
CDC60	tRNA synthetase
FRS1	tRNA synthetase
DOM34	protein synthesis, mitotic cell cycle
CKA1	mitotic cell cycle control
MIP6	RNA binding activity, mitotic cell cycle
Cluster 2 (subspace 90, 110, 130, 190)	
STE12	transcription factor (cell cycle)
CDC27	possible STE12-site
EMP47	possible STE12-site
XBP1	transcription factor
Cluster 3 (subspace 90, 110, 130, 190)	
CDC25	starting control factor for mitosis
MYO3	control/regulation factor for mitosis
NUD1	control/regulation factor for mitosis
Cluster 4 (subspace 190, 270, 290)	
RPT6	protein catabolism; complex with RPN10
RPN10	protein catabolism; complex with RPT6
UBC1	protein catabolism; part of 26S protease
UBC4	protein catabolism; part of 26S protease
Cluster 5 (subspace 70, 90, 110, 130)	
SOF1	part of small ribosomal subunit
NAN1	part of small ribosomal subunit
RPS1A	structural constituent of ribosome
MIP6	RNA binding activity, mitotic cell cycle
Cluster 6 (subspace 70, 90, 110, 130)	
RIB1	participate in riboflavin biosynthesis
RIB4	participate in riboflavin biosynthesis
RIB5	participate in riboflavin biosynthesis

Table 5.2: Results on gene expression data.

data set	# clusters/ subspaces	# correct clusters/subspaces found by			
		CLIQUE	RIS	Entropy	SURFING
06	2	1	2	0	2
07	3	1	2	0	2
08	3	1	3	0	3
16	5	0	3	0	4

Table 5.3: Comparative tests on synthetic data.

[BBB+04]. We compared the results of SURFING to the results using PCA.[2]. Only components with a eigenvalue > 1 were extracted. The varimax rotation was applied. PCA findings showed 4 components (eigenvalues of components 1-4 are 4.039, 2.612, 1.137 and 1.033) that retain 63% of the total variation. However, SURFING's best ranked single metabolites ArgSuc, Glu, Cit and Arg are not highly loaded (> 0.6) on one of four extracted components. Moreover, combinations of promising metabolites (higher dimensional subspaces) are not able to be considered in PCA. Particularly in abnormal metabolism, not only alterations of single metabolites but more interactions of several markers are often involved. As our results demonstrate, SURFING is more usable on metabolic data, taking higher dimensional subspaces into account.

Influence of Parameter k. We reran our experiments on the synthetic data sets with $k = 3, 5, 10, 15, 20$. We observed that if $k = 3$, SURFING found the correct subspaces but did not rank the subspaces first (i.e. subspaces with a less clear hierarchical clustering structure got a higher quality value). In the range of $5 \leq k \leq 20$, SURFING produced similar results for all synthetic data sets. This indicates that SURFING is quite robust regarding the choice of k within this range.

Comparison with CLIQUE. The results of CLIQUE applied to the synthetic data sets confirmed the suggestions that its accuracy heavily depends on the choice of the input parameters which is a non-trivial task. In some cases, CLIQUE failed to detect the subspace clusters hidden in the data but computed some dubious clusters. In addition, CLIQUE is not able to detect clusters of different density. Applied to our data sets which exhibit

[2]The terms PCA, eigenvalue and eigenvector are described in Section 6.3

several clusters with varying density, e.g. data set 16, CLIQUE was not able to detect all clusters correctly but could only detect (parts of) one cluster (cf. Table 5.3) — even though we used a broad parameter setting. A similar result can be reported when we applied CLIQUE to the gene expression data set. CLIQUE was not able to obtain any useful clusters for a broad range of parameter settings. In summary, SURFING does not only outperform CLIQUE by means of quality, but also saves the user from finding a suitable parameter setting.

Comparison with RIS. Using RIS causes similar problems as CLIQUE. Although the input parameters have slightly less impact, the quality of the results computed by RIS also depends on the input parameters. Like CLIQUE, in some cases RIS failed to detect the correct subspaces due to the utilization of a global density parameter (cf. Table 5.3). For example, applied to data set 16, RIS was able to compute the lower dimensional subspaces, but could not detect the higher dimensional one. The application of RIS to the gene expression data set is described in [KKKW03]. SURFING confirmed these results but found several other interesting subspaces with important clusters, e.g. clusters 5 and 6 in subspace 70, 90, 110, 130 (cf. Table 5.2). Applying RIS to the metabolome data set, the best ranked subspace contains 12 attributes which represent nearly the full feature space and are biologically not interpretable. The application of RIS to all data sets was limited by the choice of the right parameter setting. Again, SURFING does not only outperform RIS by means of quality, but also saves the user from finding a suitable parameter setting.

Comparison with Entropy. Using the quality criterion Entropy in conjunction with the proposed forward search algorithm in [DCSL02], none of the correct subspaces were found. In all cases, the subspace selection method stops at a dimensionality of 2. Possibly an exhaustive search examining all possible subspaces could produce better results. However, this approach obviously yields unacceptable runtimes. Applied to the metabolome data, the biologically relevant one-dimensional subspaces are ranked low.

5.5 Summary

In this chapter, we introduced a new method to subspace selection for clustering called SURFING which is more or less parameterless and — in contrast to most recent approaches — does not rely on a global density threshold. SURFING selects and ranks subspaces of high-dimensional data according to their interestingness for clustering. We empirically showed that the only input parameter of SURFING is stable in a broad range of settings. SURFING does not favor subspaces of a certain dimensionality. A broad, comparative experimental evaluation using synthetic and real-world data sets shows that SURFING is an efficient and accurate solution to the complex subspace clustering problem. It outperforms recent subspace clustering methods in terms of effectivity.

Chapter 6

Correlation Clustering

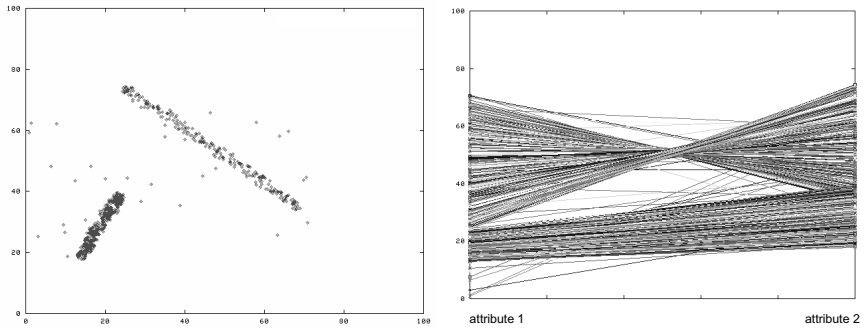
The detection of correlations between different features in a set of feature vectors is a very important data mining task because correlation indicates a dependency between the features or some association of cause and effect between them. This association can be arbitrarily complex, i.e. one or more features might be dependent from a combination of several other features. Well-known methods like the principal components analysis can perfectly find correlations which are global, linear, not hidden in a set of noise vectors, and uniform, i.e. the same type of correlation is exhibited in all feature vectors. In many applications such as medical diagnosis, molecular biology, time sequences or electronic commerce, however, correlations are not global since the dependency between features can be different in different subgroups of the set. In this chapter, we propose a method called 4C (*C*omputing *C*orrelation *C*onnected *C*lusters) to identify local subgroups of the data points, sharing a uniform but arbitrarily complex correlation. Our algorithm is based on a combination of PCA and density-based clustering (DBSCAN), has a determinate result and is robust concerning noise. A broad comparative evaluation demonstrates that for the task of correlation clustering 4C is superior to methods such as DBSCAN, CLIQUE and ORCLUS. The concepts described in this chapter have been published in [BKKZ04].

6.1 Introduction

A kind of hidden information that may be interesting to users are correlations in a data set. A correlation is a linear dependency between two or more features (attributes) of the data set. The most important method for detecting correlations is the principal components analysis (PCA) also known as Karhunen Loèwe transformation. Knowing correlations is also important and valuable because with it the dimensionality of the data set can be considerably reduced which improves both the efficiency of similarity search and data mining as well as the accuracy. Moreover, knowing about the existence of a relationship between attributes enables one to detect hidden causalities (e.g. the influence of the age of a patient and the dose rate of medication on the course of his disease or the coregulation of gene expression). The information can also be used to gain financial advantage (e.g. in stock quota analysis).

Methods such as PCA, however, are restricted because they can only be applied to the data set as a whole. Therefore, it is only possible to detect correlations which are expressed in all points or almost all points of the data set. For a lot of applications this is not the case. For instance, in the analysis of gene expression, we are facing the problem that a dependency between two genes does only exist under certain conditions. Therefore, the correlation is visible only in a local subset of the data. Other subsets may be either not correlated at all or they may exhibit completely different kinds of correlation (different features are dependent on each other). The correlation of the whole data set can be weak even if for local subsets of the data strong correlations exist. Figure 6.1 shows a simple example where two subsets of two-dimensional points exhibit different correlations.

To the best of our knowledge both concepts of clustering (i.e. finding densely populated subsets of the data) and correlation analysis have not yet been addressed as a combined task for data mining. The most relevant related approach is ORCLUS [AY00], but since it is k-medoid-based it is very sensitive to noise and the locality of the analyzed correlations is usually too coarse, i.e. the number of points taken into account for correlation analysis is too large (cf. Sections 6.2 and 6.6 for a detailed discussion).



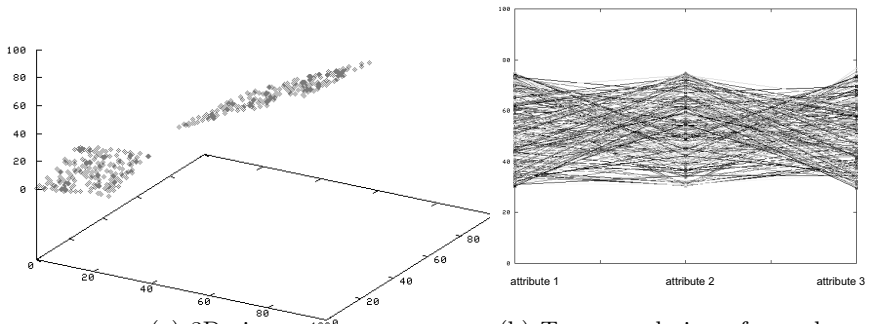
(a) 2D view. (b) Transposed view.

Figure 6.1: One-dimensional correlation lines.

In this chapter, we develop a new method which is capable of detecting local subsets of the data which exhibit strong correlations and which are densely populated (w.r.t. a given density threshold). We call such a subset a *correlation-connected cluster*.

In lots of applications, such correlation-connected clusters are interesting. For example, in E-commerce (recommendation systems or target marketing) where sets of customers with similar behavior need to be detected one searches for positive linear correlations. In DNA microarray analysis (gene expression analysis) negative linear correlations express the fact that two genes may be coregulated, i.e. if one has a high expression level, the other one is very low and *vice versa*. Usually, such a coregulation will only exist in a small subset of conditions or cases, i.e. the correlation will be hidden locally in the data set and cannot be detected by global techniques. Figures 6.1 and 6.2 show simple examples how correlation-connected clusters can look like. In Figure 6.1, the attributes exhibit two different forms of linear correlation. We observe that if for some points there is a linear correlation of all attributes, these points are located along a line. Figure 6.2 presents two examples where an attribute z is correlated to the attributes x and y, i.e. $z = a + bx + cy$. In this case, the set of points forms a two-dimensional plane.

In this chapter we propose an approach that meets both the goal of clustering and correlation analysis in order to find correlation-connected clusters. The remainder of this chapter is organized as follows: In Section 6.2



(a) 3D view. (b) Transposed view of one plane.

Figure 6.2: Two-dimensional correlation planes.

we review and discuss related work. In Section 6.3 we formalize our notion of correlation-connected clusters. Based on this formalization, we present in Section 6.4 an algorithm called 4C (*C*omputing *C*orrelation *C*onnected *C*lusters) to efficiently compute such correlation-connected clusters. In Section 6.5 we analyze the computational complexity of our algorithm while Section 6.6 contains an extensive experimental evaluation of 4C. Section 6.7 concludes the chapter.

6.2 Related Work

Traditional clustering algorithms such as k-means or the EM-algorithm search for spatial clusters which are spherically shaped. In [EKSX96] and [BC00] two algorithms are proposed which are able to find clusters of arbitrary shape. However, these approaches are not able to distinguish between arbitrarily shaped clusters and correlation clusters. The density-based clustering approach of [EKSX96] was described in detail in Chapter 2.2. In [BC00] the authors propose the algorithm FC (Fractal Clustering) to find clusters of arbitrary shape. The paper presents only experiments for two-dimensional data sets. So it is not clear whether the fractal dimension is really stable in higher dimensions. Furthermore, the shapes of clusters depend on the type of correlation of the involved points. Thus, in the case of linear correlations, it is more target-oriented to search for shapes like lines or hyperplanes than for arbitrarily shaped clusters.

As most traditional clustering algorithms fail to detect meaningful clus-

ters in high-dimensional data, in the last years a lot of research has been done in the area of subspace clustering. In the following, we are examining to what extent subspace clustering algorithms are able to capture local data correlations and find clusters of correlated points. The principal axes of correlated data are arbitrarily oriented. In contrast, subspace clustering techniques like CLIQUE [AGGR98] and its successors MAFIA [GNC99] or ENCLUS [CFZ99] or projected clustering algorithms like PROCLUS [AP99] and DOC [PJAM02] only find axis-parallel projections of the data. In the evaluation part we show that CLIQUE as one representative algorithm is in fact not able to find correlation clusters. Therefore, we focus our attention to ORCLUS [AY00] which is a k-medoid related projected clustering algorithm, allowing clusters to exist in arbitrarily oriented subspaces. The problem of this approach is that the user has to specify the number of clusters in advance. If this guess does not correspond to the actual number of clusters, the results of ORCLUS deteriorate. A second problem, which can also be seen in the evaluation part, is noisy data. In this case, the clusters found by ORCLUS are far from optimal since ORCLUS assigns each point to a cluster and thus cannot handle noise efficiently.

Based on the fractal (intrinsic) dimensionality of a data set, the authors of [STTF02] present a global dimensionality reduction method. Correlation in the data leads to the phenomenon that the embedding dimension of a data set (in other words the number of attributes of the data set) and the intrinsic dimension (the dimension of the spatial object represented by the data) can differ a lot. The intrinsic (correlation fractal dimension) is used to reduce the dimensionality of the data. As this approach adopts a global view on the data set and does not account for local data distributions, it cannot capture local subspace correlations. Therefore, it is only useful and applicable if the underlying correlation affects all data points. Since this is not the case for most real-world data, in [CM00] a local dimensionality reduction technique for correlated data is proposed which is similar to [AY00]. The authors focus on identifying correlated clusters for enhancing the indexing of high-dimensional data only. Unfortunately, they do not give any hint to what extent their heuristic-based algorithm can also be used to gain new insight into the correlations contained in the data.

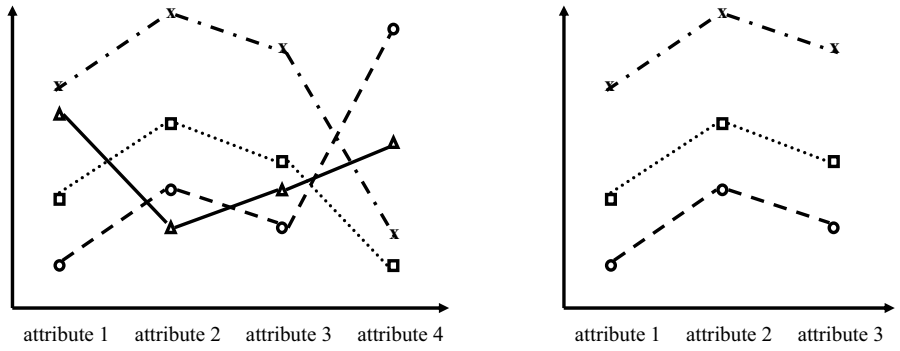


Figure 6.3: Transposed view (left) and pattern-based cluster (right) of some sample database objects.

In [YWWY02] the move-based algorithm FLOC computing near-optimal δ-clusters is presented. A transposed view of the data is used to show the correlations which are captured by the δ-cluster model (see Figure 6.3 for an illustration). Each data point is shown as a curve where the attribute values of each point are connected. Examples can be seen in Figure 6.1(b) and 6.2(b). A cluster is regarded as a subset of points and attributes for which the participating points show the same or a similar tendency rather than being close to each other on the associated subset of dimensions. The δ-cluster model concentrates on two forms of coherence, namely shifting (or addition) and amplification (or production). In the case of amplification coherence, for example, the vectors representing the points must be multiples of each other. The authors state that this can easily be transformed into the problem of finding shifting coherent δ-cluster by applying a logarithmic function to each point. Therefore, they focus on finding shifting coherent δ-clusters and introduce the metric of residue to measure the coherency among points of a given cluster. An advantage is that thereby they can easily handle missing attribute values. But in contrast to our approach, the δ-cluster model limits itself to a very special form of correlation where all attributes are positively linear correlated. It does not include negative correlations or correlations where one attribute is determined by two or more other attributes. In this cases, searching for a trend is no longer

possible as can be seen in Figure 6.1 and 6.2. Let us note that such complex dependencies cannot be illustrated by transposed views of the data. The same considerations hold for the very similar p-cluster model introduced in [WWYY02] and two extensions presented in [PZC+03, LW03].

6.3 The Notion of Correlation-Connected Clusters

In this section, we formalize the notion of a correlation-connected cluster. Intuitively, a correlation-connected cluster is a dense region of points in the d-dimensional feature space, having at least one principal axis with a low variation along this axis. Thus, a correlation-connected cluster has two different properties: density and correlation. The first aspect is discussed in detail in Chapter 2.2. In the following, we will first address the second property and then merge these ingredients to formalize our notion of correlation-connected clusters.

6.3.1 Correlation Sets

We want to identify correlation-connected clusters, i.e. regions in which the points exhibit correlation, and distinguish them from usual clusters, i.e. regions of high point density only. Thus, we are interested in all subsets of points with an intrinsic dimensionality that is considerably smaller than the embedding dimensionality of the data space, e.g. a line or a plane in a three or higher dimensional space. There are several methods to measure the intrinsic dimensionality of a point set in a region such as the fractal dimension or the principal components analysis. We choose PCA because the fractal dimension appeared to be not stable enough in our first experiments.

The PCA determines the covariance matrix $\mathbf{M} = [m_{ij}]$ with $m_{ij} = \sum_{S \in \mathcal{S}} s_i s_j$ of the considered point set $\mathcal{S}$, and decomposes it into an orthonormal Matrix $\mathbf{V}$ called *eigenvector matrix* and a diagonal matrix $\mathbf{E}$ called *eigenvalue matrix* such that $\mathbf{M} = \mathbf{VEV^T}$. The eigenvectors represent the principal axes of the data set whereas the eigenvalues represent the variance along these axes. In case of a linear dependency between two or more attributes of the point set (correlation), one or more eigenvalues are close to

zero. A set forms a λ-dimensional correlation hyperplane if $d-\lambda$ eigenvalues fall below a given threshold $\delta \approx 0$. Since the eigenvalues of different sets exhibiting different densities may differ a lot in their absolute values, we normalize the eigenvalues by mapping them onto the interval $[0, 1]$. This normalization is denoted by Ω and simply divides each eigenvalue e_i by the maximum eigenvalue e_{max}. We call the eigenvalues e_i with $\Omega(e_i) \leq \delta$ *close to zero.*

Definition 6.1 (λ-dimensional linear correlation set)
Let $\mathcal{S} \subseteq \mathcal{D}$, $\lambda \in \mathbb{N}$ ($\lambda \leq d$), $EV = e_1, ..., e_d$ the eigenvalues of the covariance matrix of $\mathcal{S}$ in descending order, i.e. $e_i \geq e_{i+1}$, and $\delta \in \mathbb{R}_0^+$ ($\delta \approx 0$). $\mathcal{S}$ forms an λ-dimensional linear correlation set *w.r.t. δ if at least $d - \lambda$ eigenvalues of $\mathcal{S}$ are close to zero, formally:*

$$\textsc{CorSet}_\delta^\lambda(\mathcal{S}) \Leftrightarrow |\{e_i \in EV \,|\, \Omega(e_i) \leq \delta\}| \geq d - \lambda$$

where $\Omega(e_i) = e_i/e_1$.

This condition states that the variance of $\mathcal{S}$ along $d - \lambda$ principal axes is low and therefore the points of $\mathcal{S}$ form an λ-dimensional hyperplane. We drop the index λ and speak of a correlation set in the following wherever it is clear from context.

Definition 6.2 (correlation dimension)
Let $\mathcal{S} \in DB$ be a linear correlation set w.r.t. $\delta \in \mathbb{N}$. The number of eigenvalues with $e_i > \delta$ is called correlation dimension*, denoted by* $\textsc{CorDim}(\mathcal{S})$.

Let us note that if $\mathcal{S}$ is a λ-dimensional linear correlation set, then $\textsc{CorDim}(\mathcal{S}) \leq \lambda$. The correlation dimension of a linear correlation set $\mathcal{S}$ corresponds to the intrinsic dimension of $\mathcal{S}$.

6.3.2 Clusters as Correlation-Connected Sets

A correlation-connected cluster can be regarded as a maximal set of density-connected points that exhibit uniform correlation. We can formalize the concept of correlation-connected sets by merging the two concepts: density-connected clusters (cf. Definition 2.7) and correlation sets (cf. Definition

6.1). The intuition of our formalization is to consider those points as core points of a cluster which have an appropriate correlation dimension in their neighborhood. Therefore, we associate each point P with a similarity matrix $\mathbf{M}_P$ which is determined by PCA of the points in the ε-neighborhood of P. For convenience, we call $\mathbf{V}_P$ and $\mathbf{E}_P$ the eigenvectors and eigenvalues of P, respectively. A point P is inserted into a cluster if it has the same or a similar similarity matrix like the points in the cluster. To achieve this goal, our algorithm looks for points that are close to the principal axis (or axes) of those points which are already in the cluster. We will define a similarity measure $\hat{\mathbf{M}}_P$ for the efficient search of such points.

We start with the formal definition of the covariance matrix $\mathbf{M}_P$ associated with a point P.

Definition 6.3 (covariance matrix)
Let $P \in DB$. The matrix $\mathbf{M}_P = [m_{ij}]$ with

$$m_{ij} = \sum_{S \in \mathcal{N}_\varepsilon(P)} s_i s_j \qquad (1 \leq i, j \leq d)$$

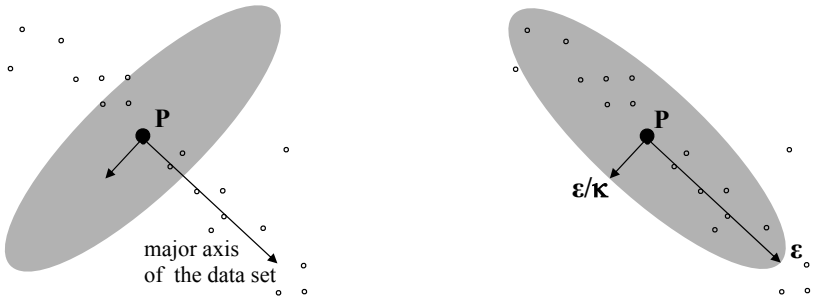
is called the covariance matrix *of the point P. $\mathbf{V}_P$ and $\mathbf{E}_P$ (with $\mathbf{M}_P = \mathbf{V}_P \mathbf{E}_P \mathbf{V}_P^{\mathbf{T}}$) as determined by PCA of $\mathbf{M}_P$ are called the eigenvectors and eigenvalues of the point P, respectively.*

We can now define the new similarity measure $\hat{\mathbf{M}}_P$ which searches points in the direction of highest variance of $\mathbf{M}_P$ (the major axes). Theoretically, $\mathbf{M}_P$ could be directly used as a similarity measure, i.e.

$$dist_{\mathbf{M}_P}(P, Q) = \sqrt{(P - Q)\mathbf{M}_P(P - Q)^{\mathbf{T}}} \quad \text{where } P, Q \in DB.$$

Figure 6.4(a) shows the set of points which lies in an ε-neighborhood of P using M_P as similarity measure. The distance measure puts high weights on those axes with a high variance whereas directions with a low variance are associated with low weights. This is usually desired in similarity search applications where directions of high variance have a high distinguishing power and, in contrast, directions of low variance are negligible.

Obviously, for our purpose of detecting correlation clusters, we need quite the opposite. We want to search for points in the direction of highest



(a) (b)

Figure 6.4: ε-neighborhood of a point P according to $\mathbf{M}_P$ (a) and $\hat{\mathbf{M}}_P$ (b).

variance of the data set. Therefore, we need to assign low weights to the direction of highest variance in order to shape the ellipsoid such that it reflects the data distribution (cf. Figure 6.4(b)). The solution is to change large eigenvalues into smaller ones and *vice versa*. We use two fixed values, 1 and a parameter $\kappa \gg 1$ rather than, for example, inverting the eigenvalues in order to avoid problems with singular covariance matrices. The number 1 is a natural choice because then the length of the corresponding semi-axes of the ellipsoid is epsilon. The parameter κ controls the "thickness" of the λ-dimensional correlation line or plane, i.e. the tolerated deviation.

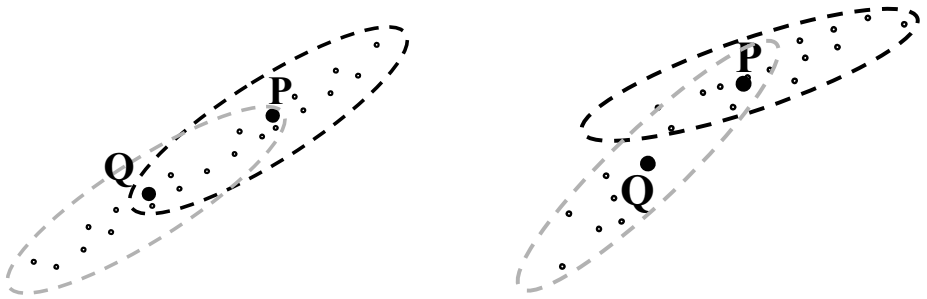
This is formally captured in the following definition:

Definition 6.4 (correlation similarity matrix)
Let $P \in DB$ and $\mathbf{V}_P$, $\mathbf{E}_P$ the corresponding eigenvectors and eigenvalues of the point P. Let $\kappa \in I\!R^+$ be a constant with $\kappa \gg 1$. The new eigenvalue matrix $\hat{\mathbf{E}}_P$ with entries $\hat{e}_i$ $(i = 1, \ldots d)$ is computed from the eigenvalues $e_1, \ldots, e_d$ in $\mathbf{E}_P$ according to the following rule:

$$\hat{e}_i = \begin{cases} 1 & if \quad \Omega(e_i) > \delta \\ \kappa & if \quad \Omega(e_i) \leq \delta \end{cases}$$

where Ω is the normalization of the eigenvalues onto $[0, 1]$ as described above. The matrix $\hat{\mathbf{M}}_P = \mathbf{V}_P \hat{\mathbf{E}}_P \mathbf{V}_P^{\mathbf{T}}$ is called the correlation similarity matrix. *The* correlation similarity measure *associated with point P is denoted by*

$$dist_P(P, Q) = \sqrt{(P - Q) \cdot \hat{\mathbf{M}}_P \cdot (P - Q)^{\mathbf{T}}}.$$



(a) $P \in \mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_Q}(Q)$. (b) $P \notin \mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_Q}(Q)$.

Figure 6.5: Correlation ε-neighborhood.

Figure 6.4(b) shows the ε-neighborhood according to the correlation similarity matrix $\hat{\mathbf{M}}_P$. As described above, the parameter κ specifies how much deviation from the correlation is allowed. The greater the parameter κ, the tighter and clearer the correlations which will be computed. It empirically turned out that our algorithm presented in Section 6.4 is rather insensitive to the choice of κ. A good suggestion is to set $\kappa = 50$ in order to achieve satisfying results. Thus, for the sake of simplicity, we omit the parameter κ in the following.

Using this similarity measure, we can define the notions of correlation core points and correlation-reachability. However, in order to define correlation-connectivity as a symmetric relation, we face the problem that the similarity measure in Definition 6.4 is not symmetric, i.e. $dist_P(P, Q) = dist_Q(Q, P)$ does in general not hold (cf. Figure 6.5(b)). Symmetry, however, is important to avoid ambiguity of the clustering result. If an asymmetric similarity measure is used in DBSCAN, a different clustering result can be obtained, depending on the order of processing, e.g. which point is selected as the starting point. Although the result is typically not seriously affected by this ambiguity effect, we avoid this problem easily by an extension of our similarity measure. The trick is to consider both similarity measures $dist_P(P, Q)$ as well as $dist_Q(P, Q)$ and to combine them by a suitable arithmetic operation such as the maximum of the two. Based on these considerations, we define the correlation ε-neighborhood as a symmetric concept:

Definition 6.5 (correlation ε-neighborhood)
Let $\varepsilon \in \mathbb{R}_0^+$. The correlation ε-neighborhood *of a point $O \in DB$, denoted by $\mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_O}(O)$, is defined by:*

$$\mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_O}(O) = \{X \in DB \mid \max\{dist_O(O,X), dist_X(X,O)\} \leq \varepsilon\}.$$

The correlation ε-neighborhood is illustrated in Figure 6.5. Correlation core points can now be defined as follows.

Definition 6.6 (correlation core point)
Let $\varepsilon, \delta \in \mathbb{R}_0^+$ and $k, \lambda \in \mathbb{N}$. A point $O \in D$ is called correlation core point *w.r.t. ε, k, δ, and λ (denoted by $\textsc{Core}_{\varepsilon,k}^{\lambda,\delta}(O)$) if its ε-neighborhood is a λ-dimensional linear correlation set and its correlation ε-neighborhood contains at least k points, formally:*

$$\textsc{Core}_{\varepsilon,k}^{\lambda,\delta}(O) \Leftrightarrow \textsc{CorSet}_\delta^\lambda(\mathcal{N}_\varepsilon(P)) \wedge |\mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_O}(O)| \geq k.$$

Definition 6.7 (direct correlation-reachability) *Let $\varepsilon, \delta \in \mathbb{R}_0^+$ and $k, \lambda \in \mathbb{N}$. A point $P \in DB$ is* direct correlation-reachable *from a point $Q \in DB$ w.r.t. ε, k, δ, and λ (denoted by $\textsc{DirReach}_{\varepsilon,k}^{\lambda,\delta}(Q,P)$) if Q is a correlation core point, the correlation dimension of $\mathcal{N}_\varepsilon(P)$ is at least λ, and $P \in \mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_Q}(Q)$, formally:*

$\textsc{DirReach}_{\varepsilon,k}^{\lambda,\delta}(Q,P) \Leftrightarrow$

(1) $\textsc{Core}_{\varepsilon,k}^{\lambda,\delta}(Q)$

(2) $\textsc{CorDim}(\mathcal{N}_\varepsilon(P)) \leq \lambda$

(3) $P \in \mathcal{N}_\varepsilon^{\hat{\mathbf{M}}_Q}(Q)$.

Correlation-reachability is symmetric for correlation core points. Both points P and Q must find the other point in their corresponding correlation ε-neighborhood.

Definition 6.8 (correlation-reachability)
Let $\varepsilon, \delta \in \mathbb{R}_0^+$ ($\delta \approx 0$) and $k, \lambda \in \mathbb{N}$. A point $P \in DB$ is correlation-reachable *from a point $Q \in DB$ w.r.t. ε, k, δ, and λ (denoted by $\textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(Q,P)$) if there is a chain of points $P_1, \cdots P_n$ such that $P_1 = Q, P_n = P$ and P_{i+1} is*

direct correlation-reachable from P_i, formally:

$$\begin{aligned}&\textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(Q,P) \Leftrightarrow \\ &\quad \exists P_1,\ldots,P_n \in \mathcal{D} : P_1 = Q \ \wedge \ P_n = P \ \wedge \\ &\quad \forall i \in \{1,\ldots,n-1\} : \textsc{DirReach}_{\varepsilon,k}^{\lambda,\delta}(P_i,P_{i+1}).\end{aligned}$$

The correlation-reachability is the transitive closure of direct correlation-reachability.

Definition 6.9 (correlation-connectivity)
Let $\varepsilon \in \mathbb{R}_0^+$ and $k \in \mathbb{N}$. A point $P \in \mathcal{D}$ is correlation-connected *to a point $Q \in \mathcal{D}$ if there is a point $O \in \mathcal{D}$ such that both P and Q are correlation-reachable from O, formally:*

$$\begin{aligned}&\textsc{Connect}_{\varepsilon,k}^{\lambda,\delta}(Q,P) \Leftrightarrow \\ &\exists o \in \mathcal{D} : \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(O,Q) \ \wedge \ \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(O,P).\end{aligned}$$

Correlation-connectivity is a symmetric relation. A correlation-connected cluster can now be defined as a maximal correlation-connected set.

Definition 6.10 (correlation-connected set)
Let $\varepsilon, \delta \in \mathbb{R}_0^+$ and $k, \lambda \in \mathbb{N}$. A non-empty subset $\mathcal{C} \subseteq DB$ is called a density-connected set *w.r.t. ε, k, δ, and λ if all points in $\mathcal{C}$ are density-connected and $\mathcal{C}$ is maximal w.r.t. density-reachability, formally:*

$\textsc{ConSet}_{\varepsilon,k}^{\lambda,\delta}(\mathcal{C}) \Leftrightarrow$

(1) *Connectivity:* $\forall O,Q \in \mathcal{C} : \textsc{Connect}_{\varepsilon,k}^{\lambda,\delta}(O,Q)$

(2) *Maximality:* $\forall P,Q \in DB : Q \in \mathcal{C} \wedge \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(Q,P) \Rightarrow P \in \mathcal{C}$.

The following two lemmata are important for validating the correctness of our clustering algorithm. Intuitively, they state that we can discover a correlation-connected set for a given parameter setting in a two-step approach: First, choose an arbitrary correlation core point O from the database. Second, retrieve all points that are correlation-reachable from O. This approach yields the density-connected set containing O.

Lemma 6.1

Let $P \in DB$. If P is a correlation core point, then the set of points which are correlation-reachable from P is a correlation-connected set, formally:

$$\textsc{Core}_{\varepsilon,k}^{\lambda,\delta}(P) \wedge \mathcal{C} = \{O \in DB \mid \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,O)\} \Rightarrow \textsc{ConSet}_{\varepsilon,k}^{\lambda,\delta}(\mathcal{C}).$$

Proof.

(1) $\mathcal{C} \neq \emptyset$:

By assumption, $\textsc{Core}_{\varepsilon,k}^{\lambda,\delta}(P)$ *and thus,* $\textsc{CorDim}(\mathcal{N}_\varepsilon(P)) \leq \lambda$.

$\Rightarrow \textsc{DirReach}_{\varepsilon,k}^{\lambda,\delta}(P,P)$

$\Rightarrow \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,P)$

$\Rightarrow P \in \mathcal{C}$.

(2) Maximality:

Let $X \in \mathcal{C}$ and $Y \in DB$ and $\textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(X,Y)$.

$\Rightarrow \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,X) \wedge \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(X,Y)$

$\Rightarrow \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,Y)$ *(since correlation reachability is a transitive relation).*

$\Rightarrow Y \in \mathcal{C}$.

(3) Connectivity:

$\forall X,Y \in \mathcal{C} : \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,X) \wedge \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,Y)$

$\Rightarrow \textsc{Connect}_{\varepsilon,k}^{\lambda,\delta}(X,Y)$ *(via P).* ◇

Lemma 6.2 *Let $\mathcal{C} \subseteq DB$ be a correlation-connected set. Let $P \in \mathcal{C}$ be a correlation core point. Then $\mathcal{C}$ equals the set of points which are correlation-reachable from P, formally:*

$$\textsc{ConSet}_{\varepsilon,k}^{\lambda,\delta}(\mathcal{C}) \wedge P \in \mathcal{C} \wedge \textsc{Core}_{\varepsilon,k}^{\lambda,\delta}(P) \Rightarrow \mathcal{C} = \{O \in DB \mid \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,O)\}.$$

Proof.

Let $\bar{\mathcal{C}} = \{O \in DB \mid \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(P,O)\}$. *We have to show that $\bar{\mathcal{C}} = \mathcal{C}$:*

(1) $\bar{\mathcal{C}} \subseteq \mathcal{C}$: obvious from definition of $\bar{\mathcal{C}}$.

(2) $\mathcal{C} \subseteq \bar{\mathcal{C}}$: Let $Q \in \mathcal{C}$. By assumption, $P \in \mathcal{C}$ and $\textsc{ConSet}_{\varepsilon,k}^{\lambda,\delta}(\mathcal{C})$.

$\Rightarrow \exists O \in \mathcal{C} : \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(O,P) \wedge \textsc{Reach}_{\varepsilon,k}^{\lambda,\delta}(O,Q)$

$\Rightarrow \text{REACH}_{\varepsilon,k}^{\lambda,\delta}(P,O)$ *(since both O and P are correlation core points and correlation-reachability is symmetric for correlation core points.)*
$\Rightarrow \text{REACH}_{\varepsilon,k}^{\lambda,\delta}(P,Q)$ *(transitivity of correlation-reachability)*
$\Rightarrow Q \in \bar{\mathcal{C}}$. ◇

6.4 The Algorithm 4C

In the following, we describe the algorithm 4C which performs one pass over the database to find all correlation clusters for a given parameter setting. The pseudocode of the algorithm 4C is given in Figure 6.6.

```
4C(SetOfPoints DB, Real ε,δ, Integer k,λ)
  // assumption: each point in DB is marked as unclassified
  for each unclassified O ∈ DB do
STEP 1. test CORE^{λ,δ}_{ε,k}(O) predicate:
    compute N_ε(O);
    if |N_ε(O)| ≥ k then
      compute M_O;
      if CORDIM(N_ε(O)) ≤ λ then
        compute M̂_O and N_ε^{M̂_O}(O);
        test |N_ε^{M̂_O}(O)| ≥ k;
STEP 2.1. if CORE^{λ,δ}_{ε,k}(O) expand a new cluster:
    generate new clusterID;
    insert all X ∈ N_ε^{M̂_O}(O) into queue Φ;
    while Φ ≠ ∅ do
      Q = first point in Φ;
      compute R = {X ∈ DB | DIRREACH^{λ,δ}_{ε,k}(Q,X)};
      for each X ∈ R do
        if X is unclassified or noise then
          assign current clusterID to X
        if X is unclassified then
          insert X into Φ;
      remove Q from Φ;
STEP 2.2. if not CORE^{λ,δ}_{ε,k}(O) O is noise:
    mark O as noise;
```

Figure 6.6: The algorithm 4C.

At the beginning, each point is marked as unclassified. During the run of 4C, all points are either assigned a certain cluster identifier or marked as noise. For each point which is not yet classified, 4C checks whether this point is a correlation core point or not (see STEP 1 in Figure 6.6). If the point is a correlation core point, the algorithm expands the cluster belonging to this point (STEP 2.1). Otherwise the point is marked as noise (STEP 2.2). To find a new cluster, 4C starts in STEP 2.1 with an arbitrary correlation core point O and searches for all points that are correlation-reachable from O. Due to Lemma 6.2, this is sufficient to find the whole cluster containing the point O. When 4C enters STEP 2.1, a new cluster identifier "clusterID" is generated which will be assigned to all points found in STEP 2.1. 4C begins by inserting all points in the correlation ε-neighborhood of point O into a queue. For each point in the queue it computes all directly correlation-reachable points and inserts those points into the queue which are still unclassified. This is repeated until the queue is empty.

As discussed in Section 6.3, the results of 4C do not depend on the order of processing, i.e. the resulting clustering (number of clusters and association of core points to clusters) is determinate.

6.5 Complexity Analysis

The computational complexity with respect to the number of data points as well as the dimensionality of the data space is an important issue because the proposed algorithms are typically applied to large data sets of high dimensionality. The idea of our correlation-connected clustering method is founded on DBSCAN, a density-based clustering algorithm for Euclidean data spaces. The complexity of the original DBSCAN algorithm depends on the existence of an index structure for high-dimensional data spaces. The worst case complexity is $O(n^2)$, but the existence of an efficient index can reduce the complexity to $O(n \log n)$ [EKSX96]. DBSCAN is linear in the dimensionality of the data set for the Euclidean distance metric. To enables user adaptability of the distance function, a quadratic form distance metric can be applied instead of the Euclidean distance metric. In this case, the time complexity of DBSCAN is $O(d^2 \cdot n \log n)$. In contrast, subspace

clustering methods such as CLIQUE are known to be exponential in the number of dimensions [AY00, AGGR98].

We begin our analysis with the assumption of no index structure.

Lemma 6.3 *The overall worst-case time complexity of our algorithm on top of the sequential scan of the data set is* $O(d^2 \cdot n^2 + d^3 \cdot n)$.

Proof. *Our algorithm has to associate each point of the data set with a similarity measure that is used for searching neighbors (cf. Definition 6.4). We assume that the corresponding similarity matrix must be computed once for each point, and it can be held in the cache until it is no longer needed. The covariance matrix is filled with the result of an Euclidean range query which can be evaluated in* $O(d \cdot n)$ *time. Then the matrix is decomposed by using PCA which requires* $O(d^3)$ *time. For all points together, we have* $O(d \cdot n^2 + d^3 \cdot n)$.
Checking the correlation core point property according to Definition 6.6, and expanding a correlation-connected cluster requires for each point the evaluation of a range query with a quadratic form distance measure which can be done in $O(d^2 \cdot n)$. *For all points together, including the above cost for the determination of the similarity matrix, we obtain an worst-case time complexity of* $O(d^2 \cdot n^2 + d^3 \cdot n)$. ◇

Under the assumption that an efficient index structure for high-dimensional data spaces [BKK96, BBJ+00] is available, the complexity of all range queries is reduced from $O(n)$ to $O(\log n)$. Let us note that we can use Euclidean range queries as a filter step for the quadratic form range queries because no semi-axis of the corresponding ellipsoid exceeds ε. Therefore, the overall time complexity in this case is given as follows:

Lemma 6.4 *The overall worst case time complexity of our algorithm on top of an efficient index structure for high-dimensional data is* $O(d^2 \cdot n \log n + d^3 \cdot n)$.

Proof. *Analogous to Lemma 6.3.* ◇

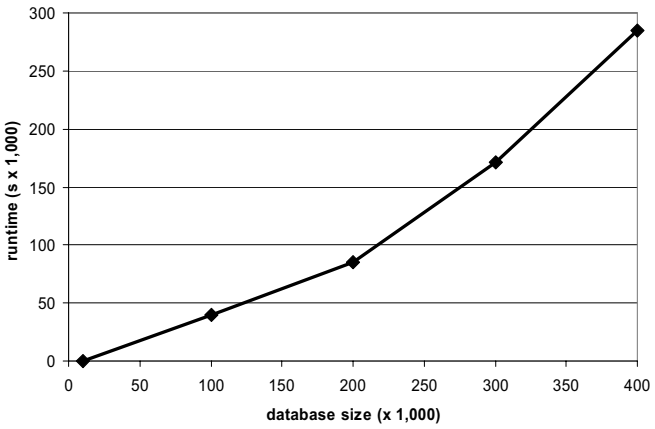


Figure 6.7: Scalability of 4C w.r.t. the database size.

6.6 Evaluation

In this section, we present a extensive evaluation of 4C. We implemented 4C as well as the three comparative methods DBSCAN, CLIQUE, and ORCLUS in JAVA. All experiments were run on a Linux workstation with a 2.0 GHz CPU and 2.0 GB RAM.

6.6.1 Efficiency

According to Section 6.5, the runtime of 4C scales superlinear with the number of input records. This is illustrated in Figure 6.7, showing the results of 4C applied to synthetic two-dimensional data of variable size.

6.6.2 Effectiveness

We evaluated the effectiveness of 4C on several synthetic data sets as well as on real-world data sets, including gene expression data and metabolome data. In addition, we compared the quality of the results of our method to the quality of the results of DBSCAN, ORCLUS, and CLIQUE. In all our experiments, we set the parameter $\kappa = 50$ as suggested in Section 6.3.2.

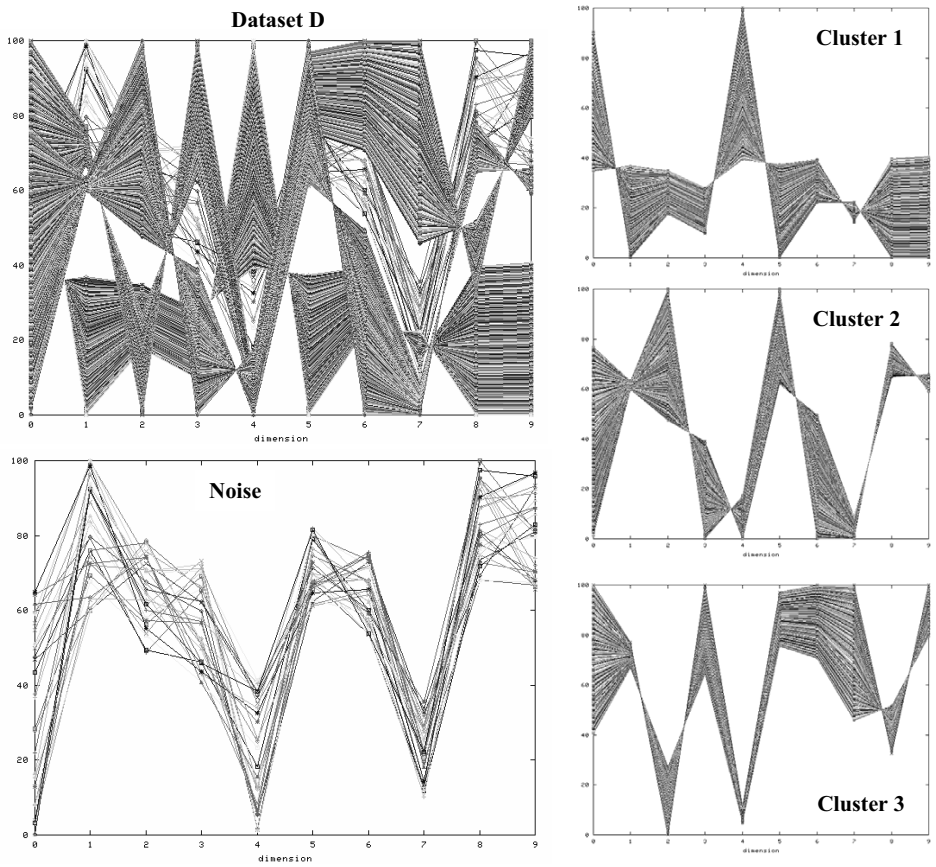


Figure 6.8: Clusters found by 4C on a 10D synthetic data set. Parameters: $\varepsilon = 10.0$, $k = 5$, $\lambda = 2$, $\delta = 0.1$.

Synthetic Data Sets

We first applied 4C on several synthetic data sets (with $2 \leq d \leq 30$) consisting of several dense, linear correlations. In all cases, 4C had no problems to identify the correlation-connected clusters. As an example, Figure 6.8 illustrates the transposed view of the three clusters and the noise 4C found on a sample 10-dimensional synthetic data set consisting of approximately 1,000 points.

Real-World Data Sets

Gene Expression Data. We applied 4C to the gene expression data set of [THC+99]. The data set is derived from time series experiments on the

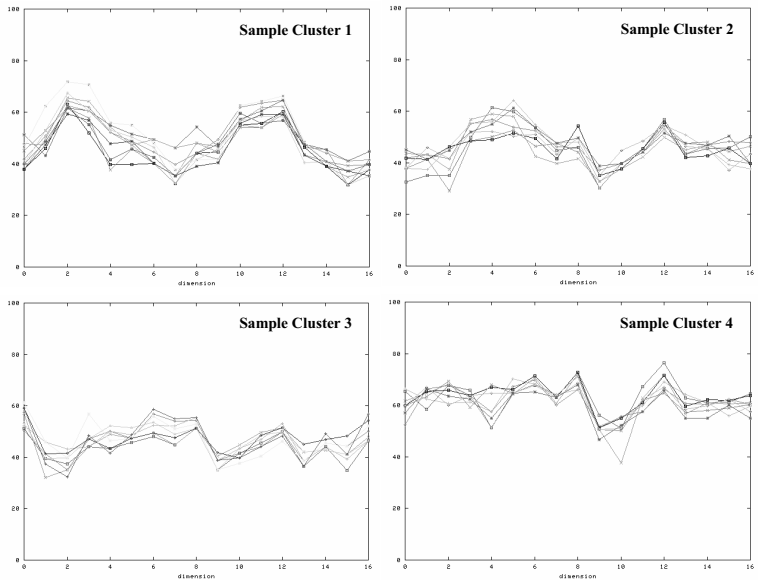


Figure 6.9: Sample clusters found by 4C on the gene expression data set. Parameters: $\varepsilon = 25.0$, $k = 8$, $\lambda = 8$, $\delta = 0.01$.

yeast mitotic cell cycle. The expression levels of approximately 3,000 genes are measured at 17 different time slots. Thus, we face a 17-dimensional data space to search for correlations, indicating coregulated genes. 4C found 60 correlation-connected clusters with few coregulated genes. The clusters contained between 10 and 20 genes, which is quite reasonable from a biological perspective. The transposed views of four sample clusters are depicted in Figure 6.9. All four clusters exhibit simple linear correlations on a subset of their attributes. Let us note that we also found other linear correlations which are rather complex to visualize. We also analyzed the results of our correlation clusters (based on the publicly available information resources on the yeast genome [Sac]) and found several biologically important implications. For example one cluster consists of several genes coding for proteins related to the assembly of the spindle pole required for mitosis (e.g. KIP1, SLI15, SPC110, SPC25, and NUD1). Another cluster contains several genes coding for structural constituents of the ribosome (e.g. RPL4B, RPL15A, RPL17B, and RPL39). The functional relationships of the genes in the clusters confirm the significance of the computed coregulation.

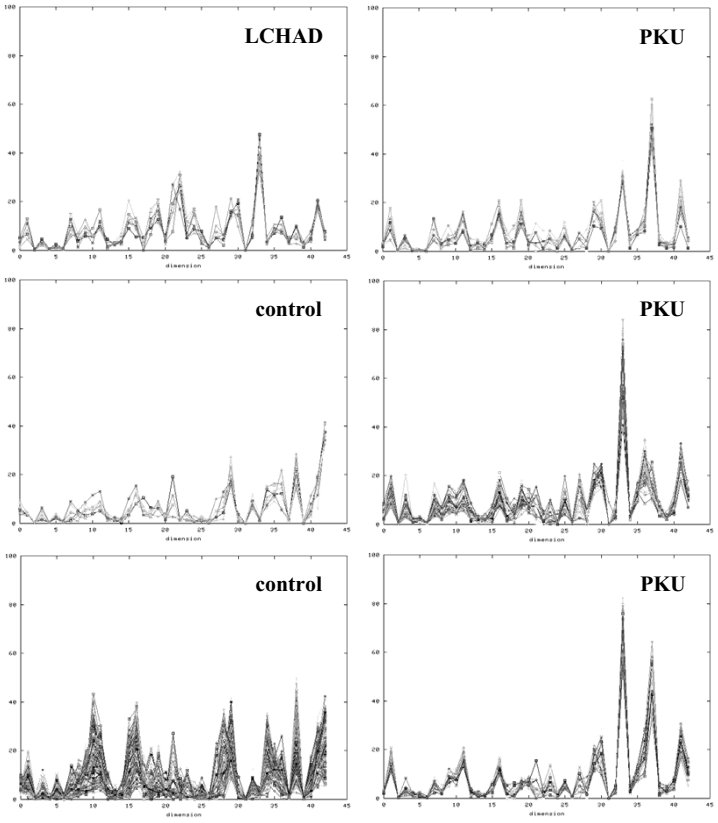


Figure 6.10: Clusters found by 4C on the metabolome data set. Parameters: $\varepsilon = 150.0$, $k = 8$, $\lambda = 20$, $\delta = 0.1$.

Metabolome Data. We applied 4C on a metabolome data set described in [LNRvK+02]. The data set consists of the concentrations of 43 metabolites in 2,000 human newborns. The newborns were labeled according to some specific metabolic diseases. Thus, the data set consists of 2,000 data points with $d = 43$. 4C detected six correlation-connected sets which are visualized in Figure 6.10. Cluster one and two (in the lower left corner marked with "control") consists of healthy newborns whereas the other clusters consists of newborns having one specific disease (e.g. "PKU" or "LCHAD"). The group of newborns suffering from "PKU" was split in three clusters. Several sick as well as healthy newborns were classified as noise.

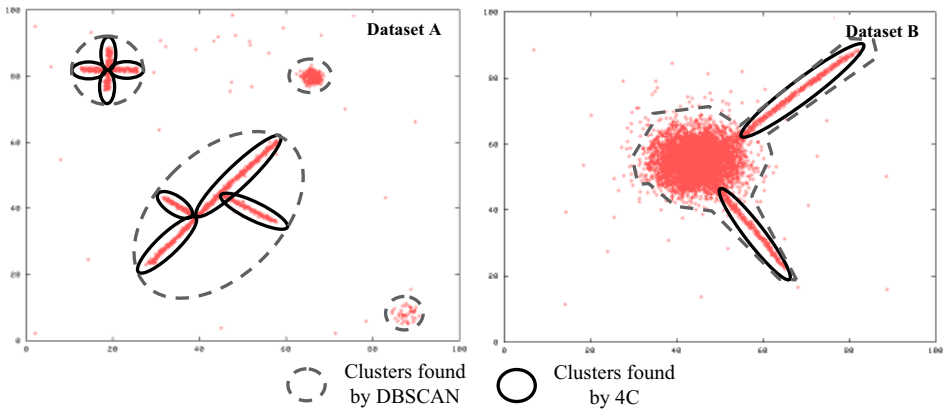


Figure 6.11: Comparison of 4C with DBSCAN.

Comparisons to Other Methods

We compared the effectiveness of 4C with related clustering methods, in particular the density-based clustering algorithm DBSCAN, the subspace clustering algorithm CLIQUE, and the projected clustering algorithm ORCLUS. For that purpose, we applied these methods on several synthetic data sets including two-dimensional data sets and higher dimensional data sets ($d = 10$).

Comparison with DBSCAN. The clusters found by DBSCAN and 4C applied to the two-dimensional data sets are depicted in Figure 6.11. In both cases, DBSCAN finds clusters which do not exhibit correlations (and thus are not detected by 4C). In addition, DBSCAN cannot distinguish varying correlations which overlap, e.g. both correlations in data set B in Figure 6.11, and treat such clusters as one density-connected set, whereas 4C can differentiate such correlations. We gain similar observations when we applied DBSCAN and 4C on the higher dimensional data sets. Let us note that these results are not unexpected since DBSCAN only searches for density-connected sets but does not search for correlations and thus cannot be applied to the task of finding correlation-connected sets.

Comparison with CLIQUE. A comparison of 4C with CLIQUE gained similar results. CLIQUE finds clusters in subspaces which do not exhibit correlations (and are not detected by 4C). On the other hand, CLIQUE is usually limited to axis-parallel clusters and thus cannot detect arbitrary correlations. These observations occur especially with higher dimensional data ($d \geq 10$ in our tests). Again, these results are not unexpected since CLIQUE only searches for axis-parallel subspace clusters (dense projections) but does not search for correlations. This empirically supports the suspicion that CLIQUE cannot be applied to the task of finding correlation-connected sets.

Comparison with ORCLUS. A comparison of 4C with ORCLUS resulted in quite different observations. In fact, ORCLUS computes clusters of correlated points. However, since it is k-medoid based, it suffers from the following two drawbacks: First, the choice of k is a rather hard task for real-world data sets. Even for synthetic data sets where we knew the number of clusters beforehand, ORCLUS often performs better with a slightly different value of k. Second, ORCLUS is rather sensitive to noise which often appears in real-world data sets. Since all points have to be assigned to a cluster, the locality of the analyzed correlations is often too coarse, i.e. the subsets of the points taken into account for correlation analysis are too large. As a consequence, the correlation clusters are often blurred by noise points and thus are hard to obtain from the resulting output. Figure 6.12 illustrates a sample three-dimensional synthetic data set, the clusters found by 4C are marked by black lines. Figure 6.13 depicts the points in each cluster found by ORCLUS ($k = 3$ yields the best result) separately. It can be seen, that the correlation clusters are — if detected — blurred by noise points. When we applied ORCLUS to higher dimensional data sets ($d = 10$), the choice of k became even more complex and the problem of noise points blurring the clusters, i.e. a too coarse locality, simply cumulated in the fact that ORCLUS often could not detect correlation clusters in high-dimensional data.

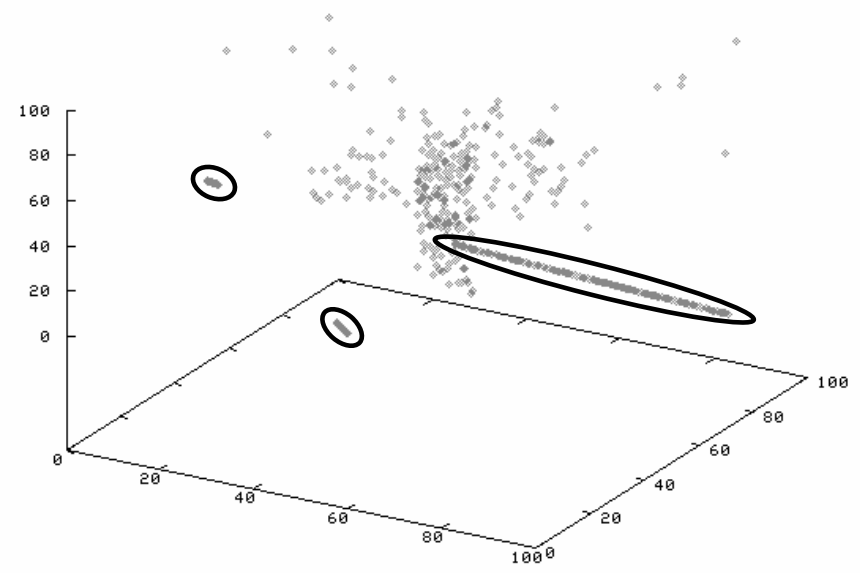


Figure 6.12: Three correlation-connected clusters found by 4C on a 3D data set. Parameters: $\varepsilon = 2.5$, $k = 8$. $\delta = 0.1$, $\lambda = 2$.

Input Parameters

The algorithm 4C needs four input parameters which are discussed in the following:

The parameter $\varepsilon \in \mathbb{R}_0^+$ specifies the size of the local areas in which the correlations are examined and thus determines the number of points which contribute to the covariance matrix and consequently to the correlation similarity measure of each point. It also participates in the determination of the density threshold a cluster must exceed. Its choice usually depends on the volume of the data space, i.e. the maximum value of each attribute and the dimensionality of the feature space. The choice of ε has two aspects. First, it should not be too small because in that case an insufficient number of points contribute to the correlation similarity measure of each point and, thus, this measure can be meaningless. On the other hand, ε should not be too large because then some noise points might be correlation-reachable from points within a correlation-connected cluster. Let us note that our experiments indicated that the second aspect is not significant for 4C (in contrast to ORCLUS).

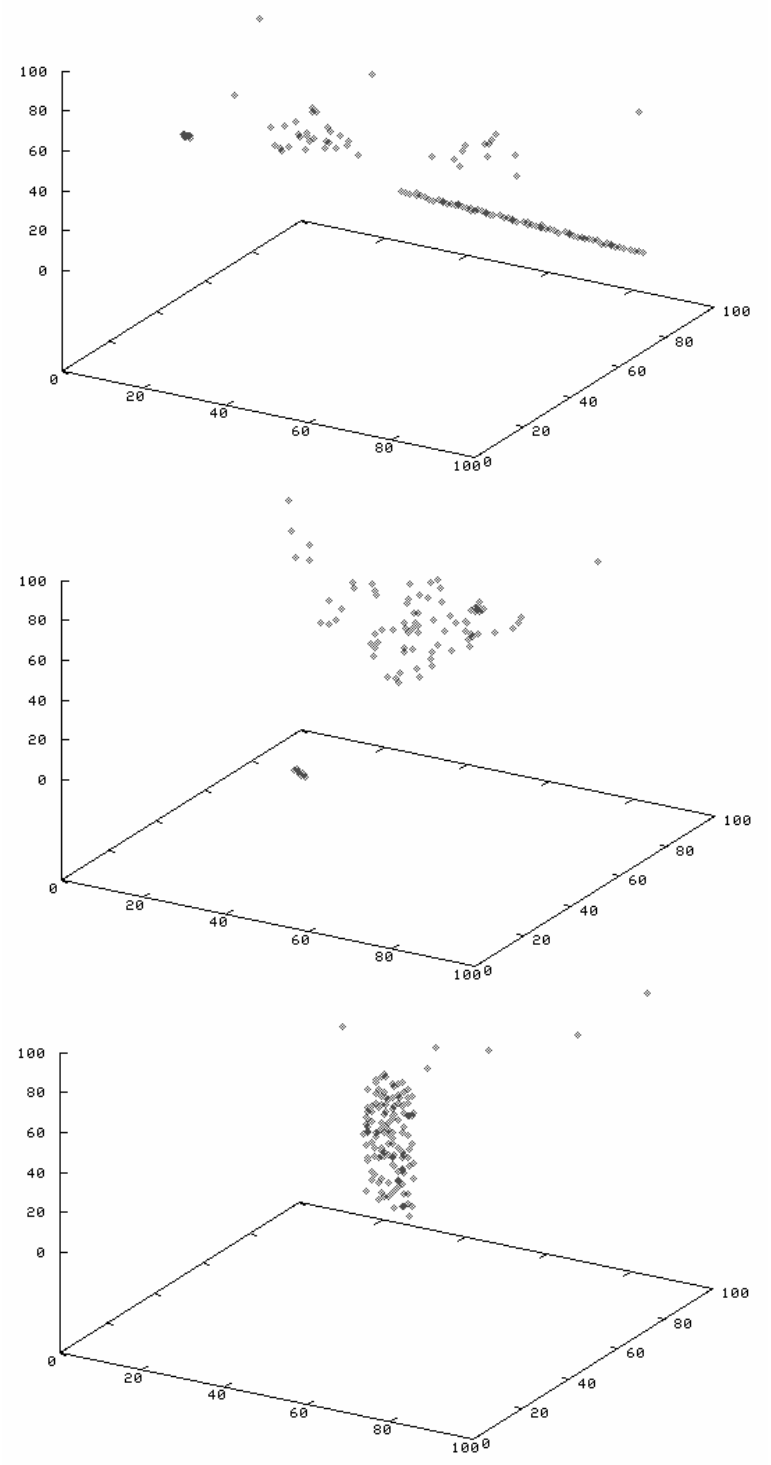


Figure 6.13: Clusters found by ORCLUS on the data set depicted in Figure 6.12. Parameters: $k = 3$, $l = 2$.

The parameter $k \in \mathbb{N}$ specifies the number of neighbors a point must find in an ε-neighborhood and in a correlation ε-neighborhood to exceed the density threshold. It determines the minimum cluster size. The choice of k should not be to small ($k \geq 5$ is a reasonable lower bound) but is rather insensitive in a broad range of values.

Both ε and k should be chosen hand in hand.

The parameter $\lambda \in \mathbb{N}$ specifies the correlation dimension of the correlation-connected clusters to be computed. As discussed above, the correlation dimension of a correlation-connected cluster corresponds to its intrinsic dimension. In our experiments, it turned out that λ can be seen as an upper bound for the correlation dimension of the detected correlation-connected clusters. However, the computed clusters tend to have a correlation dimension close to λ.

The parameter $\delta \in \mathbb{R}_0^+$ (where $0 \leq \delta \leq 1$) specifies the lower bound for the decision whether an eigenvalue is set to 1 or to $\kappa \gg 1$. It empirically turned out that the choice of δ influences the tightness of the detected correlations, i.e. how much local variance from the correlation is allowed. Our experiments also showed that $\delta \leq 0.1$ is usually a good choice.

6.7 Summary

In this chapter, we have proposed 4C, an algorithm for computing clusters of correlation-connected points. This algorithm searches for local subgroups of a set of feature vectors with a uniform correlation. Knowing a correlation is potentially useful because a correlation indicates a causal dependency between features. In contrast to well-known methods for the detection of correlations like the principal components analysis, our algorithm is capable to separate different subgroups in which the dimensionality as well as the type of the correlation (positive/negative) and the dependent features are different, even in the presence of noise points.

Our proposed algorithm 4C is determinate, robust with regard to noise, and efficient with a worst case time complexity between $O(d^2 \cdot n \log n + d^3 \cdot n)$ and $O(d^2 \cdot n^2 + d^3 \cdot n)$. In an extensive evaluation, data from gene expression analysis and metabolic screening have been used. Our experiments show

a superior performance of 4C over methods like DBSCAN, CLIQUE, and ORCLUS.

Part III

Clustering Complex Objects in Arbitrary Metric Spaces

Chapter 7

Similarity Models for Images - A Motivating Example

As we have seen in the first part of the thesis, a common approach to model data is to extract a vector of features from each object in the database and then use the Euclidean distance between those feature vectors as similarity measure for clustering. However, the effectiveness of this approach is highly dependent on the quality of the feature transformation. In this chapter, we will have a look at image data as one representative of complex objects where such a feature transformation no longer yields the desired effects. For image data, a lot of different aspects like color, texture, hue or saturation have to be integrated into the similarity model. Additional, the problem arises how to include the structural content information into the similarity model. The chapter should give the reader an impression why new techniques for clustering complex objects in arbitrary metric spaces are needed.

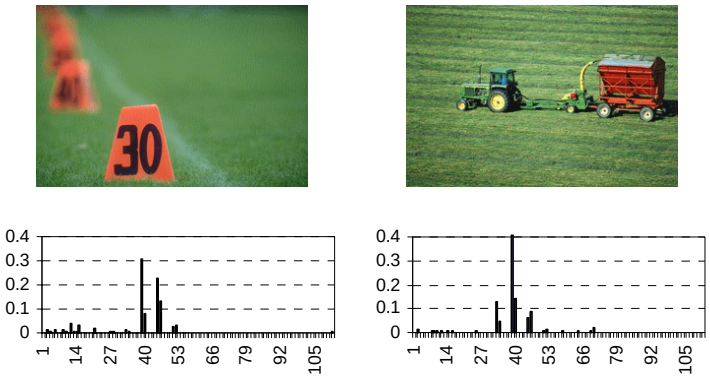


Figure 7.1: Two similar images and the corresponding 112-dimensional color histograms.

7.1 Color-Based Similarity of Images

A natural way to search for color images in a multimedia database is based on color distributions [HSE+95]. Two color images are defined to be similar if they contain approximately the same colors. This is formalized by means of a color histogram. After reducing and normalizing the color spectrum of the images to a manageable number of different colors, the images are analyzed. For each color, the ratio of pixels which are correspondingly colored is determined (cf. Figure 7.1). An obvious way to compare color histograms is to interpret them as vectors in Euclidean space. This approach leads to the difficulty that all pairs of different colors are interpreted as likewise dissimilar. In human perception, however, some colors are very similar to each other, e.g. red and orange, whereas others are very dissimilar, e.g. yellow and blue. The so-called *cross-talk* between similar colors can be taken into account if instead of the Euclidean distance between the histogram vectors the following quadratic form distance metric is used: $d_A(x,y) = \sqrt{(x-y) * A * (x-y)^T}$.

In this formula, the similarity matrix A contains the information which colors are similar to each other and to what degree. The components a_{ij} of the positive semi-definit matrix A denote the similarity of the components i and j of the respective vectors. This definition of similarity in color images is for example used by the QBIC system [FBF+94] or in [SK97].

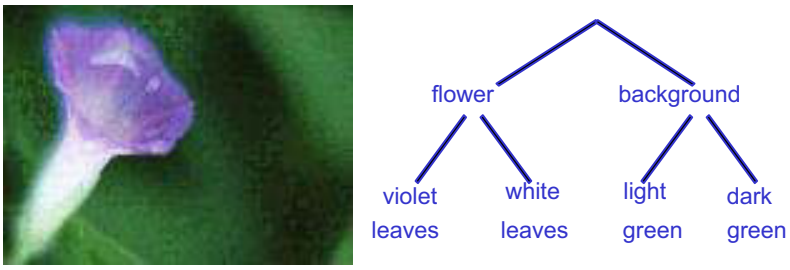


Figure 7.2: An image and its inherent structure.

7.2 Content-based Similarity of Images

Often, the user is not satisfied with getting images with a similar color, but is interested in images with a similar content or structure. Thus, numerous approaches for content-based image retrieval have been proposed in the literature. They are based on features like color [FBF+94], shape [MKL97] or texture [CHH97]. In [FCE00] a graph-based approach similar to the one described in Section 7.2.2 is used while in [TVJD95] an edit distance measure is used to measure the similarity of topological arrangements. [SWS+00] gives a nice overview of the different approaches.

Usually, images contain an inherent structure which may be hierarchical. An example can be seen in Figure 7.2. In the following, we describe two models for image representation and similarity measurement which take structural as well as content features like color into account [KKS04].

7.2.1 Image Representation as Containment Trees

To utilize the inherent structure of images for content-based retrieval, one can model them as so-called *containment trees*. Containment trees model the hierarchical containment of image regions within others (see Figure 7.3 for an illustration).

To extract the containment tree of an image, the image is first segmented based on the colors of the regions. This is done by using a region growing algorithm. The resulting segments are attributed with their color and size relative to the complete image. In a second step, the containment hierarchy is extracted from the set of segments by determining which regions are com-

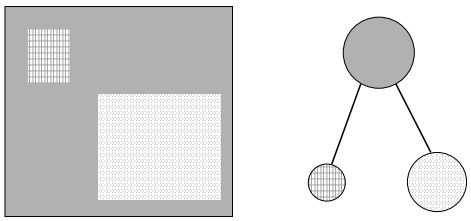


Figure 7.3: An image and its containment tree.

pletely contained in other regions. In this context, a region R_{in} is said to be contained in a region R_{cont} if for every point $p \in R_{in}$ and every straight line $L \ni p$ there exist two points $o_1, o_2 \in R_{cont}$ with $o_1, o_2 \in L$ and o_1, o_2 are on opposite sides of p.

To measure the similarity of containment trees, special similarity measures for attributed trees are necessary. A successful similarity measure for attributed trees is the edit distance. Well-known from string matching [Lev66, WF74], the edit distance is the minimal number of edit operations necessary to transform one tree into the other. The basic form allows two edit operations, i.e. the insertion and the deletion of a node. In the case of attributed nodes, the change of a node label is introduced as a third basic operation. A great advantage of using the edit distance as a similarity measure is that along with the distance value a mapping between the nodes in the two trees is provided in terms of the edit sequence. The mapping can be visualized and can serve as an explanation of the similarity distance to the user.

However, as the computation of the edit-distance is NP-complete [ZSS92], constrained edit distances like the degree-2 edit distance [ZWS96] have been introduced. The main idea behind this distance measure is that only insertions or deletions of nodes with a maximum number of two neighbors are allowed. While yielding good results, the degree-2 edit distance is still computationally complex and, therefore, of limited benefit for searching or clustering in large databases. In Chapter 9, a filter and refinement architecture for the degree-2 edit distance is presented to overcome this problem. A set of new filter methods for structural and for content-based information as well as ways to flexibly combine different filter criteria are presented.

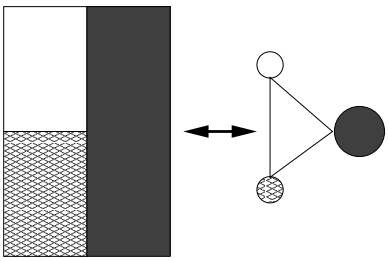


Figure 7.4: An image and its segmentation graph.

7.2.2 Image Representation as Segmentation Graphs

Graphs are another way to model images for content-based similarity search. They were successfully used for shape retrieval [HCH99], object recognition [KKV90] or face recognition [WFKvdM97]. In this section, we describe a content-based image retrieval system based on graphs which are extracted from images in a similar way as the trees in the preceding section.

To extract graphs from the images, they are segmented with a region growing technique and neighboring segments are connected by edges to represent the neighboring relationship (see Figure 7.4 for an illustration). Each segment is assigned four attribute values which are the size, the height and width of the bounding box and the color of the segment. The values of the first three attributes are expressed as a percentage relative to the image size, height and width in order to make the measure invariant to scaling.

Most known similarity measures for attributed graphs are either limited to a special type of graph or are 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*.

Definition 7.1 (edge matching distance)

Let $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ be two attributed graphs. Without loss of generality, we assume that $|E_1| \geq |E_2|$. The complete bipartite graph $G_{em}(V_{em} = E1 \cup E_2 \cup \Delta, E_1 \times (E_2 \cup \Delta))$, where Δ represents an empty dummy edge, is called the edge matching graph of G_1 and G_2. An edge matching between G_1 and G_2 is defined as a maximal matching in G_{em}. Let there be a non-negative metric cost function $c : E_1 \times (E_2 \cup \Delta) \rightarrow \mathbb{R}_0^+$. The

edge matching distance between G_1 and G_2, denoted by $d_{match}(G_1, G_2)$, is defined as the cost of the minimum-weight edge matching between G_1 and G_2 with respect to the cost function c.

The authors demonstrate that the edge matching distance is a meaningful similarity measure for attributed graphs and that it enables efficient clustering of structured data. In [KS03] there is also a filter-refinement architecture and an accompanying set of filter methods presented to reduce the number of necessary distance calculations during similarity search.

7.3 Combining Multiple Image Representations for Clustering

As we have seen in the previous two sections, there are different ways to represent image data. All those different similarity models for image data have their own advantages and disadvantages. Using for example text descriptions of images, one is able to cluster all images related to a certain topic, but those images need not look alike. Using color histograms instead, the images are clustered according to the distribution of color in the image. But as only the color information is taken into account, a green meadow with some flowers and a green billiard table with some colored balls on it can obviously not be distinguished by this similarity model. On the other hand, a similarity model taking content information into account might not be able to distinguish images of different colors.

Traditional clustering algorithms are based on one representation space and thus take only one of those representations into account during clustering. We argue that no single representation of an image models the intuitive notion of similar images adequately. Thus, for clustering complex objects like images it would be highly desirable to take different representations into account. In the next chapter, we will show how density-based clustering can be expanded to clustering complex objects like images, taking multiple representations into account.

Chapter 8

Clustering Multi-Represented Objects with Noise

Traditional clustering algorithms are based on one representation space, usually a vector space. However, in a variety of modern applications, multiple representations exist for each object. In the last chapter we have seen that images can be represented using different similarity models. In this chapter, we present an efficient density-based approach to cluster such multi-represented data, taking all available representations into account. We propose two different techniques to combine the information of all available representations dependent on the application. The evaluation part shows that our approach is superior to existing techniques. Parts of this material have been published in [KKPS04b, KKPS04a].

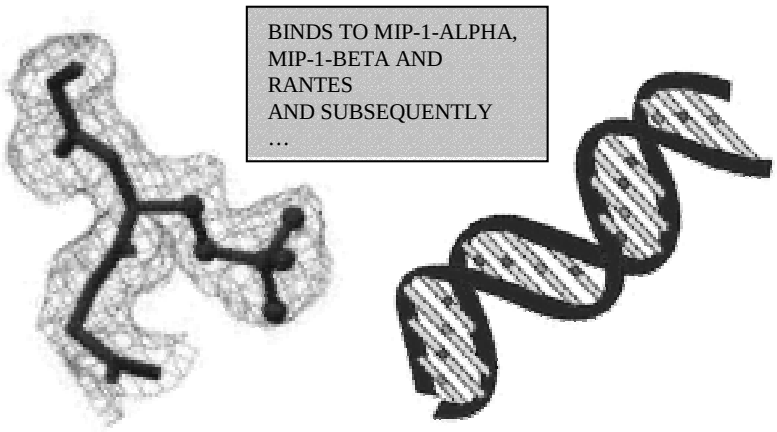


Figure 8.1: Three different representations for a protein.

8.1 Introduction

Traditional clustering methods are based on one representation space, usually a vector space of features and a corresponding distance measure. But for a variety of modern applications such as images, biomolecular data, CAD-parts or multi-media files mined from the internet, it is problematic to find a common feature space that incorporates all given information. Proteins for example are characterized by an amino acid sequence, a secondary and a tertiary structure. Additionally, protein databases such as Swissprot [BBA+03] provide meaningful text descriptions of the stored proteins (see Figure 8.1 for an illustration). In CAD-catalogues, the parts are represented by some kind of 3D model like Bézier curves, voxels or polygon meshes and additional textual information like descriptions of technical and economical key data. Another example is biometric data which consist of speech patterns, fingerprints and facial features. We call this kind of data *multi-represented data* since any data object might provide several different representations.

To cluster multi-represented data, using the established clustering methods would require to restrict the analysis to a single representation or to construct a feature space, comprising all representations. However, the restriction to a single feature space would not consider all available information and the construction of a combined feature space demands great care when constructing a combined distance function.

In this chapter, we propose a method to integrate multiple representations directly into the clustering algorithm. Our method is again based on the density-based clustering algorithm DBSCAN which is presented in Section 2.2. Since our method employs a separated feature space for each representation, it is not necessary to design a new suitable distance measure for each new application. Additionally, the handling of objects that do not provide all possible representations is integrated naturally without defining dummy values to compensate the missing representations. Last but not least, our method does not require a combined index structure, but benefits from each index that is provided for a single representation. Thus, it is possible to employ highly specialized index structures and filters for each representation instead of using very generally designed metric trees. We evaluate our method for two example applications. The first is a data set consisting of protein sequences and text descriptions. Additionally, we applied our method to the clustering of images.

The rest of the chapter is organized as follows. After this introduction, we present related work on clustering and data mining of multi-represented objects and describe the feature spaces that are used in our experiments. Section 8.3 formalizes the problem and introduces our new clustering method. In our experimental evaluation that is given in Section 8.4, we introduce a new quality measure to judge the quality of a clustering with respect to a reference clustering and display the results achieved by our method in comparison with the other mentioned approaches. The last section summarizes the chapter and presents some ideas for future research.

8.2 Related Work

8.2.1 Multi-Represented Data Mining

There are several problems that are closely related to the clustering of multi-represented data. Data mining of multi-instance objects [WFP03] is based on the precondition that each data object might be represented by more than one instance in a common data space. However, all instances that are employed are elements of the same data space and multi-instance objects

were predominantly treated with respect to classification not to clustering.

A similar setting to the clustering of multi-represented objects is the clustering of heterogenous or multi-typed objects [WZC+03, ZCM02] in web mining. In this setting, there are also multiple databases, each yielding objects in a separated data space. Each object within these data spaces may be related to an arbitrary amount of data objects within the other data spaces. The framework of reinforcement clustering employs an iterative process based on an arbitrary clustering algorithm. It clusters one dedicated data space while employing the other data spaces for additional information. It is also applicable for multi-represented objects. However, due to its dependency on the data space for which the clustering is started, it is not well suited to solve our task. Since, to the best of our knowledge, reinforcement clustering is the only other clustering algorithm that is directly applicable to multi-represented objects, we use it for comparison in our evaluation section.

8.2.2 Application Domains

In this section, we give a brief overview over the four data representations we employ in our experimental evaluation. Proteins can be regarded as sequences of the 20 amino acids. Therefore, each sequence is mapped into a 436 dimensional feature space. The first 400 features are 2-grams of successive amino acids. The last 36 dimensions are 2-grams of 6 exchange groups that the amino acids belong to [DK02]. 2-grams (a subsequence of length 2) are used to preserve the sequential character of the data. To compare the derived feature vectors, we employ Euclidian distance. To process text documents, we rely on projecting the documents into the feature space of relevant terms. Documents are described by a vector of term frequencies weighted by the inverse document frequency (TFIDF) [Sal89]. We employed cosine distance to compare the TFIDF-vectors. For images, we use two different representations. The first representation is a 64-dimensional color histogram. In this case, we use the weighted distance between those color histograms, represented as a quadratic form distance function as described in Section 7.1. The second representation are containment trees as described in Section 7.2.

8.3 Clustering Multi-Represented Objects

8.3.1 Preliminaries

Let DB be a database consisting of n objects. Let $R_i(o)$ be a function that maps the object o onto the representation i, where i is a form of representation, e.g. a tree or a graph representation of the object o. Let $R := \{R_1, ..., R_m\}$ be the set of different representations, existing for objects in DB. Each object $o \in DB$ is therefore described by at most m different representations, i.e. $o := \{R_1(o), R_2(o), ..., R_m(o)\}$. If all different representations exist for o, than $|o| = m$ else $|o| < m$. The distance function for a representation R_i is denoted by $dist_i$. We assume that $dist_i$ is symmetric and reflexive. In the following, we call the ε_i-neighborhood of an object o in one special representation R_i its local ε-neighborhood w.r.t. R_i.

Definition 8.1 (local ε_i-neighborhood w.r.t. R_i)
Let $o \in DB$, $\varepsilon_i \in \mathbb{R}_0^+$, $R_i \in R$. The local ε_i-neighborhood w.r.t. R_i *of o, denoted by $\mathcal{N}_\varepsilon^{R_i}(o)$, is defined by*

$$\mathcal{N}_\varepsilon^{R_i}(o) = \{x \in DB \mid dist_i(R_i(o), R_i(x)) \leq \varepsilon_i\}.$$

Note that ε_i can be chosen optimally for each representation.

8.3.2 General Idea of Clustering Multi-Represented Objects

The simplest way of clustering multi-represented objects is to select one representation R_i and cluster all objects according to this representation. However, this approach restricts data analysis to a limited part of the available information and does not use the remaining representations to find a meaningful clustering. Another way to handle multi-represented objects is to combine the different representations and use a combined distance function. Then any established clustering algorithm can be applied. However, this approach yields several drawbacks. First of all, the feature spaces of the different object representations might have various distance functions that are specialized to a certain kind of data, but often are not applicable to general data spaces. For example, for text objects the most established distance measure is the cosine distance, whereas trees and sequences are

often compared by variants of the edit distance. To combine these different approaches of similarity into one common distance function is difficult since certain distance functions like edit distance do not necessarily provide a finite range of values. Thus, a normalization to achieve comparability for each representation is a difficult task. Another problem of combined distance functions is the handling of missing representations. The part of the combined distance that relates to a missing representation has to be considered somehow. A common approach is to define some dummy value. However, the choice of such a dummy value might have a major influence on the distance and thus has to be considered carefully. Last but not least, the efficiency of processing ε-range queries strongly depends on the use of index structures and filters. Since these index structures are also dependent on the employed distance measures, building a common feature space usually prohibits the use of specialized index structures. Therefore, for combined data spaces only very general index structures like metric trees [CNBYM01] are applicable.

The idea of our approach is to combine the information of all different representations as early as possible, i.e. during the run of the clustering algorithm, and as late as necessary, i.e.after using the different distance functions of each representation. To do so, we adapt the core object property proposed for DBSCAN. To decide whether an object is a core object, we use the local ε-neighborhoods of each representation and combine the results to a global neighborhood. Therefore, we must adapt the predicate direct density-reachability proposed for DBSCAN. In the next two subsections, we will show how we can use the concepts of union and intersection of local neighborhoods to handle multi-represented objects.

8.3.3 Union of Different Representations

This variant is especially useful for sparse data. In this setting, the clusterings in each single representation will provide several small clusters and a large amount of noise. Simply enlarging ε would relief the problem, but on the other hand, the separation of the clusters would suffer. The union-method assigns objects to the same cluster if they are similar in at least one of the representations. Thus, it keeps up the separation of local clusters,

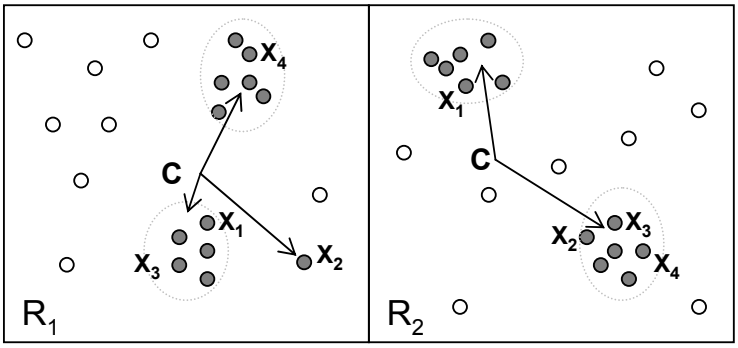


Figure 8.2: Union method: local clusters and a noise object are aggregated to a multi-represented cluster **C**.

but still overcomes the sparsity. If the object is placed in a dense area of at least one representation, it is still a core object regardless of how many other representations are missing. Thus, we do not need to define dummy values. Figure 8.2 illustrates the basic idea.

We adapt some of the definitions of DBSCAN to capture our new notion of clusters. To decide whether an object o is a union core object, we unite all local ε_i-neighborhoods and check whether there are enough objects in the global neighborhood, i.e. whether the global neighborhood of o is dense.

Definition 8.2 (union core object)
Let $\varepsilon_1, \varepsilon_2, ..., \varepsilon_m \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $o \in DB$ is called union core object*, denoted by $\text{CoreU}^k_{\varepsilon_1,..,\varepsilon_m}(o)$ if the union of all local ε-neighborhoods contains at least k objects, formally:*

$$\text{CoreU}^k_{\varepsilon_1,..,\varepsilon_m}(o) \Leftrightarrow | \bigcup_{R_i(o) \in o} \mathcal{N}^{R_i}_{\varepsilon}(o) | \geq k.$$

Definition 8.3 (direct union-reachability)
Let $\varepsilon_1, \varepsilon_2, .., \varepsilon_m \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $p \in DB$ is directly union-reachable *from $q \in DB$ if q is a union core object and p is an element of at least one local $\mathcal{N}^{R_i}_{\varepsilon}(q)$, formally:*

$$\text{DirReachU}^k_{\varepsilon_1,..,\varepsilon_m}(q,p) \Leftrightarrow \text{CoreU}^k_{\varepsilon_1,..,\varepsilon_m}(q) \wedge \exists\, i \in \{1,..,m\} : R_i(p) \in \mathcal{N}^{R_i}_{\varepsilon}(q).$$

The predicate direct union-reachability is obviously symmetric for pairs of core objects if all $dist_i$ are as demanded symmetric distance functions.

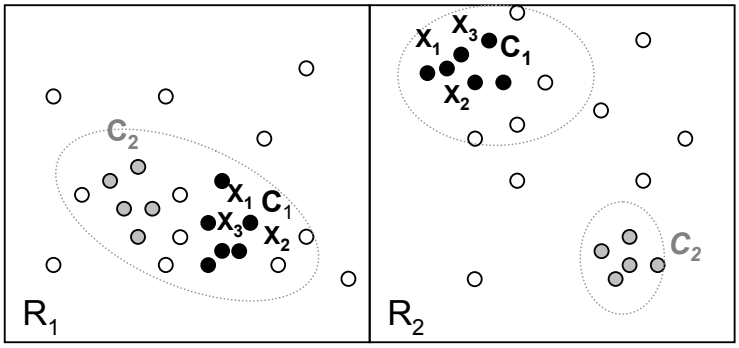


Figure 8.3: Intersection method: a local clustering is divided into the clusters C_1 and C_2.

Union-reachability and union-connectivity can be defined analogously to the original DBSCAN. A union-connected cluster is then defined as a set of union-connected objects which is maximal w.r.t. union-reachability. Thus, given the parameters $\varepsilon_1, ..., \varepsilon_m$ and k, we can discover a union-connected cluster in a two-step approach. First, we choose an arbitrary database object o, satisfying the union core object property. Second, we retrieve all objects that are union-reachable from o, thereby obtaining the cluster containing o.

8.3.4 Intersection of Different Representations

The intersection method is well suited for data where each representation includes different aspects of the data. In this case, clustering the data according to only one representation will yield rather unspecific clusters, because the data are only separated according to one specific aspect of the data. For such data, the intersection-method requires that a cluster should contain only objects which are similar according to all representations. Thus, this method is useful if all different representations exist, but the derived distances do not adequately mirror the intuitive notion of similarity. The intersection-method is used to increase the cluster quality by finding purer clusters. Figure 8.3 illustrates the basic idea.

To decide whether an object o is an intersection core object, we examine whether o is a core object in each involved representation. Of course, we use different ε-values for each representation to decide, if there are enough

objects in the local ε-neighborhood. The parameter k is used to decide, if there are enough objects in the global ε-neighborhood, i.e. the intersection of all local neighborhoods contains at least k objects.

Definition 8.4 (intersection core object)
Let $\varepsilon_1, \varepsilon_2, ..., \varepsilon_m \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $o \in DB$ is called intersection core object*, denoted by* $\text{CoreIS}^k_{\varepsilon_1,..,\varepsilon_m}(o)$ *if the intersection of all its local ε_i-neighborhoods contain at least k objects, formally:*

$$\text{CoreIS}^k_{\varepsilon_1,..,\varepsilon_m}(o) \Leftrightarrow | \bigcap_{i=1,..,m} \mathcal{N}_\varepsilon^{R_i}(o) | \geq k.$$

Using this new property, we can now define direct intersection-reachability in the following way:

Definition 8.5 (direct intersection-reachability)
Let $\varepsilon_1, \varepsilon_2, ..., \varepsilon_m \in \mathbb{R}_0^+$, $k \in \mathbb{N}$. An object $p \in DB$ is directly intersection-reachable *from $q \in DB$ if q is an intersection core object and p is an element of all local $\mathcal{N}_\varepsilon(q)$, formally:*

$$\text{DirReachIS}^k_{\varepsilon_1,..,\varepsilon_m}(q,p) \Leftrightarrow \text{CoreIS}^k_{\varepsilon_1,..,\varepsilon_m}(q) \wedge \forall i = 1,..,m : R_i(p) \in \mathcal{N}_\varepsilon^{R_i}(q) \ .$$

All the other definitions can be defined analogously to DBSCAN as described in Section 8.3.3. Figure 8.3 illustrates the effects of this method.

8.3.5 Determination of Density Parameters

In [EKSX96] a heuristic is presented to determine the ε-value of the "thinnest" cluster in the database. This heuristic is based on a diagram that represents sorted *knn*-distances of all given objects. In the case of multi-represented objects, we have to choose ε for each dimension separately, whereas k can be chosen globally. The parameter k is determined by the user. Then, the system computes the *knn*-distance diagrams for the given global k (one diagram for each representation). The user has to choose a so-called border object o for each representation. The ε for the i-th representation is given by the *knn*-distance of the border object of R_i. An example of a *knn*-distance diagram is shown in Figure 8.4. Let us note that this method still allows a certain range of ε-values to be chosen. The selection should mirror the

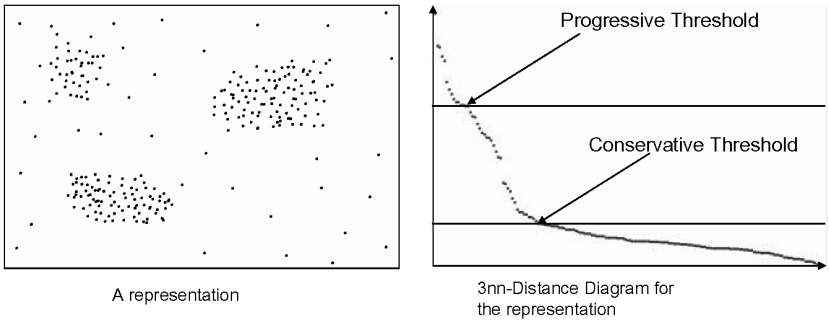


Figure 8.4: A 2D sample data set (left) and the corresponding 3-nn distance diagram (right).

different requirements of the proposed methods. For the union method, it is more advisable to choose a lower or conservative value since its characteristic demands that the elements of the local ε-neighborhood should really be similar. For the intersection-method, the ε-value should be selected progressively, i.e. at the upper rim of the range. This selection reflects that the objects of a cluster need not be too similar for a single representation, because it is required that they are similar with respect to all representations.

8.4 Performance Evaluation

To demonstrate the capability of our method, we performed a thorough experimental evaluation for two types of applications. We implemented the proposed clustering algorithm in Java 1.4. All experiments were processed on a work station with a 2.6 GHz Pentium IV processor and 2 GB main memory.

8.4.1 Deriving Meaningful Groupings in Protein Databases

The first set of experiments was performed on protein data that are represented by amino acid sequences and text descriptions. Therefore, we employed entries of the Swissprot protein database [BBA+03], belonging to five functional groups (cf. Table 8.1) and transformed each protein into a pair of feature vectors. To represent the text descriptions, we chose 100 words of medium frequency. To represent the sequence data the 436 2-gram

	Set 1	**Set 2**	**Set 3**	**Set 4**	**Set 5**
Name	Isomerase	Lyase	Signal Transducer	Oxidoreductase	Transferase
Classes	16	35	39	49	62
Objects	501	1640	2208	3399	4086

Table 8.1: Description of the protein data sets.

features already mentioned in Section 8.2 were used. Since Swissprot entries provide a unique mapping to the classes of Gene Ontology [Con00], a reference clustering for the selected proteins was available. Thus, we were able to measure a clustering of Swissprot entries by the degree it reproduces the class structure provided by Gene Ontology. To have an exact measure for this degree, we employed the class entropy in each cluster. However, there are two effects that have to be considered to obtain a fair measure of a clustering with noise. First, a large cluster of a certain entropy should contribute more to the overall quality of the clustering than a rather small cluster providing the same quality. The second effect is that a clustering having a 5% noise ratio should be ranked higher than a clustering having the same average entropy for all its clusters, but contains 50% noise.

To consider both effects, we propose the following quality measure for comparing different clusterings with respect to a reference clustering.

Definition 8.6 (quality measure)
Let O be the set of data objects, let $C = \{C_i | C_i \subset O\}$ be the set of clusters and let $K = \{K_i | K_i \subset O\}$ be the reference clustering of O. Then we define:

$$Q_K(C) = \sum_{C_i \in C} \frac{|C_i|}{|O|} \cdot (1 + entropy_K(C_i))$$

where $entropy_K(C_i)$ denotes the entropy of cluster C_i with respect to K.

The idea is to weight every cluster by the percentage of the complete data objects belonging to it. Thus, smaller clusters are less important than larger ones and a clustering providing an extraordinary amount of noise can contribute only the percentage of clustered objects to the quality. Let us note that we add 1 to the cluster entropies. Therefore, we measure the reference clustering K with the quality score of 1 and a worst case clustering,

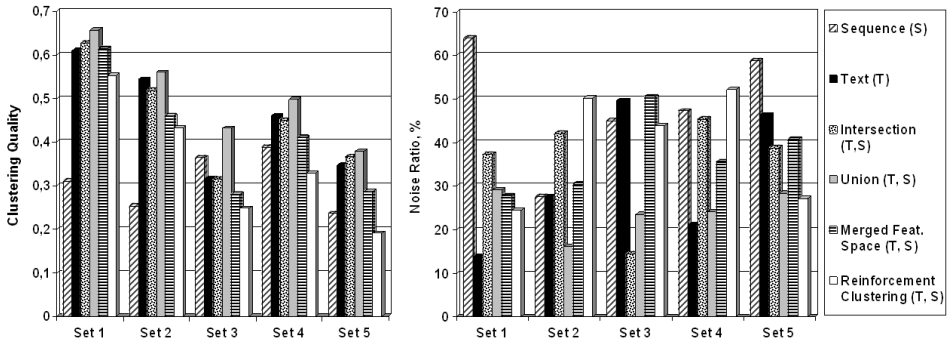
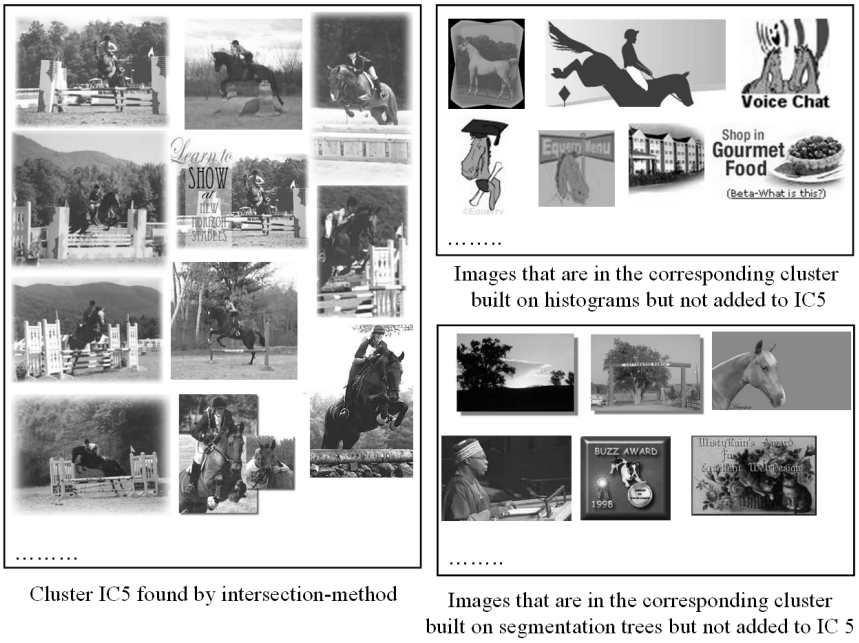


Figure 8.5: Clustering quality and noise ratio.

e.g. no clusters are found at all, with the score of 0. To relate the quality of the clustering achieved by our methods to the results of former methods, we compared it to 4 alternative approaches. First, we clustered text and sequences separately, using only one of the representations. A second approach combines the features of both representations into a common feature space and employs the cosine distance to relate the resulting feature vectors. As the only other clustering method that is able to handle multi-represented data, we additionally compared our methods to reinforcement clustering [WZC+03, ZCM02]. We used DBSCAN as underlying clustering algorithm. For reinforcement clustering we ran 10 iterations and tried several values of the weighting parameter α. The local ε-parameters were selected as described above and we chose $k = 2$. To consider the different requirements of both methods, for each data set a progressive and a conservative ε-value was determined. All approaches were run for both settings and the best results are displayed.

The left diagram of Figure 8.5 displays the derived quality for those four methods and the two variants of our method. In all five test sets, the union-method using conservative ε-values outperformed any of the other algorithms. Furthermore, the noise ratio for each data set was between 16% and 28% (cf. Figure 8.5, right), indicating that the main portion of the data objects belongs to some cluster. The intersection method using progressive ε-parameters performed comparably well, but was too restrictive to overcome the sparseness of the data as good as the union-method.



Cluster IC5 found by intersection-method

Images that are in the corresponding cluster built on histograms but not added to IC5

Images that are in the corresponding cluster built on segmentation trees but not added to IC 5

Figure 8.6: Example of an image cluster.

8.4.2 Clustering Images by Multiple Representations

Clustering image data is a good example for the usefulness of the intersection-method. As we have seen in the previous chapter, there are a lot of different similarity models for images. Using our intersection approach one is able to get the best out of all these different types of representations. Since the similarity in one representation is not really sound, the intersection-method is well-suited to find clusters of better quality for this application.

Experiments with Images from the Web

For our first test setting we used 2,150 images downloaded from the web. We used two different representations based on color histograms and segmentation trees, respectively. For the second representation, the images were first divided into segments of similar color. In a second step, a tree was created from those segments by iteratively applying a region-growing algorithm which merges neighboring segments if their colors are alike. As we do not have any class labels to measure the quality of our clustering, we can only

describe the results we achieved. In general, the clusters we got when using both representations were more accurate than the clusters we got when using each representation separately. Of course, the noise ratio increased for the intersection-method. Figure shows one representative sample cluster of images we found with the intersection-method. Using this method, very similar images are clustered together, e.g. the images contained in the left rectangle of Figure 8.6. When clustering each single representation, a lot of additional images were added to the corresponding cluster. The right rectangles of Figure 8.6 display additional images that were grouped with the corresponding cluster when clustering the images with respect to a single representation. When using the intersection-method, only the most similar images of both representations still belong to the cluster.

Experiments with TV-Images

For this experiments, our image database consisted of 1,000 color TV-images which were segmented and transformed into trees and graphs in the way described in the Sections 7.2.1 and 7.2.2. Again we used the intersection method to combine the two representations and compared the result to the results we got when clustering each representation separately.

The results we obtained when clustering the data using the graph or the tree model were quite different. With the graph model, we obtained several rather homogeneous clusters like the one depicted in Figure 8.7 but also very diverse clusters like the one shown in Figure 8.8. In general, it was possible to distinguish hockey images from the rest of the database rather well.

On the other hand, the use of the tree model only yielded one large and unspecific cluster and much noise. Obviously, this model alone is ill-suited for our image database.

But although the second model on its own did not yield any interesting results, the combination of both approaches turned out to be effective. Figures 8.9 and 8.10 show typical clusters obtained with the combination of the two models. As can be seen in Figure 8.9, the combination yielded more homogeneous clusters, as for example a cluster of insect images. Those images belonged to a big and diverse cluster for the graph model. Additionally, the

Figure 8.7: A typical homogenous cluster obtained with the graph model.

Figure 8.8: A typical diverse cluster obtained with the graph model.

Figure 8.9: A cluster of insects which could only be obtained with the combined model.

Figure 8.10: A cluster obtained with the combined model.

distinguishing power for the hockey images was preserved as shown in Figure 8.10. In general, the clusters we obtained when combining both representations were more accurate than the clusters we got using each representation separately. Obviously, the noise ratio increased when we combined the two representations.

8.5 Summary

In this chapter we discussed the problem of clustering multi-represented objects. A multi-represented object is described by a set of representations where each representation belongs to a different data space. Contrary to existing approaches, our proposed method is able to cluster this kind of data using all available representations without forcing the user to construct a combined data space. The idea of our approach is to combine the information of all different representations as early as possible and as late as necessary. To do so, we adapted the core object property proposed for DBSCAN. To decide whether an object is a core object, we use the local ε-neighborhoods of each representation and combine the results to a global neighborhood. Based on this idea, we proposed two different methods for varying applications. For sparse data, we introduced the union-method that assumes that an object is a core object if k objects are found within the union of its local ε-neighborhoods. Respectively, we defined the intersection-method for data where each local representation yields rather big and unspecific clusters. Therefore, the intersection-method requires at least k objects within the intersection of all local ε-neighborhoods of a core object. In

our experimental evaluation we introduced an entropy based quality measure that compares a given clustering with noise to a reference clustering. Employing this quality measure, we demonstrated that the union method was most suitable to overcome the sparsity of a given protein data set. To demonstrate the ability of the intersection method to increase the cluster quality, we applied it to a set of images using different similarity models.

Chapter 9

Efficient Filters for Tree Structured Data

In Chapter 7, we have seen that a successful approach to model hierarchically structured objects is using a tree representation. However, the problem of all similarity measures for attributed trees is their computational complexity and thus they are not applicable for clustering large collections of tree-structured objects. In this chapter, we propose a filter and refinement architecture to overcome this problem. We present a set of new filter methods for structural and for content-based information in tree-structured data as well as ways to flexibly combine different filter criteria. The efficiency of our methods, resulting from the good selectivity of the filters is demonstrated in extensive experiments with two real-world applications. The concepts described in this chapter have been published in [KKSS04b, KKSS04a].

9.1 Introduction

In addition to a variety of content-based attributes, complex objects typically carry some kind of internal structure which often forms a hierarchy. Several similarity measures for trees have been proposed in the literature [JWZ94, Sel77, Zha96]. These measures are well suited for hierarchical objects and have been successfully applied to website analysis [WZCS02], structural similarity of XML documents [NJ02], shape recognition [SKK01] and chemical substructure search [WZCS02], for instance. However, a general problem of all those measures is their computational complexity which makes them unsuitable for large databases. The core idea of our approach is to apply a filter criterion to the database objects in order to obtain a small set of candidate answers to a query. Then the final result is retrieved from this candidate set through the use of the original complex similarity measure. This filter-refinement architecture reduces the number of expensive similarity distance calculations and speeds up the search process. To extend this concept to the new problem of searching similar tree structures, efficient and effective filters for structural properties are required. In this chapter, we propose several new filter methods for tree structures and also demonstrate how to combine them with filters for content information in order to obtain a high filter selectivity.

In the next section, we discuss several measures for structural similarity. In Section 9.3, the concept of multi-step query processing is presented, while Section 9.4 deals with our filter methods. Finally, we present an experimental evaluation of our filters in Section 9.5 before we conclude the chapter.

9.2 Structural Similarity

Quantifying the similarity of two trees requires a structural similarity measure. There are several similarity measures for general graphs in the literature [BS98, CKS98, KKV90]. All of them either suffer from a high computational complexity or are limited to special graph types. Papadopoulos and Manolopoulos presented a measure based on certain edit operations for

general graphs [PM99]. They use the degree sequence of a graph as feature vector and the Manhattan distance between the feature vectors as similarity measure. While their measure can be calculated efficiently, it is not applicable to attributed graphs. Consequently, special distance measures for labeled trees which exploit the structure and content of trees become necessary. Jiang, Wang and Zhang [JWZ94] suggested a measure based on a structural alignment of trees. They also prove that the structural alignment problem for trees is NP-hard if the degree of the trees is not bounded. Selkow [Sel77] presented a tree-to-tree editing algorithm for ordered labeled trees. It is a first step towards the most common approach to measure tree similarity, the edit distance. The edit distance, well-known from string matching [Lev66, WF74], is the minimal number of edit operations necessary to transform one tree into the other. There are many variants of the edit distance, depending on which edit operations are allowed. The basic form allows two edit operations, i.e. the insertion and the deletion of a tree node. The insertion of a node n in a tree below a node p means that p becomes the parent of n and a subset of p's children become n's children. The deletion of a node is the inverse operation to the insertion of the node. In the case of attributed nodes, as they appear in most real-world applications, the change of a node label is introduced as a third basic operation.

Definition 9.1 (edit sequence, 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 of a sequence of edit operations $S = \langle e_1, \ldots, e_m \rangle, c(S)$ is defined as the sum of the cost of each edit operation in S, i.e. $c(S) = c(e_1) + \ldots + c(e_m)$.

Definition 9.2 (edit distance)
The edit distance between two trees t_1 and t_2, $ED(t_1, t_2)$ is the minimum cost of all edit sequences that transform t_1 into t_2: $ED(t_1, t_2) = min\{c(S)|S$ a sequence of edit operations transforming t_1 into $t_2\}$.

A great advantage of using the edit distance as a similarity measure is that along with the distance value a mapping between the nodes in the two trees is provided in terms of the edit sequence. The mapping can be visualized and can serve as an explanation of the similarity distance to the

user. This is especially important in the context of similarity search, as different users often have a different notion of similarity in mind. Here, an explanation component can help the user to adapt weights for the distance measure in order to reflect the individual notion of similarity. However, Zhang, Statman and Shasha showed that computing the edit distance between unordered labeled trees is NP-complete [ZSS92]. Obviously, such a complex similarity measure is unsuitable for large databases. To overcome this problem, Zhang proposed a constrained edit distance between trees, the degree-2 edit distance. The main idea behind this distance measure is that only insertions or deletions of nodes with a maximum number of two neighbors are allowed.

Definition 9.3 (degree-2 edit distance)
The edit distance between two trees t_1 and t_2, $ED_2(t_1, t_2)$, is the minimum cost of all degree-2 edit sequences that transform t_1 into t_2 or vice versa. A degree-2 edit sequence consists only of insertions or deletions of nodes n with $degree(n) \leq 2$ or of relabelings: $ED_2(t_1, t_2) = min\{c(S)|S$ is a degree-2 edit sequence $transforming$ t_1 into $t_2\}$.

One should note that the degree-2 edit distance is well defined in the sense that two trees can always be transformed into each other using only degree-2 edit operations. This statement is true because it is possible to build 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_1 completely and then build t_2 from scratch, resulting in a distance value for this pair of trees. In [ZWS96] an algorithm is presented to compute the degree-2 edit distance in $O(|t_1||t_2|D)$ time, where D is the maximum of the degrees of t_1 and t_2 and $|t_i|$ denotes the number of nodes in t_i. Whereas this measure has a polynomial time complexity, it is still too complex for the use in large databases. To overcome this problem, we extend the paradigm of filter-refinement architectures to the context of structural similarity search.

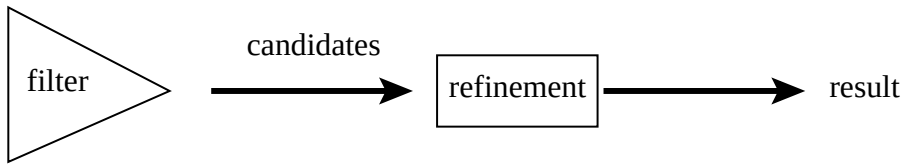


Figure 9.1: The filter-refinement architecture.

9.3 Multi-Step Query Processing

The main goal of a filter-refinement architecture, as depicted in Figure 9.1, is to reduce the number of complex and time consuming distance calculations in the query process. To achieve this goal, query processing 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 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 of all, it is essential that the filter predicate is considerably easier to determine 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 an important property. 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 there occur no false drops during the filter step. Available similarity search algorithms guarantee completeness if the distance function in the filter step fulfills the following lower-bounding property. For any two objects p and q a lower-bounding distance function d_{lb} in the filter step has to return a value that is not greater than the object exact distance d_e of p and q, i.e. $d_{lb}(p,q) \leq d_e(p,q)$. With a lower-bounding distance function it is possible to safely filter out all database objects which have a filter distance greater 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 which actually make use of the filter step. Agrawal, Faloutsos and Swami proposed such an algorithm for range search [AFS93]. In [SK98] a multi-step algorithm for k-nearest-neighbor search is presented which is optimal in the sense that the minimal number of exact distance calculations are performed during query processing.

9.4 Structural and Content-Based Filters for Unordered Trees

In this section, we introduce several filtering techniques that support efficient similarity search for tree-structured data. We learned from preliminary experiments that single-valued features including the height of a tree, the number of nodes, or the degree of a tree, are of limited use. Therefore we propose the use of feature histograms in order to represent the structural information of trees. The advantage of this extension is that there is more information provided to the filter step for the purpose of generating candidates and, thus, the discriminative power is increased. Additionally, a variety of multidimensional index structures and efficient search algorithms are available for vector data including histograms. The particular feature histograms which we propose in the following are based on the height, the degree or the label of individual nodes.

9.4.1 Filtering Based on the Height of Nodes

A promising way to filter unordered trees based on their structure is to take the height of nodes into account. A very simple technique is to use the height of a tree as a single feature. The difference of the height of two trees is an obvious lower bound for the edit distance between those trees, but this filter clearly is very coarse, as two trees with completely different structure but the same height cannot be distinguished by this filter.

A more fine-grained and more sensitive filter can be obtained by creating a histogram of node heights in a tree and using the difference between those histograms as a filter distance. A first approach is to determine the distance

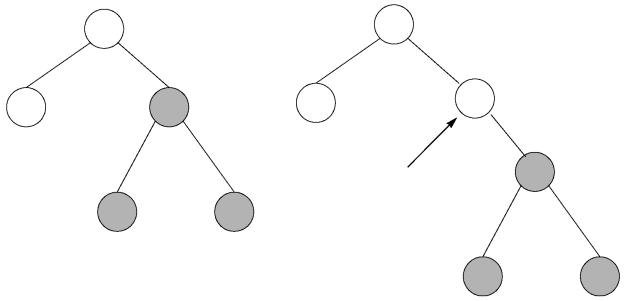


Figure 9.2: A single insertion can change the distance to the root for several nodes.

of each node in the tree to the root node and then store the distribution of those values in a histogram. Unfortunately, the distance between two such histograms is not guaranteed to be a lower bound for the edit distance or the degree-2 edit distance between the original trees. As can be seen in Figure 9.2, the insertion of a single node may change the height of all nodes in its subtree. Thus, the number of affected histogram bins is only bounded by the height of the tree.

Therefore, we propose a different approach to consider the height of a node. Instead of the distance of a node from the root, its leaf distance is used to approximate the structure of a tree.

Definition 9.4 (leaf distance)
The leaf distance $d_l(n)$ of a node n is the maximum length of a path from n to any leaf node in the subtree rooted at n.

Based on this definition, we introduce the leaf distance histogram of a tree as illustrated in Figure 9.3.

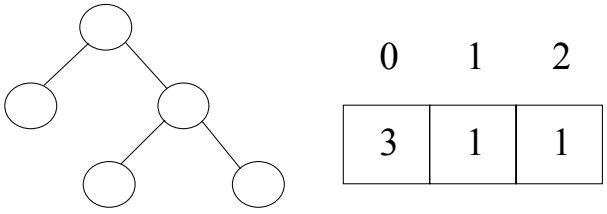


Figure 9.3: Leaf distance of nodes and leaf distance histogram.

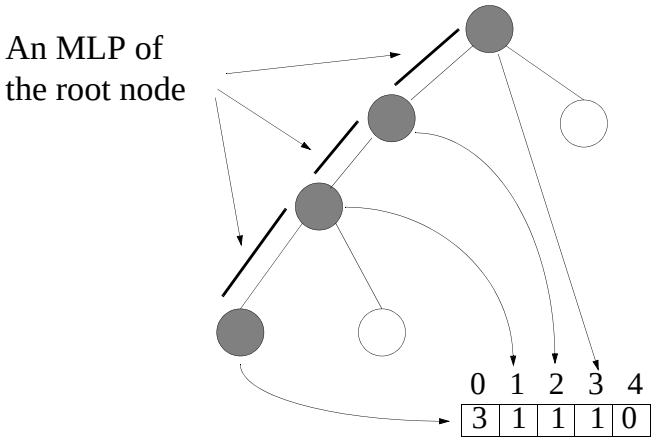


Figure 9.4: A maximum leaf path.

Definition 9.5 (leaf distance histogram)
The leaf distance histogram $h_l(t)$ of a tree t is a vector of length $k = 1 + height(t)$ where the value of any bin $i \in 0, \ldots, k$ is the number of nodes that share the leaf distance i, i.e. $h_l(t)[i] = |n \in t, d_l(n) = i|$.

For the proof of the following theorem, the definition of a maximum leaf path is useful:

Definition 9.6 (maximum leaf path)
A maximum leaf path (MLP) of a node n in a tree t is a path of maximum length from n to a leaf node in the subtree rooted by n.

An important observation is that adjacent nodes on an MLP are mapped to adjacent bins in the leaf distance histogram as illustrated in Figure 9.4.

Theorem 9.1 *For any two trees t_1 and t_2, the L_1-distance of the leaf distance histograms is a lower bound of the edit distance of t_1 and t_2:*

$$L_1(h_l(t_1), h_l(t_2)) \leq ED(t_1, t_2).$$

Proof. *Given two arbitrary trees t_0 and t_m, let us consider an edit sequence $S = \langle S_1, \ldots, S_m \rangle$ that transforms t_0 to t_m. We proceed by induction over the length $m = |S|$. If $m = 0$, i.e. $S = \langle\rangle$ and $t_0 = t_m$, the values of $L_1(h_l(t_0), h_l(t_m))$ and of $c(S)$ both are equal to zero. For $m > 0$, let us*

assume that the lower-bounding property already holds for the trees t_0 *and* t_{m-1}*, i.e.* $L_1(h_l(t_0), h_l(t_{m-1})) \leq c(\langle S1, \ldots, S_{m-1}\rangle)$*. When extending the sequence* $\langle S_1, \ldots, S_{m-1}\rangle$ *by* S_m *to* S*, the right hand side of the inequality is increased by* $c(S_m) = 1$*.*

The situation on the left hand side is as follows. The edit step S_m *may be a relabeling, an insertion or a deletion. Obviously, the effect on the leaf distance histogram* $h_l(t_{m-1})$ *is void in case of a relabeling, i.e.* $h_l(t_m) = h_l(t_{m-1})$*, and the inequality* $L_1(h_l(t_0), h_l(t_m)) = L_1(h_l(t_0), h_l(t_{m-1})) \leq c(S)$ *holds.*

The key observation for an insert or a delete operation is that only a single bin is affected in the histogram in any case. When a node ν *is inserted, it is clear that for all nodes below the insertion point the leaf distance does not change. Only the leaf distance of any predecessor of the inserted node may or may not be increased by the insertion. Therefore, if* ν *does not belong to an MLP of any of its predecessors, only the bin affected by the inserted node is increased by one. This means that in the leaf distance histogram exactly one bin is increased by one. On the other hand, if an MLP of any of the predecessors of* ν *containing* ν *exists, then we only have to consider the longest of those MLPs. Due to the insertion, this MLP grows in size by one. As all nodes along the MLP are mapped into consecutive histogram bins, exactly one more bin than before is influenced by the nodes on the MLP. This means that exactly one bin in the leaf distance histogram changes due to the insertion. As insertion and deletion are symmetric operations, the same considerations hold for the deletion of a node.*

The preceding considerations hold for all edit sequences transforming a tree t_1 *into a tree* t_2 *and particularly include the minimum cost edit sequence. Therefore, the lower-bounding relationship immediately holds for the edit distance* $ED(t_1, t_2)$ *of two trees* t_1 *and* t_2*, too.* ◇

It should be noticed that the above considerations do not only hold for the edit distance but also for the degree-2 edit distance. Therefore, the following theorem allows us also to use leaf-distance histograms for the degree-2 edit distance.

Theorem 9.2 *For any two trees t_1 and t_2, the L_1-distance of the leaf distance histograms is a lower bound of the degree-2 edit distance of t_1 and t_2:*

$$L_1(h_l(t_1), h_l(t_2)) \leq ED_2(t_1, t_2)$$

Proof. *The proof can be done analogously to the proof of theorem 9.1. Given two arbitrary trees t_0 and t_m, let us consider an edit sequence $S = \langle S_1, \ldots, S_m \rangle$ that transforms t_0 to t_m. We proceed by induction over the length $m = |S|$. If $m = 0$, i.e. $S = \langle\rangle$ and $t_0 = t_m$, the values of $L_1(h_l(t_0), h_l(t_m))$ and of $c(S)$ both are equal to zero. For $m > 0$, let us assume that the lower-bounding property already holds for the trees t_0 and t_{m-1}, i.e. $L_1(h_l(t_0), h_l(t_{m-1}))$*
$\leq c(\langle S1, \ldots, S_{m-1} \rangle)$. When extending the sequence $\langle S_1, \ldots, S_{m-1} \rangle$ by S_m to S, the right hand side of the inequality is increased by $c(S_m) \geq 1$. If an inner node is inserted or deleted, the costs $c(S_m)$ are 1 for inserting the node plus the cost for the necessary reconstruction of the tree, because only degree-2 edit operations are allowed. In all other cases $c(S_m) = 1$.

The situation on the left hand side of the equation is equal to the proof of theorem 9.1. ◇

Theorem 9.1 and 9.2 also allow us to use leaf distance histograms as a filter for the weighted edit and weighted degree-2 edit distance. This statement is justified by the following considerations. As shown above, the L_1-distance of two leaf distance histograms gives a lower bound for the insert and delete operations that are necessary to transform the two corresponding trees into each other. This fact also holds for weighted relabeling operations, as weights do not have any influence on the necessary structural modifications. But even when insert/delete operations are weighted, our filter can be used as long as there is a smallest possible weight w_{min} for an insert or delete operation. In this case, the term $(L_1(h_l(t_1), h_l(t_2))\Delta w_{min})$ is a lower bound for the weighted edit and degree-2 edit distance between the trees t_1 and t_2. Since we assume metric properties as well as the symmetry of insertions and deletions for the distance, the triangle inequality guarantees the existence of such a minimum weight. Otherwise, any relabeling of a node would be performed cheaper by a deletion and a corresponding insertion operation.

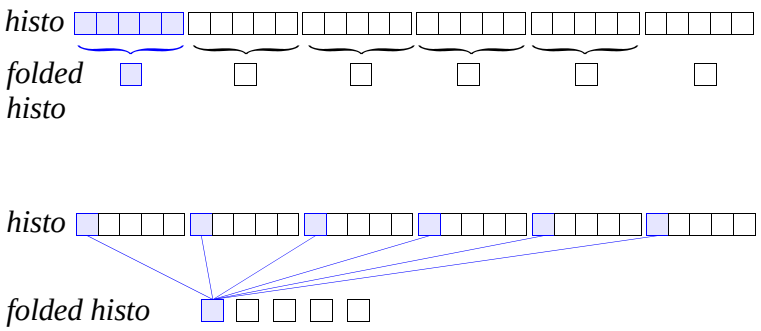


Figure 9.5: Folding techniques for histograms: The technique of Papadopoulos and Manolopoulos (top) and the modulo folding technique (bottom).

Moreover, structural differences of objects would be reflected only weakly if structural changes are not weighted properly.

Histogram Folding. Another property of leaf distance histograms is that their size is unbounded as long as the height of the trees in the database is also unbounded. This problem arises for several feature vector types, including the degree histograms presented in Section 9.4.2. Papadopoulos and Manolopoulos [PM99] address this problem by folding the histograms into vectors with fixed dimension. This is done in a piecewise grouping process. For example, when a 5-dimensional feature vector is desired, the first fifths of the histogram bins is summed up and the result is used as the first component of the feature vector. This is done analogously for the rest of the histogram bins. The above approach could also be used for leaf distance histograms, but it has the disadvantage that the maximal height of all trees in the database has to be known in advance. For dynamic data sets, this precondition cannot be fulfilled. Therefore, we propose a different technique that yields fixed-size n-dimensional histograms by adding up the values of certain entries in the leaf distance histogram. Instead of summing up adjacent bins in the histogram, we add up those with the same index modulo n, as depicted in Figure 9.5. This way, histograms of distinct length can be compared, and there is no bound for the length of the original histograms.

Definition 9.7 (folded histogram)
A folded histogram $h_{fn}(h)$ of a histogram h for a given parameter n is a vector of size n where the value of any bin $i \in 0, \ldots, n-1$ is the sum of all bins k in h with $k \bmod n = i$, i.e.

$$h_{fn}(h)[i] = \sum_{k=0\ldots(|h|-1) \wedge k \bmod n = i} h[k].$$

The following theorem justifies to use folded histograms in a multi-step query processing architecture.

Theorem 9.3 *For any two histograms h_1 and h_2 and any parameter $n \geq 1$, the L_1-distance of the folded histograms of h_1 and h_2 is a lower bound for the L_1-distance of h_1 and h_2:*

$$L_1(h_{fn}(h_1), h_{fn}(h_2)) \leq L_1(h_1, h_2).$$

Proof. *Let $len = n \cdot \lceil \frac{max(h_1,h_2)}{n} \rceil$ be the length of h_1 and h_2. If necessary, h_1 and h_2 are extended with bins containing 0 until $|h_1| = len$ and $|h_2| = len$. Then the following holds:*

$$\begin{aligned}
&L_1(h_{fn}(h_1), h_{fn}(h_2)) \\
&= \sum_{i=0}^{n-1} \left| \sum_{k=0\ldots((|h_1|-1) \atop \wedge k \ MOD \ n=i} h_1[k] - \sum_{k=0\ldots((|h_2|-1) \atop \wedge k \ MOD \ n=i} h_2[k] \right| \\
&= \sum_{i=0}^{n-1} \left| \sum_{j=0}^{(len \ DIV \ n)-1} h_1[i + j \cdot n] - \sum_{j=0}^{(len \ DIV \ n)-1} h_2[i + j \cdot n] \right| \\
&\leq \sum_{i=0}^{n-1} \sum_{j=0}^{(len \ DIV \ n)-1} |h_1[i + j \cdot n] - h_2[i + j \cdot n]| \\
&= \sum_{j=0}^{len} |h_1[k] - h_2[k] \\
&= L_1(h_1, h_2)
\end{aligned}$$

◇

9.4.2 Filtering Based on the Degree of Nodes

The degrees of the nodes are another structural property of trees which can be used as a filter for the edit distances. Again, a simple filter can be obtained by using the maximal degree of all nodes in a tree t, denoted by $degree_{max}(t)$, as a single feature. The difference between the maximal degrees of two trees is an obvious lower bound for the edit distance as well as for the degree-2 edit distance. As before, this single-valued filter is very coarse and using a degree histogram clearly increases the selectivity.

Definition 9.8 (degree histogram)
The degree histogram $h_d(t)$ of a tree t is a vector of length $k = 1+degree_{max}(t)$ where the value of any bin $i \in 0, \ldots, k$ is the number of nodes that share the degree i, i.e. $h_d(t)[i] = |n \in t, degree(n) = i|$.

Theorem 9.4 *For any two trees t_1 and t_2, the L_1-distance of the degree histograms divided by three is a lower bound of the edit distance of t_1 and t_2:*

$$\frac{L_1(h_d(t_1), h_d(t_2))}{3} \leq ED(t_1, t_2).$$

Proof. *Given two arbitrary trees t_0 and t_m, let us consider an edit sequence $S = \langle S1, \ldots, S_m \rangle$ that transforms t_0 into t_m. We proceed by induction over the length of the sequence $m = |S|$. If $m = 0$, i.e. $S = \langle\rangle$ and $t_0 = t_m$, the values of $\frac{L_1(h_d(t_0), h_d(t_m))}{3}$ and of $c(S)$ both are equal to zero. For $m > 0$, let us assume that the lower-bounding property already holds for t_0 and t_{m-1}, i.e. $\frac{L_1(h_d(t_0), h_d(t_{m-1}))}{3} \leq c(\langle S1, \ldots, S_{m-1} \rangle)$. When extending the sequence $\langle S_1, \ldots, S_{m-1} \rangle$ by S_m to S, the right hand side of the inequality is increased by $c(S_m) = 1$. The situation on the left hand side is as follows. The edit step S_m may be a relabeling, an insert or a delete operation. Obviously, for a relabeling, the degree histogram $h_d(t_{m-1})$ does not change, i.e. $h_d(t_m) = h_d(t_{m-1})$ and the inequality $\frac{L_1(h_d(t_0), h_d(t_m))}{3} = \frac{L_1(h_d(t_0), h_d(t_{m-1}))}{3} \leq c(S)$ holds.*

The insertion of a single node affects the histogram and the L_1-distance of the histograms in the following way:

1. *The inserted node n causes an increase in the bin of n's degree. That may change the L_1-distance by at most one.*

2. *The degree of n's parent node p may change. In the worst case this affects two bins. The bin of p's former degree is decreased by one while the bin of its new degree is increased by one. Therefore, the L_1-distance may additionally be changed by no more than two.*

3. *No other nodes are affected.*

From the above three points it follows that the L_1-distance of the two histograms $h_d(t_{m-1})$ and $h_d(t_m)$ changes by at most three. Therefore, the following holds:

$$\begin{aligned}
\frac{L_1(h_d(t_0), h_d(t_m))}{3} &\leq \frac{L_1(h_d(t_0), h_d(t_{m-1})) + 3}{3} \\
\frac{L_1(h_d(t_0), h_d(t_m))}{3} &\leq \frac{L_1(h_d(t_0), h_d(t_{m-1})}{3} + 1 \\
\frac{L_1(h_d(t_0), h_d(t_m))}{3} &\leq c(\langle S1, \ldots, S_{m-1}\rangle) + 1 \\
\frac{L_1(h_d(t_0), h_d(t_m))}{3} &\leq c(\langle S1, \ldots, S_{m-1}, S_m\rangle) \\
\frac{L_1(h_d(t_1), h_d(t_2))}{3} &\leq ED(t_1, t_2)
\end{aligned}$$

◇

As the above considerations also hold for the degree-2 edit distance, theorem 9.4 holds analogously for this similarity measure.

9.4.3 Filtering Based on Node Labels

Apart from the structure of the trees, the content features, expressed through node labels, have an impact on the similarity of attributed trees. The node labels can be used to define a filter function. To be useful in our filter-refinement architecture, this filter method has to deliver a lower bound for the edit cost when transforming one tree into the other. The difference between the distribution of the values within a tree and the distribution of the values in another tree can be used to develop a lower-bounding filter. To ensure efficient evaluation of the filter, the distribution of those values has to be approximated for the filter step.

One way to approximate the distribution of values is to use histograms. In this case, an n-dimensional histogram is derived by dividing the range

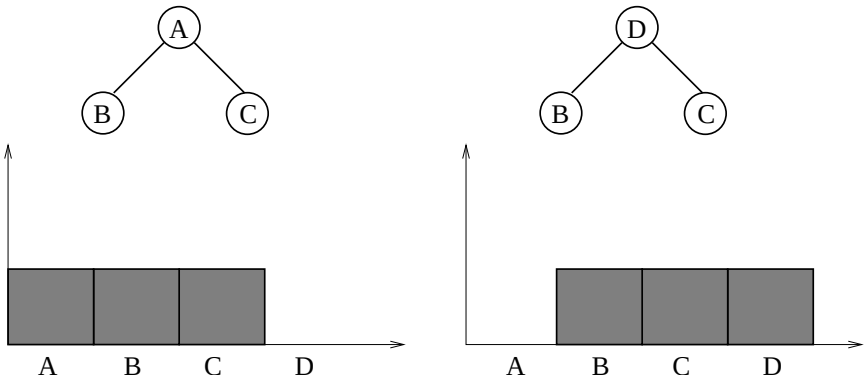


Figure 9.6: A single relabeling operation may result in a label histogram distance of two.

of the node label into n bins. Then, each bin is assigned the number of nodes in the tree whose value is in the range of the bin. To estimate the edit distance or the degree-2 edit distance between two trees, half of the L_1-distance of their corresponding label histograms is appropriate. A single insert or delete operation changes exactly one bin of such a label histogram. A single relabeling operation can influence at most two histogram bins. If a node is assigned to a new bin after relabeling, the entry in the old bin is decreased by one and the entry in the new bin is increased by one (cf. Figure 9.6). Otherwise, a relabeling does not change the histogram. This method also works for weighted variants of the edit distance and the degree-2 edit distance as long as there is a minimal weight for a relabeling operation. In this case, the calculated filter value has to be multiplied by this minimal weight in order to gain a lower-bounding filter.

This histogram approach applies to discrete label distributions very well. However, for continuous label spaces, the use of a continuous weight function which may become arbitrarily small, can be reasonable. In this case, a discrete histogram approach can not be used. An example for such a weight function is the Euclidean distance in the color space, assuming trees where the node labels are colors. Here, the cost for changing a color value is proportional to the Euclidean distance between the original and the target color. As this distance can be infinitely small, it is impossible to estimate the relabeling cost based on a label histogram as in the above cases.

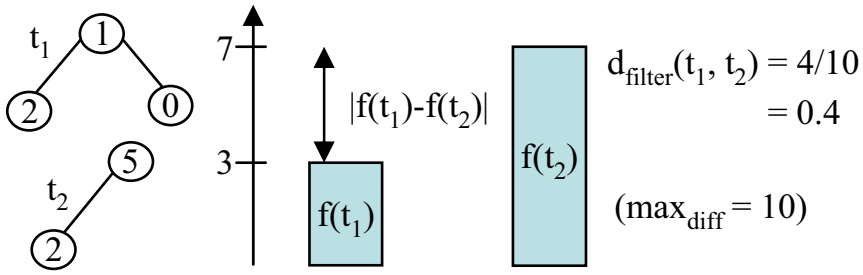


Figure 9.7: Filtering for continuous weight functions.

More formally, when using the term 'continuous weight function', we mean that the cost for changing a node label from value x_1 to value x_2 is proportional to $|x_1 - x_2|$. Let max_{diff} be the maximal possible difference between two attribute values. Then $|x_1 - x_2|$ has to be normalized to $[0, 1]$ by dividing it through max_{diff}, assuming that the maximal cost for a single insertion, deletion or relabeling is one. To develop a filter method for attributes with such a weight function, we exploit the following property of the edit distance measure. The cost-minimal edit sequence between two trees removes the difference between the distributions of attribute values of those two trees. It does not matter whether this is achieved through relabelings, insertions or deletions.

For our filter function we define the following feature value $f(t)$ for a tree t:

$$f(t) = \sum_{i=1}^{|t|} |x_i|$$

Here x_i is the attribute value of the i-th node in t and $|t|$ is the size of tree t. The absolute difference between two such feature values is an obvious lower bound for the difference between the distribution of attribute values of the corresponding trees. Consequently, we use

$$d_{filter}(t_1, t_2) = \frac{|f(t_1) - f(t_2)|}{max_{diff}}.$$

as a filter function for continuous label spaces, see Figure 9.7 for an illustration. Once more, the above considerations also hold for the degree-2 edit distance.

To simplify the presentation, we assumed that a node label consists of just one single attribute. But usually a node will carry several different attributes. If possible, the attribute with the highest selectivity can be chosen for filtering. In practice, there is often no such single attribute. In this case, filters for different attributes can be combined with the technique described in the following section.

9.4.4 Combining Filter Methods

All of the above filters use a single feature of an attributed tree to approximate the edit distance or degree-2 edit distance. As the filters are not equally selective in each situation, we propose a method to combine several of the presented filters.

A first idea to combine several filters is to create a multidimensional histogram for the cross-product of the value range of the filters. This yields a cross-product histogram whose bins contain the number of nodes in a tree with a certain feature combination. However, this approach fails because the edit distance between two trees cannot be estimated from the differences of two such histograms. The reason for this observation is that unlike in the one-dimensional case, an indeterminable number of entries in the histogram may change upon a single edit operation. For example, consider a combination of a height and a degree histogram. A single insertion may change the leaf distance of all predecessors of the inserted node. The number depends on the insertion point and cannot be determined in advance. Additionally, the predecessors may have different degrees and therefore the affected histogram bins can be distributed over the entire histogram. Consequently, the number of affected bins cannot be estimated. Therefore, it is impossible to derive a lower bound for the edit distance between two trees from the distance for their respective cross-product histograms.

Hence, we follow the different approach of combining the results of the existing methods which also allows us to integrate our filter for continuous weight functions. A very flexible way of combining different filters is to follow the inverted list approach, i.e. to apply the different filters independently from each other and then intersect the resulting candidate sets. With this

approach, separate index structures for the different filters have to be maintained, and for each query a time-consuming intersection step is necessary. To avoid those disadvantages, we concatenate the different filter histograms and filter values for each object and use a combined distance function as a similarity function.

Definition 9.9 (combined distance function)
Let $C = d_i$ be a set of distance functions for trees. Then, the combined distance function d_c is defined to be the maximum of the component functions:

$$d_C(t_1, t_2) = max\{d_i(t_1, t_2)\}.$$

Theorem 9.5 *For every set of lower-bounding distance functions $C = \{d_{low}(t_1, t_2)\}$, i.e. for all trees t_1 and t_2 $d_i(t_1, t_2) \leq ED(t_1, t_2)$, the combined distance function d_C is a lower bound of the edit distance function d_{ED}:*

$$d_C(t_1, t_2) \leq ED(t_1, t_2).$$

Proof. *For all trees t_1 and t_2, the following equivalences hold:*

$$\begin{aligned} d_C(t_1, t_2) &\leq ED(t_1, t_2) \Leftrightarrow \\ max\{d_i(t_1, t_2)\} &\leq ED(t_1, t_2) \Leftrightarrow \\ \forall d_i : d_i(t_1, t_2) &\leq ED(t_1, t_2) \end{aligned}$$

The final inequality represents the precondition. ◇

Justified by theorem 9.5, we apply each separate filter function to its corresponding component of the combined histogram. The combined distance function is derived from the results of this step.

Again, the above considerations also hold for the degree-2 edit distance. Therefore, theorem 9.5 allows us to use the combined histogram distance function with the degree-2 edit distance as similarity measure, too.

9.5 Experimental Evaluation

For our tests, we implemented a filter-refinement architecture according to the optimal multi-step k-nearest-neighbor search approach as proposed in [SK98]. We used k-nearest neighbor queries instead of directly showing results for range queries. Since for k-nearest neighbor queries the result set size is known beforehand, the interpretation of the experimental results is easier. Naturally, the positive effects, which we show in the following experiments for k-nearest neighbor queries, also hold for range queries and for all data mining algorithms based on range queries or k-nearest neighbor queries, e.g. density-based clustering, k-nearest neighbor classification. As similarity measure for trees, we implemented the degree-2 edit distance algorithm as presented in [ZWS96]. The filter histograms were organized in an X-tree [BKK96] or an M-tree [CPZ97] in case of combined histograms. All algorithms were implemented in Java 1.4 and the experiments were run on a workstation with a Xeon 1,7 GHz processor and 2 GB main memory under Linux.

To show the efficiency of our approach, we chose two different applications: an image database and a database of websites which are described in the following.

9.5.1 Image Databases

The images we used for our experiments were taken from three real-world databases: a set of 705 black and white pictographs, a set of 8,536 commercially available color images and a set of 43,000 color TV-Images. We extracted trees from those images in a two-step process. First, the images were divided into segments of similar color by a segmentation algorithm. In the second step, a tree was created from those segments by iteratively applying a region-growing algorithm which merges neighboring segments if their colors are similar. This is done until all segments are merged to a single node. As a result, we obtain a set of labeled unordered trees where each node label describes the color, size and horizontal as well as vertical extension of the associated segment. Table 9.1 shows some statistical information about the trees we generated.

	number	number of nodes			height			maximal degree		
	of images	max	min	Ø	max	min	Ø	max	min	Ø
commercial images	8,536	331	1	30	24	0	3	206	0	18
TV-images	43,000	109	1	24	13	0	3	71	0	11
pictographs	705	113	3	13	2	1	1	112	2	12

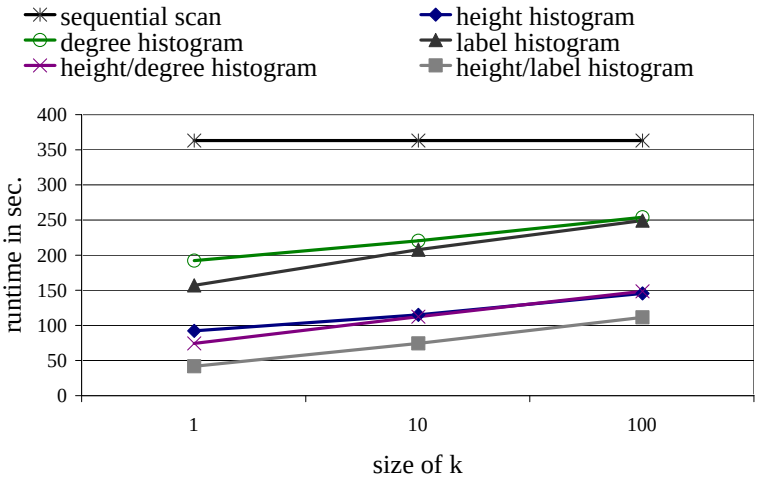
Table 9.1: Statistics of the image data set.

For the first experiments, we used label histograms as described in Section 9.4.3. To derive a discrete label distribution, we reduced the number of different attribute values to 16 different color values for each color channel and 4 different values each for size and extensions. We used a relabeling function with a minimal weight of 0.5. Later on we also show some experiments where we did not reduce the different attribute values and used a continuous weight function for relabeling.

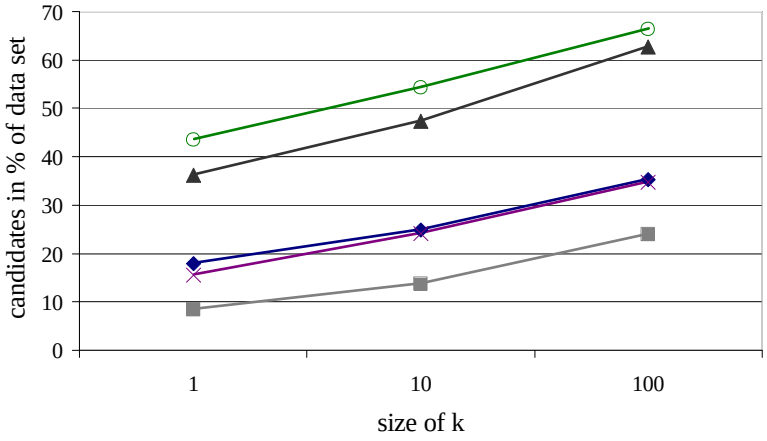
Comparison of our Filter Types.

For our first experiment we used 10,000 TV-images. We created a 10-dimensional height and a 10-dimensional degree histogram and combined them to a 20-dimensional height/degree histogram as described in Section 9.4.4. We also built a 24-dimensional combined label histogram which considered the color, size and extensions of all node labels (6 attributes with histograms of size 4). Finally, the 34-dimensional combination of this combined label histogram and the 10-dimensional height histogram was taken as another filter criterion.

We ran 100 k-nearest-neighbor queries (k = 1, 10, 20, 50) for each of our filters. Figure 9.8 shows the selectivity of our filters, measured in the average number of candidates with respect to the size of the data set. The figures show that filtering based solely on structural (height or degree histogram) or content-based features (label histogram) is not as effective as their combination. Figure 9.8 also shows that for this data the degree filter is less selective than the height filter. The method which combines the filtering based on the height of the nodes and on content features is most effective. Figure 9.8 additionally depicts the average runtime of our filters compared



(a)



(b)

Figure 9.8: Runtime (a) and number of candidates (b) for k-nn queries on 10,000 color TV-images.

to the sequential scan. As one can see, we reduced the runtime by a factor of up to 5. Furthermore, the comparison of the two diagrams in Figure 9.8 shows that the runtime is dominated by the number of candidates, whereas the additional overhead, due to the filtering, is negligible.

Influence of Histogram Size.

In a next step we tested to what extent the size of the histogram influences the size of the candidate set and the corresponding runtime. The results for nearest neighbor queries on 10,000 color TV-images are shown in Figure 9.9. With increasing dimension, the number of candidates as well as the runtime decrease. The comparison of the two diagrams in Figure 9.9 shows that the runtime is again dominated by the number of candidates, while the additional overhead, due to higher dimensional histograms, is negligible.

Scalability of Filters versus Size of Data Set.

For this experiment, we united all three image data sets and chose three subsets of size 10,000, 25,000 and 50,000. On these subsets we performed several representative 5-nn queries. Figure 9.10 shows that the selectivity of our structural filters does not depend on the size of the data set.

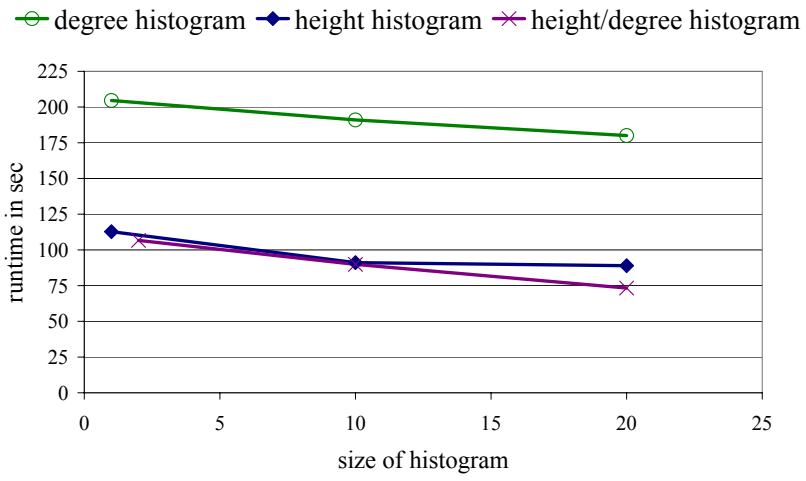
Runtimes for the Creation of the Filters

For each filter criterion we created an X-tree or an M-tree storing the filter histograms. Figure 9.11 shows the runtimes for the creation of these trees for 10,000 color images. Even for the most complex filter criterion the creation time is rather moderate.

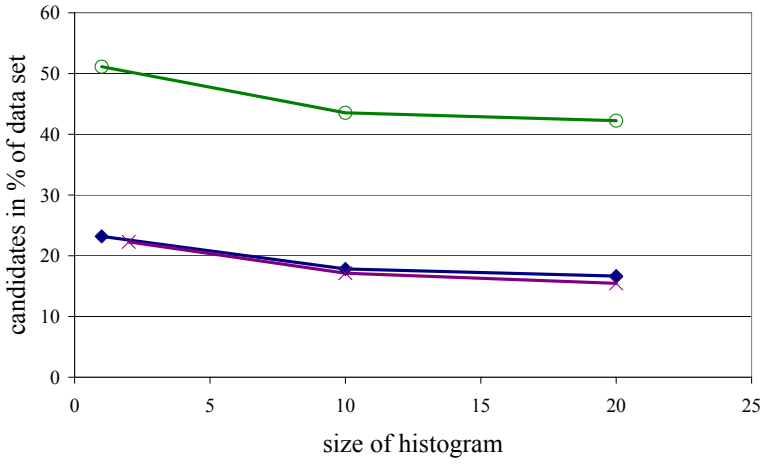
The creation also scales well with an increasing number of images. For example, the creation of an M-tree for 28-dimensional combined height and label histograms of 50,000 images took 733 seconds.

Comparison of Different Filters for a Continuous Weight Function.

As mentioned above, we also tested our filters when using a continuous weight function for relabeling. For this experiment, we used again 10,000 TV-images. Figure 9.12 shows the results averaged over 200 k-nn queries.



(a)



(b)

Figure 9.9: Influence of dimensionality of histograms.

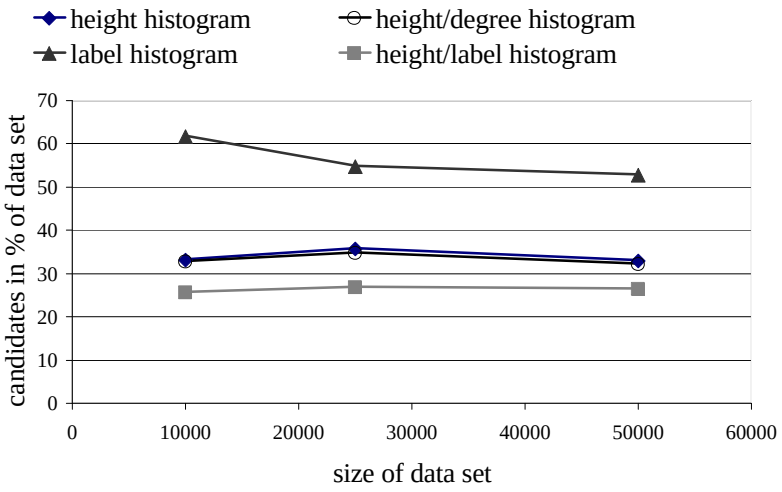


Figure 9.10: Scalability w.r.t. the size of the data set.

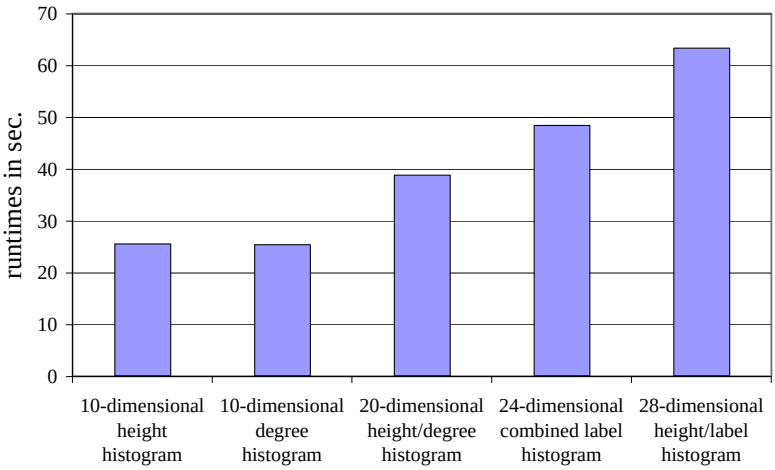
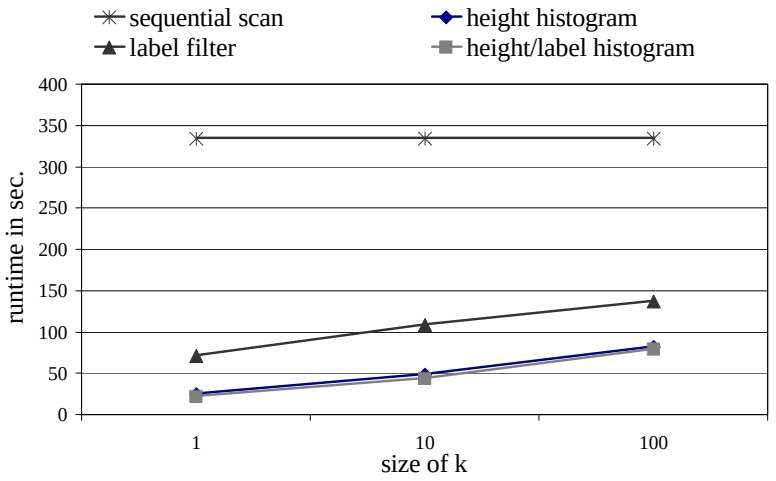
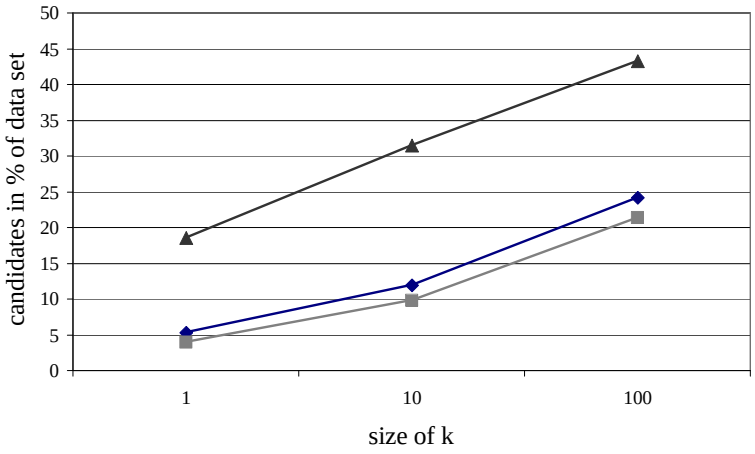


Figure 9.11: Runtimes for filter creation.



(a)



(b)

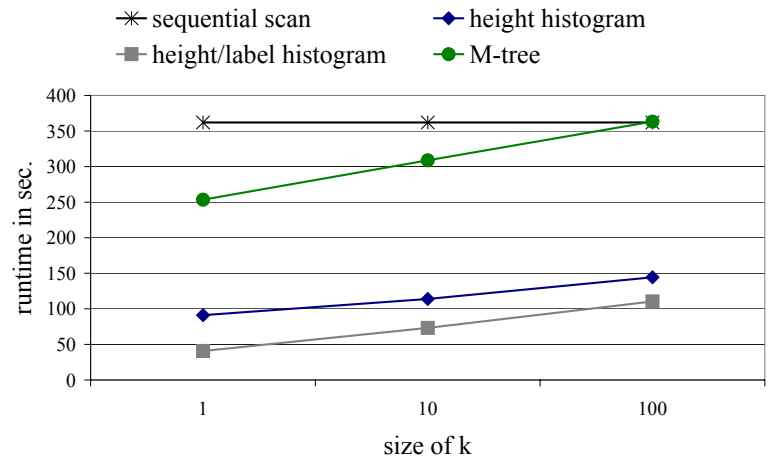
Figure 9.12: Runtime (a) and number of candidates (b) when using a continuous weight function.

In this case, both the height histogram and the label filter are very selective. Unfortunately, the combination of both does not further enhance the runtime. While there is a slight decrease in the number of candidates, this is used up by the additional overhead of evaluating two different filter criteria.

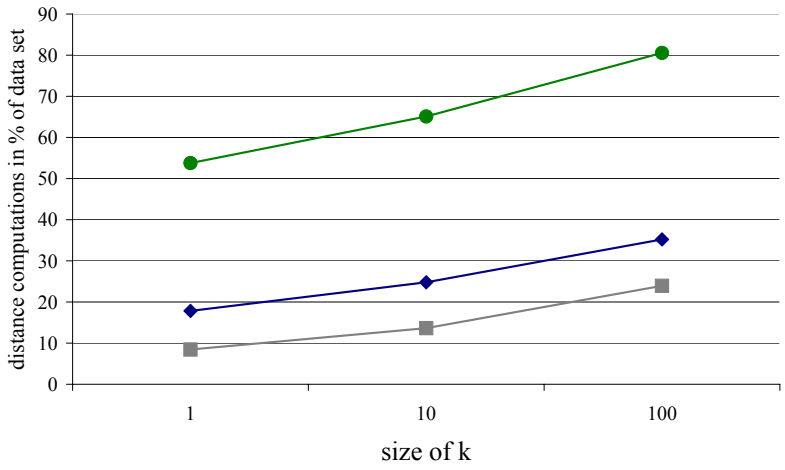
Comparison with a Metric Tree.

In [CNBYM01] other efficient access methods for similarity search in metric spaces are presented. In order to support dynamic data sets, we maintain the filter histograms in data structures that can be updated at any time. Therefore, we chose to compare our filter methods to the M-tree which analogously is a dynamic index structure for metric spaces. We implemented the M-tree as described in [CPZ97], using the best split policy mentioned there.

The creation of an M-tree for 1,000 tree objects already took more than one day, because of the split policy that has quadratic time-complexity. On the other hand, the time for the creation of the filter vectors was in the range of a few seconds. As can be seen in Figure 9.13, the M-tree outperformed the sequential scan for small result set sizes. However, all of our filtering techniques significantly outperform the sequential scan and the M-tree index for all result set sizes. This observation is mainly due to the fact that the filtering techniques reduce the number of necessary distance calculations far more than the M-tree index. This behavior results in speed-up factors between 2.5 and 6.2 compared to the M-tree index and even higher factors compared to a simple sequential scan. Consequently, our multi-step query processing architecture is a significant improvement over the standard indexing approach.



(a)



(b)

Figure 9.13: Runtime (a) and number of distance computations (b) of filter methods compared to the M-tree.

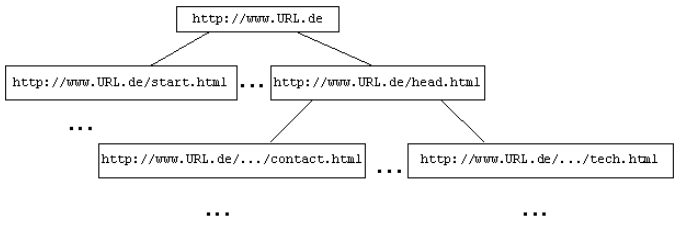


Figure 9.14: Part of a website tree.

9.5.2 Website Graphs

As demonstrated in [WZJS94], the degree-2 edit distance is well suitable for approximate website matching. In website management it can be used for searching similar websites. In [EKS02] website mining is described as a new way to spot competitors, customers and suppliers in the world wide web.

By choosing the main page as the root, one can represent a website as a rooted, labeled, unordered tree. Each node in the tree represents a webpage of the site and is labeled with the URL of that page. All referenced pages are children of that node and the borders of the website where chosen carefully. See Figure 9.14 for an illustration.

For our experiment, we used a compressed form of the 207 websites described in [EKS02], resulting in trees that have 67 nodes on average. We ran 5-nn-queries on this data. The results are shown in Figure 9.15. We notice that even if the degree filter produces a lot more candidates than the height filter, it results in a better runtime. This is due to the fact that it

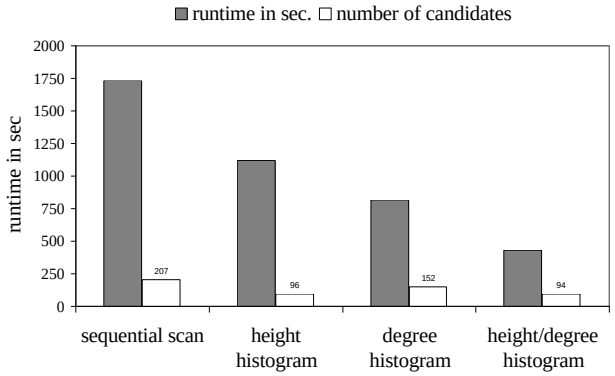


Figure 9.15: Average runtime and number of candidates for 5-nn queries.

filters out those trees, where the computation of the degree-2 edit distance is especially time-consuming. Using the combination of both histograms, the runtime is reduced by a factor of 4.

9.6 Summary

In this chapter, we presented a new approach for efficient similarity search in large databases of tree structures. Based on the degree-2 edit distance as similarity measure, we developed a multi-step query architecture for similarity search in tree structures. For structural as well as for content-based features of unordered attributed trees, we suggested several filter methods. These filter methods significantly reduce the number of complex edit distance calculations which are necessary for similarity search. The main idea behind our filter methods is to approximate the distribution of structural and content-based features within a tree by means of feature histograms. Furthermore, we proposed a new technique for folding histograms and a new way to combine different filter methods in order to improve the filter selectivity. We performed extensive experiments on two sets of real data from the domains of image similarity and website mining. Our experiments showed that filtering significantly accelerates the complex task of similarity search for tree-structured objects. Moreover, it turned out that no single feature of a tree is sufficient for effective filtering, but only the combination of structural and content-based filters yields good results.

Chapter 10

Combining Metric Indexing and Filtering

The general problem of many similarity measures for complex objects is their computational complexity which makes them unusable for large databases. As we have already seen, this is a strong handicap for all approaches where many similarity range queries have to be performed, e.g. for clustering multi-represented objects. In this chapter, we combine and extend the two techniques of multi-step query processing, presented in the previous chapter, and metric index structures to improve the performance of range query processing. The efficiency of our methods is demonstrated in extensive experiments on real-world data including graphs, trees and vector sets. Parts of this material have been published in [KKPS04a].

10.1 Introduction

Density-based clustering algorithms like DBSCAN or OPTICS (cf. Chapter 2) and the techniques presented in the previous chapters are based on range queries for each database object. As each range query requires a lot of exact distance calculations, these algorithms are only applicable to large collections of complex objects if those range queries are supported efficiently. When clustering complex objects, the necessary distance calculations are usually significantly more expensive than the necessary disk accesses. So the ultimate goal for enhancing the efficiency of clustering is to save as many distance calculations as possible.

One approach to improve the performance of range queries is to use a filter-refinement architecture as described in the previous chapter. Another possibility is the use of a metric index structure. In [CNBYM01] several efficient access methods for similarity search in metric spaces are presented. In most real-world applications a static index structure is not acceptable, so dynamic index structures like the M-tree [CPZ97] are applied.

In this chapter, we show that those concepts can beneficially be combined and that through the combination a significant speed-up compared to both separate approaches can be achieved. We discuss how the two approaches can be combined and present some other techniques to improve the efficiency of range query processing. Filters can easily be used to speed up the creation and the traversing of a metric index structure like the M-tree [CPZ97]. Additionally, caching can be used to prevent that the same distance calculations are performed more than once. As DBSCAN [EKSX96], for example, is only interested in getting all objects in the ε- neighborhood of a given query object, but does not need to know the actual distances, we introduce the concept of "positive pruning" to save further distance calculations.

The remainder of the chapter is organized as follows. In Section 10.2 we present our techniques used to save costly distance calculations while performing range queries. The performance gain of our new techniques is presented in Section 10.3, while Section 10.4 concludes the chapter.

10.2 Efficient Range-Queries on Complex Objects

As we have seen in the previous chapter (cf. Section 9.3), multi-step query processing can be used to reduce the number of costly object distance calculations. A second concept to improve the performance of query processing on complex objects are metric index structures. Several index structures for pure metric spaces have been proposed in the literature (see [CNBYM01] for an overview). A well-known dynamic index structure for metric spaces is the M-tree [CPZ97]. The M-tree, which is explained in detail in Section 10.2.1, aims at providing good I/O-performance as well as reducing the number of distance computations. In the following, we will demonstrate the ideas for range queries with the M-tree as index structure and arbitrary filters fulfilling the lower-bounding criterion. It has to be noted that the techniques can also be applied to similar metric index structures like the Slim-tree [TTSF00].

This section is organized as follows. After introducing the necessary concepts for similarity range queries using the M-tree, we present the concept of "positive pruning" in Section 10.2.2. In Section 10.2.3, we combine the two worlds of direct metric index structures and multi-step query processing based on filtering. Furthermore, we show in this section that filters cannot only be used for improving the query response time of an M-tree, but also for efficiently creating an instance of an M-tree. In Section 10.2.4, we show how caching can be applied to accelerate the processing of similarity range queries.

10.2.1 Similarity Range Queries using the M-tree

The M-tree (*metric tree*) [CPZ97] is a balanced, paged and dynamic index structure that partitions data objects not by means of their absolute positions in the multidimensional feature space, but on the basis of their relative distances in this feature space. The only prerequisite is that the distance function between the indexed objects is metric. Thus, the M-tree's domain of applicability is quite general, and all sorts of complex data objects can be organized with this index structure.

The maximum size of all nodes of the M-tree is fixed. All database ob-

jects O_d or references to them are stored in the leaf nodes of an M-tree along with their feature values and the distance $d(O_d, P(O_d))$ to their parent object $P(O_d)$. Inner nodes contain so-called *routing objects* which correspond to database objects to whom a *routing role* was assigned by a promoting algorithm that is executed whenever a node has to be split. Additional to the object description and the distance to the parent object, routing objects O_r also store their *covering radius* $r(O_r)$ and a pointer $ptr(T(O_r))$ to the root node of their subtree, the so-called *covering tree* of O_r. For all objects O_d in this covering tree, the condition holds that the distance $d(O_r, O_d)$ is smaller or equal to the covering radius $r(O_r)$. This property induces a hierarchical structure of an M-tree, with the covering radius of a parent object always being greater than or equal to all covering radii of their children and the root object of an M-tree storing the maximum of all covering radii.

Range queries are specified by a query object O_q and a range value ε by which the answer set is defined to contain all the objects O_d from the database that have a distance to the query object O_q of less than or equal to ε:

Definition 10.1 (similarity range query)
Let $\mathcal{O}$ be a domain of objects and $DB \subseteq \mathcal{O}$ be a database. For a query object $O_q \in \mathcal{O}$ and a query range $\varepsilon \in \mathbb{R}_0^+$, the similarity range query simRange *: $\mathcal{O} \times \mathbb{R}_0^+ \mapsto 2^{DB}$ returns the set*

$$\mathit{simRange}(O_q, \varepsilon) = \{O_d \in DB | dist(O_d, O_q) \leq \varepsilon\}.$$

Given a query object O_q and a similarity range parameter ε, a similarity range query $\mathsf{simRange}(O_q, \varepsilon)$ starts at the root node of an M-tree and recursively traverses the whole tree down to the leaf level, thereby pruning all subtrees which certainly contain no result objects.

A description of simRange in pseudocode and the recursive procedure rangeSearch used to traverse the M-tree is given in Figure 10.1.

The subtree of a routing object O_r can be pruned if the absolute value of the distance of the routing object's parent object O_p to the query object O_q, $d(O_p, O_q)$, minus the distance between O_r and O_p is greater than the covering radius of O_r plus ε:

```
1   simRange(queryObject O_q, range ε) → ResultSet
2     result = NIL;
3     rangeSearch(root, O_q, ε);
4     return result;
```

```
1   rangeSearch(Node N, queryObject O_q, range ε)
2     O_p := parent object of node N;
3     if N is not a leaf then
4       for each O_r in N do
5         if |d(O_p, O_q) − d(O_r, O_p)| ≤ r(O_r) + ε
6         then
7           compute d(O_r, O_q);
8           if d(O_r, O_q) <= r(O_r) + ε then
9             rangeSearch(ptr(T(O_r), O_q, ε);
10          end if
11        end if
12      end for
13    else
14      for each O_d in N do
15        if |d(O_p, O_q) − d(O_d, O_p)| ≤ ε then
16          compute d(O_d, O_q);
17          if d(O_d, O_q) ≤ ε then
18            add O_d to result;
19          end if
20        end if
21      end for
22    end if
```

Figure 10.1: Pseudocode description of similarity range search on M-trees.

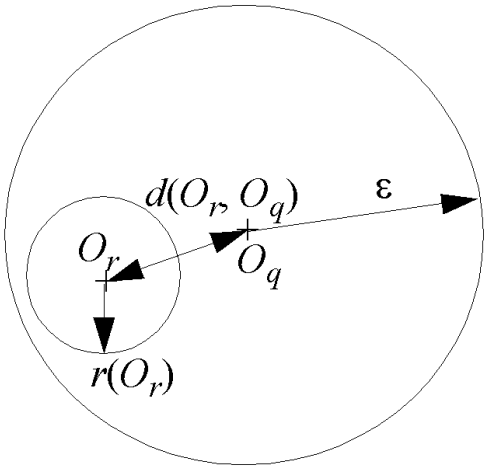


Figure 10.2: Positive pruning for the M-tree.

$$|d(O_p, O_q) - d(O_p, O_r)| > r(O_r) + \varepsilon.$$

A proof for this is given in [CPZ97]. Thus, as the distance between O_p and O_q has already been computed when accessing a node N, subtrees can be pruned without further distance computations (see line 5 of the algorithm in Figure 10.1).

10.2.2 Positive Pruning

A hierarchical index structure, like the M-tree, is composed of directory nodes with routing objects O_r which represent all objects in their respective subtree $T(O_r)$. For all objects $O \in T(O_r)$, $d(O_q, O_r) \leq r(O_r)$ holds. Efficient processing of range queries on the original M-tree is based on the concept of "negative pruning". During the query processing, certain subtrees are excluded from the search based on the following formula: $d(O_q, O_r) > r(O_r) + \varepsilon$ (see line 7 of the algorithm in Figure 10.1).

In this section, we introduce the concept of "positive pruning". If a directory node is completely covered by the query range, we can report all objects on the leaf level of the M-tree without performing any cost intensive distance computations (cf. Figure 10.2).

```
1       rangeSearch(Node N, queryObject O_q, range ε)
                                ⋮
7           compute d(O_r, O_q);
7a          if d(O_r, O_q) + r(O_r) ≤ ε then
7b            report all objects in T(O_r);
8           else if d(O_r, O_q) <= r(O_r) + ε then
                                ⋮
```

Figure 10.3: Adaptation of similarity range search on M-trees for positive pruning.

Lemma 10.1 *Let $O_q \in \mathcal{O}$ be a query object and $\varepsilon \in \mathbb{R}_0^+$ a query range. Furthermore, let O_r be a routing object in an M-tree with a covering radius $r(O_r)$ and a subtree $T(O_r)$. Then the following statement holds:*

$$d(O_r, O_q) + r(O_r) \leq \varepsilon \Rightarrow \forall O \in T(O_r) : d(O, O_q) \leq \varepsilon$$

Proof. *The following inequalities hold for all $O \in T(O_r)$ due to the triangle inequality and due to $d(O_r, O_q) + r(O_r) \leq \varepsilon$:*

$$\begin{aligned} d(O, O_q) &\leq d(O, O_r) + d(O_r, O_q) \\ &\leq r(O_r) + d(O_r, O_q) \leq \varepsilon. \end{aligned}$$

◇

In the case of negative pruning, we skip the recursive tree traversal of a subtree $T(O_r)$ if the query range does not intersect the covering radius $r(O_r)$. In the case of positive pruning, we skip all the distance calculations involved in the recursive tree traversal if the query range completely covers the covering radius $r(O_r)$. In this case, we can report all objects stored in the corresponding leaf nodes of this subtree without performing any further distance computations. Figure 10.3 shows how this concept can be integrated into the original method rangeSearch as depicted in Figure 10.1.

This approach is very beneficial for accelerating density-based clustering on complex objects. DBSCAN, for instance, only needs the information whether an object is contained in $\mathsf{simRange}(O_q, \varepsilon) = \{O \in DB | d(O, O_q) \leq \varepsilon\}$ but not the actual distance of this object to the query object O_q.

10.2.3 Combination of Filtering and Indexing

The M-tree reduces the number of distance calculations by partitioning the data space even if no filters are available. Unfortunately, the M-tree may suffer from the navigational cost related to the distance computations during the recursive tree traversal. On the other hand, the filtering approach heavily depends on the quality of the filters.

When combining both approaches, these two drawbacks are reduced. We use the filter distances to optimize the required number of exact object distance calculations needed to traverse the M-tree. Thereby, we do not save any I/O cost compared to the original M-tree, as the same nodes are traversed, but we save a lot of costly distance calculations that are necessary for the traversal. In the following, we call this combination *filtering M-tree*. This filtering M-tree stores the objects along with their corresponding filter values within the M-tree. A similarity query based on the filtering M-tree always computes the filter distance values prior to the exact distance computations. If a filter distance value is already a sufficient criterion to prune branches of the M-tree, we can avoid the exact distance computation. If we have several filters, the filter distance computation always returns the maximum value of all filters.

The pruning quality of the filtering M-tree benefits from both the quality of the filters and the clustering properties of the index structure. In the following, we will show that the number of distance calculations used for range queries as well as for the creation of an M-tree can be optimized by using lower-bounding filters.

Range Queries

Similarity range queries are used to retrieve all objects from a database which are within a certain similarity range from the query object (cf. Definition 10.1). By computing the filter distance prior to the exact distance, we can save many distance computations. Based on the following lemma, we can prune many subtrees without computing the exact distances between a query object O_q and a routing object O_r (cf. Figure 10.4).

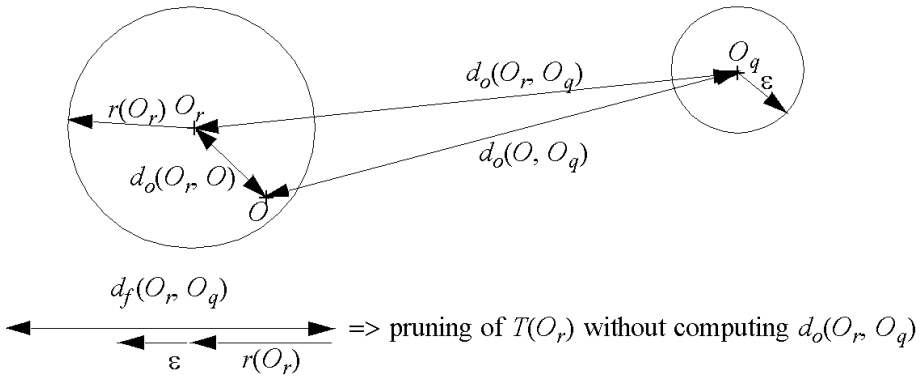


Figure 10.4: Similarity range query based on the filtering M-tree.

Lemma 10.2 *Let $\mathcal{O}$ be a set of objects and $DB \subseteq \mathcal{O}$ a database. Furthermore, let $d_o, d_f : \mathcal{O} \times \mathcal{O} \mapsto \mathbb{R}_0^+$ be two distance functions for which d_f lower bounds d_o, i.e. $\forall O_1, O_2 \in \mathcal{O} : d_f(O_1, O_2) \leq d_o(O_1, O_2)$ holds. Let $O_q \in \mathcal{O}$, $\varepsilon \in \mathbb{R}_0^+$. For each routing object $O_r \in DB$ with covering radius $r(O_r) \in \mathbb{R}_0^+$ and subtree $T(O_r)$ the following statement holds:*

$$\forall O \in T(O_r) : (d_f(O_q, O_r) > r(O_r) + \varepsilon) \Rightarrow d_o(O_q, O) > \varepsilon.$$

Proof. *As $\forall O_1, O_2 \in \mathcal{O} : d_f(O_1, O_2) \leq d_o(O_1, O_2)$ holds, the following statement is true:*

$$d_f(O_q, O_r) > r(O_r) + \varepsilon \Rightarrow d_o(O_q, O_r) > r(O_r) + \varepsilon.$$

Based on the triangle inequality and our assumption that $d_o(O, O_r) \leq r(O_r)$, we can prove the above lemma as follows:

$$\begin{aligned} & d_f(O_q, O_r) > r(O_r) + \varepsilon \\ \Rightarrow\ & d_o(O_q, O_r) > r(O_r) + \varepsilon \\ \Rightarrow\ & d_o(O_q, O_r) - r(O_r) > \varepsilon \\ \Rightarrow\ & d_o(O_q, O_r) - d_o(O, O_r) > \varepsilon \\ \Rightarrow\ & d_o(O_q, O) > \varepsilon \end{aligned}$$

$\diamond$

```
1     rangeSearch(Node N, queryObject O_q, range ε)
                              ⋮
5       if |d(O_p, O_q) − d(O_r, O_p)| ≤ r(O_r) + ε
6       then
6a        compute d_f(O_r, O_q)
6b        if d_f(O_r, O_q) <= r(O_r) + ε then
7           compute d(O_r, O_q);
8           if d(O_r, O_q) <= r(O_r) + ε then
                              ⋮
15      if |d(O_p, O_q) − d(O_d, O_p)| ≤ ε then
15a       compute d_f(O_d, O_q)
15b       if d_f(O_d, O_q) ≤ ε then
16          compute d(O_d, O_q);
17          if d(O_d, O_q) ≤ ε then
                              ⋮
```

Figure 10.5: Adaptation of similarity range search on M-trees for filtering.

Let us note that a similar optimization can be applied to the objects stored on the leaf level with the assumption that their 'covering radius' is 0. Figure 10.5 shows how this concept can be integrated into the original method rangeSearch of Figure 10.1.

Construction of an M-tree

Filters can also be used for accelerating the creation of an M-tree.

Insert. They can be used to accelerate the function which decides which tree to follow during the recursive tree-traversal of the insert operation. The main idea is that we sort all objects according to the filter distance and then walk through this sorted list. Thereby, we first test those candidates which might not lead to an increase in the covering radius. If we detect a routing object for which no increase is necessary, we postpone the reporting of this object. We first investigate all routing objects which are closer to the given query object and possibly also do not have to increase their covering radius. If several of those routing objects exist, we take the one closest to the inserted object. If no such routing object exists, we walk through the list until we have found the routing object which leads to a minimal increase

of its covering radius. Let us note that this idea is closely related to the optimal multi-step k-nearest neighbor search algorithm [SK98].

Split. If a node overflow occurs due to an insertion, the node has to be split adequately. The 'ideal' split strategy should promote two new routing objects such that for the resulting regions volume and overlap are minimized. Several different strategies for splitting a node are described in [CPZ97]. There the authors show that in most cases it is the best strategy to minimize the maximum of the resulting covering radii. This strategy, which is called *mM_Rad*, is also the most complex in terms of distance computations. It considers all possible pairs of objects and after partitioning the set of entries promotes the pair of objects for which the maximum of the two covering radii is minimal. Given a set of n entries and two routing objects, the generalized hyperplane decomposition is used to assign each of the n objects to one of the two routing objects. Although this leads to unbalanced splits, experimental results show that it is superior to techniques resulting in a balanced distribution.

The filter distances can also be used to speed up the split of an M-tree node. The main idea is that we generate a priority queue containing pairs of promoting objects based on the filter distances. We walk through this list and if we detect that the *mM_Rad* value based on the filters is higher than the best already found *mM_Rad* value based on exact distance computations, we can stop. Thus, we do not necessarily have to test all $O(n^2)$ pairs of promoting objects. Again this approach is similar to [SK98]. Furthermore, if we test two actual promoting objects O_{p1} and O_{p2}, we have to assign an object O either to O_{p1} or to O_{p2}. This test can be accelerated by computing first the actual distance between O and the promoting object for which the filter distance is smaller. If the resulting exact distance is still smaller than the filter distance to the other promoting object, we can save on the second exact distance computation.

10.2.4 Caching Distance Calculations

In this section, we present a further technique which helps to avoid costly distance computations for index construction and query processing.

Cache-Based Construction

If we have to cope with distance computations which are more expensive than accessing secondary storage, we suggest to store the already processed distance computations to disk. Especially when splitting the same overflowing node repeatedly, accessing stored distance computation values can speed up the insertion process since otherwise the same distances are computed several times.

Cache-Based Range Queries

Efficient query processing of range queries also benefits from the idea of caching distance calculations. During the navigation through the M-tree directory, the same distance computations may have to be carried out several times. Although each object O is stored only once on the leaf level of the M-tree, it might be used several times as routing object. Furthermore, we often have the situation that distance calculations carried out on the directory level have to be repeated at the leaf level.

As shown in Figure 10.1, a natural way to implement range queries is by means of recursion resulting in a depth-first search. We suggest to keep all distance computations in main memory which have been carried out on the way from the root to the actual node. After leaving the node, i.e. when exiting the recursive function, we delete all distance computations carried out at this node. This limits the actual main memory footprint to $O(h \cdot b)$ where h denotes the maximum height of a tree and b denotes the maximum number of stored elements in a node. Even in multi-user environments this rather small worst-case main memory footprint is tolerable. The necessary adaptations of the rangeSearch algorithm are drafted in Figure 10.6.

```
1       rangeSearch(Node N, queryObject O_q, range ε)
                                    ⋮
6         distCache(N, O_r, O_q);
                                    ⋮
16        distCache(N, O_d, O_q);
                                    ⋮
22        end if
22a       deleteCache(N);
```

```
distCache(Node N, Object O_1, Object O_2) → float
  result = hashtable.lookup(O_1, O_2);
  if result = null then
    result = compute d(O_1, O_2);
    hashtable.add(N, O_1, O_2, result);
  end if
  return result;
```

```
deleteCache(Node N)
  hashtable.delete(N);
```

Figure 10.6: Adaptation of similarity range search on M-trees for caching.

10.3 Evaluation

To show the efficiency of our approach, we chose the applications and data types described in Section 1.2 and performed extensive experiments. All algorithms were implemented in Java 1.4 and the experiments were run on a workstation with a Xeon 1.7 GHz processor and 2 GB main memory under Linux. We implemented the M-tree as described in [CPZ97]. As in all cases, the time for distance calculations was dominating the runtime of a range query, we only show the number of distance calculations and not the runtime.

10.3.1 CAD Vector Set Data

For the experiments with this data type, we used the similarity model presented in [KBK+03], where CAD objects were represented by a vector set

consisting of 7 vectors in 6D. All experiments were carried out on a data set containing 5,000 CAD objects from an American aircraft producer. As distance measure between sets of feature vectors we used the minimal matching distances which can be computed in $O(k^3)$, where k denotes the cardinality of the point set, using the Kuhn-Munkres [Kuh55, Mun57] algorithm. As filter we used the centroid filter introduced in [KBK+03].

Creation of the M-tree

The generation of the optimized M-tree was carried out without caching (cf. Figure 10.7) and with caching (cf. Figure 10.8). Without caching, the number of necessary distance calculations is very high due to the repeated splitting of nodes. Note that the number of distance calculations for one node split is quadratic w.r.t. the number of elements of this node. In this case, our nodeSplit algorithm only needs 1/4 of the distance calculations while still producing the same M-tree. If we apply caching, the overall number of required distance computations is much smaller. This is due to the fact that many distance values necessary for splitting a node can be fetched from disk. In this case, our $findSubTree$ function allows us to reduce the number of required distance calculations even further, i.e. the number of distance computations is bisected. To sum up, both optimizations, which are based on the exploitation of available filter information, allow us to build up an M-tree much more efficiently.

Range Queries

Figure 10.9 and 10.10 show in which way the different approaches for range query processing depend on the chosen ε-value. Figure 10.9 shows that for the investigated data set the original M-tree is the worst access method for all ε-values. On the other hand, the pure filter performs very well. For this data set, reasonable ε-values for density-based clustering would be ~ 1 for DBSCAN and ~ 2 for OPTICS. In this parameter range, our approach clearly outperforms both the filter and especially the original M-tree.

In Figure 10.10 one can see that for small ε-values we benefit from the filtering M-tree, whereas for higher values we benefit from caching and pos-

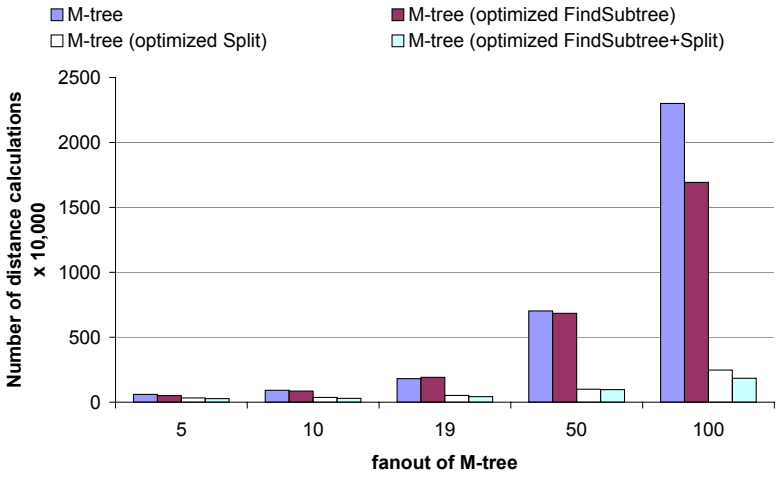


Figure 10.7: Creation without caching distance calculations.

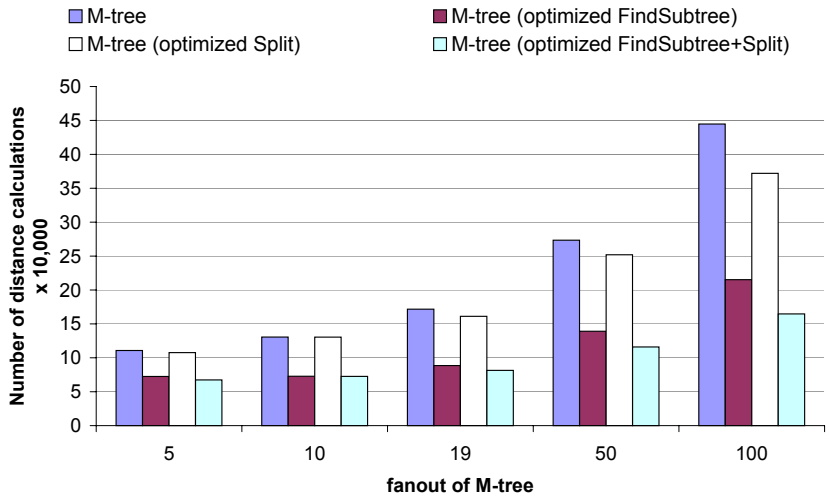


Figure 10.8: Creation with caching distance calculations.

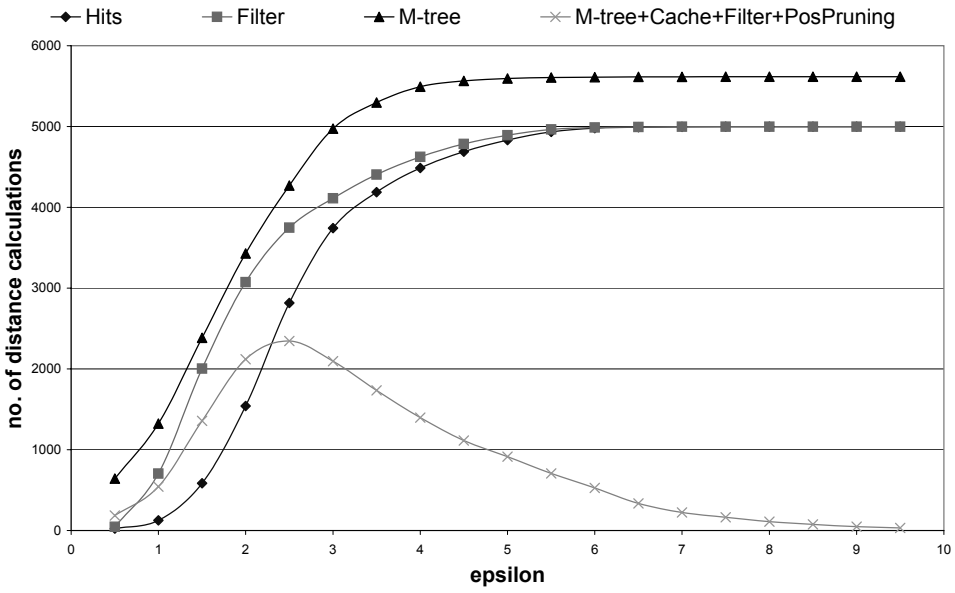


Figure 10.9: Comparison of our best technique to M-tree and filtering for vector set data .

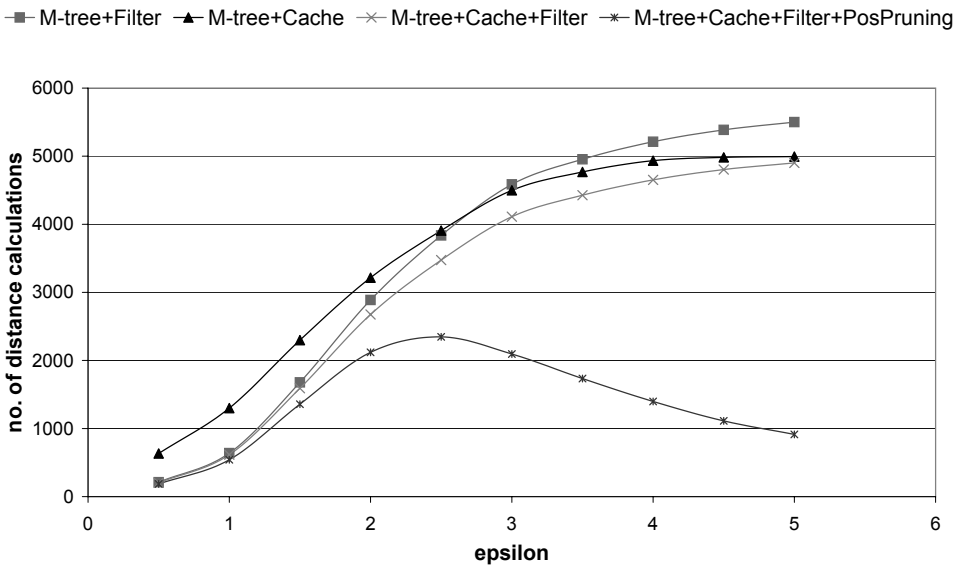


Figure 10.10: Comparison of our techniques for vector set data.

itive pruning.

Furthermore, we clustered the data set using OPTICS [ABKS99] which forms the basis for the visual data mining tool presented in Section 2.3. With a suitable parameter setting for OPTICS, we achieved a speed-up of 16% compared to the centroid filter, 33% compared to the original M-tree, and 104% compared to the sequential scan. Let us note that the average cardinality of the result set of each range query was almost 2,000 which limits the best achievable speed-up to 150%.

10.3.2 Image Data

As we have seen in Chapter 7, image data is a good example for multi-represented complex data. In Chapter 8 it is shown that the presented approach for clustering multi-represented objects is able to get the best out of different types of representations. For this multi-represented clustering algorithm it is very important that the necessary range queries are supported efficiently. Here we present some experiments for image data represented as trees or graphs. For this representations the efficiency of range query processing is especially important because of the complex similarity measure.

Tree Structured Image Data.

We used the tree description of images as presented in Section 7.2.1. As similarity measure for the resulting trees, we used again the degree-2 edit distance and the filter refinement architecture as described in Chapter 9. We used a sample set of 10,000 color TV-Images. For the experiments, we chose reasonable epsilon values for the multi-represented clustering algorithm.

Figure 10.11 shows that we achieve a significant speed-up compared to the original M-tree. As can be seen, we also outperform the pure filtering approach.

Graph Structured Image Data.

We used the graph description of images as presented in Section 7.2.2 and the edge matching distance and the image data set as described in [KS03]. The filter presented in this paper is almost optimal, i.e. the number of unnecessary distance calculations during query processing is very low. Even in this case, our technique is as good as the filter.

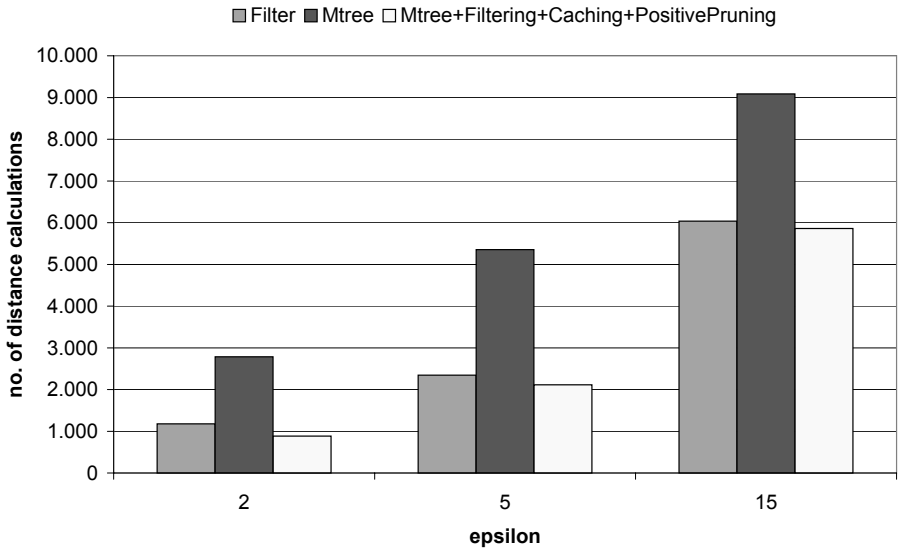


Figure 10.11: Comparison of our best technique to M-tree and filtering for tree structured data.

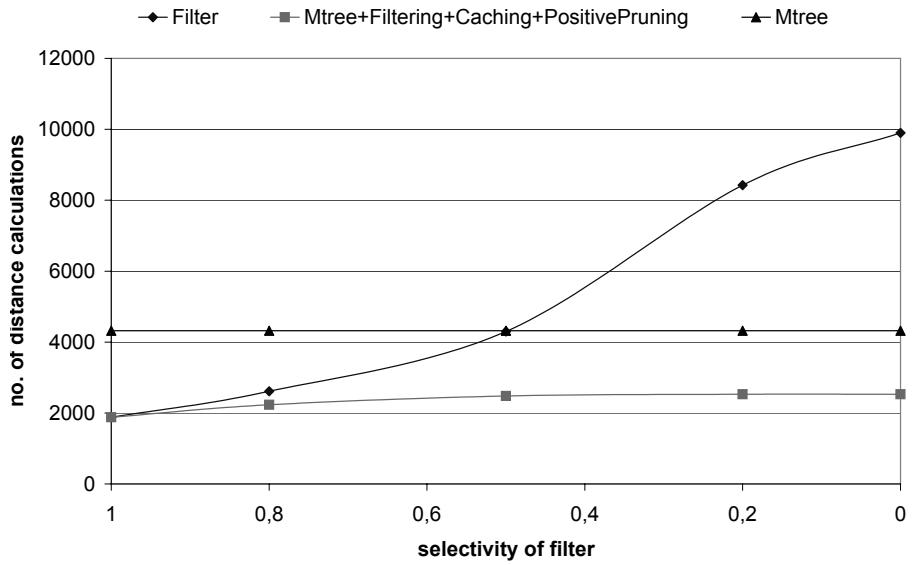


Figure 10.12: Comparison of our techniques for graph data.

To show the robustness of our approach with respect to the filter selectivity, we reduced it in a stepwise process. We weighted the original filter distances with constant factors to decrease the filter selectivity. Figure 10.12 shows that independent from the filter selectivity our approach outperforms the original M-tree by a factor of almost 2 and is at least as good as the pure filtering approach.

10.4 Summary

In this chapter, we showed that there are a lot of interesting application areas for density-based clustering of complex objects. Density-based clustering is based on similarity range queries where the similarity measures used for complex objects are often computationally very complex which makes them unusable for large databases. To overcome the efficiency problems, metric index structures or multi-step query processing are applied. We combined and extended these approaches to achieve the best from two worlds. More precisely, we presented three improvements for metric index structures, i.e. positive pruning, the combination of filtering and indexing, and caching. In a broad experimental evaluation based on real-world data sets we showed that our approach achieves a significant speed-up for similarity range queries. By means of our new techniques, the clustering of complex multi-represented objects can be extended to larger databases.

Part IV

Conclusions

Chapter 11

Summary and Future Research Directions

Within the KDD process, data mining is the application of algorithms to discover patterns and trends in large databases. Clustering is one of the most important data mining tasks. The methods and concepts presented in this thesis contribute to the field of clustering complex objects. This chapter summarizes the main contributions of this thesis (Section 11.1) and shows some directions for future work (Section 11.2).

11.1 Summary of Contributions

The rapidly increasing amount of data stored in databases requires efficient and effective data mining methods to gain new information contained in the collected data. Clustering is one of the primary data mining tasks and aims at detecting subgroups of similar data objects. This thesis contributes to the field of clustering complex objects. New and original solutions, extending the density-based clustering approach, are proposed. In the following, we give a detailed summary of these contributions.

11.1.1 Clustering High-Dimensional Vector Data

Part II dealt with the problem of subspace clustering which is an active area of research. After discussing recent work on clustering high-dimensional data, we presented how the density-based clustering approach can be extended to clustering high-dimensional data.

The algorithm SUBCLU (density-connected *Sub*space *Clu*stering) was proposed which automatically and efficiently computes all "flat" subspace clusters DBSCAN would have found if applied to all possible subspaces. SUBCLU was applied to a real-world gene expression data set, outperforming comparative subspace clustering approaches in terms of effectivity and yielding a significant amount of important biological information.

Additionally, two subspace selection techniques, RIS (*R*anking *I*nteresting *S*ubspaces) and SURFING (*SU*bspaces *R*elevant *F*or cluster*ING*) were proposed for the subspace clustering problem. RIS and SURFING rank the subspaces according to their clustering quality instead of directly computing the subspace clusters. Afterwards a user can choose some subspaces from a list sorted by clustering quality and apply his own (e.g. hierarchical) clustering algorithm to the particular subspaces. The advantage of RIS and SURFING is that they can be combined with a hierarchical clustering algorithm. The combination of RIS or SURFING with OPTICS was applied to gene expression data, yielding further important insights that were missed by SUBCLU.

Furthermore, we combined the density-based clustering notion with principal component analysis, a primitive to measure correlation. Based on

this combination, a formalization of correlation-connected clusters was presented. We proposed an efficient algorithm called 4C (*C*omputing *C*orrelation *C*onnected *C*lusters) to compute such correlation-connected clusters and applied this method on a gene expression data set and on a metabolome data set. 4C shows a significant accuracy gain compared to other clustering methods.

In summary, we proposed four new techniques for clustering high-dimensional data which extend the density-based clustering notion. The benefit of the proposed methods is that the advantages of the powerful density-based clustering model are conserved.

11.1.2 Clustering Complex Objects in Arbitrary Metric Spaces

Often, complex objects can not be represented by a feature transformation. In this case, more complex similarity models are used to capture the intuitive notion of similarity. Part III dealt with the challenges of clustering such complex data in arbitrary metric spaces.

First, a new solution to handle multi-represented complex objects was presented. A lot of complex objects provide more than one form of representation to capture the intuitive notion of similarity. Using our new approach for multi-represented data, more information is available during the clustering process and thereby more accurate clusters are generated.

Afterwards we addressed the efficiency aspect of this new approach. As most multi-represented objects have complex similarity measures, the execution of the necessary range queries has to be supported efficiently. We showed how a filter-refinement architecture can be used to enhance the runtime of query-processing for tree-structured objects. Therefore, we presented several filters, i.e. structural and content-based filters, for the degree-2 edit distance, one of the common similarity measures for tree-structured objects. The evaluation part showed that filtering significantly accelerates the runtime of query processing for images or websites represented as trees.

To further improve the performance of range query processing on complex objects, we combined the approach of filter-refinement with metric indexing. We demonstrated how filters can be used to improve the perfor-

mance of metric indices like the M-tree. Additionally, we proposed positive pruning and caching to enhance the efficiency of range query processing with the M-tree. Our experiments with three different complex similarity models showed a significant speed-up.

To sum up, by means of our new techniques, clustering can be extended to larger collections of complex objects. Compared to existing approaches, using only one representation for clustering, we increased the effectivity of clustering complex objects.

11.2 Potentials for Future Work

At the end of this thesis let us emphasize that our work opens up a wide range of potentials for future work.

For the clustering of high-dimensional vector data, we want to point out two interesting research directions.

In part II of this thesis, we concentrated on the effectivity of clustering high-dimensional data. We did not address the efficiency of the four presented approaches. All of them are based on the execution of partial range queries (range queries in arbitrary subspaces of the original data space) or partial k-nearest neighbor queries (k-nearest neighbor queries in arbitrary subspaces of the original data space). To date, there is no index structure efficiently supporting those queries in arbitrary subspaces. Thus, an important approach for future work is the development of new techniques to support partial range or partial k-nearest neighbor queries. An open question is, for example, if traditional index structures which originally cannot be applied to this problem can be adopted to solve this problem.

Another encouraging research direction is to further examine the idea of correlation clustering. While 4C is an interesting and promising algorithm for correlation clustering, there are still unsolved problems. Currently, 4C can only detect correlations of a fixed correlation dimension. However, two k-dimensional correlations can, for example, form a $(k + 1)$-dimensional correlation. It would be interesting to investigate how the concepts of correlation-connected clusters could be extended to find correlation hierarchies. Furthermore, searching for non-linear correlations is another impor-

tant research direction. Combining the density-based clustering notion with other correlation primitives like fractal dimension or Hough transformations might be a promising approach.

Let us point out two interesting directions for future research in the field of clustering complex arbitrary metric data.

The presented approach for clustering multi-represented data can be extended in many directions. First of all, combining more than two representations poses the interesting question if all representation should be intersected or united. One can of course imagine a combination of the intersection or union method. It would be interesting to investigate how the user can be aided in his task to find out which technique is best suited for his data. Another interesting challenge is to extend our method to an multi-instance and multi-represented clustering. In this setting, each object may be represented by several instances in some of the representations. And again, a hierarchical version of this approach would be highly desirable.

In the last chapter we introduced an optimized M-tree. One could imagine to use this optimized M-tree for effectively and efficiently navigating through arbitrary metric data sets, similar to the approach BOSS presented in Section 2.3. Each directory node of an M-tree consists of objects representing all elements stored in the corresponding spherical subtrees. Thus, the tree itself can be regarded as a hierarchical clustering which, additionally, efficiently supports all kinds of similarity queries. Furthermore, the optimizations introduced in this paper allow to build up an optimized M-tree much more efficiently than carrying out a complete hierarchical density-based clustering. In order to increase the quality, i.e. to minimize the overlap between subtrees of the optimized M-tree, one could carry out update operations similar to update operations on Slim-trees [TTSF00], i.e. use a variant of the slim-down algorithm trying to keep the tree tight. Of course the trade-off between quality and efficiency of this new dynamic data-mining browsing tool had to be elaborated.

List of Figures

List of Tables

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 Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA*, pages 49–60, 1999.

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

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. "Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'98), Seattle, WA*, pages 94–105, 1998.

[AKKS99] M. Ankerst, G. Kastenmüller, H.-P. Kriegel, and T. Seidl. "3D Shape Histograms for Similarity Search and Classification in Spatial Databases". In *Proc. 6th Int. Symposium on Large Spatial Databases (SSD'99), Hong Kong, China*, pages 207–226, 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 Int. Conf. on Very Large Databases (VLDB'95), Zurich, Switzerland*, pages 490–501, 1995.

[AP99] C. C. Aggarwal and C. Procopiuc. "Fast Algorithms for Projected Clustering". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'99), Philadelphia, PA*, 1999.

[AS94] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules". In *Proc. 20th Int. Conf. on Very Large Databases (VLDB'94), Santiago, Chile*, pages 487–499, 1994.

[AY00] C.C. Aggarwal and P.S. Yu. "Finding Generalized Projected Clusters in High Dimensional Space". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'00), Dallas, TX*, 2000.

[BBA+03] B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M.J. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. "The SWISS-PROT Protein Knowledgebase and its Supplement TrEMBL in 2003". *Nucleic Acid Research*, 31:365–370, 2003.

[BBB+97] S. Berchtold, C. Böhm, B. Braunmüller, D. Keim, and H.-P. Kriegel. "Fast Parallel Similarity Search in Multimedia Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'97), Tucson, AZ*, pages 1–12, 1997.

[BBB+04] C. Baumgartner, C. Böhm, D. Baumgartner, G. Marini, K. Weinberger, B. Olgemöller, B. Liebl, and A. A. Roscher. "Supervised machine learning techniques for the classification of metabolic disorders in newborns". *Bioinformatics*, 2004. in press.

[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 Int. Conf. on Data Engineering (ICDE'00), San Diego, CA*, pages 577–588, 2000.

[BBKK97] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. "A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space". In *Proc. ACM PODS Symp. on Principles of Database Systems (PODS'97), Tucson, AZ*, pages 78–86, 1997.

[BC00] D. Barbara and P. Chen. "Using the Fractal Dimension to Cluster Datasets". In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'00), Boston, MA*, pages 260–264, 2000.

[BDL95] M. Berry, S. Dumais, and T. Landauer. "Using Linear Algebra for Intelligent Information Retrieval". *SIAM Review*, 37(4):573–595, 1995.

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

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

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

[BKKZ04] C. Böhm, K. Kailing, P. Kröger, and A. Zimek. "Computing Clusters of Correlation Connected Objects". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'04), Paris, France*, pages 455–466, 2004.

[Bou96] A. Bouguettaya. "On-Line Clustering". *IEEE Transactions on Knowledge and Data Engineering*, 8(2):333–339, 1996.

[BS98] H. Bunke and K. Shearer. "A graph distance metric based on the maximal common subgraph". *Pattern Recognition Letters*, 19(3-4):255–259, 1998.

[CFZ99] C.-H. Cheng, A.W.-C. Fu, and Y. Zhang. "Entropy-Based Subspace Clustering for Mining Numerical Data". In *Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'99), San Diego, CA*, pages 84–93, 1999.

[CHH97] J.F. Cullen, J.J. Hull, and P.E. Hart. "Document Image Database Retrieval and Browsing Using Texture Analysis". In *Proc. 4th Int. Conf. Document Analysis and Recognition (ICDAR'97), Ulm, Germany*, pages 718–721, 1997.

[CKS+88] P. Cheeseman, J. Kellu, M. Self, J. Stutz, W. Taylor, and D. Freeman. "AUTOCLASS: A Bayesian Classification System". In *Proc. 5th Int. Conf. on Machine Learning, Ann Arbor, MI*, pages 54–64, 1988.

[CKS98] G. Chartrand, G. Kubicki, and M. Schultz. "Graph Similarity and Distance in Graphs". *Aequationes Mathematicae*, 55(1-2):129–145, 1998.

[CM00] K. Chakrabarti and S. Mehrotra. "Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces". In *Proc. 26th Int. Conf. on Very Large Databases (VLDB'00), Cairo, Egypt*, pages 89–100, 2000.

[CNBYM01] E. Chavez, G. Navarro, R. Baeza-Yates, and J.L. Marroquin. "Searching in metric spaces". *ACM Computing Surveys*, 33(3):273–321, 2001.

[Con00] The Gene Ontology Consortium. "Gene Ontology: Tool for the Unification of Biology". *Nature Genetics*, 25:25–29, 2000.

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

[DCSL02] M. Dash, K. Choi, P. Scheuermann, and H. Liu. "Feature Selection for Clustering – A Filter Solution". In *Proc. of the 2rd IEEE Int. Conf. on Data Mining (ICDM'02), Maebashi City, Japan*, pages 115–122, 2002.

[DK02] M. Deshpande and G. Karypis. "Evaluation of Techniques for Classifying Biological Sequences". In *Proc. 6th Pacific-Asia Conf. on Advances in Knowledge Discovery and Data Mining (PAKDD'02), Taipei, Taiwan*, pages 417–431, 2002.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data via the EM Algorithm". *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[EKS+98] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. "Incremental Clustering for Mining in a Data Warehousing Environment". In *Proc. 24th Int. Conf. on Very Large Databases (VLDB'98), New York, NY*, pages 323–333, 1998.

[EKS02] M. Ester, H.-P. Kriegel, and M. Schubert. "Web Site Mining: A new way to spot Competitors, Customers and Suppliers in the World Wide Web". In *Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (SIGKDD'02), Edmonton, Canada*, pages 249–258, 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 Int. Conf. on Knowledge Discovery and Data Mining (KDD'96), Portland, OR*, pages 226–231, 1996.

[Eve81] B. Everitt. *"Cluster Analysis"*. Heinemann, London, 1981.

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

[FCE00] C.-S. Fuh, S.-W. Cho, and K. Essig. "Hierarchical color image region segmentation for content-based image retrieval system". *IEEE Transactions on Image Processing*, 9(1):156–162, 2000.

[Fis87] D. H. Fischer. "Knowledge Acquisition via Incremental Conceptual Clustering". *Machine Learning*, 2(2):139–172, 1987.

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

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

[Fuk90] K. Fukunaga. *"Introduction to Statistical Pattern Recognition"*. Academic Press, San Diego, CA, 1990.

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

[GNC99] S. Goil, H.S. Nagesh, and A. Choudhary. "MAFIA: Efficient and Scalable Subspace Clustering for Very Large Data Sets". Tech. Report No. CPDC-TR-9906-010, Center for Parallel and Distributed Computing, Dept. of Electrical and Computer Engineering, Northwestern University, 1999.

[GRS98] S. Guha, R. Rastogi, and K. Shim. "CURE: An efficient Clustering Algorithm for Large Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'98), Seattle, WA*, pages 73–84, 1998.

[HCH99] B. Huet, A. Cross, and E. Hancock. "Shape Retrieval by Inexact Graph Matching". In *Proc. IEEE Int. Conf. on Multimedia Computing Systems (ICMCS'99), Florence, Italy*, pages 722–776, 1999.

[HK98] A. Hinneburg and D. A. Keim. "An Efficient Approach to Clustering in Large Multimedia Databases with Noise". In *Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining (KDD'98), New York, NY*, pages 58–65, 1998.

[HK99] A. Hinneburg and D. A. Keim. "Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering". In *Proc. 25th Int. Conf. on Very Large Databases (VLDB'99), Edinburgh, UK*, 1999.

[HK00] J. Han and M. Kamber. *"Data Mining: Concepts and Techniques"*. Morgan Kaufman, San Francisco, CA, 2000.

[HSE+95] J. Hafner, H.S. Sawhney, W. Equitz, M. Flickner, and W. Niblack. "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 Int. Conf. on Management of Data (SIGMOD'91), Denver, CO*, pages 208–217, 1991.

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

[JD88] A. K. Jain and R. C. Dubes. *"Algorithms for Clustering Data"*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[JKP04] E. Januzaj, H.-P. Kriegel, and M. Pfeifle. "DBDC: Density Based Distributed Clustering". In *Proc. 9th Int. Conf. on Extending Database Technology (EDBT 2004), Heraklion, Greece*, pages 88–105, 2004.

[Jol86] I. T. Joliffe. *"Principal Component Analysis"*. Springer-Verlag, New York, 1986.

[JWZ94] T. Jiang, L. Wang, and K. Zhang. "Alignment of Trees - An Alternative to Tree Edit". *Proc. Int. Conf. on Combinatorial Pattern Matching (CPM'94), Asilomar, CA*, pages 75–86, 1994.

[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 Int. Conf. on Management of Data (SIGMOD'03), San Diego, CA*, pages 587–598, 2003.

[KCMP01] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. "Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'01), Santa Barbara, CA*, pages 151–162, 2001.

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

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

[KKK04] K. Kailing, H.-P. Kriegel, and P. Kröger. "Density-Connected Subspace Clustering for High-Dimensional Data". In *Proc. SIAM Int. Conf. on Data Mining (SDM'04), Lake Buena Vista, FL*, pages 246–257, 2004.

[KKKW03] K. Kailing, H.-P. Kriegel, P. Kröger, and S. Wanka. "Ranking Interesting Subspaces for Clustering High Dimensional Data". In *Proc. 7th European Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD'03), Cavtat-Dubrovnik, Croatia*, pages 241–252, 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 Int. Conf. on Database Systems for Advanced Applications (DASFAA'03), Kyoto, Japan*, pages 27–36, 2003.

[KKPS04a] K. Kailing, H.-P. Kriegel, M. Pfeifle, and S. Schönauer. "Efficient Indexing of Complex Objects for Density-based Clustering". In *to appear in Fifth Int. Workshop on Multimedia Data Mining (MDM/KDD'04), Seattle, WA*, 2004.

[KKPS04b] K. Kailing, H.-P. Kriegel, A. Pryakhin, and M. Schubert. "Clustering Multi-Represented Objects with Noise". In *Proc. 8th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'04), Sydney, Australia*, pages 394–403, 2004.

[KKS98] G. Kastenmüller, H.-P. Kriegel, and T. Seidl. "Similarity Search in 3D Protein Databases". In *Proc. 6th German Conf. on Bioinformatics (GCB'98), Cologne, Germany*, 1998.

[KKS04] K. Kailing, H.-P. Kriegel, and S. Schönauer. "Content-Based Image Retrieval Using Multiple Representations ". In *to appear in Proc. 8th Int. Conf. on Knowledge-Based Intelligent Information and Engineering Systems (KES'04), Wellington, New Zealand*, 2004.

[KKSS04a] K. Kailing, H.-P. Kriegel, S. Schönauer, and T. Seidl. "Efficient Similarity Search for Hierachical Data in Large Databases". In *Proc. 9th Int. Conf. on Extending Database Technology (EDBT 2004), Heraklion, Greece*, pages 676–693, 2004.

[KKSS04b] K. Kailing, H.-P. Kriegel, S. Schönauer, and T. Seidl. "Efficient Similarity Search in Large Databases of Tree Structured Objects (Poster)". In *Proc. 20th Int. Conf. on Data Engineering (ICDE'04), Boston, MA*, page 835, 2004.

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

[KR90] L. Kaufman and P. J. Rousseeuw. *"Finding Groups in Data: An Introduction to Cluster Analysis"*. John Wiley & Sons, 1990.

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

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

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

[Lev66] V. Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals". *Soviet Physics-Doklady*, 10:707–710, 1966.

[LNRvK+02] B. Liebl, U. Nennstiel-Ratzel, R. von Kries, R. Fingerhut, B. Olgemöller, A. Zapf, and A. A. Roscher. "Very High Compliance in an Expanded MS-MS-Based Newborn Screening Program Despite Written Parental Consent". *Preventive Medicine*, 34(2):127–131, 2002.

[LW03] J. Liu and W. Wang. "OP-Cluster: Clustering by Tendency in High Dimensional Spaces". In *Proc. of the 3rd IEEE Int. Conf. on Data Mining (ICDM'03), Melbourne, FL*, pages 187–194, 2003.

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

[MKL97] B.M. Mehtre, M.S. Kankanhalli, and W.F. Lee. "Shape Measures for Content Based Image Retrieval: A Comparison". *Information Processing and Management*, 33(3):319–337, 1997.

[Mun57] J. Munkres. "Algorithms for the Assignment and Transportation Problems". *Journal of the SIAM*, 5(1):32–38, 1957.

[NGC01] H.S. Nagesh, S. Goil, and A. Choudhary. "Adaptive Grids for Clustering Massive Data Sets". In *1st SIAM Int. Conf. on Data Mining (SDM'01), Chicago, IL*, 2001.

[NH94] R. T. Ng and J. Han. "Efficient and Effective Clustering Methods for Spatial Data Mining". In *Proc. 20th Int. Conf. on Very Large Databases (VLDB'94), Santiago, Chile*, pages 144–155, 1994.

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

[PJAM02] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. M. Murali. "A Monte Carlo Algorithm for Fast Projective Clustering". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'02), Madison, WI*, pages 418–427, 2002.

[PM99] A. Papadopoulos and Y. Manolopoulos. "Structure-based Similarity Search with Graph Histograms". In *Proc. DEXA/IWOSS Int. Workshop on Similarity Search, Florence, Italy*, pages 174–178, 1999.

[PZC+03] J. Pei, X. Zhang, M. Cho, H. Wang, and P. S. Yu. "MaPle: A Fast Algorithm for Maximal Pattern-based Clustering". In *Proc. of the 3rd IEEE Int. Conf. on Data Mining (ICDM'03), Melbourne, FL*, pages 259–266, 2003.

[Sac] Saccharomyces Genome Database (SGD). http://www.yeastgenome.org/. (visited: Oktober/November 2003).

[Sal89] G. Salton. *"Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer"*. Addison-Wesley, Boston, MA, 1989.

[SCZ98] G. Sheikholeslami, S. Chatterjee, and A. Zhang. "WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases". In *Proc. 24th Int. Conf. on Very Large Databases (VLDB'98), New York, NY*, pages 428–439, 1998.

[SEKX98] J. Sander, M. Ester, H.-P. Kriegel, and X. Xu. "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and its Applications". *Data Mining and Knowledge Discovery, An International Journal*, 2(2):169–194, 1998.

[Sel77] S. Selkow. "The Tree-to-Tree Editing Problem". *Information Processing Letters*, 6(6):184–186, 1977.

[Sib73] R. Sibson. "SLINK: An Optimally Efficient Algorithm for the Single-Link Cluster Method". *The Computer Journal*, 16(1):30–34, 1973.

[SK97] T. Seidel and H.P. Kriegel. "Efficient User-Adaptable Similarity Search in Large Multimedia Databases". In *Proc. 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece*, pages 506–515, 1997.

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

[SKK01] T. B. Sebastian, P. N. Klein, and B. B. Kimia. "Recognition of Shapes by Editing Shock Graphs". In *Proc. 8th Int. Conf. on Computer Vision (ICCV'01), Vancouver, Canada*, volume 1, pages 755–762, 2001.

[SSZ+98] P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P Brown, D. Botstein, and B. Futcher. "Comprehensive Identification of Cell Cycle-Regulated Genes of the

Yeast Saccharomyces Cerevisiae by Microarray Hybridization.". *Molecular Biolology of the Cell*, 9:3273–3297, 1998.

[STTF02] E. P. M. d. Sousa, C. Traina, A. J. M. Traina, and C. Faloutsos. "How to Use Fractal Dimension to Find Correlations between Attributes". In *Proc. KDD-Workshop on Fractals and Self-similarity in Data Mining: Issues and Approaches, Edmonton, Canada*, pages 26–30, 2002.

[SWS+00] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. "Content-Based Image Retrieval at the End of the Early Years". *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 22(12):1349–1380, 2000.

[THC+99] S. Tavazoie, J. D. Hughes, M. J. Campbell, R. J. Cho, and Church G. M. "Systematic Determination of Genetic Network Architecture". *Nature Genetics*, 22:281–285, 1999.

[TTSF00] C. Jr. Traina, A. Traina, B. Seeger, and C. Faloutsos. "Slim-Trees: High Performance Metric Trees Minimizing Overlap between Nodes". In *Proc. 7th Int. Conf. on Extending Database Technology (EDBT'00), Konstanz, Germany*, pages 51–65, 2000.

[TVJD95] H.D. Tagare, F.M. Vos, C.C. Jaffe, and J.S. Duncan. "Arrangement - A Spatial Relation between Parts for Evaluating Similarity of Tomographic Section". *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 17(9):880–893, 1995.

[WF74] R. A. Wagner and M. J. Fisher. "The String-to-String Correction Problem". *Journal of the ACM*, 21(1):168–173, 1974.

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

[WFP03] N. Weidmann, E. Frank, and B. Pfahringer. "A Two-Level Learning Method for Generalized Multi-instance Problems". In *Proc. 14th European Conferenc on Machine Learning (ECML'03), Cavtat-Dubrovnik, Croatia*, pages 468–479, 2003.

[WWYY02] H. Wang, W. Wang, J. Yang, and P. S. Yu. "Clustering by Pattern Similarity in Large Data Set". In *Proc. ACM*

SIGMOD Int. Conf. on Management of Data (SIGMOD'02), Madison, WI, pages 394–405, 2002.

[WYM97] W. Wang, J. Yang, and R. Muntz. "STING: A Statistical Information Grid Approach to Spatial Data Mining". In *Proc. 23rd Int. Conf. on Very Large Databases (VLDB'97), Athens, Greece*, pages 186–195, 1997.

[WZC+03] J. Wang, H. Zeng, Z. Chen, H. Lu, L. Tao, and W. Ma. "ReCoM: reinforcement clustering of multi-type interrelated data objects". In *Proc. 26th Int. Conf. on Research and Development in Information Retrieval (SIGIR'03), Toronto, Canada*, pages 274–281, 2003.

[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.

[WZJS94] J. Wang, K. Zhang, K. Jeong, and D. Shasha. "A System for Approximate Tree Matching". *IEEE Transactions on Knowledge and Data Engineering*, 6(4):559–571, 1994.

[XEKS98] X. Xu, M. Ester, H.-P. Kriegel, and J. Sander. "A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases". In *Proc. 14th Int. Conf. on Data Engineering (ICDE'98), Orlando, FL*, pages 324–331, 1998.

[YWWY02] J. Yang, W. Wang, H. Wang, and P. S. Yu. "Delta-Clusters: Capturing Subspace Correlation in a Large Data Set". In *Proc. 18th Int. Conf. on Data Engineering (ICDE'02), San Jose, CA*, pages 517–528, 2002.

[ZCM02] H. Zeng, Z. Chen, and W. Ma. "A Unified Framework for Clustering Heterogeneous Web Objects". In *Proc. 3rd Int. Conf. on Web Information Systems Engineering (WISE'02), Singapore*, pages 161–172, 2002.

[Zha96] K. Zhang. "A Constrained Editing Distance Between Unordered Labeled Trees". *Algorithmica*, 15(6):205–222, 1996.

[ZRM96] T. Zhang, R. Ramakrishnan, and Livny M. "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'96), Montreal, Canada*, pages 103–114, 1996.

[ZSS92] K. Zhang, R. Statman, and D. Shasha. "On the editing distance between unordered labeled trees". *Information Processing Letters*, 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*, 7(1):43–57, 1996.

Curriculum Vitae

Karin Kailing was born on August 12, 1973 in Pfaffenhofen/Ilm, Germany. She attended primary school and high school in Pfaffenhofen where she received her high school degree in 1993.

From September 1993 to September 1995 she was trained as industrial clerk at the Siemens AG in Munich.

She entered the *Ludwig-Maximilians-Universität (LMU)* in November 1995, studying Computer Science with a minor in Psychology. In May 2001 she passed the final examination with distinction and received her diploma degree. During this time, she worked part-time for the OSRAM GmbH. Her diploma thesis on the topic *'Analyse und Entwurf einer Software-Lösung zur aktiven Zeitwirtschaft mit SAP'*(Analysis and Design of a Software Solution for Positive Time Management in SAP R/3) was supervised by Professor Rolf Hennicker and Assessor Joachim Potthast, OSRAM GmbH.

In Mai 2001, Karin Kailing joined the research group of Professor Hans-Peter Kriegel at the *LMU* where she works as a research and teaching assistant. Her research interests include knowledge discovery and similarity search in large standard and multimedia databases.