Density-Based Clustering in large Databases using Projections and Visualizations

Dissertation

zur Erlangung des akademischen Grades

Dr. rer. nat.

vorgelegt der

Mathematisch-Naturwissenschaftlich-Technischen Fakultät
(mathematisch-naturwissenschaftlicher Bereich)
der Martin-Luther-Universität Halle-Wittenberg

verteidigt am 19.12.2002

von Alexander Hinneburg
geb. am: 27.02.1975 in Halle (Saale)

Gutachter:
1. Prof. Dr. Daniel A. Keim (Konstanz)
2. Prof. Dr. Heikki Mannila (Helsinki)
3. Prof. Dr. Stefan Posch (Halle)

Halle (Saale), den 19.2.2003

urn:nbn:de:gbv:3-000004638
[http://nbn-resolving.de/urn/resolver.pl?urn=nbn%3Ade%3Agbv%3A3-000004638]

Lebenslauf

Persönliche Daten

Name	Alexander Hinneburg
geboren am	27.02.1975
in	Halle (Saale)
Staatsangehörigkeit	deutsch
Familienstand	verheiratet

Schulausbildung

1981-1982	Polytechnische Oberschule in Merseburg
1982-1989	Polytechnische Oberschule in Bad Schmiedeberg
1982-1990	Polytechnische Oberschule in Pretzsch
1990-1993	Dom-Gymnasium Merseburg, Abschluß Abitur

Universitätsausbildung

1993 – 1997	Studium der Informatik (Nebenfach Mathematik) an der Martin-Luther-Universität Halle-Wittenberg, Abschluß Dipl.-Informatiker
Februar 1998	Forschungsaufenthalt an der RWTH Aachen
November 1997 – Mai 1998	Forschungspraktikum am FB Biochemie, MLU Halle, AG "'Molecular Modelling"'
seit Mai 1998	wissenschaftlicher Mitarbeiter an der Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik

Alexander Hinneburg

Erklärung

Hiermit erkläre ich, dass ich diese Arbeit selbständig und ohne fremde Hilfe verfasst habe. Ich habe keine anderen als die von mir angegebenen Quellen und Hilfsmittel benutzt. Die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen sind als solche kenntlich gemacht worden. Ich habe mich bisher nicht um den Doktorgrad beworben.

Alexander Hinneburg

Acknowledgments

At this point I want to express my thanks to all the people who supported me during the last years while I have been working on my thesis.

I am most grateful to my advisor Professor Dr. D. A. Keim for his excellent scientific guidance over the years, which made this work possible. I would like to thank him for the fruitful discussions and his continued support. Many results in reported here present joint efforts with him.

I also would like to thank Professor Dr. H. Mannila and Professor Dr. S. Posch for their useful suggestions and their willingness to serve as co-referees.

This work has been carried out at the Institute for Computer Science at the Martin-Luther University of Halle-Wittenberg. I am grateful to my colleagues for creating a pleasant working atmosphere. Special thanks go to Dr. A. Herrmann for her continued moral support, to Michael Schaarschmidt for the lively discussions, to Markus Wawryniuk who read large parts of the thesis and was alway ready for discussions. I am also thankful to Dirk Habich, Thomas Lange and Marcel Karnstedt for implementing the current version of *HD-Eye* and the libraries of the separator framework and for their willingness and promptness in integrating new ideas.

Finally, I would like to express my gratefulness to my wife Iris for her love, patience and support during the time of writing this work.

Halle, August 2002

Alexander Hinneburg

Contents

Chapter 1

Introduction

The amount of data stored in databases grows with an increasing rapid pace. As the capacity of human analysts is limited, there is a strong need for automated and semi-automated methods to extract useful knowledge from large data sets. Data mining attempts to find unexpected, useful patterns in large data bases. Due to the new nature of the problems in this context contributions from many related research fields have been proposed in data mining. Despite of innovative solutions, problems arise from the different backgrounds of the contributors, which make the comparison of the proposed algorithms highly non-trivial. So, today's data mining systems often consist of a collection of different algorithms, however, without proper guidances, which algorithm is best used in a given application context. Also the differences between the results of the algorithms are not fully understood and characterized.

In the following paragraphs we give a short overview on the current situation. There are three basic methodologies used in data mining, namely association rule mining, classification and clustering.

Association rules are first defined on transactions, each consisting of several items. Such data often appear in the context of market basket analysis where each transaction is a set of items bought by a customer. After the introduction of association rules many other applications have been identified, where such data are produced. The goal is to find rules (consisting of premise and conclusion) between itemsets, for which exist a minimal number of examples in the database and which are true in a given minimal percentage of the stored cases fulfilling the premise.

Classification attempts to learn a mechanism from a given set of pre-classified examples which is able to correctly assign (non-classified) objects to a class from a given set of classes based on feature attributes describing the object. There are several methods known to do this task. A very popular method is to use decision trees. This method also delivers a set of interpretable rules describing the classification process. The ability to explain the results is a very important aspect in data mining, because this makes it possible to get deeper insights to the problem and so to find useful unknown knowledge.

Clustering, which is investigated in this work, groups objects into clusters based on the similarity between the objects, so that similar objects are in the same group and objects from different groups are dissimilar. Often the similarity is determined from features, describing the objects. Clustering is used for different purposes, e.g. finding of natural classes or as a data reduction method. Typical applications are customer segmentation, document or image categorization as well as class finding in scientific data.

As mentioned above, data mining is a strongly interdisciplinary research field, with many contributions from statistics, machine learning, databases and visualization. These different research fields came together to form the new research field data mining, because the problems in the context of rapidly growing amounts of information require integral, holistic approaches. In that way data mining can be seen as an united effort to handle knowledge extraction from very large data sources. There are many statements in the literature and in key note talks saying that the research and the problems in this context are not covered by a single research field of the mentioned contributors.

The basic techniques mentioned above have been studied for a long time in several of contributing research areas, however with different backgrounds and research goals. As the main problem of the early data mining research has been scaling algorithms to large databases, many different trade-offs between result quality and runtime have been proposed, which are motivated by the different research backgrounds. Beside the positive effects of this research interaction, as a result, many algorithms are known solving strongly related problems, but trading quality for runtime in very different ways, making it nearly impossible for the user to understand the differences.

For example database oriented researchers proposed density-based clustering algorithms supported by multidimensional indices, without making use of the well-established theory about density estimation from statistics [32]. For basically the same problem (finding arbitrary shaped cluster), solutions have been proposed from the machine learning community using analogies to neural signal processing in the brain [36] and from image processing people using wavelets, a technique, which is often used for image compression [101].

The disadvantages for data mining is the limited use of the algorithms. So the database oriented solution depends on the existence and the good performance of a multi-dimensional index, which is not guaranteed in a typical data mining scenario. Since the machine learning solution is an on-line method, no statements can be made about the state of convergence and whether all data points are taken into account. The wavelet solution is rather limited to two-dimensional data, because it treats the data like a two-dimensional image. The user has to be aware of all these conditions and their impacts to the result, when the appropriate algorithm has to be chosen.

One contribution of this work is the development of a new consistent framework for clustering and related problems (like outlier detection and noise filtering), which is based on sound statistical theory and allows to chose a reasonable compromise between quality and runtime within the framework. The advantage is that the overall setting does not change and one can focus on the scaling problem. The main contribution of our framework is the decoupling of density estimation and clustering scheme. While the choice of the density estimation method has to do with scaling, the selection of the appropriate clustering scheme is a semantic question depending on the application context. There are two improvements. The first improvement is of technical nature, because the new concept of decoupling density estimation and clustering scheme leads to new more scalable algorithms. Secondly, the usability of clustering is improved by the framework, because semantic decisions are separated from technical scaling problems.

Today, there is a large amount of data stored in traditional relational databases. This is also true for databases of complex 2D and 3D multimedia data such as image, CAD, geographic, and molecular biology data. It is obvious that relational databases can be seen as high-dimensional databases (the attributes correspond to the dimensions of the data set), but it is also true for multimedia data which - for an efficient retrieval - are usually transformed into high-dimensional feature vectors such as color histograms [44], shape descriptors [61, 81], Fourier vectors [107], and text descriptors [72]. In many of the applications mentioned above, the databases are very large and consist of millions of data objects with several tens to a few hundreds of dimensions.

As an insight from relying on the statistical theory of density estimation, we learned that clustering in high-dimensional feature space is very limited, because the huge volume of such spaces can not be sampled sufficiently with data points even when gigabytes of data are used. Scott and Härdle [47] gave an impressing example to illustrate this situation and then they asked:

> "Should we give up now that we know that smoothing[1] high dimensions is almost impossible unless we have billions of data points that we can't analyze effectively? No, we could still try to pursue the goal to extract the most interesting low dimensional feature."

We followed this advise and investigated the problem of finding clusters in low dimensional projected spaces. This problem is much more difficult than the traditional clustering problem, because the number of possible projections is very large. Note that standard dimensionality reduction techniques like principal component analysis can not be used for this problem, because these techniques

[1] In the statistical literature smoothing is often used as a synonym for density estimation.

attempt to find a single projection, which is applied to all data points and so to all clusters. The new aspect of projected clustering is, that each cluster may have its own relevant projection.

In the first part of chapter 4 we develop a new notion of similarity as well as a generalized definition for nearest neighbor search, which takes this aspect into account. The new contribution is that a metric is not used anymore for the whole feature space, but serves as similarity measure only in a region around the query point using only a subset of the attributes.

There are only a few publication available for projected clustering in the literature. We review shortly the existing algorithms and explain their problems. Our main contribution in this part is the development of a new projected clustering algorithm, which overcomes weighty drawbacks of the other ones. We apply our algorithm to several real data sets and show its usability in different application contexts.

The final chapter deals with the usability of the previously developed automated clustering methods. As in the application of clustering always some semantic decisions are involved, the user has to be enabled to understand the impact of her/his decisions. Visualizations can be very effective to support this understanding and to bridge the semantic gap between the user and the clustering system. The term 'semantic gap' describes a situation where it is difficult for the user to communicate her/his needs and expectations to a software system. The bridging of the sematic gap or semantic chasm is one of the most challenging tasks, which are faced by today's research in information retrieval, human computer interfaces and with growing importance also in data mining.

The main challenge for clustering and projected clustering is to find a meaningful definition of similarity. For high-dimensional data different alternatives are possible, which differ by their weighting of the used attributes. The finding of axes-parallel projections, which allow the meaningful separation of clusters, is a simplified variant of the problem. However, it is highly non-trivial for the user to communicate a definition of meaningful to the clustering system in a formal way. Here we are facing an instance of the problem of bridging the semantic gap.

We developed an system called *HD-Eye*, which integrates several (semi-) automated clustering method and interactive visualizations. With the help of the visualizations the user may select meaningful projections, directly specify partial cluster-descriptions, tune parameters according to her/his intention or understand the results of automated clustering procedures. The system is especially useful for data exploration tasks, because the visualizations can also partially show hidden structure in the data or may disclose unknown properties, which are not captured by more formal summaries. This stimulates the user to generate different hypothesis about the data and serves in that way the exploration process.

HD-Eye also supports the semantic decisions needed for projected clustering, by allowing the comparison of different models for the found clusters. This is an important aspect in projected clustering, since the choice of the projections are crucial for the relevance and meaningfulness of the results.

Many of the results presented in this work have been published before in different conference proceedings and journals. Most of the articles present joint work with Prof. Dr. Daniel Keim. In [50,53] we published an new clustering algorithm called *DENCLUE* based on density estimation. In [55] we investigated a first clustering algorithm using projection and in [52] we developed a new notion for projected nearest neighbor search. First results from our visual clustering system *HD-Eye* have been published in [51]. We gave also a tutorial about clustering at several conferences [54, 56, 67], which builds the basis for the related work chapter. All materials presented in this dissertation is original work except the basic definitions on density estimation and the WARPing methods presented in chapter 3.

There is also ongoing development of software packages which implements the *HD-Eye* system and the ideas of the introduced separator framework for clustering. The software packages are not described in the thesis. A first prototype of *HD-Eye* was demonstrated at the SIGMOD'02 conference [57].

Chapter 2

Related Work

2.1 Common Concepts

Clustering has been successfully applied in many areas, for instance statistical description of biological phenomena, use in social investigation, psychology, statistical interpretation of business data and many more. The general purpose for doing clustering is to group similar objects together, so that unsimilar objects are in different groups and to build by this procedure an abstract but useful model of the part of the real world, which is investigated in the application. Second order purposes can be data reduction, discretization of continuous attributes, outlier finding, detection and description of natural classes and noise filtering.

The first purpose has to do with knowledge discovery, the second is more technical and deals with statistics, machine learning and data mining. Each approach of the related work described in the next sections is more or less dedicated to some of the second order purposes. Each specific approach serves in the context as a tool to reach the general goal of knowledge discovery and mostly defines implicitly the apriori blurred terms 'similar' and 'useful'. The translation of these terms into the application context is the most challenging task for successful clustering.

Clustering generally requires two preconditions: a set of objects, which should be clustered and a similarity function. There are two main ways to meet these requirements. The first is to provide the objects as a set of abstract symbols and the distance function as a distance matrix, which stores the distances or similarities of an object to each other. This approach avoids many difficulties of the second, because no transformation of the data is needed. But in case of a large object set this approach is prohibitive, because of the quadratic growth of the computational costs caused by the distance matrix. The second approach translates the meaning of the objects into vectors of fixed lengths, which can be seen as an enumeration of attributes. Then a mathematical distance function over the vector space is chosen to serve as a distances or similarity function for the objects. Using properties of the vector space, algorithms has been developed, which show a subquadratic behavior [1]. However, in contrast to the first approach it remains largely unchecked whether the distances or similarities resulting from the translation into the feature vector space and the chosen distance are useful for all combinations of objects. The work involved in the translation is called data preparation and is often separated from the clustering step. Since the application context in this work implies large objects sets, only algorithms based on the second approach are explored. As a further restriction the attributes have to be of numeric type, which can be ordered.

All clustering algorithms for this type of data estimate the probability density in the vector space. The estimated density generally depends on input parameters from the user. All clustering techniques use the density information to build groups of data points. There are two extreme possibilities to group the data points, which are useless from the point of knowledge discovery. First, the grouping of the all data points into one cluster does not reveal a new contribution. But also taking each object as a cluster of its own is not an gain of information. So any algorithm tries

[1] Often particular assumptions about the data distribution or the result are necessary for this.

to find a clustering between the both extremes by examining the data distribution according to the chosen technical approach and the parameter setting. However there is no general criterion of how to choose the best approach and the best parameter setting for an application, because of the a-priori unknown translation of 'similarity' and 'useful' from the objects and their application context into the feature vector representation and the used cluster paradigm.

2.2 Classification of Methods

In this section approaches proposed in the literature are classified and described according to the used method of density estimation and the clustering paradigm. So far as possible the interconnection between the approaches is extracted but also historical aspects are considered. The classes are *model-based*, *linkage-based* and *density-based* approaches.

2.2.1 Model-Based Approaches

The approaches in this class are called model-based, because the used algorithms adopt a fixed model to a given data set. Since the model is in general smaller then the data set these algorithms are often used for data compression. In the statistical literature the methods of this class are also called parametric, because parameters of a model are adopted, in opposite to non-parametric methods, which construct a result and return it as a not predefined model. Model-based algorithms can be formulated as an optimization problem. The idea behind is, that the parameters should be optimally estimated according to an optimization function and the used model. Note that optimally estimated cluster centroids are not necessarily meaningful within an application context. E.q. when the clusters have arbitrary shapes or different average densities and sizes centroid based approach are likely to split clusters.

An early published, but still relevant algorithm is k-**means** also referred as **LBG** [75, 76]. The aim is to find for a given finite data set $D \subset F$ positions of a set of k centroids $P = \{p_1, \ldots, p_k\}$ in a feature space $F \subset \mathbb{R}^d$, which minimize the quantization error. Given a data set $D = \{x_1, \ldots, x_n\} \subset F$, a set of reference vectors (centroids) $P = \{p_1, \ldots, p_k\} \subset F$, a distance function $x, y \in F, d(x, y) \rightarrow \mathbb{R} = \sum_{i=1}^{d}(x_i - y_i)^2$ and an index function $I(x) = min\big\{i : \forall j \in \{1, \ldots, k\} d(p_i, x) \leq d(p_j, x)\big\}$ the quantization error is

$$E[D, P] = \frac{1}{\#D} \sum_{x \in D} min\big\{d(p, x), p \in P\big\} = \frac{1}{\#D} \sum_{x \in D} d(x, p_{I(x)})$$

The LBG algorithm finds a local optimum with an iterative procedure. LBG initializes the reference vectors at randomly chosen data points. In an iteration step it calculates for each reference vector the set $R_c = \{x : x \in D, I(x) = c\}$ of data points which have the centroid p_c as the nearest centroid. Each reference vector p_c is moved to the mean of R_c. Since this movement of the reference vectors may cause a change of the sets R_c the iteration is done again. This step, also called *Lloyd Iteration*, produces a lower or equal quantization error [41]. The iteration stops if the centroids do not change any more. The result of LBG can be seen as a centroidal *Voronoi* partitioning of the data space F, where the mean of each Voronoi cell is at the position of the centroid. There are different interpretations of the result, namely the direct use of the centroids as cluster centers and alternatively vector quantization as a data reduction method not as a clustering method. In case of the first interpretation the clusters have to have round and compact shape and nearly equal density. The data compression works best, if no uniformly distributed points are in the data. The run time complexity is in $O(n \cdot d \cdot k)$.

A drawback of LBG is to assume the data can be averaged to calculate a mean. There are cases, where the mean of a set of data points is not defined or meaningless. An alternative to LBG is **CLARANS** [82], which uses medoids instead of centroids. Medoids are special data points, which are selected as representatives. The optimization function is the same as for LBG, but instead of the *Lloy Iteration* CLARANS uses a bounded search heuristic to approximate the gradient. The

authors introduced the parameters *num_local* which is the number of iterations and *max_neighbor* which is the number of tests per iteration to replace one medoid by a better data point. The search space is formalized as a graph. Each node corresponds to a configuration of the medoids and has $n \cdot k$ edges. The number of nodes is $\frac{n!}{k!(n-k)!}$. Due to the heuristic search and the very large optimization the algorithms can not guarantee to reach a local optimum. So the algorithm is well suited for low dimensional, small to medium sized data sets.

The LBG or CLARANS algorithms are increasingly sensitive to the initialization when the data space becomes sparse. The problem of initialization has been studied in [26, 27]. The authors use different heuristics to find a good initialization and to avoid poor local minimums.

The algorithms LBG-U [37] and k-harmonic means [22, 113] use different strategies to overcome the initialization problem. The authors of **k-harmonic means** use a different optimization function to minimize the quantization error, namely the harmonic mean:

$$OPT[D,P] = \frac{1}{\#D} \sum_{x \in D} ha\big\{d(p,x) : p \in P\big\} = \frac{1}{\#D} \sum_{x \in D} \frac{k}{\sum_{p \in P} 1/d(x,p)^q}$$

This bases on the observation that the harmonic mean, which is defined as $ha\{a_1, \ldots, a_k\} = k/\sum_{i=1}^{k} \frac{1}{a_i}$, behaves more like a min function than an average function. The parameter q is used for a weighting function. The derived update formula for a centroid uses not only information about the nearest points but also about the position of the other centroids $p_j \in P$:

$$p_j = \sum_{i=1}^{k} \frac{1}{d(x_i,p_j)^{q+2}\big(\sum_{l=1}^{k} \frac{1}{d(x_i,p_l)^q}\big)^2} \cdot x_i \Big/ \sum_{i=1}^{k} \frac{1}{d(x_i,p_j)^{q+2}\big(\sum_{l=1}^{k} \frac{1}{d(x_i,p_l)^q}\big)^2}$$

The authors show empirically for 2-dimensional data sets that k-harmonic means finds independently from the initialization a better local optimum than LBG.

LBG-U overcomes a poor local optimum by using non-local jumps, which moves not well utilized centroids to better positions. Therefore the author introduces the utility of a prototype $U(p) = E[D, P\backslash\{p\}] - E[D,P]$ as a measure how useful is the actual position of the centroid p for the data approximation. After the termination of the LBG algorithm the centroid p with the minimal utility is moved near to the centroid p', which causes the largest quantization error. Then LBG is invoked again. The iteration stops when the quantization error is not improved any more. The introduction of non-local jumps makes LBG-U independent from the initialization and the algorithms finds a better local optimum than LBG.

The other research branch where model-based cluster algorithms has been developed is machine learing. The methods are here also referred as *unsupervised learning*. The focus is the storage and compression the history of presented data in a model. In contrast to the statistical algorithms, which scan the whole database to derive the update information (batch mode), machine learning algorithms update the model immediately after a single data object has been processed. This mode is called online mode and models the behavior of organisms in a more natural way. In the recent literature online-algorithms are studied in the context of data streaming, as the immediate processing of data fit the streaming model.

A well established technique are ***Kohonen*-maps** [70, 89], also known as self organizing maps (SOM). A map consists of k centroids, which are linked by edges according to a fixed topology, like a 2D grid. The map is initialized randomly and trained for a data set by repetitively picking a randomly chosen data point $x \in D$, determining the nearest centroid $p_{I(x)}$, adopting the centroid $p_{I(x)} = \alpha \cdot (x - p_{I(x)})$ towards the picked data point according to a learning rate α as well as the topological neighbors of $p_{I(x)}$ with a learning rate $\beta \leq \alpha$. To achieve convergence the learning rates α and β decrease with the number of processed data points.

The **neural gas** algorithm [78] is an online learning algorithm without a fixed topology. In contrast to other methods the topology is not represented by edges between the nodes. The training algorithm starts with a fixed number of k centroids $P = \{p_1, \ldots, p_k\}$ which are randomly initialized. In each training iteration a data point $x \in D$ is randomly chosen. Then all centroids are placed in ascending order according to the distance to the picked data point x. These order

is represented as an index sequence $(i_0, i_1, \ldots, i_{k-1})$ starting with the closed centroid p_{i_0} to the farthest one p_{i_k}. All centroids are updated according to the following formula:

$$\Delta p_i = \epsilon \cdot h_\lambda(l) \cdot (x - p_i), \text{ with } 0 \leq l \leq k-1 \text{ and } p_i = p_{i_l};\ h_\lambda(l) = e^{-l/\lambda}$$

The ϵ and λ are time-depended functions and decrease with the iteration counter t, $0 \leq t \leq t_{max}$ according to $\lambda(t) = \lambda_i(\lambda_f/\lambda_i)^{t/t_{max}}$ and $\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{max}}$. The parameters used in the simulation in [78] are $\lambda_i = 10,\ \lambda_f = 0.01,\ \epsilon_i = 0.5,\ \epsilon_f = 0.005,\ t_{max} = 40000$. The iteration stops when the iteration counter t reaches t_{max}. Since the map has no predefined topology it can adopt arbitrary distributions. However, the plasticity (number of centroids) and the parameters for the learning rates have to be specified for the algorithm.

A method to find a topology represented by edges for a given set of centroids is **Competitive Hebbian Learning** [77, 79]. The desired topology graph connects neighbored centroids by edges. In the general case this is a Delaunay graph, which is very costly to derive for high dimensional data. Competitive Hebbian learning derives an induced Delaunay Graph by masking the original Delaunay triangulation with a data distribution. In the induced Delaunay graph centroids are connected, if the maximum of probability density at the common border of the both corresponding Voronoi faces is greater than zero. The algorithm takes an initialized set of centroids $P = \{p_1, \ldots, p_k\}$ and a set of data points $D \subset \mathbb{R}^d$. The induced Delaunay graph $G = (V, E)$ is initialized as undirected graph with $V = P$ and $E = \emptyset$. Then it picks randomly data points from D until a maximum number t_{max} is reached. For each data point the nearest and the second nearest centroid p_1, p_2 is determined. If there is no edge between p_1 and p_2 in E an new edge is inserted $E = E \cup \{(p_1, p_2)\}$. Martinetz and Schulten [79] proved that the resulting induced Delaunay graph is a subgraph of the original Delaunay graph for the given set of centroids P. The method works for all vector quantization techniques.

Fritzke proposed in [36] a algorithm called **growing neural gas**, which combines neural gas and Hebbian learning with a growing strategy. The method starts with two centroids connected by an edge. Similar to the Kohonen map the training algorithm picks randomly a data point x, determine the nearest and second nearest centroid p_1, p_2. If both centroids are connected by an edge, the age of the edge is set to zero else a new edge with zero age is inserted. Then p_1 and the topological neighbors are moved towards x according to a constant learning rate. In the reminder of the iteration step all age counter of the edges outgoing from p_1 (the nearest centroid) are incremented and edges older than a_{max} are deleted. In case of isolated centroids result from the deletion, these centroids will be deleted as well. The growing strategy bases on the approximation error, which is associated to each centroid and incrementally derived during the iteration. The approximation error of nearest centroid p_1 is updated with $\delta E_1 = d(p_1, x)^2$. An new centroid is inserted every lth iteration using the following procedure. The centroid p_q with the largest approximation error E_q and the neighboring centroid p_f of p_q with largest error are determined. The new centroid $p_r = 0.5(p_q + p_f)$ gets the position between the selected centroids p_q and p_f. The approximation errors of p_q, p_f are reduced by $\delta E_q = -\alpha E_q$ resp. $\delta E_f = -\alpha E_f$ with $0 < \alpha < 1$ and the approximation error of the new centroid p_r is $E_r = 0.5(E_q + E_f)$. At last the approximation error of all prototypes is decreased by $\delta E_c = -\beta E_c$. Fritzke used for the experiments in his publications the following parameter settings: $l = 200,\ \alpha = 0.5,\ a_{max} = 88,\ \beta = 0.0005$. As stop criterion can be used the net size or the average approximation error. In contrast to the combination of neural gas and Hebbian learning the advantage of growing neural gas is that the net size has not to be predefined and all intermediate training states are valid approximations of the data distribution.

2.2.2 Linkage-Based Approaches

The next class of clustering methods are linkage-based approaches. There are a number of different hierarchical linkage methods known, for example single, complete and average linkage. In this section we describe extensions of the centroid and the single-linkage approach, which have become a focus of recent data mining research. Centroid and single-linkage generate a hierarchy

of clusters (also called dendrogram). Both start in the agglomerative case with the data points as clusters and merges in each iteration step the two clusters with the smallest distance. The iteration runs until only one cluster remains. The distance between two clusters A and B, which are sets of data points, is the distance between $mean(A)$ and $mean(B)$ in case of centroid linkage, the minium/maximum distance between a point from A and a point from B for single/complete linkage. The computation of a complete hierarchy is very costly for large data sets because in each iteration all pairwise distances between the current clusters have to be computed. A way to circumvent the high complexity is to determine only a part of the hierarchy.

BIRCH [115] determines the upper part of the hierarchy using a variant of centroid linkage which depends on the order of the processed data points. For this a R-tree like structure called CF-tree is build, which stores summaries about all data points seen so far. In contrast to a R-tree the CF-tree represents the data points by centroids with variance. The entries of a CF-tree are additive that means nodes on higher levels are the sum of the sons. BIRCH stores only the upper part of the complete hierarchy, which has a size lower than a given threshold M. The size of the tree is controlled by the variance of the leafs, which must not exceed a threshold $t \geq 0$. In case the tree becomes too large the variance threshold is increased so that the leafs can absorb more data. The new tree is rebuilt by inserting the leafs of the old tree. Since the allowed variance in the leafs is larger the new tree is smaller than the previous one. The output of BIRCH are the centroids of the leaf level, which can be used like any other result from a vector quantization method. In contrast to k-means the number of centroids depends on the size of the given memory, the order and the distribution of the data. BIRCH needs only one scan over the data to build the CF-tree.

In case of single-linkage clustering the smallest part of an hierarchy is a horizontal cut, which corresponds to a minimum linkage distance $\epsilon > 0$ and a partitioning $C = \{C_1, \ldots, C_m\}$ of the data set D. Such a flat single-linkage clustering fulfills three main conditions:

1. Non-Emptiness: $C_i \neq \emptyset$ for $i = 1, \ldots, m$
2. Connectivity: $x \in C_i$ and $dist(x, y) \leq \epsilon \Rightarrow y \in C_i$
3. Maximality: There is no $i, j \in \{1, \ldots, m\}, i \neq j$ that $(C_i \cup C_j)$ fulfills the first both conditions.

A simple algorithm, which can directly determine such a clustering (not in an agglomerative way), takes each data point as a node of a graph and inserts an edge between two data points if the distance between them is smaller than ϵ. The clusters C_i are the connected components of the graph. The clusters may have arbitrary shape in multi-dimensional spaces, since a cluster is represented by all points which belong to the cluster. This is the main difference to vector quantization methods, which represent a cluster only by one point and produce clusters of compact and nearly round shape. The run time of the flat single-linkage algorithm is in $O(n^2)$, because for the determination of the edges all pairwise distances have to be computed.

The past data mining research on this algorithm dealt with two main issues, namely the improvement of the effectivity and scaling of the algorithm to large data sets. At first, we review the research on improved effectivity. A disadvantage of single-linkage is, that two dense groups of points which are linked by a chain of close points are determined as one cluster by single-linkage. The undesired effect is also called chaining effect.

An simple method to reduce the chaining effect was published by Wishart [111]. He proposed a preprocessing step in which all data points, having less than $c \in \mathbb{N}$ other data points in their ϵ-neighborhood are removed. This heuristic removes points of the chain. The single-linkage algorithm is applied to the reduced data set. This idea has been taken up by Sander et al. who proposed the DBSCAN-algorithm [32], which defines a cluster in a very similar way like Wishart. The only difference to Wisharts method is, that also points within the neighborhood of a core point (a data point with more than $MinPts \in \mathbb{N}$ data points in the ϵ-neighborhood) belong to a cluster. The main contribution of the DBSCAN approach deals with the efficiency issue which is discussed later.

Since it is difficult to tune the parameters ϵ and c (or $MinPts$ in the publications of Sander et al.) Xu et al. [112] proposed an algorithm which has no parameter but assumes the points are uniformly distributed within the clusters. Instead of DBSCAN's core point condition the authors

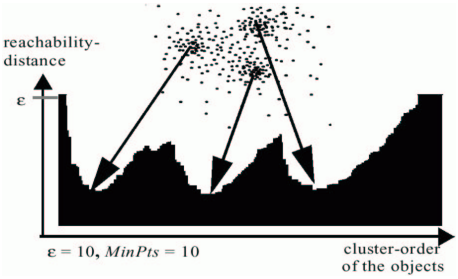


Figure 2.1: Example illusttration of the cluster-ordering found by OPTICS. In the figure a cluster corresponds to a valley. (Taken from [12], page 6)

require for the nearest neighbor distance of the points in a cluster to be normally distributed which is the case if the data points fill the region of the cluster uniformly. The condition is checked via the χ^2-test on the theoretical nearest neighbor distance distribution and the observed one.

Both extensions of single-linkage reduce the chaining effect by introducing a special condition which describes the border of a cluster. So the algorithms can be seen as single-linkage plus an additional stop criterion, which decides locally whether a point can be linked to a cluster or not.

In case of clusters with different densities the approach fails since the border of the clusters can not be uniquely described. To deal with different densities in the data the OPTICS approach [12] computes the lower part of the single-linkage hierarchy. OPTICS is an further development of DBSCAN so the core point condition to reduce the chain effect is also integrated. The lower part of a single-linkage hierarchy corresponds to linkage distances $0 \leq \epsilon \leq \epsilon_{max}$. The lower part is not a hierarchy tree, but consists of many small subtrees. OPTICS determines instead of the subtrees an ordering of the data points so that the points belonging to the same subtree appear in sequence in the ordering. To each point in the sequence a value reciprocal to the density (called reachability distance) is attached. So hierarchical clusters can be recognized as nested valleys in a graph with the point ordering on the x-axis and the reachability distance on the y-axis. The determination of the ordering is a single-linkage process which is guided by the density of the points to link. In contrast to standard single-linkage the graph scan is implemented as breadth first search using a priority queue instead of a simple one. The priority of data points (or nodes of the graph) correspond to their density measured by the k-nearest neighbor distance within an ϵ_{max}-neighborhood. So points with high density are processed earlier than low density point at the border of an cluster. The ordering produced is the processing order of the points. The priority queue guides the graph scan from any point of the cluster to the most dense region and from there it discovers the cluster in a manner of concentric circles until the border is reached. In case a path to another more dense area is found the points there are explored first. So regions of different densities can be handled. The output of OPTICS is usually displayed by a graph like in figure 2.1. Different levels of the partial hierarchy are extracted by cutting the graph with horizontal lines and taking all points in a valley below a line as one cluster. In that manner a partial dendrogram can be determined, which captures the cluster structure in the data.

In the last part of this subsection we want to review the work on making single-linkage and the variants more efficient. The standard partitioning algorithm has to check the distances between all pairs of points, whether the distance is below the linkage distance and might establish an edge between the points. This causes a quadratic run time of the standard algorithm. Sander et al. proposed to use a spatial index structure like R^*-tree or any other advanced spatial index structure supporting near neighbor queries. Using such index structures the ϵ-surrounding of a data point can be retrieved more efficiently. Assuming the index structure can deliver points in an ϵ-surrounding in $O(\log n)$ the run time of whole clustering comes down to $O(n \cdot \log n)$. The assumption is only true for low and medium dimensional data and small query regions. Studies on index structures have shown that the efficiency decreases with growing dimensionality and will be beaten by a linear scan for high dimensionality. Another scaling variant is to approximate single-linkage clusters. An approximation method for single-linkage clusters using a quad-tree like grid structure has been

proposed by Muntz et al. [109]. Another method [28] pre-compresses the data using BIRCH and does the single-linkage clustering on the compressed data.

2.2.3 Density-Based Approaches + KDE

Density-based clustering has been early studied in the field of applied statistics. The idea is to estimate the probability density function using observations which are given by multi-dimensional data points. In the second step the clusters are constructed using the density function. In early work [38, 94] the clusters are constructed using the gradient of the density function. The methods apply to each data point an iterative hill climbing procedure. A cluster is defined by a local maximum of the density function and consists of the data points, which converge to the maximum. The algorithms are designed for small data sets. The determination of the proposed density functions at one point in the data space requires distance calculations to each data point. Since in each iteration the density and the density gradient is determined for all data points the algorithms have a runtime in $O(n^2)$ (n is the number of data points).

Density estimation is a separate research area in the statistic community with many applications including cluster and regression analysis. There are a number of different techniques to construct a density estimate using data points, namely multi-dimensional histograms, averaged-shifted histograms [96], frequency polygons and kernel density estimation [98, 105, 108]. Kernel density estimation has become a widely studied framework and in which nearly all estimation methods can be formulated. The idea of kernel density estimation is to model the density as a sum of the influences of the data points. The influence of a data point is given by a kernel function, which is symmetric and has the maximum at the data point. Examples for kernel functions are Gaussian bell curve, square wave function or nearest-neighbor distance function. The density function takes higher values than the simple kernels of the points in regions, where some kernel functions have a significant overlap. Since in the general case the kernel function of an data point is in the whole data space above zero the determination of the density function which is the sum of kernels at different positions has no zero added, which could be excluded from the summation. So the complexity of a kernel density function for the determination of the density at a single point is in $O(n)$, which makes the function very costly for large applications. The WARPing-framework [47] proposes to pre-aggregate the data in advance and to use the aggregated points. A special case of WARPing using BIRCH as aggregation method has been published by Zhang et.al [116]. The complexity of the density function can be reduced from $O(n)$ down to $O(k)$, where k is the number of used centroids with $k \ll n$.

In the KDD literature other density-based approaches than DBSCAN [32] has been proposed. This methods makes use of histogram-based density estimation. The approach called grid-clustering by Schikuta [93] constructs a grid-file like data structure and scan connected grid cells according to the density in decreasing order. The output is a similar ordering like the one from OPTICS, but grid clustering starts directly in the center of a cluster.

The wavecluster approach [101] constructs a fine 2D histogram of the data (only 2D can be handled by this approach). In a second step the bin values are taken as grey signals of an image and transformed using wavelets to clean out noisy parts of the data. After the transformation the clusters are determined as connected regions with a bin value above zero. The wavelet approach enables also the finding of clusterings with different resolutions.

2.3 Projected Clustering

The problem of projected clustering is motivated by the observation that high dimensional data spaces are sparsely populated with data points despite the large size of the used data sets. This is due to the fact that the number of data points needed to fill a high dimensional space grows exponentially with the number of dimensions. There are different consequences of this fact. In Beyer et al. [21] is shown that nearest neighbor search becomes instable with growing dimensionality. The lemma shown has very general conditions met by many data sets. Another consequence formulated

by statisticians is that density estimation becomes insignificant in high dimensional spaces due to the lack of observations with respect to the volume of the space [105]. These consequences limit the effectivity of clustering in high dimensional spaces, since cluster heavily rely on these techniques. A way out is to find clusters in subspaces of the original high dimensional space [47]. There are some recent research activities in the KDD community which address the problem of finding clusters within subspaces.

Aggrawal et al. [7] proposed an apriori-like algorithm called CLIQUE which identifies subspaces with clusters. The algorithm bins the dimensions into several intervals and find in an apriori-like fashion dense combinations of intervals from different dimensions. The involved dimensions of the found combinations stretch a subspace which contains clusters. In the following clustering step all connected dense interval combinations are grouped together. Experiments have shown that the algorithm is sensitive to the initial binning. In case of a coarse binning many combinations are dense (or frequent) and so the candidate sets of apriori becomes very large. This results into a poor performance of CLIQUE. If the binning has fine granularity the algorithm produces dense combinations, which rarely consists of more than three or four dimensions. So the dimensionality of the found subspaces is low. The algorithm scales linear in the number of data point but quadratic or super-quadratic in the number of initial dimensions.

Other approaches by Aggarwal et al. [3, 4] follow the strategy to partition the data into initial clusters and reduce the dimensionality of these subsets to a given lower dimensionality. The dimensionality reduction strategies are integrated into a k-means like framework. During the reduction process clusters which become similar in the subspaces can be merged. In both approaches clusters are described by a center point and a vector set spanning the subspace. In the first approach the subspaces are restricted to be axes-parallel. The reduction process is guided by a variance criterion measuring the deviation of data points from the center in each dimension. Dimensions with a large deviation are greedily removed until a given minimal dimensionality is reached. The second approach allows arbitrary oriented subspaces. Here, the spanning vectors are determined by separate applications of principal component analysis to the initial cluster subsets. Since both approaches determine a partition of the data set, the algorithms can not detect whether a data point belongs to more than one projected clusters within different subspaces. A more detailed review of recent work of projected clustering is in section 4.2.

2.4 Outlier detection

The problem of finding outliers in large data sets is related to the clustering problem, because some clustering algorithms indirectly are able to find outliers. On a raw level, from the clustering perspective an outlier is an object, which can not be assigned to any of the found clusters with large confidence. In fact it might be useful to examine whether the concepts for clustering may serve for outlier detection.

The outlier detection problem has been extensively studied in the field of statistics. A generally accepted characterization of the problem is given by Hawkins: "an outlier is an observation that deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [48]. A large number of tests for outliers are known in the statistical literature [17]. All these tests are designed for univariate data and make assumptions about the underlying distribution. In the data mining literature four different definitions of outliers have been proposed, which do not make assumptions on the data distribution and can handle multivariate data.

Knorr and Ng introduce distance based outliers [68, 69]. The definition requires an appropriate chosen distance function, which measures the dissimilarity between two objects. An outlier is assumed to be dissimilar to more than p percent objects of the data set. An object is dissimilar to another one if the distance between both exceeds an given threshold D. The approach by Knorr and Ng identifies all the objects with less than $(1 - p)$ percent of objects in their D surrounding. The authors propose two algorithms, first a cell based algorithm which is linear in the number of data points but only scales up to four dimensions and second a nested-loop algorithm with a complexity of $O(d \cdot N^2)$ and a very low constant.

A variant of distance based outliers [87] uses the nearest neighbor distance to compute outliers. Here, the authors define the n points with the largest nearest neighbor distance to be outliers. The algorithms proposed in [87] uses index structures for low dimensional data and the nested loop join for high dimensional data.

The second approach called local outliers has been proposed by Breunig et al. [28, 29]. The authors developed a measure called LOF based on the reachability distance also used for the OPTICS clustering algorithm. The LOF measurement for a data point x is high when the reachability distance is much larger than the average reachability distance in the neighborhood of x. Data points with a high LOF value are reported as outliers. The concept of the LOF value allows to find outliers in regions of different average density. Such outliers can not be found by the previous approaches. In case of local outliers the determination of the reachability distance of an object and so the LOF-value requires a k-nearest neighbor query. For the efficient handling of these queries a multi-dimensional index structure like R-tree or X-tree has been used by the authors. The experiments show that this works well for low and medium-dimensional data sets. Since for high dimensional data sets the performance of the index structure becomes linear, the LOF-method by Breunig et al. converges to a quadratic runtime.

The work of Han et al. [62] improves the efficiency of the LOF-method using the following concepts. First, the authors say, it is not necessary to compute the LOF and so the nearest neighbor distance for all points of the data set, since only the n points with the largest LOF are outliers. The authors use the BIRCH algorithms to partition the data into clusters and determine for each partition an lower and upper bound for the nearest-neighbor distance and so for the LOF-measure. In that way they are able to compute candidate partitions which possibly contain the top-n local outliers. The other partitions are pruned. In the last step only the candidate partitions are used to compute the n local outliers with the largest LOF-value.

An interesting approach by Aggarwal and Yu [5] deals with outliers in high dimensional spaces and the inherent sparsity of such data spaces. The focus of their study is to find data points in low dimensional projections where the density of the points deviates significantly from the expected one. The approach partitions the dimensions into intervals. The intervals from different dimensions can be combined to grid cells in the subspace, which is spanned by the used dimensions. The expected density of a point in a grid cell is the product of the densities in the used intervals, while the measured density is the number of data points in the grid cell. The authors use an evolutionary search strategy to find grid cells in subspaces, where the measured density is significantly lower than the expected one. The authors demonstrated on small, classified data sets from the UCI machine learning repository that their notion of outliers is useful to find instances belonging to small classes, which can be seen as outliers. However, due to the exponential search space of the problem the evolutionary search heuristic can not guarantee to find all points matching the proposed outlier definition.

2.5 Summary

In this section the previous work related to clustering is summarized. The model-based algorithms (except CLARANS) can be seen as vector quantization methods. The vector quantization methods are efficient, that means they have an acceptable asymptotic run time for large databases $O(n \cdot d \cdot k)$ in case of batch methods and $O(t_{max} \cdot d \cdot k)$ in case of online learning. The algorithms differ in the shown effectiveness, which can be measured by the quantization error. To use the algorithms as cluster algorithms for general cluster types, post processing is required in all cases, since a standalone vector quantization method detects only spherical clusters of nearly equal density and size.

The standard k-means/LBG algorithm with random initialization finds a local optimum, which depends on the chosen initial positions for the centroids. LBG-U and k-harmonic means overcome this drawback, which find better local optimums than LBG.

The online methods from the machine learning research do not guarantee to find a local minimum of the quantization error function, since they are designed for different purposes. A great

advantage is the topology, which can be used for the clustering. Neural gas/ Competitive Hebbian Learning and Growing Neural Gas outperform with the adaptive topology the Kohonen map (fixed topology), since they are able to avoid anomalies in the topology structure (see [89]). However the online learning methods require an extensive parameterization, which prevent the use of the methods in a general data mining tool.

In general, linkage based algorithms produce a hierarchy of clusters, but their high runtime complexity (super quadratic) is prohibitive in case of large data sets.

Centroid linkage and BIRCH deliver like other vector quantization algorithms sets of centroid which are a compressed version of the original data. BIRCH is an efficient, but order-dependent variant of centroid linkage which needs only a single scan over the data and determines the upper part of the complete hierarchy.

Other research on linkage-based algorithms focuses on extensions of single linkage. In contrast to vector quantization this type of clustering algorithms can detect arbitrary shaped clusters. The extensions on effectiveness improvements of single linkage are the handling of the chaining effect (DBSCAN) and the determination of the lower part of the single linkage hierarchy (OPTICS). Improvements on the efficiency include the use of spatial indexes which reduces the runtime complexity to $O(n \log n)$ for low dimensional data.

Density estimation is a statistical research area with strong relations to clustering. There are first approaches of density based clustering algorithms, which use multivariate density functions to determine a clustering. However, the algorithms do not scale to large data sets. Interesting new results on efficient density estimation have not been integrated into cluster algorithms. Recent approaches into that direction mainly use histogram-based density estimation.

Beside full dimensional clustering the new branch of projected clustering has been developed. This new field is motivated by many results, showing the decreasing efficiency and effectiveness of methods when the dimensionality of the data grows high. First interesting results have been proposed recently, which combine clustering with the frequent itemset algorithm apriori and with principal component analysis.

Outlier detection is also strongly related to clustering. Since here objects should be found which can be assigned to a cluster, this problem is complementary to the clustering problem. The proposed concepts include distance-based outliers, local outliers and outliers in projected spaces. For the first two concepts efficient algorithms have been proposed for low dimensional data. However for high dimensional data only quadratic algorithms are available. Outlier detection in projections is an interesting problem but due to the exponential growing size of the search space (in number of dimensions) only heuristic methods are applicable.

The review of the related work has shown that scaling a (quadratic) clustering algorithms often means trading result-quality against efficiency. On the other hand some algorithms introduced concepts to improve the quality of the results. The proposed algorithms can be seen as a combination of scaling strategy and (improved) clustering strategy. An open problem is how to build a consitent framework which can integrate existing methods and allows an exploration of new combinations of different scaling and clustering strategies.

Such a framework should also be extensible to the projected clustering problem. New search strategies for the very large space of projections are needed. It is also an important question how to interprete projected clusters and how different clustering paradigms can serve for full dimensional clustering in this new context.

The last issue is how to integrate expert knowledge about the application domain into the unknown clustering model. When such knowledge is hard to define or the mining goal is only informally given visualization and user interaction might be helpful.

Chapter 3

A Clustering Framework based on Primitives

3.1 Basic Definitions

For defining the clustering primitives, we need the basic definitions of the data space, data set and density functions.

Definition 1 (Data Space, Data Set)
The d-dimensional data space $F = F_1 \times \ldots \times F_d$ is defined by d bounded intervals $F_i \subset \mathbb{R}, 1 \leq i \leq d$. The data set $D = \{x_1, \ldots, x_n\} \subset F \subset \mathbb{R}^d$ consists of n d-dimensional data points $x_i \in F, 1 \leq i \leq n$.

The **probability density function** is a fundamental concept in statistics. In the multivariate case we have random variables from which we can observe data points in F. The density function f gives a natural description of the distribution in F and allows probabilities to be found from the relation

$$P(x \in Reg) = \int_{x \in Reg} f(x)dx, \quad \text{for all regions } Reg \subset F$$

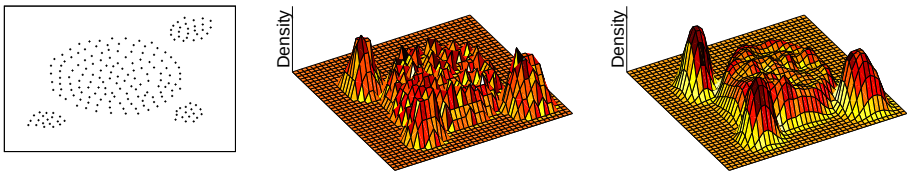
The density function is a basis to build clusters from observed data points. A very powerful and effective method to estimate the unknown density function f in a nonparametric way from a sample of data points is kernel density estimation [98, 105].

Definition 2 (Kernel Density Estimation)
Let $D \subset F \subset \mathbb{R}^d$ be a data set, h be the smoothness level and $\| \cdot \|$ an appropriate metric with $x, y \in F, dist(x, y) = \|x - y\|$. Then, the **kernel density function** $\hat{f^D}$ *based on the kernel density estimator K is defined as:*

$$\hat{f^D}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \quad \textit{with } 0 \leq \hat{f^D} \textit{ and } \int K(x)dx = 1$$

In the statistics literature, various kernels K have been proposed. Examples are square wave and Gaussian functions. An example for the density function of a two-dimensional data set using a square wave and Gaussian kernel is provided in figure 3.1. A detailed introduction to kernel density estimation can be found in [98, 105, 108].

There had been various scaling methods for density estimation proposed in the literature [37, 50, 108, 109, 115]. Table 3.1 presents an overview with the storage complexity and the run time complexity needed to estimate the density at a given point $x \in F$. Note that beside the runtime complexity, the computational costs for building a density estimator can be an important issue. There are different types of density estimators. Local kernels (which are zero for a distant



(a) Data Set (b) Square Wave (c) Gaussian

Figure 3.1: Example for density functions of a two-dimensional data set using a square wave kernel(b) and a Gaussian kernel(c).

Table 3.1: Overview about the storage size and the run time complexity of different scaling methods for density estimation (n: number of data points, d: number of dimensions). The run time complexity is the time needed to estimate the density at a single point $x \in F$.

Type	Size	Time
KDE	$O(n)$	$O(n)$
Local Kernel + mult. dim. Index	$O(n)$	O(range query time)
Histogram (Heap)	$O(n)$	$O(log(n))$
Histogram (Array)	$O(2^d)$	$O(1)$
k centroids, $k \ll n$	$O(k)$	$O(k)$

regions with regards to the particular data point) require only the retrieval of data points in a local neighborhood around the point of estimation. The retrieval can be supported by an multi-dimensional index. Heap-based histograms store only the used grid cells but have to organize the cells in the search structure like heaps or hash maps. Array-based histograms stores all grid cells and have an constant access time, but waste a lot of memory. The output of BIRCH [115] and LBG [75] are k centroids approximating the data set. So the centroids instead of the data points can be used to estimate the density. The histogram and k centroids methods can be integrated into the kernel density estimation concept by using binned kernel estimators [47, 97, 104, 108].

3.2 Definition of the Separator Framework

3.2.1 Separators

Many approaches to clustering large data sets of numerical vectors use density estimation [56]. Density estimation measures the probability of observations of data points in a certain region and the density function contains a lot of local aggregated information about the distances between the data points. Actually this more abstract data description can serve as a basic method for clustering algorithms for numerical vector data. In our approach we strictly distinguish between density estimation and clustering, since density estimation allows the generation of clusters according to different schemes. This distinction has the following advantages: (a) since density estimation can be very costly wrt. resources, the clustering algorithm can be more easily adopted to technical application constraints (runtime, space) by varying the density estimator, (b) since density estimators have different accuracies one can find a tradeoff between quality and efficiency, without changing the clustering scheme (c) density estimators can be easily reused in several applications of a clustering scheme, which reduces the overall runtime of the analysis.

We define the term density estimator as a function, which estimates the unknown density function f based on a set of data points. The estimator can be one of the types presented in table 3.1. Note, the same type of estimator can be the output of different algorithms.

Definition 3 (Density Estimator)
A density estimator is a function $\hat{f}^D \colon F \to \mathbb{R}, D \subset F \subset \mathbb{R}^d$ *with* $\int_{\mathbb{R}^d} \hat{f}^D(x)dx = 1$*, which estimates the point density in the continuous data space* F *according to the given sample data set* D*.*

Clustering algorithms basically use density estimation to determine which points should be joined into a group and so separated from other points. The reader may note that there is a dualism in the description of clusters. First one can describe why points are joined to form groups and the second possibility is to describe why points are separated into different groups. In case of hierarchical algorithms the dualism is known as agglomerative (bottom-up) and divisive (top-down) generation of clusters. For the design of our framework we used the divisive methodology since we believe that an analytical (divisive) approach is more natural and easier to understand in the context of data exploration than a synthetical (agglomerative) one.

More detailed, our new framework employs a similar concept for clustering as decision trees do for classification. A decision tree consists of a number of nodes, each containing a decision rule which splits the data to achieve purer label sets in the child nodes. In our case we do not use class labels to find a split but a density estimator. The equivalent to the decision rule is the separator, which partitions and labels the points of the data space according to a clustering scheme.

Definition 4 (Separator)
A separator is a point labeling function $S \colon F \to \{0, \ldots, Split(S)-1\}$ *which uses a density estimator to assign each point in the continuous data space* F *a label (integer) to distinguish different clusters or groups of clusters. The number of separated regions is* $Split(S) > 0$*.*

For convenience, given a set of point $A \subset \mathbb{R}^d$ the subset of points $\{x \in A : S(x) = i\}$ is denoted as $S^i(A)$ and $S^i(A) \cap S^j(A) = \emptyset, i \neq j, 1 \leq i, j < Split(S)$ and that a separator partitions the data space so that $\bigcup_{i=0}^{Split(S)-1} S^i(A) = A$. Note that there might exist empty regions $0 \leq i < Split(S), S^i(A) = \emptyset$.

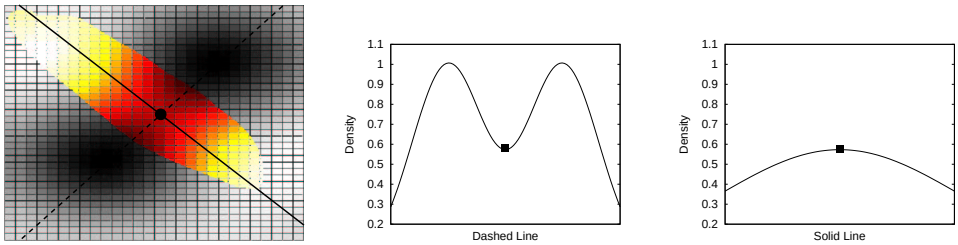
A separator is intended to describe the usage of clustering. The usage – we call it the clustering scheme – determines how the blurred terms "similarity" and "useful grouping" are translated into a computable formalism. In fact, there are different useful translations in the literature (BIRCH [115], DBSCAN [32], DBCLASD [112] etc.). These algorithms produce results according to different clustering schemes, which are not comparable.

However, clusterings according to the same clustering scheme are comparable via a quality function. The quality function is equivalent to an index function like the Gini index in the decision tree concept. Since the choice of the separator type (clustering scheme) is highly semantically influenced by application domain knowledge there cannot exist a general quality function rating all types of separators based on statistical information about the given data set. However, there is an individual quality measure for each clustering scheme, which rates how well the scheme fits the data.

We found four different classes of clustering schemes, namely data compression with centroids, the density based single linkage scheme, noise separation and outlier detection.

- **Data compression** (vector quantization) with k centroids has the goal to minimize the distortion error, which averages the distance from the data points to their nearest centroid. A cluster is described by a single point (centroid) $p \in F$ and contains all data points having p as the nearest neighbor among the k centroids. So the k centroids represent the data set and due to the nearest neighbor rule they form a Voronoi partitioning of the data space. Using a typical metric like the euclidian metric the resulting clusters tend to be compact and approximate d-dimensional spheres. So the data set is lossy compressed to k centroids, while the average error (the average radius of the spheres) is minimized. The usage of this scheme corresponds to the question: What is a smaller representation of the data points, when the data should be represented by centroids?

  The natural measure for the quality of data compression is the distortion error. For a set of



(a) Two 2D Clusters (b) Dashed Line (c) Solid Line

Figure 3.2: The quality of the density based single linkage separator is determined as the maximum density at the border between the clusters (solid line). Subfigure (a) shows the overall picture with two clusters and the color coded density, (b) the density at the dashed line and (c) the density at the solid line. The separation quality in the example is 0.57.

centroids $P = \{p_1, \ldots, p_k\}$ and a data set the distortion error is defined as:

$$E(P, D) = \frac{1}{n} \sum_{x \in D} dist(p_{I(x)}, x)^2$$
$$\text{with } I(x) = min\big\{i \colon dist(p_i, x) \leq dist(p_j, x)\ \forall j \in \{1, \ldots, n\}\big\}.$$

Most algorithms for centroid placement try to find positions for the centroids which have a low distortion error.

- The **density-based single-linkage scheme** defines clusters as regions in the data space, which are (a) connected, (b) of maximal size and (c) where the density is at all points in a cluster region larger than a minimum threshold. Since in general a cluster according to this scheme is arbitrary shaped, it can not be represented by a single centroid and the nearest neighbor rule. Clusters according to the density-based single-linkage scheme are useful to approximate complex correlations. The corresponding question is here: What are the arbitrary shaped regions in the data space, which get densely populated by the underlying data generation processes? The scheme focuses on finding natural classes in the data.

  The quality measure for the density-based single-linkage scheme has not a straight forward definition. Clusters of this type are defined by the minimum density at the border, while shape and compactness have no influence. Clusters are separated by low density valleys. We define the separation quality q_{sep} $(0 \leq q_{sep} \leq 1)$ depending on the maximum density at the borders of the cluster regions in the continuous data space F. The border points of the cluster regions determined by a separator S wrt. a distance function $dist(\cdot, \cdot)$ are defined by:

  $$Border(S) = \{x \in F \text{ and } \forall \epsilon > 0 \colon \exists i, j \in \mathbb{N}, 0 \leq i, j < Split(S) \text{ and}$$
  $$x_a \in S^i(F), x_b \in S^j(F), i \neq j \colon dist(x, x_a) \leq \epsilon \text{ and } dist(x, x_b) \leq \epsilon\}.$$

  The formula says a border point x has the property, that in each arbitrary small ϵ-neighborhood around x there are points of the continuous data space with different cluster labels. This definition is valid since a separator labels not only the data points but all points of the continuous data space. The separation quality is defined as:

  $$q_{sep}(S) = 1 - MAX\big\{\hat{f^D}(x) |\ x \in Border(S)\big\}$$

  Figure 3.2 shows the separator quality of an example data set. The color shows the density on the border between the two clusters and the separator quality corresponds to the inverse of the maximum density on the border.

- **Noise separation** partitions the data space into two regions, which are not necessarily connected. In one region (noise) the density is low and nearly constant (uniform distribution).

This scheme works like a filter, which discernes clustered data from uniformly distributed data.

The goal in the noise separation scheme is to find areas with nearly uniformly distributed data points and low density. Like any statistical distribution a uniform data distribution can be tested with a χ^2-test (or another distribution test). The test variable – the χ-value – can be used as quality measure (see section 3.3.4 for more details).

- **Outlier** detection separates data points, which deviate strongly from the other data points. In the literature the concepts of distance based [68,69,87] and density based outliers [29] had been proposed. Since in case of clusters with different densities the second concept is the more relevant definition [29], we focus only on density based outliers. The focus here is to find data points, which deviate from their neighborhood wrt. to the estimated density.

  Outliers are similar to noise since they do not belong to a cluster either. For density based outliers the density at the outlier point x is compared with the average density at in the local neighborhood $LN(x)$. In [29] the authors proposed a measurement called LOF to rate the outlier degree of a data point. We used an extension of this measurement as quality measure. In contrast to [29], where a very specific notion of density estimation is used, we present an extended version of LOF, which works for general density functions:

$$LOF(x) = \frac{\frac{1}{\#(LN)} \sum_{x' \in LN} \hat{f}(x')}{\hat{f}(x)},\ x \in D,\ LN \subset D.$$

  The ratio $LOF(x)$ is high, when the average density of the data points in the local neighbor hood LN is high and the density at x is low.

Cluster separators with different schemes can be applied to different parts of the data with individual parameters. The decision which cluster scheme applies best to a part depends on the application knowledge or the task at hand, which makes it impossible in the general case to determine the best cluster scheme from the observed data. Our framework enables interactive decisions, where the human analyst can chose a clustering scheme which is semantically meaningful.

3.2.2 Separator Tree

Like decision trees which are an assemblage of classification rules forming a classificator a separator tree is a collection of cluster separators forming a cluster model for the given data, wrt. the user's intention. A separator tree is defined as follows:

Definition 5 (Separator Tree)
A separator tree T is a tree which corresponds to a recursive partitioning of a data set D. A node v of T corresponds to a cluster region $R_v \subset F$. Except the leafs each node v has an assigned separator S_v splitting the corresponding region R_v. The ith son of v corresponds to the region $S_v^i(R_v)$. The region of the root node is the whole data space F.

Next we introduce the split operation, which can be used to grow the tree. The split operation takes a leaf node l of a separator tree T, assigns a separator S to l, and defines in that way the new son nodes of l. The region R of l is decomposed into subregions, each containing different clusters or groups of clusters. An example for the split operation is given in figure 3.3.

In the merge operation two or more independent separators can be joined to get a smaller separator tree. Two separators are independent, if they can be found without applying one of the other separators in advance. The merge operation of separators is defined as follows:

Definition 6 (Merge Operation)
Let be $S_1, \ldots, S_p$ separators. The result of a merge operation of $S_1, \ldots, S_p$ is defined as the mapping function

$$S(x) = S_1 \times \ldots \times S_p = \sum_{i=1}^{p} \left(S_i(x) \cdot \prod_{j=0}^{i-1} Split(S_j) \right) \text{ with } Split(S_0) = 1$$

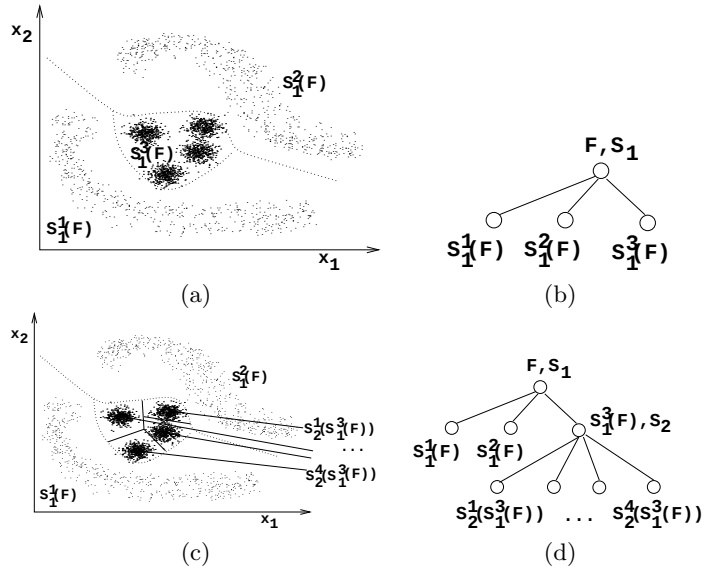


Figure 3.3: Examples of growing the separator tree using a data compression separator with four centroids.

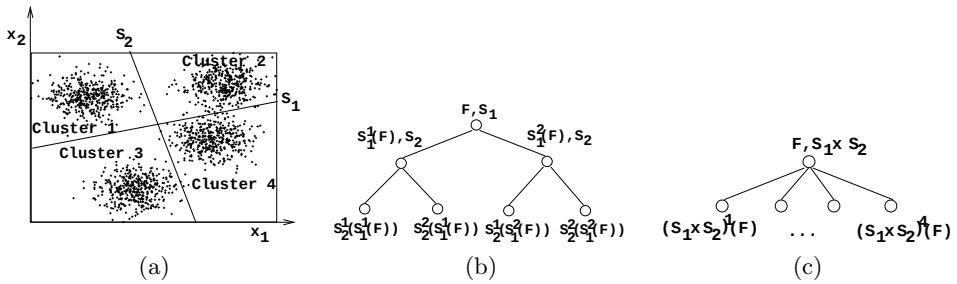


Figure 3.4: Examples of Merge: Since separator S_1 and S_2 are independent of each other, that means the application order is nonrelevant, the merge operation can form a separator $(S_1 \times S_2)$ which is the conjunction of S_1 and S_2.

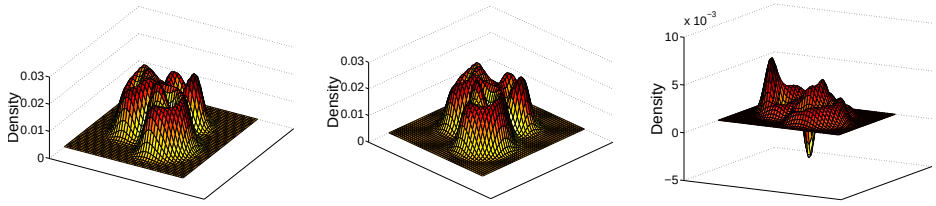
and the Split function is definded by

$$Split(S) = \prod_{i=1}^{p} Split(S_i).$$

The result of an merge operation can be thought of as an linearized arbitrary grid. The merge operation can be used to make the separator tree more compact which intends to improve the ability to interprete the tree. Figure 3.4 provides an example.

3.3 Clustering Primitives

In the previous section we formally introduced the separator tree framework for clustering with the definitions of density estimators, separators with qualities and the separator tree. We identified density estimators and separators as clustering primitives. In the first part (section 3.3.1) of this section we evaluate different density estimators wrt. efficiency and effectiveness. In the rest of the section we propose new or redesigned algorithms for separators, which work with all types of density estimators.



(a) $\hat{f}^D$, $n = 200$ (b) $\hat{f}^P$, $k = 50$ (c) $\hat{f}^P - \hat{f}^D \leq 10\%$

Figure 3.5: Comparison of the kernel density function with an approximation based on centroids for an 2D Example.

3.3.1 Density Estimators

The naive implementation of the standard kernel density function (see section 3.1) requires a number of vector difference operations in $O(n)$ (where n is the number of data points) to determine the density at a single point $x \in F$. Since the size of a data set may be very large, the frequent use of the density function is computationally expensive. There has been done much research to find more efficient density estimation methods.

The WARPing framework (Weighted Averaging of Rounded Points) [47, 108] describes a way to develop efficient density estimation methods based on prebinning. Approaches according to WARPing are (a) the k centroids method [116], (b) histograms [98] and (c) average shifted histograms (ASH) [96, 98].

Another approach bases on local kernels with multidimensional index structures (partial range queries) [32,112]. Since the efficiency of the multidimensional index structures is decreasing in high dimensional spaces [110], we focus on the WARPing approaches, which can guarantee a sublinear runtime behavior for determining the density at a single point.

Density estimation based on k centroids uses the sufficient statistics (average, variance and number of data points) of the Voronoi cells, which are given by the positions of the k centroids. The positions are determined by a placement algorithm (vector quantization). Using the WARPing framework Zhang et. al. proposed in [116] a density function based k centroids, which approximates the standard density function. Let $P = \{p_i \in F : 1 \leq i \leq k\}$ be the set of centroids, $I(x) = min\{i : dist(x, p_i) \leq dist(x, p_j) \forall j \in \{1, \ldots, k\}\}$ the index function, which delivers the minimal index of the nearest centroid p_i, $p_i(D) = \{x \in D : I(x) = i\}$ the set of data points, which have p_i as the nearest centroid, $n_i = \#p_i(D)$ the number and σ_i the standard deviation of the data points in $p_i(D)$. The density function based on the Voronoi prebinning and the Gaussian kernel for a given smothing factor $h \in \mathbb{R}$ is:

$$\hat{f^P}(x) = \frac{1}{n} \sum_{i=1}^{k} \frac{n_i}{\sqrt{2\pi}\sqrt{\sigma_i^2 + h^2}^d} \cdot exp\left(- \frac{(x - p_i)^2}{2(\sigma_i^2 + h^2)} \right), \; 0 \leq \hat{f^P}.$$

The function needs space in $O(k \cdot d)$ and has a runtime complexity of $O(k \cdot d)$ for a density estimation at a single point, which is a significant reduction if $k \ll n$. An example (figure 3.5) demonstrates the impact of the data reduction and the loss of accuracy.

To produce a Voronoi binning several vector quantization methods can be used, namely random uniform sampling, k-means or its variants (LBG [75], LBG-U [37], k harmonic means [114]), online algorithms like neural gas [78], growing neural gas [36] or centroid linkage or its popular variant BIRCH [115]. The runtime complexities of these algorithms except centroid linkage are linear in the number of data points.

Now we compare the effectiveness of this density estimator in combination with different placement algorithms. The effectiveness of vector quantization is typically measured by the distortion error, which measures the average distance from the data points to its nearest centroid. Since we want to measure the effectiveness wrt. density estimation we use a relative comparison with the standard kernel density function. This comparison method has been proposed in [116] to evaluate

the quality of the density functions by averaging the relative density difference at several breakpoints. Using a standard kernel density function with Gaussian kernel as reference function the relative density difference is defined as:

$$D(\hat{f}, x) = \frac{2 \cdot |\hat{f}_{KDE}(x) - \hat{f}(x)|}{\hat{f}_{KDE}(x) + \hat{f}(x)}$$

We evaluate the performance of different placement algorithms (random sampling, LBG, LBG-U and BIRCH) on synthetic and real data sets. We measure the performance by the averaged density difference relative to the standard KDE function using 100 randomly chosen breakpoints [116] and by the quantization error. For the first experiments we configured the methods to use 20 centroids. For the experiments we used synthetic data of two types, namely (1) the clustered data are in one cluster (normally distributed) and (2) the clustered data are spread over 50 cluster (normally distributed) of different sizes. In the experiments we used 20% noise and 80% cluster data. For each type we varied the dimensionality from $d = 2$ to $d = 20$.

Figure 3.6 shows the quantization error and the averaged density difference depending on the dimensionality for both types of synthetic data. Figures 3.6 (a,c) and (e,g) show, that the distortion error as well as the averaged density difference grow with the dimensionality.

Since the curves are close together the reader may conclude from the experiments that it is more or less the same, which placement algorithm is used. To devitalize this argument we take a closer look at the results. Since random sampling is the simplest placement method we plotted all measures relative to random sampling to check out whether the more sophisticated methods are an improvement or not (see figure 3.6(b,d,f,h)). In order to measure the improvement over random sampling in percent we transformed the measurements as follows:

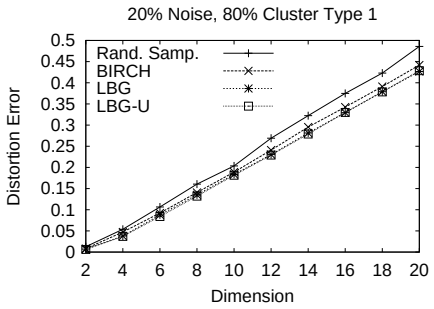
$$Rel.D_{RS}(\hat{f}, x) = \left(1 - \frac{D(\hat{f}, x)}{D(\hat{f}_{RS}, x)}\right) \cdot 100 \text{ and } Rel.E_{RS}(P, D) = \left(1 - \frac{E(P, D)}{E(P_{RS}, D)}\right) \cdot 100$$

The plots in figure 3.6 show that LBG and LBG-U, which minimize the distortion error, also have the smallest density difference and the largest improvement over random sampling for low dimensional data. That leads to the conclusion that minimizing the distortion error improves the quality of the density estimate. However, the results for the averages density difference also show that all types of prototype based density estimation degenerate with growing dimensionality, which is due to the so called "curse of dimensionality". So beyond a dimensionality of 16 the improvement of density estimate by the minimization of the distortion error has only minor effects. The figure also shows that BIRCH has some instabilities, which might be due to the dependence of the order of the data points.

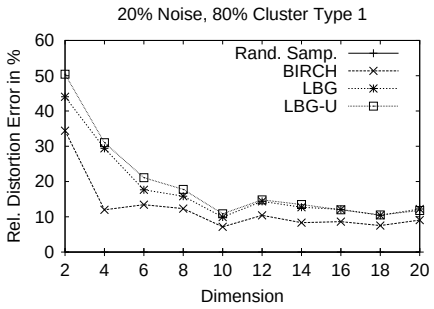
It is well understood that the degeneration of the density estimate can be deferred by using more centroids [53,116], but this does not change the whole situation. Figure 3.7 shows the average density difference for random sampling with different numbers of centroids (1%,5%,10% of the size of the data set). The results on both types of synthetic data show that the degeneration of the density estimate can be weakened but not removed. Note, that with a growing number of centroids the desired run complexity reduction gets lost, because the k increases up to the order of n and the run time complexity of $O(k \cdot d)$ of the centroids based density function approaches $O(n \cdot d)$, which is the standard kernel density function. So the conclusion of this part is that density estimation can be accelerated in low dimensional spaces by representing the data with centroids and sufficient statistics. However, this approach is limited to data with a dimensionality $d < 12$.

The other efficient methods for density estimation are the histogram and the average shifted histogram, which is an improved variant. A histogram can be thought as a multidimensional grid. In [50,53] we proposed the key idea of our efficient implementation of high dimensional histograms, which consists in storing only the populated grid cells. Considering the minimal bounding rectangle $MBR = [Min, Max]$, $(Min, Max \in F^d)$ of the data set and $h_1, \ldots, h_d$ as the bin width in each dimension, the histogram may contain

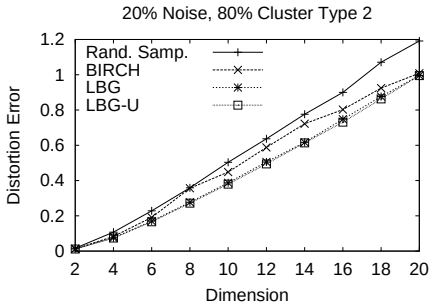
$$M = \prod_{i=1}^{d} \left\lceil \frac{|Min_i - Max_i|}{h_i} \right\rceil$$



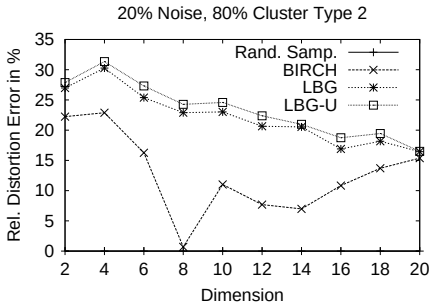
(a) Distortion Error



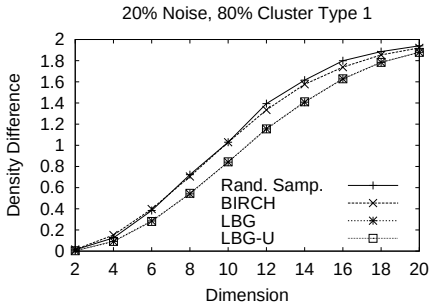
(b) Improvement over Random Sampling



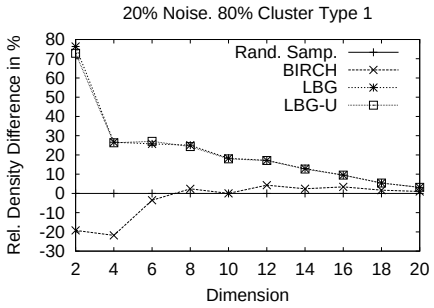
(c)Distortion Error



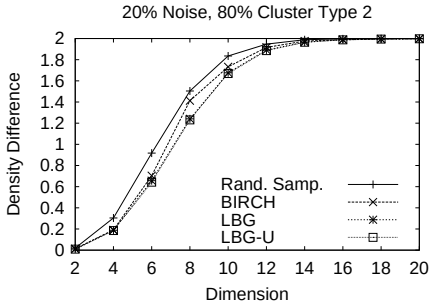
(d) Improvement over Random Sampling



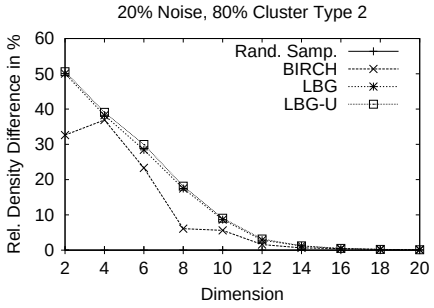
(e) Density Difference



(f) Improvement over Random Sampling

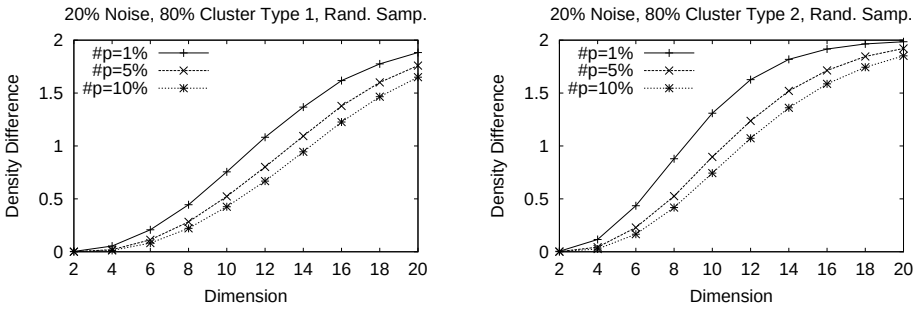


(g) Density Difference



(h) Improvement over Random Sampling

Figure 3.6: Comparison of density estimation based on k prototypes: We used for all experiments data with 10000 data points (20% noise, 80% cluster, type 1 and 2) with different dimensionality. We configured the methods to use 20 prototypes. Each measurement is the average of 10 runs. We compared the effectiveness of the placement algorithms random sampling, BIRCH, LBG, LBG-U, measured by the distortion error (a-d) and the density difference to the standard density estimation (e-h) (lower is better). Part (b,d,f,h) shows the improvement over random sampling in percent (higher is better).



(a) Density Difference (b) Density Difference

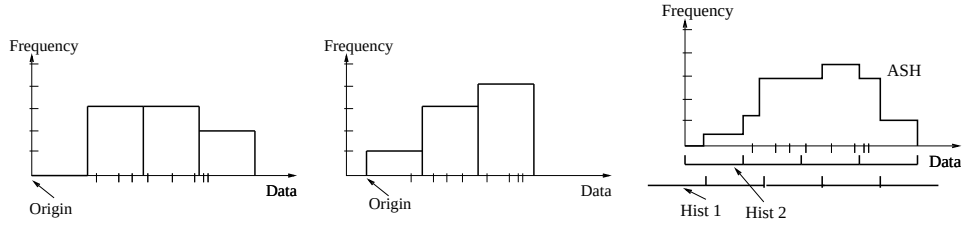
Figure 3.7: We investigated the influence of the number of prototypes on the quality of prototype based density estimation using random sampling as an example placement method. This does not limit the results, since we showed that the improvement over random sampling wrt. density estimation decreases with growing dimensionality. We used our synthetic data of type 1 and 2 with 10000 data points and set the number of prototypes to 1%,5% and 10% of the size of the data set. Each measured value is averaged over 10 runs.

bins. The number of possible bins M grows exponentially with the number of dimensions. However, since a populated bin has to contain at least one data point, the number of populated bins is at most n. In case of $n \ll M$ only the populated bins should be stored, which can be efficiently done in a heap data structure. For the heap construction one has to determine for a given point x a unique and sortable key for the bin of x. If the MBR is known such a key can be efficiently determined by linearization. When the MBR is not known the key can be determined by discretization of the axes and converting the discrete vector into a string which can be sorted lexicographically. Usable heap data structures are randomized search trees for main memory [80] or B^+ trees for storage on hard disk. In the latter case, the histogram can be determined with standard SQL 92 queries and can be stored in a relational database table with an index. The following example SQL statement generates a equi-distant 15-dimensional histogram with $(0, 0, \ldots, 0)$ as origin and a bin width of 0.2 from the numerical attributes col1, ...,col15 of table "DataTable" by producing the key of a bin and the corresponding bin count:

```
select  concat(
round(col1/0.2)," ",round(col2/0.2)," ",round(col3/0.2)," ",
round(col4/0.2)," ",round(col5/0.2)," ",round(col6/0.2)," ",
round(col7/0.2)," ",round(col8/0.2)," ",round(col9/0.2)," ",
round(col10/0.2)," ",round(col11/0.2)," ",round(col12/0.2)," ",
  round(col13/0.2)," ",round(col14/0.2)," ",round(col15/0.2)
) as key_attr,
count(*) as density
from DataTable
group by key_attr
```

The number of possible grid cells M can be used to decide whether the histogram should be stored in an array (storing of all cells, if $M < n$, storage complexity $O(2^d)$, access complexity $O(1)$) or in a heap (storing of the used cells, if $M \gg n$, storage complexity $O(n)$, access complexity $O(\log n)$). For low dimensional data spaces the array is preferable, because it provides constant access time to a grid cell, while the heap method provides logarithmic access time, but since only the used grid cells are stored in the heap it has a storage complexity in $O(n)$, which is much lower than the space needed for arrays for a higher dimensionality. Let be $h1, \ldots, h_d$ the bin widths in each dimension the density function provided by an histogram is:

$$x \in F,\ \hat{f}(x) = \frac{histogram[key(x)].count}{n \cdot h_1 \cdot \ldots \cdot h_d}$$



(a) histograms for the same data (b) ASH

Figure 3.8: Part (a) shows two histograms with the same bin width but different origins smoothing the same data. The histograms are quite different demonstrating the impact of the choice of the origin. Part (b) shows the average shifted histogram (ASH, $m = 2$) constructed from the both histograms of (a).

where *histogram* is the data structure, which stores the information of the grid cells and provides access to them using the *key*-function. The *count* variable stores the number of data points in the cell. The building runtime to initialize the histogram is $O(n)$ for array based and $O(n \cdot \log(n))$ for heap based histograms. There are also a lot of helpful rules and measures available in the statistical literature for the problem how to determine useful parameters for a histograms. For an overview see [98].

There are several drawbacks of histograms. One important is the dependency of the density estimate from the histogram origin. For an example see figure 3.8 (a), which demonstrates the impact of the choice of the origin. To weaken this undesired dependency Scott proposed in [96] to use average shifted histograms (ASH). An ASH consists of m simple histograms which have the same bin width but the origin is shifted by a part of the bin width. Figure 3.8 (b) gives an example for the univariate case. To estimate the density at a point x the bin counts for x in each histogram are determined and averaged.

$$x \in F,\ \hat{f}(x) = \frac{1}{m} \sum_{i=1}^{m} \frac{histogram_i[key(x)].count}{n \cdot h_{i,1} \cdot \ldots \cdot h_{i,d}}$$

For our experiments we used the following shifts (for d-dimensional data): $o_1 = j/lh \cdot e_1, \ldots, o_d = j/lh \cdot e_d$, $o_{diag} = j/lh \cdot [1, \ldots, 1]$ and $o_0 = [0, \ldots, 0]$, where e_i is the unit vector with a 1 in dimension i and 0 else, $1 \leq j \leq l-1$ and $l = 2, 3$. This means that we have to construct $((d \cdot (l-1))+2),\ l = 2, 3$ d-dimensional histograms.

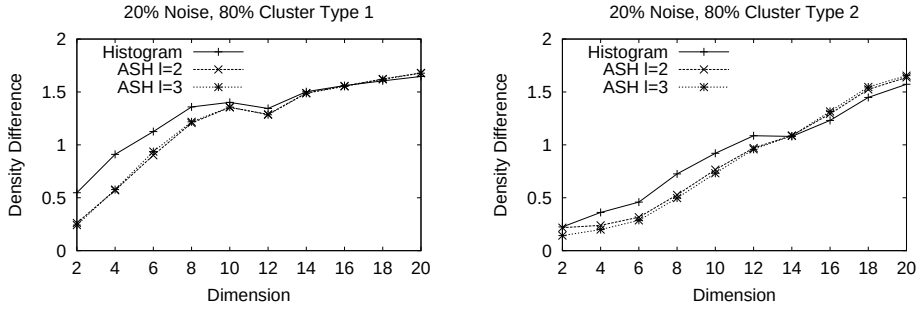
In figure 3.9 we show the results of the histogram based density estimation for the two types of synthetic data. Also this density estimator degenerates with growing dimensionality. Figure 3.9(b) shows that for density estimation in low dimensional spaces with average shifted histograms the quality can be improved by using more histograms for the average, that means larger values for l. However this also increases the computational costs.

In table 3.2 we showed a comparison of all methods using a real data set from the US census bureau. The data consists of the geographic position of the averaged households at block level ($n = 276000, d = 2$).

In figure 3.10 we compared the k centroid method and the histogram. For simplicity we used only random sampling and the simple histogram. The result on the synthetic data show that centroid based density estimation is better for low dimensional spaces ($d < 8$) and histogram based density estimation is more suitable for medium dimensionality ($8 \leq d \leq 16$). For high dimensional spaces ($d > 16$) both methods do not work well. This result is supported by statements from [47, 98, 105], which say density estimation becomes statically insufficient in high dimensional spaces. This is the starting point for all density based cluster algorithms.

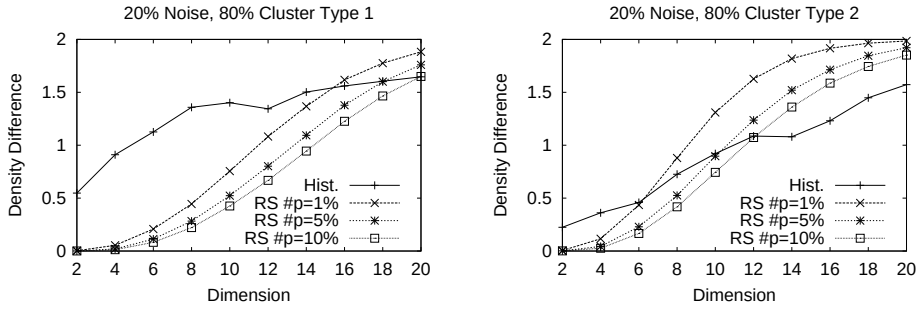
3.3.2 Data Compression

Data compression has the goal to represent the data points $D = \{x_1, \ldots, x_n\}$ with k centroids $P = \{p_1, \ldots, p_k\} \subset F \subset \mathbb{R}^d$. The assignment of the points to the centroids is typically done using



(a) Density Difference (b) Density Difference

Figure 3.9: The figures show the average density difference for the histogram estimator, the ASH with l=1 and the ASH with l=2. The number of shifted histograms per dimension is denoted with l. The bin width in each dimension is set to $(max[i] - min[i])/2$.



(a) Density Difference (b) Density Difference

Figure 3.10: The plots show the averaged density difference of the histogram density estimator and of the prototype based density estimator with $k = 1\%, 5\%, 10\%$ of the size of the data set.

Table 3.2: The table shows a comparison of all density estimation methods using a real data set from the US census bureau. The data consists of the geographic position of the averaged households at block level ($n = 276000, d = 2$). The results show that centroid based density estimation performs better that histograms. It also shows that LBG and LBG-U outperform all other methods wrt. estimation quality.

Data	Algo.	k,l	Time/sec	Density Diff
US Census	Rand. Samp.	$k = 100$	0.8	0.00836112
		$k = 1000$	3.2	0.00110772
	BIRCH	$k = 100$	0.8	0.00470599
		$k = 1000$	3.2	0.00439966
	LBG	$k = 100$	0.8	0.00165451
		$k = 1000$	3.2	0.0009156
	LBG-U	$k = 100$	0.8	0.00159414
		$k = 1000$	3.2	0.0001112
	Histogram	-	0.3	0.441989
	ASH	l=2	0.5	0.331553
	ASH	l=3	0.6	0.264106

the nearest neighbor rule. This means that a data point is represented by its nearest centroid. The euclidian metric is often used as distance function. The natural measure for the quality of data compression is the distortion error. For a set of centroids P and a data set the distortion error is defined as:

$$E(P,D) = \frac{1}{n}\sum_{x\in D} dist(p_{I(x)},x)^2 \text{ with } I(x) = min\big\{i|dist(p_i,x) \leq dist(p_j,x)\ \forall j \in \{1,\ldots,n\}\big\}.$$

Most algorithms for centroid placement try to find positions for the centroids which have a low distortion error. For convenience we denote the subset of D with $I(x) = i, 0 \leq i < k$ as set D_i and the corresponding part of the data space F as F_i with the probability density f_i. To minimize the distortion error two conditions are necessary [41], firstly the nearest neighbor condition (a point x has to be mapped to its nearest centroid) and the centroid condition (in case of euclidian distance: $p_i = cent(F_i) = E[x|\ x \in F_i]$). There are different placement algorithms working directly on the data set, which is assumed to estimate the probability distribution in F. In this case the centroid function reduces to the arithmetic average:

$$cent(F_i) = \int_{F_i} x f_i(x)dx \approx \frac{1}{|D_i|}\sum_{x\in D_i} x$$

The key idea of LBG is to do Lloyd iteration steps until the algorithm converges. In the Lloyd iteration each data point x is assigned to its nearest centroid. After determining the subsets D_i each centroid p_i is set to the centroid of F_i estimated by the mean of D_i. These moves may change the D_is, which is the issue for the next iteration. The process converges when the Lloyd iteration does not change the data point assignments anymore.

Our new algorithm approximates the centroid of a region using directly an estimate of the probability density. First we set the density function for the part of the data space F_i to:

$$f_i(x) = \begin{cases} c\cdot f(x) & \text{if } x \in F_i \\ 0 & \text{else} \end{cases}$$

where $f(x)$ is the global density function and c a constant to assure $\int f_i(x)dx = 1$. So we can derive $cent(F_i)$ using Monte-Carlo integration [86] as follows:

$$cent(F_i) = \int_{F_i} x f_i(x)dx = \frac{1}{\int_{F_i} f(x)dx}\int_{F_i} x f(x)dx \approx \frac{1}{\sum_{x\in B}\hat{f}(x)}\sum_{x\in B} x\hat{f}(x)$$

where B is a uniform sample from F_i, which can be generated using numerical random number generators. Our idea for the modification of the original LBG algorithm is to approximate the most right formula using a density estimator and uniform sampling in F_i instead of the evaluation of the mean of D_i. The algorithm 1 implements this idea.

The density based LBG algorithm has two additional parameters m_{it} and m_{sample}, which set the number of iterations and the number of uniform sample points for the integral approximation. Note, that in contrast to random sampling uniform sampling selects the points without bias of the data points. The points are uniformly selected in the data space F, which is described by the minimal bounding rectangle (MBR). Please note that due to the uniform character of the sample points x, the lengths of the Δp_i vectors after the execution of the inner loop do not reflect the degree of convergence of the whole algorithm. That's why we chose to use a fixed number of runs of the inner loop instead of coupling the execution of it with the state of convergence. This makes our algorithm more similar to an online algorithm, which also does not guarantee convergence.

Since a uniformly selected point with the corresponding density may represent many data points the value for m_{it} can be smaller than n. This is helpful in case of clustered data with high density areas. The theoretical run time of DB-LBG is in $O(m_{it}\cdot m_{sample}\cdot d(k+$ density estimation time)).

The decoupling of density estimation and centroid placement leads to useful strategies of improving the original LBG algorithm. First to trade quality against run time by using faster but

Algorithm 1 Density Based LBG

Require: $k, m_{it}, m_{sample} \geq 1$, F described by the MBR, $\hat{f}(x)$ a well set density estimator
  $P \leftarrow$ UniformSample(F, k)
  for $i = 1$ to m_{it} **do**
    $\forall 0 \leq i < k : \Delta p_i \leftarrow \vec{0}, n_i \leftarrow 0$
    for $j = 1$ to m_{sample} **do**
      $x \leftarrow$ UniformSample$(F, 1)$
      $\Delta p_{I(x)} \leftarrow \Delta p_{I(x)} + \hat{f}(x) \cdot x$
      $n_i \leftarrow n_i + \hat{f}(x)$
    end for
    $\forall 0 \leq i < k : p_i \leftarrow \Delta p_i / n_i$
  end for

less accurate center approximations than the one based on nearest neighbor rule and the whole data set. Second, since the used density estimator may require less memory than the data set the new algorithm is more suitable for large data sets which do not fit into main memory.

Now we evaluate the algorithms experimentally on synthetic and real data sets. First we used the k' centroid estimator with random sampling (to avoid confusion we denote the number of centroids for the density estimator with k' in this subsection) to estimate the density function $\hat{f}(x)$. Figure 3.11 shows the average estimation errors versus the needed runtimes of the algorithms. The left point of a line shows run time and distortion error for density based LBG and the right point for the original LBG. The figure shows results for data set of dimension $d = 2, 4, 6, 8, 10, 12, 14, 16$ (from bottom to top). Note that the line connects only the corresponding measurements but has no additional meaning.

The experiments show that using the k' centroids estimator a small loss in quality enables a large performance gain. The other important advantage of the density estimator is the much lower memory requirements, namely $O(kd + k'd)$ for DB-LBG in contrast to $O(kd + nd)$ for LBG which needs a materialized data set. In our experiments the number of centroids where to set to $k' = 20$ in opposite to $n = 100000$. Since the k' centroid density estimator can be built in a single scan of the data set the run time complexity and I/O costs of the $DB - LBG$ algorithm are significantly reduced compared to disk based LBG (however, we did not use disk based LBG in the runtime experiments to have a fair comparison).

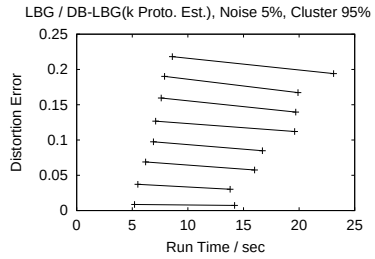
We also made experiments using the average shifted histogram as density estimator. Using this estimator DB-LBG also reaches a comparable quality on our data sets but only for low dimensional data a lower run time than LBG is achieved. The histogram does reduce the space requirements only in the low dimensional case. The results for a synthetic data set are presented in table 3.3.

In case of data exploration, it is often required to run LBG several times on the same data set to tune the parameters to the requirements of the application. In this case the total run time costs reduce dramatically, because the density estimator has to be computed only one time. In Figure 3.12 we show for LBG and DB-LBG (k' centroids and ASH) the number of repetitions versus the total runtime required. It shows that the additional costs for time intensive estimator construction quickly amortizes when the density estimator is used more often and DB-LBG outperforms LBG by magnitudes.

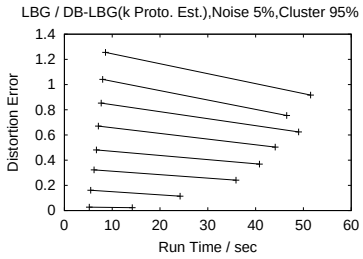
We performed experiments with real data namely molecular biology data (d=19, n=100000), US census data (d=2, n=254000) and cad data (d=11, n=36000). The results are presented in table 3.4, which shows that DB-LBG has a much lower runtime than LBG with a small loss of accuracy in case of low dimensional large data sets.

3.3.3 Density-based Single-Linkage

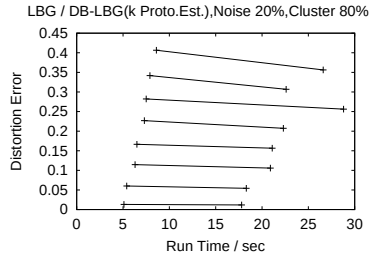
One of the most popular approaches to clustering is the single-linkage algorithm [103]. In general, a single-linkage clustering with a given linkage distance l is defined as follows.



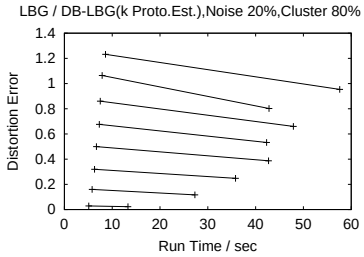
(a) 5% noise + 95% cluster (1 normal distribution)



(b) 5% noise + 95% cluster (50 normal distributions)



(c) 20% noise + 80% cluster (1 normal distribution)



(d) 20% noise + 80% cluster (50 normal distributions)

Figure 3.11: Comparison of the density based LBG with the original LBG: The left point of a line shows run time and distortion error for density based LBG and the right point for the original LBG (average values of 10 runs). The figure shows results for data sets of dimension $d = 2, 4, 6, 8, 10, 12, 14, 16$ (from bottom to top). Note, the line connectd the corresponding measurements but has no additional meaning. We used for DB-LBG the k-prototype density estimator with 20 prototypes, which were randomly selected from D. The additional parameters were set to $m_{iter} = 10$ and $m_{sample} = 50$.

Table 3.3: Comparison of the Density based LBG with the original LBG using the ASH as density estimator. The measurements are averages of 10 runs.

Data	Dim	Density Est.	Alg.	Run Time	Dist. Error
Noise: 20%, Cluster 80%, (50 Norm. Dist.)	2	ASH	DB-LBG	7.8	0.027
		k' centroids	DB-LBG	5.1	0.029
		-	LBG	13.9	0.023
	4	ASH	DB-LBG	16.3	0.159
		k' centroids	DB-LBG	5.8	0.159
		-	LBG	23.4	0.116
	6	ASH	DB-LBG	27.9	0.348
		k' centroids	DB-LBG	6.3	0.319
		-	LBG	36.1	0.247
	8	ASH	DB-LBG	42	0.584
		k' centroids	DB-LBG	6.7	0.499
		-	LBG	39.7	0.385
	10	ASH	DB-LBG	59	0.885
		k' centroids	DB-LBG	7.3	0.676
		-	LBG	44	0.530

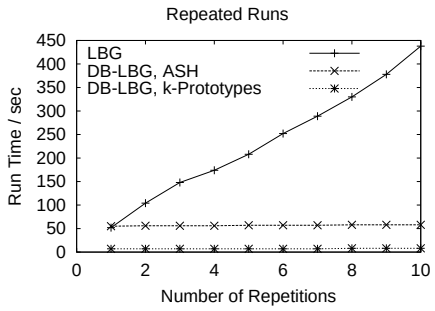


Figure 3.12: Comparison of the density based LBG with the original LBG in the case of repeated runs on the same data set with changed parameters. We used the 10 dimensional data set with 20% noise and 80% cluster (50 normal dist.). The runtime for LBG grows linearly. For DB-LBG we used the k' prototype estimator and the ASH. The run time of DB-LBG grows in both cases very slowly since the repeated building of the density estimator can be avoided.

Table 3.4: Comparison of the density based LBG with the original LBG using the ASH as density estimator on real data. The measurements are averages of 10 runs. The used real data sets are molecular biology data (d=19, n=100000), cad data (d=11, n=36000) and US census data (d=2, n=254000).

Data	Density Est.	Alg.	Run Time	Dist. Error
Molecular Biology Data	ASH	DB-LBG	176	140523
	k' centroids	DB-LBG	8.9	121905
	-	LBG	317.5	89754
CAD Data	ASH	DB-LBG	30	4.93385
	k' centroids	DB-LBG	2	3.05562
	-	LBG	40	1.46551
US Census	ASH	DB-LBG	16	13.243
	k' centroids	DB-LBG	10	13.263
	-	LBG	82	13.328

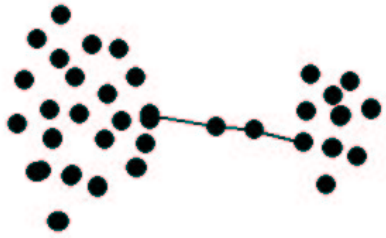


Figure 3.13: 2-dim. example for the chaining effect

Definition 7 (Single-Linkage Clustering)
Given a set of objects O, the partition $C = \{C_1, \ldots, C_m\}$ of O is a single-linkage clustering for a distance function $\mathrm{dist}(\cdot,\cdot)$ *and a given linkage distance $l > 0$, iff the following properties are satisfied for all $C_i, i = 1, \ldots, m$.*

1. *$C_i \neq \emptyset$*
2. *If the object $o \in C_i$, then are all objects o' in the l-environment of o $\{o' \in O : \mathrm{dist}(o, o') \leq l\}$ belong to C_i.*
3. *There are no non-empty subsets C', C'' of C_i, which satisfy 1. and 2.*

There are many algorithms to determine such a clustering. A general problem of the single-linkage method is that is produces sometimes clusters which are not homogeneous, especially if clusters of similar objects are linked by a chain. When the data contains noise, this is likely to occur. Figure 3.13 illustrates such a case. A method to reduce the chaining effect has been proposed by Wishart [111] (see also [23]). Wishart suggests as a preprocessing step to remove all objects o from the data set for which

$$\#\{o' \in D : \mathrm{dist}(o, o') \leq l\} < den, den \in \mathbb{N} \tag{3.1}$$

and then to apply the single-linkage method to the reduced data set. The preprocessing removes the points which are in regions of low density in the data space and in that way reduces the chaining effect. In the following text we assume that the objects are given by feature vectors and the distance is the Euclidian metric in this space. With this assumption definition 7 together with the additional condition 3.1 forms the density-based single-linkage scheme.

The concept has been slightly modified reinvented by Ester et al. in [32, 63] with the focus on scaling the single-linkage method. They supported the near neighbor search of the l-environment with spatial index structures like R*-tree and X-tree to reduce the overall complexity from $O(n^2)$ to $O(n \cdot time_{retrieval})$ which is for low dimensional spaces $O(n \log n)$.

Despite some promising results in [32, 63] the concept of single-linkage consisting of linkage distance and the number of points in the neighborhood remained. In [50, 53] we showed that density-based single-linkage can be simulated using kernel density estimation with square wave functions. In our new approach we redesign the density-based single-linkage concept to work with general density functions, which leads to interesting new efficient algorithms for building such clusterings.

Based on the ideas in [50, 53], we give a definition for density-based single-linkage which makes no assumption about the employed density estimation function. We replace the condition 2 in definition 7 that linked points are near enough ($dist(x, x') < l$) and the additional condition 3.1 that more than *den* points have to be in the l-environment by the condition, that the density on the line between x and x' is not below a given threshold value ξ.

Definition 8 (Density-Based Single-Linkage Clustering)
Given a set of objects described by the set of feature vectors $D = \{x_1, \ldots, x_n\}$, the partition $C = \{C_1, \ldots, C_l\}$ of D is a density-based single-linkage clustering for a given density function $\hat{f}(\cdot)$ and a noise level ξ, iff the following properties are satisfied for all $C_i, i = 1, \ldots, l$.

1. $C_i \neq \emptyset$
2. *If* $x \in C_i$, *then are all* $x' \in C_i$ *with:* $\forall t \in [0,1] : \hat{f}(x + t \cdot (x' - x)) \geq \xi$.
3. *There are no non-empty subsets* C', C'' *of* C_i, *which satisfy 1. and 2.*

Points $x \in D$ *with* $\hat{f}(x) < \xi$ *are called outlier.*

The new requirement 2. enforces that the density between x and x' is high and so enough points are between x and x' to get them at least transitively linked. With values for ξ which are larger than zero also the additional condition of Wishart can be fulfilled. Note that the clustering may vary depending on quality of the underlying density function. The intuition behind this definition is that a cluster is formed by a connected region R of the data space and the density at all points of R is above the noise threshold ξ. The clusters are isolated by low density valleys. For our separator approach we need an algorithm, which finds the valley with the lowest density which separates two clusters.

By definition the original single-linkage problem has quadratic runtime complexity in the number of data points, since each data point has to be compared with each other whether they can be linked. For density functions, which are zero in some regions, combinations of data points can be excluded from the search. However, in general this regions are not known. To make the algorithm scalable we propose first to reduce the set of data points $D = \{x_1, \ldots, x_n\}$ to a set of centroids $P = \{p_1, \ldots, p_k\}$. This can be done with the DB-LBG algorithm which accepts general density functions or any other placement method in special cases. The reduction decreases the number of objects from n to k with $k \ll n$.

After reducing the data set, the second task is to determine the connected high density regions based on the centroid representation of the data set. The naive approach starts with a complete, undirected and weighted graph, called cluster graph:

$$G = (V, E, w) \text{ with } V = \{p_i : 1 \leq i \leq k\},\ E = \big\{(p_i, p_j) : i, j \in \{1, \ldots, k\}\big\}$$
$$\text{and } \forall e = (p, p') \in E,\ w(e) = min\big\{\hat{f}(p + t \cdot (p' - p))\big\},\ t \in [0,1].$$

Since the density function is not in analytical form available the implementation of the weight determination works with a discretization of the line between the centroids and estimates the density on $r \geq 1, r \in \mathbb{N}$ points at the line between p and p'. So the time for the determination of a weight is in $O(r \cdot time_{DE})$ where $time_{DE}$ is the time needed for the density estimation at a single point.

Centroids belonging to the same high density region are at least transitively linked in the cluster graph with edges of large weights. To separate such regions edges are deleted in the ascending order of their weights until the graph splits into two connected components. The threshold for the noise level ξ is set to the weight of the last deleted edge. All nodes with a lower density than ξ are assumed to approximate noise points and are collected in a special prototype subset P_0. Algorithm 2 describes the method in pseudo code.

The disadvantage of the naive approach is that in the cluster graph the number of edges is $\binom{k}{2}$. To ensure that high density regions are connected in the initial cluster graph the Delaunay graph of the centroids can also be used. Informally, the Delaunay graph connects neighboring centroids by an edge. In low dimensional spaces ($d = 2, 3$) the Delaunay graph can be computed in $O(k \log k)$ time, however for higher dimensionality the costs grow exponentially. Since the Delaunay graph in two dimensional spaces is planar the number of edges is bounded by $\#(E) \leq 3 \cdot k - 6$. However, for $d \geq 3$ the Delaunay graph may be the complete graph for which the number of edges is $\binom{k}{2}$ ([16], page 357). For a survey on Delaunay graphs and the dual structure – the Voronoi diagram – we refer to [16].

To use the advantage of Delaunay graphs (having less edges as the complete graph) also in multidimensional spaces with $d > 3$ the initial cluster graph can be given as induced Delaunay graph of the centroids which is an subgraph of the full Delaunay graph. For large data sets and low dimensional spaces the induced Delaunay graph is statistically convincing and so also fulfills

Algorithm 2 Density-Based Single-Linkage Separator

db_slink_separator($G(V = P, E, w), \hat{f}$ **)**

Require: $G(V = P, E, w)$ initial cluster graph

Ensure: $P_1, P_2, \ldots$ contains the centroids in connected high density regions isolated by density valleys with maximal density ξ (separation quality), P_0 contains prototypes approximating noise

1: $\xi \leftarrow 0$, $P_0 \leftarrow \emptyset$
2: **while** G is connected **do**
3: determine e with $w(e) = min\{w(e'), e' \in G.E\}$
4: $\xi \leftarrow w(e)$
5: G.delete_edge(e)
6: $P_0 \leftarrow P_0 \cup \{p \in G.V, \hat{f}(p) < \xi\}$
7: delete all nodes p from G with $\hat{f}(p) < \xi$
8: **end while**
9: $P_1, P_2, \ldots \leftarrow$ Determine_Connected_Components(G)
10: **return(**$P_1, P_2, \ldots, P_0, \xi$**)**

the requirements for the initial cluster graph. According to Martinetz and Schulten [77, 79] the induced Delaunay graph is defined as follows:

$$\begin{aligned} G_{ID} = (V, E, w) \text{ with } V = \{p_i : 1 \leq i \leq k\} \text{ and } E = \big\{ & (p_i, p_j) : \exists x \in D \text{ with} \\ & (i = I(x) \text{ and } j = I2(x, i)) \text{ or } (j = I(x) \text{ and } i = I2(x, j)) \big\} \\ & \text{with } I(x) = min\{i : dist(x, p_i) \leq dist(x, p_j) \forall j \in \{1, \ldots, k\}\} \\ & \text{and } I2(x, i_{out}) = min\{i : \ i \neq i_{out} \text{ and } dist(x, p_i) \leq dist(x, p_j) \forall j \in \{1, \ldots, k\}\}. \end{aligned}$$

In contrast to [79] we extended the simple induced Delaunay graph to an weighted, undirected graph, reflecting the density between the centroids. An example is shown in figure 3.14.

The simple induced Delaunay graph can be determined with a linear scan over the data, which costs time in $O(d \cdot n \cdot k)$. The determination of the weight costs time in $O(\#(E_{ID} \cdot r \cdot time_{DE})$ with $\#(E_{ID})$ denoting the number of edges in the induced Delaunay graph.

Now we compare the induced Delaunay graph and the complete graph. The number of edges of the induced Delaunay graph are in general lower or equal to the number of edges of the complete graph. The induced Delaunay graph has additional costs of an linear scan over the data. The use of this graph is preferable when the costs of the linear scan are amortized by the saved costs of density estimation for the lower number of edges. More formally, the use of the induced Delaunay graph G_{ID} has only advantages over the complete graph if G_{ID} is statistically convincing and $n \cdot k + \#(E_{ID}) \cdot r \cdot time_{DE} < \binom{k}{2} \cdot r \cdot time_{DE}$. From the inequations result the following conditions for the usage of the induced Delaunay graph instead of the complete cluster graph in case of the k centroid and the histogram density estimator:

$$k \text{ centroids: } \#(E_{ID}) < \frac{k(k-1)}{2} - \frac{n}{r} \text{ and Histograms: } \#(E_{ID}) < \frac{k(k-1)}{2} - \frac{n \cdot k}{r \cdot \log n}$$

In case of histograms, if $\log n$ is much smaller than k the complete graph is preferable because of the low costs for histogram density estimation.

In figure 3.15 we show the determined number of edges for the induced Delaunay graph for uniformly distributed data, randomly selected centroids and growing dimensionality. Uniformly distributed data are the worst case for the induced Delaunay graph, since the almost all possible edges are inserted. Figure 3.15 (a) shows that with growing dimensionality the number of edges approaches the theoretical bound for of $\binom{k}{2}$. Part (b) shows the number of edges which are saved in the induced Delaunay graph and the bound, when this number becomes to small to prefer this method.

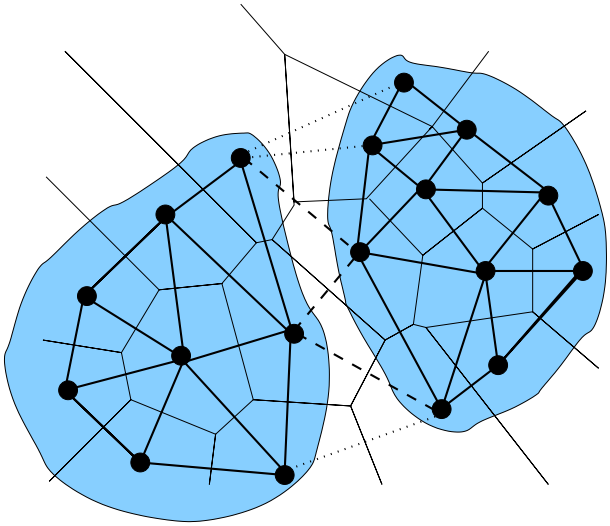
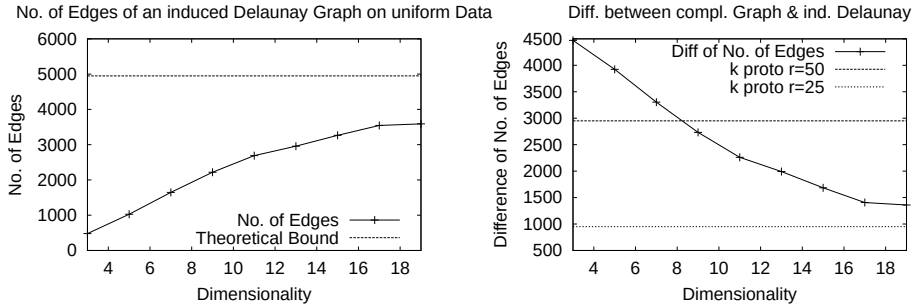


Figure 3.14: The figure schematically shows a Voronoi partitioning (thin solid lines) with an underlying data distribution (shaded areas) and the induced Delaunay graph (thick solid and dashed lines). The thick dotted edges belong to the full Delaunay graph, but are not inserted into the induced graph. The thick dashed edges have a low density and are deleted by the density based single linkage separator.



(a) (b)

Figure 3.15: Part (a) shows the number of edges of the induced Delaunay graph. The data sets used are uniformly distributed and have $n = 100000$ data points. The graph has $k = 100$ vertices and the upper bound for the number of edges is $\binom{100}{2} = 4950$. Part (b) shows the difference $\binom{100}{2} - \#(E_{ID})$. The lines show when both graphs have the same costs for the case the k prototype estimator and $r = 25$ and $r = 50$ is used. The complete graph gets better when the difference of edges is below the line.

From the inequation and the empirical observations we conclude that for data with a dimensionality $3 < d \leq 8$, when high accuracy is desired (larger values for r) and the k centroid estimator is used, the induced Delaunay graph is preferable as cluster graph. In case of $d \leq 3$ the full Delaunay graph should be used, because this graph can be computed without scanning the data points. In all other cases the use of the complete graph as initial cluster graph is the choice with the lowest costs.

The output of the separator algorithm is a partition of P into at least two subsets of centroids which approximates connected clusters and possibly a subset of centroids which approximates outliers P_0. If the initial cluster graph is not connected (which may only happen when an induced Delaunay graph is used) the while loop is not executed and it is possible that more than two clusters groups have been separated by the algorithms. When the initial cluster graph is connected, during the while loop the edges with lowest weights are deleted until the graph splits into two components. A data point x can be labeled (or assigned to a cluster) by looking for the nearest centroid $p_{I(x)} \in P$ and determining the index i of the subset of centroids with $p_{I(x)} \in P_i$. So the separating density valley is approximated by the Voronoi edges which correspond to the deleted edges in the induced Delaunay graph.

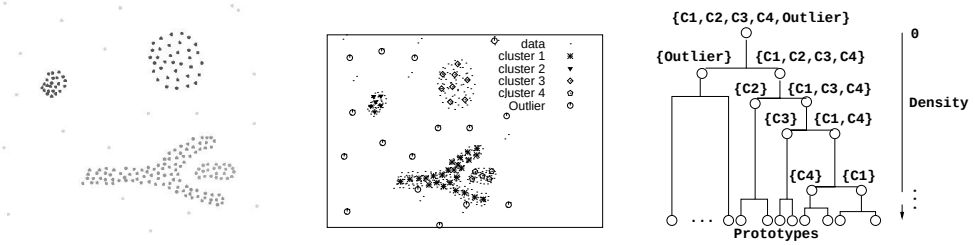
A recursive variant of the separator algorithm can be used to approximate the single-linkage hierarchy. In this case the subsets of centroids as well as the corresponding subgraphs of the cluster graph are inputs of recursive calls of the separator algorithm. The algorithm for approximating the hierarchy is described in pseudo code in algorithm 3. The intermediate cluster descriptions C can be stored in a tree to form the hierarchy.

Algorithm 3 Approximation of the Single-Linkage Hierarchy

db_slink_hierarchy($G(V = P, E, w), \hat{f}$ **)**
Require: $G(V = P, E, w)$ cluster graph
1: **if** $\#(P) = 1$ **then**
2: **return**
3: **end if**
4: $(P_1, P_2, \ldots, P_0, \xi) \leftarrow$ **db_slink_separator(**$G(V = P, E, w), \hat{f}$ **)**
{We assume G contains only nodes with $\hat{f}(p) \geq \xi$ and edges with $w(e) \geq \xi$ after the call.}
5: $C \leftarrow \{P_1, P_2, \ldots\}$ {Note that P_0 does not belong to the actual clustering C.}
6: **for all** $P_i \in C$ **do**
7: $V_{sub} \leftarrow P_i$, $E_{sub} \leftarrow \{e(p, p') : p, p' \in P$ and $e \in G.E$
8: **db_slink_hierarchy(**$G_{sub}(V_{sub}, E_{sub}, w), \hat{f}$ **)**
9: **end for**

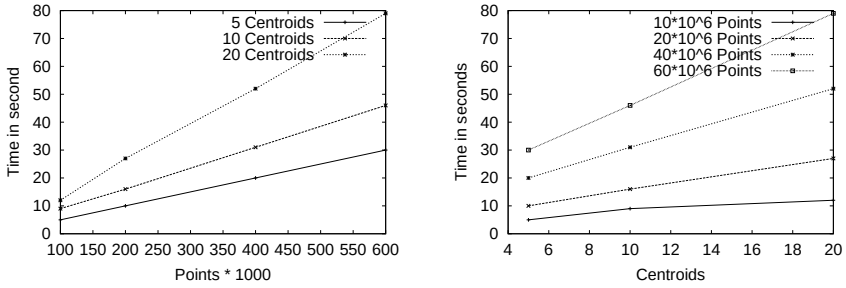
Figures 3.16 (a-b) shows a comparison of clusters found by DBSCAN and the approximation determined using **db_slink_hierarchy**. The results show that DBSCAN and the density-based single-linkage separator find arbitrary shaped cluster with comparable quality. Figure 3.16 (c) shows the upper part of the corresponding hierarchy. Relevant parts of the hierarchy can be integrated into the separation tree by using the density-based single-linkage separator recursively until all clusters are separated. The separation quality of the clusters on each level is reflected by the noise level ξ (or splitting density), which increases for each recursive splitting. Higher splitting density indicates low separation quality. The separator tree can also be used to exclude branches of the hierarchy form further separation. In that way arbitrary shaped clusters of different densities can be handled.

The runtime behavior of the whole process of clusters separation is dominated by the density estimator. In the runtime experiments we used LBG-U as a placement algorithm for the centroids of the density estimator and as reduction algorithm with the induced Delaunay Graph on the data set with dimensionality $d = 5$. In figure 3.17 we show empirically the overall runtime behavior of the density-based single-linkage separator + density estimator depending on the number of data points and the number of centroids. The linear runtime of the density-based single-linkage separator is a good improvement over existing methods like DBSCAN or OPTICS whose runtimes strongly



(a) DBSCAN (b) Primitives $k = 60$ (c) Dendrogram

Figure 3.16: Comparison of clusters found by DBSCAN and the approximation determined using the cluster primitives (k=60, density threshold=0.005) for an 2D Example of 400 data points. Part (a,b) show that DBSCAN and the density-based single-linkage separator are comparable wrt. the quality of the result. Part (c) shows the upper part of the single-linkage hierarchy.



(a) Time vs. No. of Points (b) Time vs. No. of Centroids

Figure 3.17: Part (a) shows the overall runtime behavior of the density-based single-linkage separator + the LBG-U based density estimator depending on the number of data points. The experiments indicate that the separator has a linear runtime wrt. number of data points. This is a good improvement over of existing methods like DBSCAN or OPTICS whose runtimes strongly depend on the underlying index. The use of the index causes a superlinear runtime behavior for both methods.

depend on the underlying index. The use of the index causes a superlinear runtime behavior for both methods.

3.3.4 Noise & Outlier Separators

Real data often contains objects which do not fit into clusters because the objects are independent from the rest. Our assumption is, that after the object's transformation into vector data, these objects are *noise* and are uniformly distributed in the feature space F or they are *outliers* and have a high LOF value.

The filtering of uniformly distributed noise is very important since the most placement methods for centroids are sensitive to noise and produce low quality results. It is well known that noise can be effectively filtered by multidimensional histograms (grids) [50, 55], but other efficient density estimators may be used. For the noise separator we use the observation that the density of uniformly distributed points is low and nearly equal. This fact is captured in the frequency distribution, which shows how often a density value occurs at a data point $x \in D$. For independent, uniformly distributed data points the normalized frequency distribution follows a binomial distribution $B(n, p)$ with n is the number of data points and p is the probability that a randomly picked point $x \in F$ (according to the uniform distribution) has density $den \in \mathbb{R}$. For a small p and a large n the binomial distribution

$$P_{binom}(k) = \binom{n}{k} p^k (1-p)^{n-k},\ 0 \leq k \leq n$$

can be approximated by a Poisson distribution

$$P_{poisson}(k) = \frac{\lambda^k \cdot e^{-\lambda}}{k!},\ \lambda = np,\ 0 \leq k \leq n$$

with the mean $\mu = \lambda = np$ (for further information about the probability distributions see [84]). Since it is hard to directly estimate p, we determine it from the mean using the relation $p = \mu/n$.

The data may contain data points which are not uniformly distributed, so we want to find the density threshold t with the property, that the frequency distribution below t follows a binomial distribution. Then a point $x \in D$ is noise if $\hat{f}(x) \leq t$. To find an appropriate value for t, the algorithm determines the χ^2-value for the intervals $[0, t']$, $t' \in [t_{min}, t_{max}]$ of the frequency distribution wrt. the estimated binomial distribution in the given interval. The χ^2-value reflects how well the interval $[0, t']$ fits a binomial distribution $B(p, n_{t'})$ with $\mu = mean([0, t'])$, $p = \mu/n_{t'}$ and $n_{t'} = \#\{x \in D \text{ and } \hat{f}(x) \leq t'\}$. Note that the observed frequency distribution has to be renormalized according to $n_{t'}$ in each step. The result t is the density threshold with the lowest χ^2-value, that means the frequency distribution in $[0, t]$ has the best fit with the estimated Binomial distribution for this interval. Algorithm 4 show this procedure in pseudo code. As alternative to the χ^2-test also the Kolmogorow-Smirnow test with the associated test variable can be used. The algorithm needs a lower bound t_{min} for the threshold t to have enough points to estimate the binomial distribution. A good choice for t_{min} is when $n_{t_{min}}$ is larger then 1000. The upper bound t_{max} is only for time saving purposes and can be set to $t_{max} = \infty$.

Algorithm 4 Noise Separator

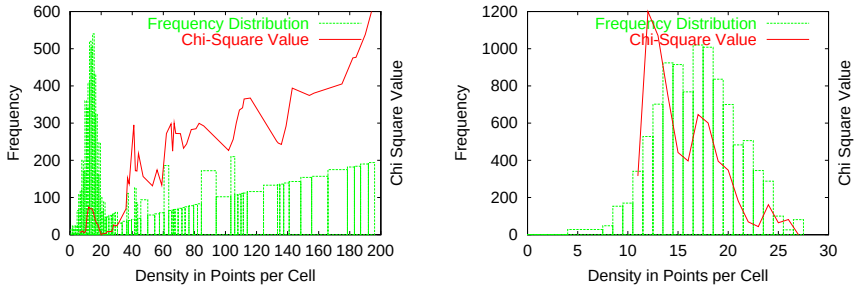
noise_separator$(D, \hat{f}, t_{min}, t_{max}, r)$

Require: The data set $D = \{x_1, \ldots, x_n\}$, a well set density estimator $\hat{f}(\cdot)$, minimal and maximal values for the desired density threshold t, the resolution $r > 0$ of the frequency plot.

Ensure: t is the density threshold in$[t_{min}, t_{max}]$ with best quality (lowest χ^2-value).

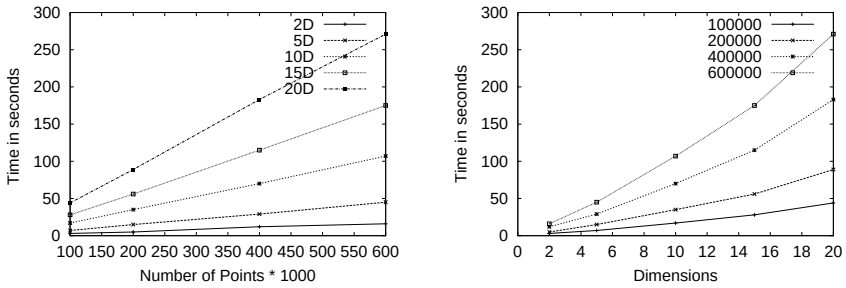
1: $max\_density \leftarrow max\{\hat{f}(x), x \in D\}$
2: **if** $t_{max} = \infty$ **then**
3: $\quad t_{max} \leftarrow max\_density$
4: **end if**
5: $frequency[0, \ldots, r-1] \leftarrow [0, \ldots, 0]$
6: **for all** $x \in D$ **do**
7: $\quad i \leftarrow \lfloor \frac{\hat{f}(x)}{max\_density} \cdot r \rfloor$
8: $\quad frequency[i] \leftarrow frequency[i] + 1$
9: **end for**
10: $i_{best} \leftarrow 0$, $q_{best} \leftarrow \infty$, $i_{min} \leftarrow \lfloor \frac{t_{min}}{max\_density} \cdot r \rfloor$, $i_{max} \leftarrow \lfloor \frac{t_{max}}{max\_density} \cdot (r-1) \rfloor$
11: $n_{t'} \leftarrow \sum_{i=0}^{i_{min}} frequency[i]$
12: **for** $i = i_{min}$ to i_{max} **do**
13: $\quad test[0, \ldots, i] \leftarrow frequency[0, \ldots, i]/n_{t'}$
$\quad$ {The division by $n_{t'}$ applies to each component which renormalizes the frequency distribution to the new interval.}
14: $\quad \mu \leftarrow mean(test)$, $p = \frac{\mu}{n_{t'}}$
15: $\quad q_i \leftarrow \chi^2\text{-function}\big(test, B(\mu, p)\big)$
16: $\quad$ **if** $q_i < q_{best}$ **then**
17: $\quad\quad q_{best} \leftarrow q_i$, $i_{best} = i$
18: $\quad$ **end if**
19: $\quad n_{t'} \leftarrow n_{t'} + frequency[i]$
20: **end for**
21: **return**$(t = \frac{i}{r-1} \cdot max\_density,\ q_{best})$

Figure 3.18 shows the frequency distribution and the χ^2 values for two different data sets. The experiments base on a histogram density estimator. Note that the density values and frequency



(a) Uniform Noise and 1 Cluster (b) Uniform Noise

Figure 3.18: Experiment for noise level estimation for 5D data sets with 5000 data points. A low χ^2-value corresponds to a good fit of the estimated Binomial distribution and the observed frequency distribution. For $t = 21$ (experiment (a)) 94% of the uniformly distributed points are separated.



(a) Time vs. No. of Points (b) Time vs. Dimensionality

Figure 3.19: Run Time Behavior of the Noise Separator based on a histogram density estimator

are not normalized in the figure. The data set used in figure 3.18(a) consists of uniform noise and one Gaussian cluster while the data set used in figure 3.18(b) only contains uniform noise. Both data sets have the dimensionality 5 and contain 5000 data points. In the figure, a good value for t corresponds to a small χ^2-value. For the data set shown in figure 3.18(a), our noise separation algorithm determines a noise level threshold of $t = 21$ (the first value when going from left to right, where the χ^2-values becomes zero) which almost perfectly separates the noise data from the cluster data (94% of the noise points are separated). For the data set shown in 3.18(b), our algorithm determines $t = 27$ as threshold with the result that all of the data is recognized as noise. Note that in the general case, the noise regions do not need to be connected in the feature space.

An interactive version of the algorithm may use a similar plot like in figure 3.18 to allow a good choice to t.

The noise separator itself has to store only the frequency distribution, which is very small. But the needed density estimator may have storage costs in $O(n)$, when a heap based histogram is used. But alternative density estimators are possible like the centroid based estimator, which might be combined with random sampling, LBG, LBG-U or BIRCH as placement methods. The runtime is dominated by the runtime needed for density estimation, since the operations on the small frequency distribution are negligible. In figure 3.19 we show empirically the run time behavior of the noise separator based on a histogram density estimator depending on the number of data points and the dimensionality.

The **outlier separator** has to determine the LOF value defined in section 3.2.1 for the data points and marks points with a high LOF value as outlier. The concept of LOF has been proposed by Breunig et al. in [29] and bases mainly on k nearest neighbor distances. In fact, to determine the LOF value the authors defined a local reachability density (in short $ldr(p)$) of an object p by averaging the k nearest neighbor distance of neighboring objects o' of p. It is important to see that $ldr(p)$ is a mixture of the naive estimator and the k nearest neighbor estimator (see [105]

pages 13, 19). Han, Tung and Jin [62] proposed a more scalable algorithm to find the top n_{top} local outliers which relies on a spatial data structure. However, the theoretical foundations in this paper have been left unchanged. The disadvantages of this algorithms are that their performance strongly depends on the underlying index, which is used to support the nearest-neighbor query. Since indices do not perform well with high dimensional data the methods becomes computationally expensive and the runtime changes from $O(n \log n)$ to $O(n^2)$.

We redesigned the LOF concept to work on general density functions. The general LOF value is defined as

$$LOF(x) = \frac{\frac{1}{\#(LN)} \sum_{x' \in LN} \hat{f}(x')}{\hat{f}(x)},\ x \in D,\ LN = \{x' \in F : dist(x, x') = l\}.$$

The ratio $LOF(x)$ is high, when the average density of the points in the local neighbor hood LN is high and the density at x is low. Note that LN does not contain data points, but randomly generated points around x. The number of points in LN is a parameter for the measurement. For small samples of the local neighborhood it is better to use the maximum density instead of the average density, since sample points in low density regions around x can distort the LOF value.

The determination of the average or the maximum density in the local neighborhood can be done by randomly picking points $x' \in F$ on the fly which are distributed equidistantly around the examined data point x. There is no need to store the points in LN. To find the top n_{top} local outliers we propose algorithm 5.

Algorithm 5 Local Outlier Separator

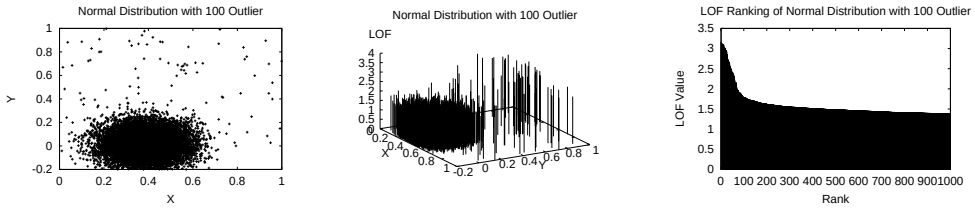
local_outlier_separator$(D, \hat{f}, m_{sample}, n_{top}, l)$

Require: The data set $D = \{x_1, \ldots, x_n\}$, a well set density estimator $\hat{f}(\cdot)$, the number of sample points m_{sample} for the local neighborhood and the number l of the top local outliers.

Ensure: D_0 contains the top l local outliers (data points with highest LOF-values).

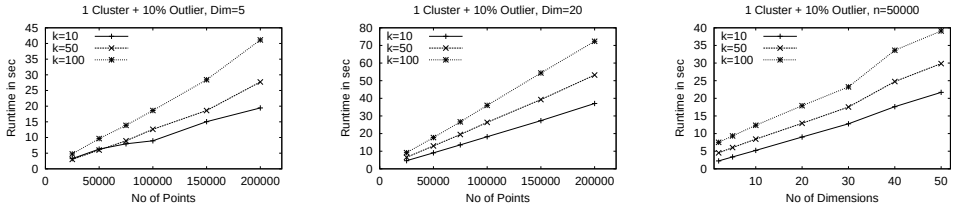
1: $D_0 \leftarrow \emptyset$
2: **for all** $x \in D$ **do**
3: $LN \leftarrow$ gen_sample(x, m_{sample})
$\{LN$ is a set of randomly generated sample points around x with $\forall x' \in LN :\ dist(x, x') = l\}$
4: $LOF(x) \leftarrow \frac{\frac{1}{\#(LN)} \sum_{x' \in LN} \hat{f}(x')}{\hat{f}(x)}$
5: **if** $\#(D_0) < n_{top}$ **then**
6: $D_0 \leftarrow D_0 \cup \{x\}$
7: **end if**
8: **if** $\#(D_0) = n_{top}$ and $\exists y \in D_0 : LOF(y) < LOF(x)$ **then**
9: $D_0 \leftarrow D_0 - \{y\}$
10: $D_0 \leftarrow D_0 \cup \{x\}$
11: **end if**
12: **end for**
13: **return**(D_0)

Runtime and space requirements depend on the underlying density estimator. For the experiments we used the k centroid density estimator. For this type of density estimator algorithm 5 becomes a runtime complexity of $O(n \cdot m_{sample} \cdot k \cdot d + n \cdot \log n_{top})$, where the first term corresponds to the LOF determination and the second to the filtering of the top n_{top} local outliers. If the LOF value should be determined for all points the last term changes to $n \log n$. The complexity for finding the top n_{top} outliers is much lower than those of the algorithms in [29, 62]. Figure 3.20 and 3.21 shows the results of our experiments.



(a) 2D data set (b) LOF Values (c) LOF Ranking

Figure 3.20: Part (a) shows the two dimensional data consisting of 10000 normally distributed points and 100 uniformly distributed points, which simulates the outliers. Part (b) shows the DB-LOF value for the data points, which states that the uniformly distributed points have a much higher LOF-value. Part (c) shows a ranking of the data points with the LOF values, starting with the highest LOF-value. The figure allows a visual determination of the top n local outliers, by looking for the knee in the plot. The figure shows that our DB-LOF notion also allows a nearly perfect detection of the outliers. The k centroid density estimator with $k = 100$ and random sampling as placement method have been used in the experiment.



(a) (b) (c)

Figure 3.21: For the experiments we used DB-LOF in combination with the k centroids density estimator with random sampling as placement algorithm. For this combination we expect a linear runtime for DB-LOF. In the experiments (a,b) we used 5 and 20-dimensional data with one cluster and 10% outliers. The number of the top outliers has been set to $n_{top} = 150$. Part (a) and (b) shows the runtimes of DB-LOF depending on the number of data points for different settings for k of the density estimator. For both data sets we get a linear total runtime which is a significant improvement over existing LOF algorithms, which depend on the use of an index. Part (c) shows the behavior of DB-LOF wrt. dimensionality. Also in this experiment we observed the expected linear runtime behavior. The small perturbations are due to caching effects of the operation system.

3.4 Improvements

In this section we discuss the improvements of our framework on existing approaches. The first improvement comes from the decoupling of density estimation and clustering scheme, which allows to find a reasonable tradeoff between efficiency and accuracy. As a result, methods with a better runtime complexity can be developed without loss of much accuracy.

The other improvement is that the separation framework better fits the situation of knowledge discovery and data mining, because within the framework semantic decisions (e.g. which type of separator is meaningful for an application) remain as such and are not transformed into quantitative parameterization problems [42] or into numerical optimizations of global cluster validation metrics [45]. We believe that the separator framework provides a language, which is useful for communicating problems related to clustering in a common scenario in knowledge discovery and data mining.

3.4.1 Improvements of Complexities

The first improvement comes from the decoupling of density estimation and clustering scheme, which allows to construct a clustering algorithm as a combination of a density estimation method and a specific clustering scheme. The choice of the density estimation method has a large impact on the quality and the efficiency of the whole clustering algorithm. It is a disadvantage that existing clustering algorithms strongly integrated both concepts, which results into a large variety of different algorithms. This issue is reported in a recent paper [33] with the title 'Why so many clustering algorithms?'. Our framework could be a helpful answer to this question because the number of clustering algorithms is reduced to a small number of primitives. This approach sets free much potential to find a reasonable tradeoff between efficiency and accuracy by combining primitives with the desired features.

So runtimes of the algorithms DBSCAN, OPTICS, and the LOF-Outlier detection strongly depend on the performance of the used indices. In the best case the complexity of the algorithms is in $O(n \log n)$. However it is well known that the best case performance decreases with increasing the dimensionality [110], which causes that the runtime complexity of the algorithms goes up to $O(n^2)$ which is prohibitive for large data bases. In section 3.3.3 and 3.3.4 we showed that using the centroid-based density estimator the runtime of the density-based single-linkage separator as well as the density-based LOF outlier detector can be brought down to $O(n)$ which is a significant performance gain, with only a small loss of accuracy. Using more expensive density estimators the quality can be improved but on the costs of a larger runtime. The advantage of the separator framework is that the tradeoff can be found by varying the separators and not by trying different clustering algorithms, which might use different parameters and different clustering schemes.

3.4.2 Benefits of the Framework

A first advantage of the separator framework is that growing the separator tree is a good way to apply the same separator method to different subsets using specially adopted parameter settings. This feature enables the user to build cluster hierarchies in a easy way. Cluster hierarchies often capture much more information about the structure of the data than non-hierarchic methods.

The separator framework also allows to integrate different clustering schemes into a global clustering model. The advantage here is that different schemes can be applied to subsets of the data. In that way the method can firstly generalize other methods and secondly integrate them to build more complex models for the data.

The second advantage is that the separator framework provides a language, which is useful for communicating problems related to clustering in a common scenario in knowledge discovery and data mining. This means that the design of different separators with different quality functions makes clear that there are different clustering schemes which are not comparable and that the choice of a clustering scheme is semantically dependent from the application.

In contrast to other clustering algorithms where density estimation is strongly integrated with the clustering scheme, the separator framework allows to handle the question of efficiency independently from the question of which clustering scheme is used. Efficiency questions can be handled by parameter setting, e.g. a larger number of centroids provides often better results but increases the runtime of the method. However, the question of which separator type is suitable for a given application remains as a semantic decision in the framework, which can only made by the user. By making the decision explicit the framework model provides a natural language for communicating problems related to clustering. The rationales why the decision is meaningful is left to the user. There are many approaches which are not aware of this semantic problem and offer highly biased solutions. However, we believe that in the field of knowledge discovery and data mining semantic decisions needs rationales, which can be only developed by the user, because only the user can decide what is meaningful for a given application. As our framework allows such semantic decisions, it is very suitable for a knowledge discovery context.

Chapter 4

Clustering in Projected Spaces

As mentioned in the previous chapter, clustering in high dimensional spaces has difficulties. In section 3.3.1 we showed that density estimation in high dimensional spaces degenerates wrt. effectiveness. The reason is the increasing sparsity of the data space, which comes from the exponential growing volume of the data space without a corresponding growth of the data sets. So from the statistical point of view we face the situation that we want to estimate a function over a fast growing attribute space with a nearly constant number of sample points. So it is obvious that the results loose significance in high dimensional spaces.

We approach the problem of clustering high dimensional data from different directions. In the first section in this chapter, we examine the behavior of distance metrics and similarity in high dimensional spaces, which we published in [52]. As a result we found a more general definition for nearest neighbor search in high dimensional spaces.

Since clustering is very dependent on the applied similarity notion, this leads us to the intuition of projected clusters and provides us with a useful interpretation. We published a preliminary approach to that problem in [55]. In section 4.3 we explore a new strategy and develop an new algorithm for mining projections. In contrast to recent approaches to projected clustering which starts with the high dimensional space, partition the data into subsets and reduce the dimensionality of the subsets, we start with low dimensional projections and combine them to higher dimensional ones using a frequent item set algorithm like the apriori algorithm. Another question is how many low dimensional projections are needed to find projected clusters. In the last sections of this chapter we present experimental results of our new approach to projected clustering and introduce an extension of our approach to more complex projected clusters.

4.1 Similarity in high dimensional Spaces

In the context of vector data it is a very common concept to use a vector metric as dissimilarity function. That's why we focus in this section on nearest neighbor search to find similar objects for a given query object.

Nearest neighbor search in high dimensional spaces is an interesting and important, but difficult problem. The traditional nearest neighbor problem of finding the nearest neighbor x_{NN} of a given query point $q \in \mathbb{R}^d$ in the database $D \subset \mathbb{R}^d$ is defined as

$$x_{NN} = \{x' \in D | \forall x \in D, x \neq x' : dist(x', q) \leq dist(x, q)\}.$$

Finding the closest matching object is important for a number of applications. Examples include similarity search in geometric databases [71, 81], multimedia databases [34, 100], and data mining applications such as fraud detection [24, 49], information retrieval [10, 90] among numerous other domains. Many of these domains contain applications in which the dimensionality of the representation is very high. For example, a typical feature extraction on an image will result in hundreds of dimensions.

Nearest neighbor problems are reasonably well solved for low dimensional applications for which efficient index structures have been proposed. Starting with the work on the R-Tree [43], a wide variety of multidimensional indexes have been proposed which work well for low dimensional data (see [40] for a comprehensive overview). These structures can support a wide range of queries such as point queries, range queries, or similarity queries to a predefined target. Many empirical studies have shown that traditional indexing methods fail in high dimensional spaces [20, 21, 110]. In such cases, almost the entire index is accessed by a single query. In fact, most indexes are handily beaten by the sequential scan [110] because of the simplicity of the latter.

However, as recent theoretical results [21] show, questions arise if the problem is actually meaningful for a wide range of data distributions and distance functions. This is an even more fundamental problem, since it deals with the *quality issue* of nearest neighbor search, in opposite to the *performance issue*. If the nearest neighbor problem is not meaningful to begin with, then the importance of designing efficient data structures to do the search is secondary. Here we deal with the quality issue of nearest neighbor search, and examine several theoretical and practical aspects of performing nearest neighbor queries in high dimensional space.

There can be several reasons for the meaninglessness of nearest neighbor search in high dimensional space. One of it is the sparsity of the data objects in the space, which is unavoidable. Based on that observation it has been shown in [21] that in high dimensional space all pairs of points are almost equidistant from each other for a wide range of data distributions and distance functions. In such cases, a nearest neighbor query is said to be *unstable*. However, the proposition of [21] is not that the difference between the distance of the nearest and the farthest data point to a given query point approaches zero with increasing dimensionality, but the authors proved that this difference does not increase as fast as the distance from the query point to the nearest points when the dimensionality goes to infinity. It is still an open question whether and when nearest neighbor search in high dimensional spaces is meaningful. One objective of this work is to qualify the results reported in [21].

It is useful to understand that high-dimensional nearest neighbor problems often arise in the context of data mining or other applications, in which the notion of similarity is not firmly predecided by the use of any particular distance function. Often applied metrics are instances of the L_p metric ($p = 1$, Manhattan; $p = 2$, euclidian) based on a comparision of all dimensions. In this context, many interesting questions arise as to whether the current notion of nearest neighbor search solves the right problem in high dimensions. If not, then what is the nearest neighbor in high dimensions? What is the meaning of the distance metric used? One of the problems of the current notion of nearest neighbor search is that it tends to give equal treatment to all features (dimensions), which however are not of equal importance. Furthermore, the importance of a given dimension may not even be independent of the query point itself.

In this section, we report some interesting experiments on the impact of different distance functions on the difference between the nearest and farthest neighbor. As we will see, our findings do not contradict the findings of [21] but provide interesting new insights. We discuss why the concept of nearest neighbor search in high dimensional feature spaces may fail to produce meaningful results. For that purpose, we classify the high dimensional data by their meaning. Based on our discussion and experiments, we introduce a new generalized notion of nearest neighbor search which does not treat all dimensions equally but uses a quality criterion to assess the importance of the dimensions with respect to a given query. We show that this generalized notion of nearest neighbor search, which we call *projected nearest neighbor search*, is the actually relevant one for a class of high dimensional data and develop an efficient and effective algorithm which solves the problem.

The projected nearest neighbor problem is a much more difficult problem than the traditional nearest neighbor problem because it needs to examine the proximity of the points in the database with respect to an unknown combination of dimensions. Interesting combinations of dimensions can be determined based on the inherent properties of the data and the query point which together provide some specific notion of locality. Note that the projected nearest neighbor problem is closely related to the problem of projected clustering [3, 4] which determines clusters in the database by examining points and dimensions which also define some specific notion of data locality.

d	Dimensionality of the data space
N	Number of data points
$\mathcal{F}$	1-dimensional data distribution in $(0, 1)$
$\mu_{\mathcal{F}}$	Mean of $\mathcal{F}$
X_d	Data point from $\mathcal{F}^d$, each coordinate follows $\mathcal{F}$
$dist_d(\cdot, \cdot)$	Symmetric distance function in $[0, 1]^d$, with $dist_d(\cdot, \cdot) \geq 0$ and triangle inequality
$\|\cdot\|$	Distance of a vector to the origin $(0, \ldots, 0)$
$Dmax_d = \max\{\|X_d\|\}$	maximum distance from the origin
$Dmin_d = \min\{\|X_d\|\}$	minimum distance from the origin
$P[e]$	Probability of event e
$E[X]$, $var[X]$	Expected value and variance of a random variable X
$Y_d \rightarrow_p c$	A sequences of vectors $Y_1, \ldots$ converges in probability to a constant vector c if: $\forall \epsilon > 0 \; lim_{d \rightarrow \infty} P[dist_d(Y_d, c) \leq \epsilon] = 1$

Table 4.1: Notations and Basic Definitions

4.1.1 Nearest Neighbor Search in high-dimensional Spaces

The results of [21] show that the relative contrast of the distances between the different points in the data set decreases with increasing dimensionality. In this section we first present some interesting theoretical and practical results which extend the results presented in [21]. The outcome is very interesting since – despite the pessimistic conclusions of [21] – the results show that meaningful nearest-neighbor search in high dimensions may be possible under certain circumstances.

Theoretical Considerations

Let us first recall the important result discussed in Beyer et. al. [21] which shows that in high dimensions nearest neighbor queries become unstable. Let $Dmin_d$ be the distance of the query point to the nearest neighbor and $Dmax_d$ the distance of the query point to the farthest neighbor in d-dimensional space (see table 4.1 for formal definitions).

The theorem by Beyer et. al. states that under certain rather general preconditions the difference between the distances of the nearest and farthest points ($Dmax_d - Dmin_d$) does not increase with dimensionality as fast as $Dmin_d$. In other words, the ratio of $Dmax_d - Dmin_d$ to $Dmin_d$ converges to zero with increasing dimensionality. Using the definitions given in table 4.1, the theorem by Beyer et al. can be formally stated as follows.

Theorem 1
If $lim_{d \rightarrow \infty} \; var\left(\frac{\|X_d\|}{E[\|X_d\|]}\right) = 0$, *then*

$$\frac{Dmax_d - Dmin_d}{Dmin_d} \rightarrow_p 0.$$

Proof: See [21]. ■

The theorem shows that in high dimensional space the difference of the distances of farthest and nearest points to some query point[1] does not increase as fast as the minimum of the two. This is obviously a problem since it indicates poor discrimination of the nearest and farthest points with respect to the query point.

[1] For our theoretical considerations, we consistently use the origin as the query point. This choice does not affect the generality of our results, though it simplifies our algebra considerably.

Metric	$Dmax - Dmin$ converges against
L_1	$C_1 * \sqrt{(d)}$
L_2	C_2
$L_k, k \geq 3$	0

Table 4.2: Consequences of Theorem 2

It is interesting however to observe that the difference between nearest and farthest neighbor ($Dmax_d - Dmin_d$) does not necessarily go to zero. In contrast, the development of ($Dmax_d - Dmin_d$) with d largely depends on the distance metric used and may actually grow with the dimensionality for certain distance metrics. The following theorem summarizes this new insight and formally states the dependency between ($Dmax_d - Dmin_d$) and the distance metric used. It allows to draw conclusions for specific metrics such as the Manhattan distance (L_1), Euclidean metric (L_2), and the general k-norm L_k.

Theorem 2
Let $\mathcal{F}$ be an arbitrary distribution of two points and the distance function $\|\cdot\|$ be an L_k metric. Then,

$$lim_{d\to\infty} E\left[\frac{Dmax_d^k - Dmin_d^k}{d^{1/k-1/2}}\right] = C_k,$$

where C_k is some constant dependent on k.

Proof: see [52]. ■
We can easily generalize the result for a database of N uniformly distributed points. The following theorem provides the result.

Theorem 3
Let $\mathcal{F}$ be an arbitrary distribution of n points and the distance function $\|\cdot\|$ be an L_k metric. Then,

$$C_k \leq lim_{d\to\infty} E\left[\frac{Dmax_d^k - Dmin_d^k}{d^{(1/k)-(1/2)}}\right] \leq (n-1)\cdot C_k,$$

where C_k is some constant dependent on k.

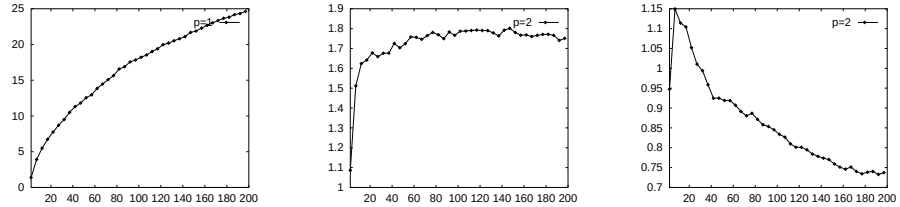
Proof: If C is the expected difference between the maximum and minimum of two randomly drawn points, then the same value for n points drawn from the same distribution must be in the range $[C, (n-1)\cdot C]$. ■

A surprising consequence of theorem 2 is that the value of $Dmax_d - Dmin_d$ grows (in absolute terms) as $d^{(1/k)-(1/2)}$. As a result, $Dmax_d - Dmin_d$ increases with dimensionality as $\sqrt{d}$ for the Manhattan metric (L_1 *metric*). The L_1 metric is the only metric for which the absolute difference between nearest and farthest neighbor increases with the dimensionality. It is also surprising that for the Euclidean metric (L_2 *metric*), $Dmax_d - Dmin_d$ converges to a constant, and for distance metrics L_k for $k \geq 3$, $Dmax_d - Dmin_d$ converges to zero with increasing d. These consequences of theorem 2 are summarized in table 4.2.

Experimental Confirmation

We performed a series of experiments to confirm these theoretical results. For the experiments we used synthetic (uniform and clustered) as well as real data sets. In figure 4.1, we show the average $Dmax - Dmin$ of a number of query points plotted over d for different data distributions. Note that the resulting curves depend on the number of data points in the data set.

Note that these experimental results are no contradiction to the results of [21]. The reason that even for the L_1 and L_2 metrics $\frac{Dmax_d - Dmin_d}{Dmin_d} \to_p 0$ is that $Dmin_d$ grows faster with d than



(a) Manhattan (L_1) (b) Euclid (L_2) (c) $L_3 Metric$

Figure 4.1: $|Dmax - Dmin|$ depending on d for different L_k metrics (uniform data)

$Dmax_d - Dmin_d$. In case of the L_1 metric, $Dmin_d$ grows linearly with d and in case of the L_2 metric, $Dmin_d$ grows as $\sqrt{d}$ with d. As a result, for the L_1 metric $lim_{d\to\infty}\frac{\sqrt{d}}{d} = 0$ and for the L_2 metric $lim_{d\to\infty}\frac{C_2}{\sqrt{d}} = 0$.

The theoretical and experimental results of this section show that for L_k metrics with $k \geq 3$, nearest neighbor search in high dimensional spaces is meaningless while for the L_1 and L_2 metrics the distances may reveal important properties of the data.

4.1.2 Problems of high dimensional data and meaningful nearest neighbor

In one- or two-dimensional spaces, it is usually rather easy to understand the properties of the data and identify the data distribution. It is safe to assume that all dimensions are equally relevant and that a standard (Euclidean) metric provides meaningful results. In general, this is not true in the high-dimensional case.

To get a deeper understanding of the nature of high dimensional data, it is important to uncover the meaning of the dimensions. High dimensional data points or feature vectors are typically derived from complex real world objects like products, images, CAD data, etc. There are three main methods to derive a high dimensional feature vector from complex real world objects, namely

- enumerating some properties of the objects (irreversible transformation),
- determining histograms which describe some statistical properties of the objects (irreversible transformation) or
- transforming the full description of the objects into a feature vector (reversible transformation).

In the following, we examine the impact of the three potential sources of high dimensional data to the meaningfulness of the nearest neighbor problem.

Enumeration of Properties: We use an example in order to elucidate this case. For our example we assume that we want to compare cars. Comparing cars is often done by deriving various properties of the cars such as motor power, equipment, design and so on. Each measurement forms a dimension which is only related to the other measurements of the same object. When users query the car data base, they can select or weight the importance of the different properties, and in that way each user is able to form his own meaningful distance metric. The reason why a user can easily perform a meaningful nearest neighbor search is that the dimensions are directly interpretable by the user. By omitting some of the dimensions and by weighting them the user can control the degree of abstraction for the nearest neighbor search. In our experience, the dimensionality of such data is in the medium range (10 to 50).

Determination of Histograms: Histograms are often used to produce high dimensional data because they allow a flexible description of complex properties of real world objects. Examples are color histograms [44], word counts for document retrieval and text mining [72, 90] and census data [83]. Each bin of the histogram is taken as a single dimension. The information transformation from the real world object into the histogram is an irreversible process which means that some

information about the object is lost. The user of a histogram data base has to be aware of this. The goal of the query has to match the reduced information of the transformed object. On the other hand the histogram may contain information about aspects (for instance the background in an image) the user wants to abstract from. In that case, the information in the histogram must be reduced to the relevant portion. However, in contrast to the enumeration method the users are generally not able to specify the reduction because they usually do not know the underlying transformation. Another difference to the previous method is that it is not useful to group the dimensions independent from the users and the query points. In general all possible groupings are potentially meaningful. First approaches to deal with that problem of query specification are reported in [13, 34]. In general the connection between the information in the histograms and the semantic information of the objects is weak. The dimensionality of such data can vary from the medium to large range (10 to 1000).

Full Feature Description: The third method is to use the description of complex objects directly as a feature vector. The advantage is that all information about the object is stored in the feature vector and that the object is reconstructible from the vector. However, often the real world objects do not allow a representation as a feature vector with fixed length. Examples for data which allow such a representation are molecular biology data [31]. Like the histogram data, it is also not meaningful to group the dimensions to sensible units independently from the query point and/or the user. Due to the possibility of reconstruction, the semantic aspects are strongly connected to the information stored in the feature vectors.

The three types of high dimensional data relate to different aspects of *meaningfulness*. In general there is not a single meaningful nearest neighbor for a query, but the user has to select the desired aspects. For the first category of high dimensional data, the user is able to specify his/her notion of 'meaningfulness' (the actual relevant aspects) by his knowledge about the real world objects. To deal with the second and third types of data, the user needs help from the data creator or the database system to specify the 'meaningful' aspects. But how does a specification assistance for the relevant aspects may look like? For certain applications, there are data dependent methods which use interaction in the selection process [34]. In this part of the work, we focus on a data independent method which selects the relevant dimensions automatically by extracting and rating additional information about the data distributions.

As a second question we investigate how far a single metric can serve as a similarity measure for the second and third type of data. We already stated that for those types of data the relevant dimensions (attributes) depend on the query point and the intention of the user. If the meaningfulness of a metric depends on the query point, then a metric can not serve as a measure of similarity between the query object and all other objects. In other words, a metric which is only based on the relevant attributes (which are assumed to be a subset of all attributes) can only serve as a criterion for similarity in a local environment of the query point. Objects (or data points) outside this environment are incomparable to the query object, because they may have other relevant attributes. In summary one can say that for the second and third type of data, the relationship between the metric and the intended similarity measure becomes weaker with increasing distance to the query point. As a consequence, meaningful metrics for high dimensional data spaces have to be varied according to the considered query point and data objects under consideration. Our generalized notion of nearest neighbor search which is presented in the next section provides an automatic adaptation of the similarity measure in order to allow a meaningful nearest neighbor search in high dimensional space.

4.1.3 Generalized Nearest Neighbor Search

From the previous sections we have seen, that the problem of finding a meaningful nearest neighbor in high dimensional spaces consists of the following two steps: First, an appropriate metric has to be determined, and second, the nearest neighbor with respect to this metric has to be determined. The first step deals with selecting and weighting the relevant dimensions according to the users intention and the given query point. This step is obviously rather difficult since it is difficult to select and weight the relevant dimensions among hundreds of dimensions. The basic idea of our

approach is to automatically determine the relevant dimensions for a given query point based on the properties of the data distribution. Although our approach can not guess the users intention, in general the data distribution contains highly relevant information and allows a much better and more meaningful nearest neighbor search.

Definition

In this section, we propose a generalization of the nearest neighbor search problem which remains meaningful in high-dimensional spaces. The basic idea of our new notion of nearest neighbor search is to use a quality criterion to dynamically determine which dimensions are relevant for a given query point and use those dimensions to determine the nearest neighbor[2]. The space of all subsets of dimensions can also be seen as the space of orthogonal projections of the data set, and the problem can therefore be defined as an optimization problem over the space of projections. In the following, we formalize our generalized notion of nearest neighbor search. First, we formally introduce a quality criterion which is used to rate the usefulness of a certain combination of dimensions (projection).

Let $D = \{x_1, \ldots, x_n\},\ x \in \mathbb{R}^d$ be a database of d-dimensional feature vectors, $x_q \in \mathbb{R}^d$ the query point, $p : \mathbb{R}^d \to \mathbb{R}^{d'},\ d' \leq d$ a projection, and $dist(\cdot, \cdot)$ a distance function in the projected feature space.

Definition 9 (Quality Criterion)
The **quality criterion** *is a function $C(p, x_q, D, dist) \to \mathbb{R},\ C \geq 0$ which rates the quality of the projection with respect to the query point, database, and distance function. In other words, the quality function rates the meaningfulness of the projection p for the nearest neighbor search.*

In section 4.1.4 we develop a useful quality criterion based on the distance distribution of the data points to the query point within a given projection.

Let P be the space of all possible projections $p : \mathbb{R}^d \to \mathbb{R}^{d'},\ d' \leq d$ and $\forall x \in \mathbb{R}^d : p(p(x)) = p(x)$. To find a meaningful nearest neighbor for a given query point x_q we have to optimize the quality criterion C over the space of projections P.

Definition 10 (Generalized Nearest Neighbor Search)
A meaningful nearest neighbor for a given query point $x_q \in \mathbb{R}^d$ is the point[3]

$$x_{NN} = \left\{ x' \in D | \forall x \in D, x \neq x' : dist\big(p_{best}(x'), p_{best}(x_q)\big) \leq dist\big(p_{best}(x), p_{best}(x_q)\big) \right\}$$

where $p_{best} = \left\{ p \in P | \underset{p:\mathbb{R}^d \to \mathbb{R}^{d'}, d' \leq d}{MAX} \{\ C(p, x_q, D, dist)\ \} \right\}$.

Solving the generalized nearest neighbor problem is a difficult and computation intensive task. The space of all general projections P is infinite and even the space of all axes-parallel projections has exponential size. In addition, the quality function C is a-priori unknown and therefore, it is difficult to find a general and efficiently computable solution of the problem. In the next section, we develop an algorithm which provides a very general solution of the problem.

4.1.4 Generalized Nearest Neighbor Algorithm

The most important but difficult task in solving the generalized nearest neighbor problem is to find the relevant projections. As mentioned in the previous subsections, this decision is in general query and data dependent which makes the problem computationally difficult. For our following

[2]Note that the nearest neighbor determined by our approach might be different from the nearest neighbor based on all dimensions.

[3]Note that our definition can be easily generalized to solve the k-nearest neighbor problem by fixing the selected projection and determining the k nearest neighbors.

considerations, we restrict the projections to the class of axes-parallel projections, which means that we are searching for meaningful combinations of dimensions (attributes). The restricted search space has still an exponential size with respect to dimensionality, which makes enumeration impossible for higher dimensionality. In order to keep our algorithm generic and allow different quality criteria (cf. subsection 4.1.4), our first approach was to use general optimization algorithms such as random search, genetic and greedy optimization, for which the implementations can be made largely independent of the specific problem structure. In random search, random combinations of dimensions are evaluated in terms of the quality criterion, and the best projection is returned. The genetic algorithm uses multiple populations which are mutated and combined based on the quality criterion, and the greedy algorithm directly uses the best one-dimensional projections which are combined into higher-dimensional ones. All three algorithms are sketched in pseudo code (see figures 6, 7 and 8).

Algorithm 6 Random Optimization

random_search $(x_q, d_{tar}, D, C, dist, no\_iter)$
  $p_{best}.quality \leftarrow 0$
  for $i \leftarrow 0$ **to** no_iter **do**
    $p \leftarrow$ generate_random_projection(d_{tar})
    $p.quality \leftarrow C(p, x_q, D, dist)$
    if $p_{best}.quality < p.quality$ **then**
      $p_{best} \leftarrow p$
    end if
  end for
  return(p_{best})

Algorithm 7 Genetic Optimization

genetic_search $(x_q, d_{tar}, D, C, dist, no\_iter)$
  $population \leftarrow \emptyset$, $pop\_size \leftarrow 100$, $elite \leftarrow 10$, $child \leftarrow 80$
  for $i := 0$ **to** pop_size **do**
    $p \leftarrow$ generate_random_projection(d_{tar})
    $p.quality \leftarrow C(p, x_q, D, dist)$
    $population.insert(p)$
  end for
  for $i \leftarrow 0$ **to** no_iter **do**
    $new\_pop \leftarrow \emptyset$
    insert the $elite$ best projection into new_pop
    for $j \leftarrow elite$ **to** $elite + child$ **do**
      {projections with high quality have higher probability to be selected for cross-over}
      $parent1 \leftarrow$ randomly select a projection from old_pop
      $parent2 \leftarrow$ randomly select a projection from old_pop
      $child \leftarrow$ gen. a new proj. by comb. $parent1$, $parent2$
      $child.quality \leftarrow C(p, x_q, D, dist)$
      new_pop.insert($child$)
    end for
    qualify and insert $pop\_size - (elite + child)$ random projections into new_pop
    $population \leftarrow new\_pop$
  end for
  select the best projection p_{best} and return it

The results of the first experiments showed that none of the three algorithms was able to find the relevant subset of dimensions. Even for synthetic data, for which the relevant subset of dimensions is known, only a subset of the relevant dimensions was found. We found that random search had been only useful to check whether a given quality criterion is effective on a specific data set or not. If the random search does not find any projection with good quality, both genetic and greedy

Algorithm 8 Greedy Optimization

greedy_search $(x_q, d_{tar}, D, C, dist, p_{tmp})$
  set of selected dimensions $S \leftarrow \emptyset$ or from p_{tmp}
  for $i \leftarrow 0$ **to** dim_{tar} **do**
    pick the dimension $k_i \notin S$ such that the quality of the projection based on $S \cup \{k_i\}$ is maximal
    $S \leftarrow S \cup \{k_i\}$
  end for
  return $(p_{best}(S))$

algorithm are likely to fail in finding a good projection as well. However, in cases when random search does not fail, the genetic search provides much better results. The greedy algorithm assumes that the influence of a dimension on the quality is independent from other dimensions. In general, this assumption is not true for real data sets. A crucial problem is that one-dimensional projections of high dimensional data usually do not contain much information and so the greedy algorithm picks the first dimensions randomly and is therefore not useful for selecting the first dimensions. It turned out, however, that the greedy algorithm can be used effectively to refine results from random or genetic search.

Algorithm 9 Generalized Nearest Neighbor Algorithm

p_nn_search $(x_q, d_{tar}, D, C, dist)$
  $d_{tmp} \leftarrow$ between 3 to 5
  $no\_iter \leftarrow$ between 10 to 20
  $p_{tmp} \leftarrow$ genetic_search($x_q, d_{tmp}, D, C, dist, no\_iter$)
  $p_{best} \leftarrow$ greedy_search($x_q, d_{tar}, D, C, dist, p_{tmp}$)
  $x_{NN} \leftarrow$ p_nn_search($x_q, D, dist, p_{best}$)
  return(x_{NN})

Our algorithm to determine the relevant subset of dimensions is therefore based on a combination of the genetic and the greedy algorithm. For determining the first three to five dimensions, we use a genetic algorithm and for extending the result to more dimensions we use a greedy-based search. Figure 9 shows the pseudocode of the algorithm. For controlling the degree of abstraction and improving the efficiency, we use the target dimensionality $d_{tar} = d' \leq d$ as a parameter of the algorithm. If the genetic algorithm determines the first five of the relevant dimensions and the greedy algorithm the remaining ones, the complexity of our algorithm is

$$O((5 \cdot \#(Iterations) \cdot PopulationSize + d \cdot (d_{tar} - 5)) \cdot O(\text{Quality Determination})).$$

Distance Distributions

In this section we develop a quality measure based on the distance distribution with respect to the query point. The distance distribution of a data set D with respect to a query point x_q is the distribution of distances of the data points $x \in D$ from x_q. More formally, we have to consider the probability that the distance of a query point x_q to another data point is smaller than a threshold $dist_t$:

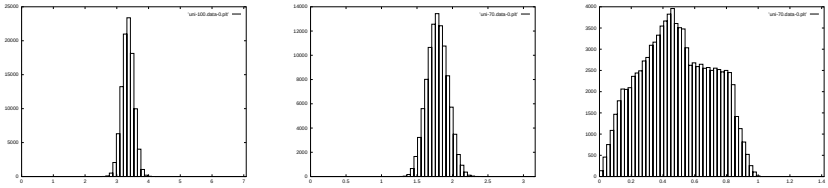
$$\Phi(dist_t) = P[dist(x_q, x) < dist_t], x \in D, dist_t \in \mathbb{R}$$

The corresponding probability density is

$$f(dist_t) = \Phi'(dist_t).$$

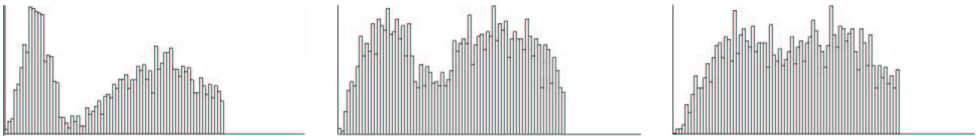
Note that $\Phi(dist_t)$ is not continuous and therefore we can only estimate the probability density $f(dist_t)$. In this subsection, we use simple histograms showing the distances of the data points from random query points.

To examine how typical distance distributions look like, we examine the distance distribution in different dimensionality. Let us first consider the case of high-dimensional uniform data. We know



(a) 50 Dimensions (b) 10 Dimensions (c) 2 Dimensions

Figure 4.2: Distance Distribution of Uniform Data



(a) all Dimensions (b) one non-rel. Dim. (c) two non-rel. Dim.

Figure 4.3: Distance Distribution of Data

that in this case the distances are meaningless. Figure 4.2 shows typical distance distributions[4] of a 50-dimensional data set consisting of 100,000 data points uniformly distributed in $[0, 1]^d$. Figure 4.2 (a)-(c) shows typical projections[5] onto randomly chosen 50, 10, and 2 dimensions. The distance distribution has always one peak which means that all data points are basically in one large distance cluster from the query point. As a consequence from the theorem in [21] the peak gets sharper as the dimensionality to the query point grows. We avoid this effect for our quality criterion by estimating the density only in the range $[d_{min}, d_{max}]$, because this effect is common to mostly all distributions and from section 4.1.1 we conclude that this effect does not necessarily tell something about the meaningfulness of the nearest neighbor. From the discussion in section 4.1.2 we conclude that a meaningful distance distribution should show two peaks. The nearer peak is formed by the points which are comparable to the query point (the metric is related to a type of similarity). The other peak – in most cases the larger one – is formed by those points which are incomparable to the query point because other attributes are relevant for those data objects. However, with respect to the currently used attributes they are assumed to behave like uniformly distributed data.

How to detect a two peak distance distribution? Our idea is to use kernel density estimation (see [105] for an introduction) to smooth the distribution and suppress random artifacts. To measure the quality we increase the kernel width (smoothing factor) until the smoothed distribution yields only two maxima. The obtained kernel width is h_1. Then we increase the kernel width until the distance distribution yields only one maximum. This results in the kernel width h_2. We use the difference between the smoothing factor for one maximum and for two maxima $h_2 - h_1$ as our quality criterion to measure the similarity of a current distance distribution with a distance distribution that yields two significant peaks. To get rid of possible disturbances in the distribution, which may also result in two maxima, we use only the k nearest percent of the data.

Figure 4.3 shows distance distributions of data, which contains uniformly distributed data and a projected cluster, which means that these points follow a Gaussian distribution in some dimensions and a uniform distribution in the others. Figure 4.3(a) shows the distance distribution in a projection where all dimensions are relevant, which means that all selected dimensions are used in the definition of the projected cluster. In Figure 4.3(b), one relevant dimension is replaced by a non-relevant and in Figure 4.3(c) two relevant dimensions are replaced by non-relevant ones. In 4.3(c) the two peak structure is hard to recognize and the quality criterion gives no hint on the hidden relevant dimensions. From these observations we can conclude that the genetic algorithm can only optimize projections with a dimensionality of 3-5. If the dimensionality is higher the quality criterion degenerates to an oracle and the algorithm can only guess a good projection –

[4] In case of uniform data, the distance distribution is always similar independent of the chosen query point.
[5] In case of uniform data, the distance distribution always looks the same independent of the chosen projection.

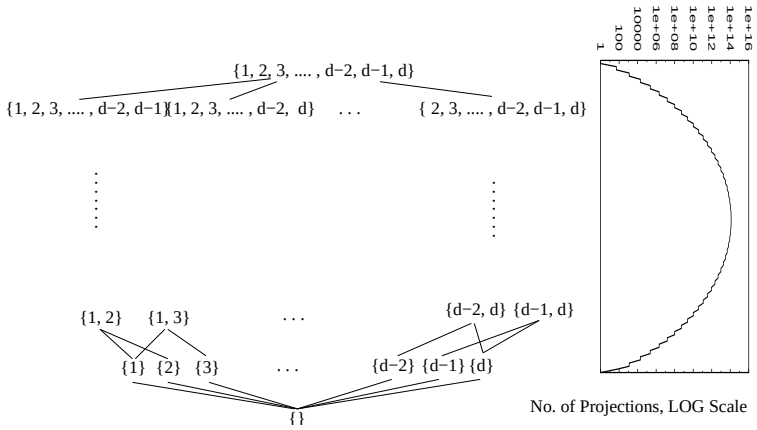


Figure 4.4: The space of axes parallel projections forms a lattices of subsets. The plot on the right side shows the number of projections of a fixed dimensionality d', which is like $\binom{d}{d'}$. The plot comes from a 50 dimensional space, but the proportions are similar in other spaces.

and the probability to guess a good projection in high dimensional data is rather low.

4.1.5 Summary

In this part of the chapter, we developed a generalized notion of nearest neighbor search in high dimensional spaces. In [52] we showed that our new notion is highly relevant in practical applications and improves the *effectiveness* of the search. The basic idea is to determine a relevant subset of dimensions depending on the query point and the data distribution by an optimization process which rates the distance distribution for the selected subset of dimensions according to an elaborate quality criterion. We also discussed some interesting aspects of using different L_p-distance metrics for finding the nearest neighbor. Our new technique for solving the generalized nearest neighbor problem is not only valuable for allowing a more meaningful and effective nearest neighbor search in high dimensional spaces but it also provides a better understanding of the data and the relevant notion of proximity. The ventilations leads us to a better understanding why clustering in projections can be useful.

4.2 Problems of existing Approaches for Projected Clustering

From the projected nearest neighbor problem we learned that the similarity between objects is better rendered by a distance metric in a low dimensional feature space than in a high dimensional one. So clustering in low dimensional projections of high dimensional spaces may yield several potentials to discover unknown structure in the data. In the first part of the section we will examine three general observations on mining projected spaces and draw connections to existing algorithms.

Firstly, we characterize the space of projections and restrict ourself to axes parallel projections. This subset of possible projections forms a subset lattice of the set of dimensions, which is sketched in figure 4.4. The complete enumeration of this space is not possible due its exponential size. Especially in the middle of the lattice the number of projections is very large, so any search strategy in this part is helplessly lost. We conclude from the figure that a search strategy has to focus on the high or low dimensional part of the subspace lattice.

Second, we want to recall the observation from the beginning of the chapter regarding the number of data points. To estimate the density in the space we can use only a nearly constant

number of data points, which becomes insignificant when the dimensionality gets higher. Growing dimensionality means exponential growing of the space volume, which is sampled by a constant number of data points. So we can expect that only the low dimensional projections yield significant information.

The last observation is that for projected clustering two tasks are necessary: the finding of the projections and the grouping of the data into clusters. Both tasks are dependent in the following way: the choice of the projection determines how similarity is defined and this definition induces the particular clustering of the data. The assumption for projected clustering is that a projection is meaningful for only a subset of the data points. Since the associated subsets of different projections may overlap in general a data point may belong to multiple projected clusters. This is different from full-dimensional clustering, where due to a global notion of similarity, clusters are found as partitions or nested partitions (in the hierarchical case) of the data. So algorithms should be able to assign data points to different clusters without assuming a cluster hierarchy. We argue that this leads to a fundamental change in the design of clustering algorithms.

Now we shortly review existing algorithms for the problem and draw connections to the general observations. Here we will focus on the method by which the space of projections is searched and in which order the two tasks (projection finding and data partitioning) are processed.

The algorithms PROCLUS [3] and ORCLUS [4] start in the full dimensional space and partition the data into many subgroups (seeds), reduce the dimensionality for the subgroups and join them if appropriate. So the algorithms mine the projection space top down following a greedy strategy. In each step of the greedy strategy the worst dimensions are removed. The dimensions are independently rated using statistical properties like variance or singular value. After the reduction the data points are reassigned to the cluster centers. During an iteration both algorithms use the following order of the two tasks: first partition the data, then finding of projections. Since the data points are assigned to exactly one cluster (partitioning) the algorithms can not detect overlapping clusters.

The CLIQUE algorithm [7] mines the projection space bottom up by searching quantitative frequent item sets (histogram bins) which are assembled to clusters on a single linkage basis. The order of tasks is first making the quantitative data discrete by partitioning the data set into regular histogram bins. Second, projections are searched by determining frequent item sets of the discrete data. The resulting frequent multidimensional histogram bins are used as building blocks for clusters. Overlapping clusters are possible here, but the clusters have to be reassembled in the projections from the frequent histogram bins.

The three algorithms have in common that they first split the data into arbitrary subgroups and then try to reassemble the subgroups to clusters according to their statistical and geometric properties. The subgroups are used to find projections with clusters. However, all algorithms have to deal with the problem to reassemble the previously splitted clusters.

Problems of Cluster Splitting We start with an examination of the impact of splitting the data first wrt. to high dimensionality as described in [55]. To investigate this issue we discuss the properties of different data distributions for an increasing number of dimensions. Let us first consider uniformly distributed data. It is well-known that uniform distributions are very unlikely in high-dimensional space. From a statistical point of view, it is even impossible to determine a uniform distribution in high-dimensional space a-posteriori. The reason is that there is no possibility to have enough data points to verify the data distribution by a statistical test with sufficient significance. Assume we want to characterize the distribution of a 50-dimensional data space by an grid-based histogram and we split each dimension only once at the center. The resulting space is cut into $2^{50} \sim 10^{14}$ cells. If we generate one trillion data points by a uniform random data generator, we get about 10^{12} cells filled with one data point which is about one percent of the cells. Since the grid is based on a very coarse partitioning (one cutting plane per dimension), it is impossible to determine a data distribution based on one percent of the cells. The available information could justify a number of different distributions including a uniform distribution. Statistically, the number of data points is not high enough to determine the distribution of the data.

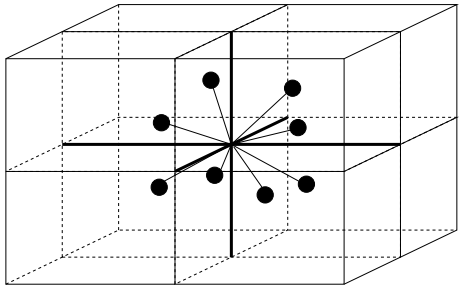


Figure 4.5: Example Scenario for a Normal Distribution, $d = 3$

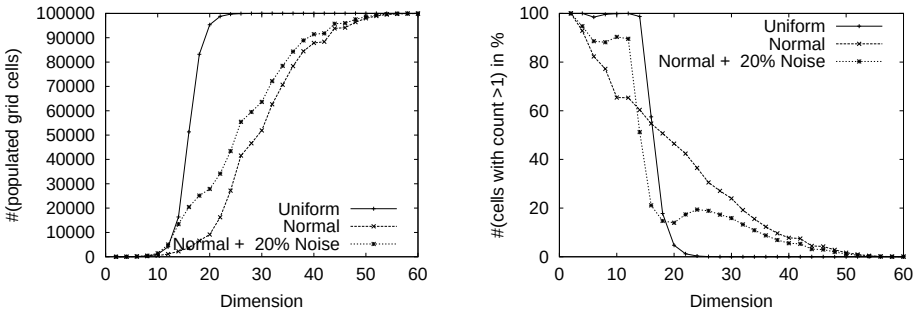
The problem is that the number of data points can not grow exponentially with the dimension, and therefore, in high-dimensional space it is generally impossible to determine the distribution of the data with sufficient statistical significance. (The only thing which can be verified easily is that the projections onto the dimensions follow a uniform distribution.) As a result of the sparsely filled space, it is very unlikely that data points are nearer to each other than the average distance between data points, and as a consequence, the difference between the distance to the nearest and the farthest neighbor of a data point goes to zero in high-dimensional space (see [21] for a recent theoretical proof of this fact).

Now let us look at normally distributed data. A normal distribution is characterized by the center point (expected value) and the standard deviation (σ). The distance distribution of the data points to the expected point follows a Gaussian curve but the direction from the expected point is randomly chosen without any preference. An important observation is that the number of possible directions from a point grows exponentially in the number of dimensions. As a result, the distance among the normally distributed data points increases with the number of dimensions although the distance to the center point still follows the same distribution. If we consider the density function of the data set, we find that it has a maximum at the center point although there may be no data points very close to the center point. This results from the fact that it is likely that the data points slightly vary in the value for one dimension but still the single point densities add up to the maximal density at the center point. The effect that in high dimensional spaces the point density can be high in empty areas is called the *empty space phenomenon* [105], Page 93 and [95].

To illustrate this effect, let us consider normally distributed data points in $[0, 1]^d$ having $(0.5, \ldots, 0.5)$ as center point and a grid based on splitting at 0.5 in each dimension. The number of directions from the center point now directly corresponds to the number of grid cells which is exponential in d (2^d). As a consequence, most data points will fall into separate grid cells (Figure 4.5 shows an example scenario for $d = 3$). In high dimensions, it is unlikely that there are any points in the center and that populated cell are adjacent to each other on a high-dimensional hyperplane which is again an explanation of the high inter-point distances.

To show the effects of high-dimensional spaces on split-first clustering approaches, we performed some interesting experiments based on using a simple grid as described and counting the number of populated grid cells. Figure 4.6 (a) shows the total number of populated cells (containing at least one data point) depending on the dimensionality. In the experiments, we used three data sets consisting of 100000 data points generated by a uniform distribution, a normal distribution with $\sigma = 0.1$ with a center uniformly distributed in $[0, 1)^d$ and a combination of both (20% of the data is uniformly distributed)[6]. Based on the considerations discussed above, it is clear that for the uniformly distributed data as many cells as possible are populated which is the number of available cells (for $d \leq 16$) and the number of data points (for $d \geq 20$). For normally distributed data, the number of populated grid cells is always lower but still converges against the number of data points for higher dimensions due to the many directions the points may vary from the center point. The third data set is a combination of the other two distributions and the speed of convergence

[6]The data points follows independently generated distributions in the projections.



(a) (b)

Figure 4.6: The figure (a) shows the number of populated grid cells for different data distributions (uniform, normal, normal + 20% noise), each sampled with 100000 data points. Figure (b) shows the percentage of cells with more than one data point in.

is between uniformly and normally distributed data. Figure 4.6 (b) shows the percentage of grid cells with more than one data point. The plot shows that for all data distributions the number of such cell goes towards zero in high dimensional spaces, so grid cells with more than one data point becomes unlikely in high dimensional spaces.

A general problem of clustering in high-dimensional spaces arises from the fact that the cluster centers can not be as easily identified as in lower dimensional cases. In grid-based approaches it is possible that clusters are split by some of the (d-1) dimensional cutting planes and the data points of the cluster are spread over many grid cells. Let us use a simple example to exemplify this situation. For simplification, we use a grid where each dimension is split only once. In general, such a grid is defined by d $(d-1)$-dimensional hyperplanes which cut the space into 2^d cells. All cutting planes are parallel to $(d-1)$ coordinate axes. By cutting the space into cells, the naturally neighborhood between the data points gets lost. A worst case scenario could be the following case. Assume the data points are in $[0,1]^d$ and each dimension is split at 0.5. The data points lie on a hypersphere with a small radius $\epsilon > 0$ round the split point $(0.5, 0.5, \ldots, 0.5)$. For $d > 40$, most of the points would be in separate grid cells despite the fact that they form a cluster. Note that there are 2^d cells adjacent to the split point. Figure 4.5 tries to show this situation of a worst case scenario for a three-dimensional data set. In high-dimensional data, this situation is likely to occur.

The experiments correspond directly to the CLIQUE approach since it also partitions the data space by binning the dimensions which results in a grid which probably splits clusters. CLIQUE connects adjacent grid cells and treat the connected cells as one cluster object. A naive approach to find the adjacent populated cells is to test all possible neighboring cells of a populated cell whether they are also populated. This approach however is prohibitively expensive in high-dimensional spaces because of the exponential number of adjacent neighbor grid cells. The other possibility is the test all populated grid cells whether they are adjacent to the actual one. However, this approach has quadratic run time in the number of populated grid cells, which is for high dimensional spaces in $O(n^2)$.

Similar arguments applies to the data splitting used in PROCLUS and ORCLUS. Here the data is splitted into Voronoi cells defined by centroids. The data points are assigned to a number of clusters seeds (centroids or medoids) using the nearest neighbor rule. The assumptions here is that each initial partition induced by a seed is homogenous and comes more or less from the same cluster. To guarantee this precondition the volumes of the Voronoi cells have to be sufficiently small. Otherwise the quality of the clustering may decrease. Since the distances between the data points grow fast in high dimensional spaces, a Voronoi splitting which meets the requirement of homogeneity ends with about one data point per Voronoi cell. This causes high costs for refining and merging the cells.

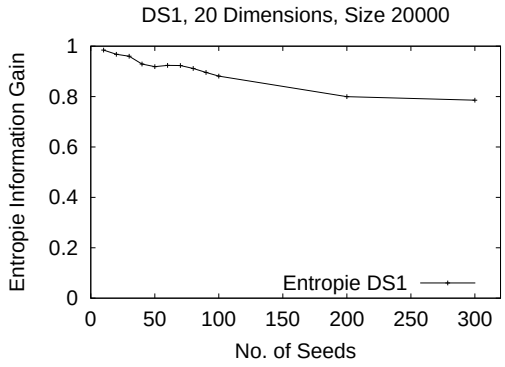


Figure 4.7: The diagram shows the entropy depending on the number of randomly chosen seeds for data set DS1. The data set consists of 20000 data points with 20 dimensions and contains two equally sized projected clusters. Each axes-parallel projected cluster has 3 relevant and 17 non-relevant dimensions. To simulate the initialization of ORCLUST we used random sampling to determine the seeds. The figure shows the entropy of the induced partitioning (using the nearest neighbor rule), which measures the homogeneity of the subsets. Low entropy (near to zero) indicates that the subsets mostly contain points from a single cluster. The entropy goes down with increasing the number of seeds, however the runtime of ORCLUS increases with the number of seeds quadratically. Because of the high computational costs we could not test an seed set with zero entropy. In cases the zero entropy condition is not met the ORCLUST algorithm is likely to converge to false relevant dimensions.

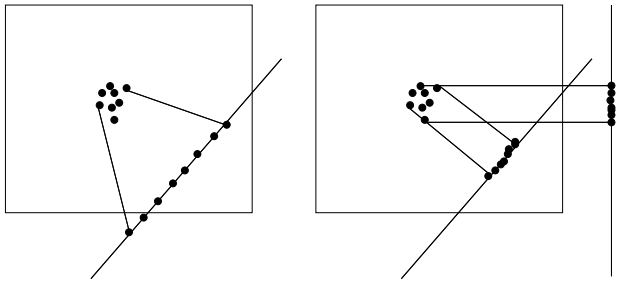
To elaborate the splitting issue we simulated the initialization of ORCLUS [7] by picking seeds randomly from the data as described in the paper [4]. We generated a 20-dimensional data set (to which is later referred as data set DS1) with two axes-parallel projected clusters of same size, each with three relevant dimensions (relevant dimensions are normally distributed, non-relevant dimensions are uniformly distributed). Since the data used has been labelled we could examine the entropy of the partitioning $D_1, \ldots, D_k$ of the data set D induced by the set of seeds S and the nearest neighbor rule. The entropy of the partitioning is defined as

$$entropy(D_1, \ldots, D_k) = \sum_{i=1}^{k} \frac{\#D_i}{\#D} \cdot entropy(D_i); \quad entropy(D_i) = -\sum_{j=1}^{c} p_j \log_2 p_j$$

with p_j denoting the frequency of cluster j in subset D_i. The entropy measures the homogeneity of the partitions according to the cluster labels. The entropy is near zero when the initial partitions are homogeneous. Figure 4.7 shows the dependence of the entropy from the number of seeds. Since the clusters in the data have only a few relevant dimensions and many non-relevant dimensions nearest neighbor rule for the initial partitioning is dominated by the non-relevant information. Note that ORCLUS's runtime depends quadratically on the number of seeds. So if the data contains many non-relevant dimensions, which means a large number of seeds are needed, the algorithm is not applicable due to the high computational costs. However, when the partitioning is not homogeneous (only a small number of seed is used) the local dimensionality reduction (principal component analysis) goes wrong, because the spawning vectors of the relevant subspace of one cluster are averaged (and so distorted) with the non-relevant dimensions from the other clusters. This causes, that the algorithm is likely to converge to false relevant dimensions.

As a consequence, any approach which considers the connections for handling the effect of splitted clusters will not work effectively and efficiently on large databases, and therefore another solution guaranteeing the effectiveness while preserving the efficiency is necessary for an effective clustering of high-dimensional data.

[7] Unfortunately the original ORCLUS algorithm is not available from IBM due to a pending patent.



(a) general (b) contracting

Figure 4.8: General and Contracting Projections.

4.3 A new projected clustering Algorithm

Our new approach searches first for good low dimensional projections and then groups the data into clusters. A projection has a high quality when data could be separated without splitting a cluster. So our approach has not to reassemble previously splitted clusters. Before examining projected clusters we introduce some general definitions and proof an important lemma. First we give the definition of contracting projections. The use of such projections avoids distortions of the projected data. The formal definition is given by:

Definition 11 (Contracting Projection)
A contracting projection for a given d-dimensional data space F and an appropriate metric $\|\cdot\|$ is a linear transformation P defined on all points $x \in F$

$$P(x) = Ax \text{ with } \|A\| = \max_{y \in F}\left(\frac{\|Ay\|}{\|y\|}\right) \leq 1 \ .$$

Figure 4.8 shows an example for general and contracting projections. An important question is how to separate clusters in projected spaces and ensure at the same time that no other cluster is splitted in the original high dimensional space. We proof an important lemma for the correctness of our non-cluster splitting approach, which states that the density at a point x' in a contracting projection of the data is an upper bound for the density at the points $x \in F$ with $P(x) = x'$ in the original space.

Lemma 12 (Upper Bound Property)
Let $P(x) = Ax$ with $P : \mathbb{R}^d \to \mathbb{R}^{d'}$ be a contracting projection, $P(D)$ the projection of the data set D, and $\hat{f}^{P(D)}(x')$ the density for a point $x' \in P(F)$. Then,

$$\forall x \in F \text{ with } P(x) = x' : \ \hat{f}^{P(D)}(x') \geq \hat{f}^{D}(x) \ .$$

with

$$\hat{f^D}(x) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right) \text{ and } \hat{f}^{P(D)}(x) = \frac{1}{nh^{d'}}\sum_{i=1}^{n} K\left(\frac{P(x) - P(x_i)}{h}\right).$$

Proof: First, we show that the distance between points becomes smaller by the contracting projection P. According to the definition of contracting projections, for all $x, y \in F$:

$$\|P(x) - P(y)\| = \|A(x - y)\| \leq \|A\| \cdot \|x - y\| \leq \|x - y\|$$

The density function which we assume to be kernel based depends monotonically on the distance of the data points. Since the distances between the data points in the projection becomes smaller, the density in the projected space $P(F)$ grows. ■

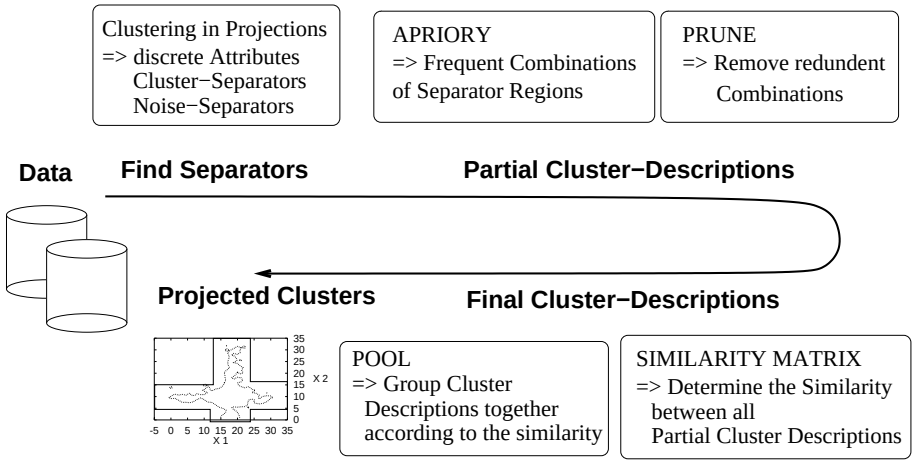


Figure 4.9: The first step is to find noise and cluster separators in projections of the data. This can be seen as a transformation of the continuous vector data into discrete transaction. A transaction consists of the separator regions which include the original data point. Second, the transactions are mined for frequent itemsets, which are frequent occurring combinations of separator regions. A frequent combination can be seen as a partial cluster description. To keep the complexity low redundant combination are pruned. In the last steps the similarities between the combinations are derived. Similar combinations are pooled to final cluster descriptions.

The assumption that the density is kernel based is not a real restriction. There are a number of proofs in the statistical literature that non-kernel based density estimation methods converge on a kernel based method [98]. Note that Lemma 12 is a generalization of the Monotonicity Lemma in [7]. Lemma 12 allows to determine clusters in a projection and ensures that the density at the border of an cluster does not exceed a fixed value.

In the following we characterize axes parallel projected clusters and develop an algorithm to determine such clusters. An axes parallel projected cluster is defined on a subset of dimensions (relevant attributes) and undefined on the other dimensions (non-relevant attributes). The points of such a cluster are assumed to follow independent distributions in all dimensions, that means there are no dependencies between the attributes. We also assume the non-relevant attributes to be uniformly distributed on the whole attribute ranges. The relevant attributes are clustered in sub-intervals of the attribute ranges.

Now we sketch the outline of our new algorithm. The first step is to find noise and cluster separators in projections of the data. This can be seen as a transformation of the continuous vector data into discrete transaction. A transaction consists of the separator regions which include the original data point. Second, the transactions are mined for frequent itemsets, which are frequent occurring combinations of separator regions. A frequent combination can be seen as a partial cluster description. To keep the complexity as low as possible redundant combinations are pruned. In the last steps the similarities between the combinations are derived. Similar combinations are pooled to final cluster descriptions. A final cluster description captures the properties of a projected cluster. The main difference is that the multiple cluster descriptions may refer to the same data point. So non-hierarchic, overlapping clusters may be found. The outline of our algorithm for projected cluster is sketched in figure 4.9.

4.3.1 Finding Separators

In this section we describe how to the find separators which serve for the discretization of the continuous vector data without splitting clusters. This step has a strong impact on the whole method. We propose to find the separators by looking only on the projected data (in a low dimensional subspace). For simplicity we describe first the case of axes-parallel projected clusters

and extent this approach to more general projected clusters afterwards.

Since axes parallel projected clusters are assumed to have no dependencies between the attributes the information to decide whether an single dimension is relevant or not can be obtained from the one dimensional projection onto the examined attribute. The challenge is that in the projection the distributions of all clusters are jammed and it is difficult to separate them. Two cases are possible, firstly distributions of two clusters are jammed and overlap each other, secondly a cluster distribution and non-relevant distributions are mixed. The separation of the distributions is not perfectly possible in the general case since data points drawn from different distributions may be projected onto the same position. So the tasks for examining an one dimensional projection is to separate the clusters from each other and from the non-clustered rest.

Due to the assumed independency of the data our algorithm examines only the one dimensional axes parallel projections to find good separators. A separator in this context consists of several $d-1$ dimensional hyperplanes, each defined by a split point in the same one-dimensional projection. The split points partition the attribute range into subintervals, however the separator partitions the feature space F into the same number of grid cells (or slices). A good split point is required to have low density in the projection, because this gives a low upper bound for the density on the $d-1$ dimensional hyperplane and minimizes the risk to split a cluster (see lemma 12).

Due to the two separation tasks we introduce two different separators, an one dimensional cluster separator and a noise separator. First we describe how to find the split points for cluster separation. Good cluster separating split points are local minima of the one dimensional density function. The quality of the separator built from a set of split points is the maximal density in the projection at a split point. The algorithm looks for separating minima of the density function which e.g. are not at a border of an attribute range and have sufficiently enough data points at both sides. The separating minima are found by examining the smoothed gradient of the density function and determining zero points of the gradient function. The density function is estimated as discrete histogram. The smoothed gradient of the discrete histogram function is defined as

$$grad(x, \hat{f}^{P(D)}) = \frac{1}{s}\left(\sum_{i=1}^{s} f^{P(D)}(x - i \cdot \epsilon) - \sum_{i=1}^{s} f^{P(D)}(x + i \cdot \epsilon)\right),\ \epsilon \in \mathbb{R}, s \in \mathbb{N}$$

where ϵ is the bin width of the histogram and s is the smoothing factor describing how many bins at both sides have to be averaged. The gradient is smoothed to make the procedure robust against small disturbances. A zero point x of smoothed gradient function is a minimum of the density function, if and only if $x - \epsilon < 0$ and $x + \epsilon > 0$. The other zero points are maxima. Figure 4.10 (b) shows the smoothed gradient function with the minima of the density function (a). Choosing a minimum of the density function as split point for a separator reduces also the probability of cluster splitting. However, not all minima necessarily separate clusters. To make a minimum to a separating one, the maximum density of both, the left and right neighboring intervals have to be above the noise threshold. In case of figure 4.10 (b) two of the three minima $\{2, 3, 4\}$ have to be deleted to make the remaining minimum to a separating one. In such a case the remaining minimum is chosen as the one with the lowest split density. Part (c) of figure 4.10 shows the cluster separator with the separating minima determined by the smoothed gradient. Algorithm 10 takes a set of minima M, the histogram density function f and the noise threshold ξ and shows in pseudo code how to determine the separating minima.

The intervals between the remaining, separating minima are directly mapped to separator regions using the hyperplanes defined by the split points. Formally the cluster separator for dimension j is given by an ascending ordered set of split points $SC = \{split_0, \ldots, split_{l_j-1}\}$. The separator function uses the set of split points SC and returns for an point $x \in F$ the minimal index of the split points, which are larger than the projection of the point.

$$x \in F,\ S_j^C(x) = \begin{cases} i_{min} & \text{if } \exists i_{min} = min\{i : i \in \{0, \ldots, l_j - 1\} \text{ and } P(x) \leq split_i\} \\ l_j & \text{else } P(x) > split_{l_j-1} \end{cases}$$

For convenience $S_j^{C,i}(F) \subset F$ denotes the subset of F with $x \in S_j^{C,i}(F) \Rightarrow S_j^C(x) = i$.

Algorithm 10 Determination of Separating Minima

separating_min$(M, f^{P(F)}, \xi)$

Require: $M = \{x_1, \ldots, x_{last}\}$ a set of ascending ordered minima, $f^{P(F)}$ the density function in projection P and ξ the noise threshold.

Ensure: M_{sep} contains only separating minima.

1: **for all** $x_i \in M$ **do**
2: $x_i.lMax \leftarrow$ Determine the maximum density in the left interval $[x_{i-1}, x_i]$
3: $x_{min}.rMax \leftarrow$ Determine the maximum density in the right interval $[x_{i-1}, x_i]$
{The left interval for x_1 is $[min, x_1]$ and the right interval for the last minium is $[x_{last}, max]$.}
4: **end for**
5: $i \leftarrow 1$; $M_{del} \leftarrow \emptyset$
6: **while** $x_i \in M$ and $x_i.lMax < \xi$ **do**
7: $M_{del} \leftarrow M_{del} \cup \{x_i\}$; $i \leftarrow i + 1$
8: **end while**
9: $left \leftarrow 0$; $right \leftarrow 0$
10: **for** i to $last$ **do**
11: **if** $x_i.lMax \geq \xi$ **then**
12: $left \leftarrow i$
13: **end if**
14: **if** $x_i.rMax \geq \xi$ **then**
15: $right \leftarrow i$
16: **end if**
17: **if** $left \neq 0$ and $right \neq 0$ **then**
18: $x_{remain} \leftarrow x_{remain} \in \{x_i : left \leq i \leq right\}$ and $f^{P(F)}(x_{remain}) \leq f^{P(F)}(x_i)$.
19: $M_{del} \leftarrow M_{del} \cup \{x_i : left \leq i \leq right\} - \{x_{remain}\}$
20: $left \leftarrow 0$; $right \leftarrow 0$
21: **end if**
22: **end for**
23: **if** $left \neq 0$ **then**
24: $M_{del} \leftarrow M_{del} \cup \{x_i : left \leq i \leq last\}$
25: **end if**
26: $M_{sep} \leftarrow M - M_{del}$
27: **return**(M_{sep})

The other task for examining a one dimensional projection is to separate clusters from the non-clustered rest. A noise separator can be used to separate points following a non-relevant distribution in the current dimension from other points, which are clustered in the dimension. Noise separators are only determined when no cluster separator is found.

To determine the noise threshold the methods described in section 3.3.4 could be used or the threshold is estimated by hand from an example visualization of the density. Due to the precondition that no separating minimum exists in the projection, a noise separator may consist of one or two split points, which mark intersections of the noise level with the density function. This corresponds to the following cases:

1. One split point: only the left or right part of the density function is above the noise level
2. Two split points: a middle part of the density function is completely above the noise level

As a consequence the density in exactly one interval is completely above the noise threshold. This interval is labeled as cluster interval. The following formula shows how a noise separator labels a point

$$x \in F,\ S_j^N(x) = \begin{cases} 1 & \text{if } f^{P(F)}(P(x)) \geq \xi \\ 0 & \text{else} \end{cases}$$

A point x with label 0 is marked as noise and with label 1 as cluster. Similar to the cluster separator $S_j^{N,i}(F) \subset F$ denotes the subset of F with $x \in S_j^{N,i}(F) \Rightarrow S_j^N(x) = i$. Figure 4.10 (d-f) shows an application of the noise separator.

The whole procedure described in this section is sketched in algorithm 11. To find projected clusters our algorithm tries to find for each axes parallel projection a cluster separator and, if no cluster separator exists, a noise separator. Note that for a given noise threshold a cluster separator or a noise separator do not necessarily exist. The found separators are collected in the sets S^C and S^N. Both separator algorithms require an histogram of each one-dimensional projection. Such

Algorithm 11 finding of Separators

separation(D, ξ)

```
Determine histograms for all projections P_1, ..., P_d
S^C ← ∅, S^N ← ∅ {Sets of Separators (Cluster, Noise)}
for all histograms h_1, ..., h_d do
   S^C ← S^C ∪ findClusterSeparator(h_i, ξ)
   if no Cluster Separator Found then
      S^N ← S^N ∪ findNoiseSeparator(h_i, ξ)
      if no Noise Separator found then
         mark the current dimension as non-relevant
      end if
   end if
end for
return(S^C, S^N)
```

histograms can be determined in one linear scan of the data set. The separator algorithms itself have also a linear runtime in the size of the histogram. So the runtime of this step is linear in the number of data points. Please note that our separator finding step has the two important difference to the CLIQUE approach. Firstly, CLIQUE splits the data using all dimensions and secondly, it uses equi-distant splits, which do not take the data distribution into account. This makes it very likely, that clusters are spread over multiple grid cells. As mentioned above, it is very costly to reassemble the grid cells, which belong to the same cluster.

4.3.2 Determining partial Cluster Descriptions

After determining separators in the projections the question arises how to use the determined separators. Each separator region can be seen as a primitive cluster description assigning data

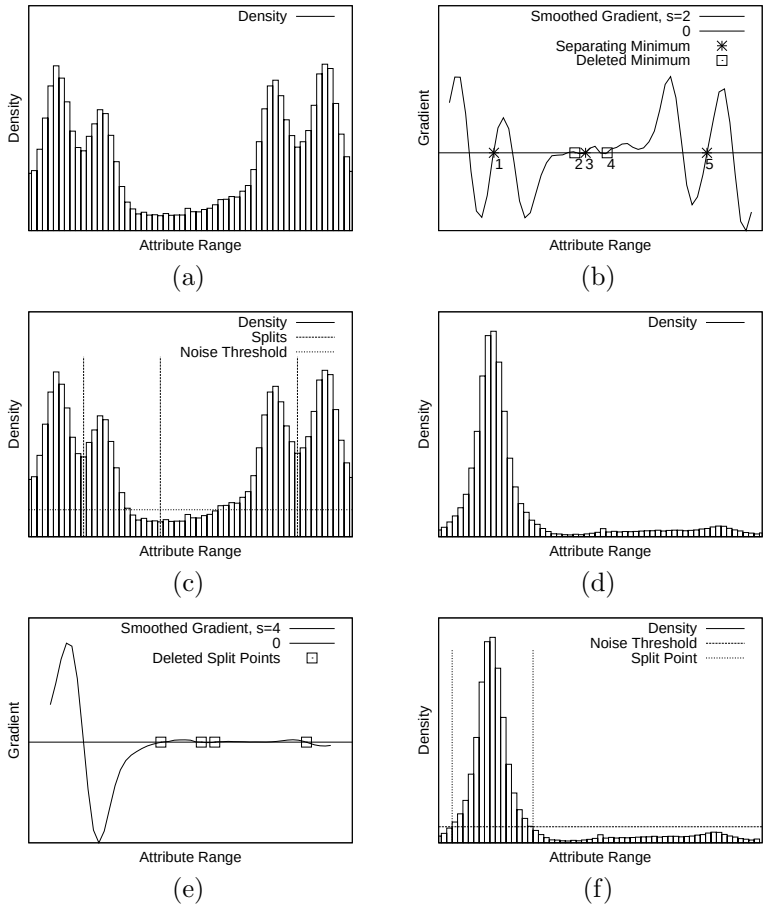


Figure 4.10: Part (a) shows a density function (histogram) of a one dimensional projection (source: attribute 5 of the molecular biology data). Part (b) shows the smoothed gradient (s=2) with the zero points for the minimums. Since not all minimums separates clusters, two of $\{2, 3, 4\}$ have to be deleted. The minimum with the smallest split density remains. Part (c) shows the separator with the three separating minimums. Part (d-f) shows a case where the noise threshold can be used to find noise splits. All gradient splits points are deleted since they do not separate clusters (see e). The noise splits are defined by the intersections of the noise level with the density function (see f).

points to a clusters. The point groups described by the separator regions may overlap, however, it is unknown which separator regions correspond to the same projected cluster.

Since the separators are independently determined a naive approach would merge all separators (see the merge-operation in section 3.2.1). This forms a grid, which partitions the data space F and so the data set D. Due to our restriction to one dimensional axes parallel projections the grid is a regular grid like in figure 4.11(a). Not all grid cells contain clusters, so a simple method to find cells with clusters is desired. An intuitive approach is to require the grid cells with clusters to contain at least a given number $minsup \in \mathbb{N}$ of data points. The problem is that the information from the one dimensional projections allows no determination of the number of points in the multidimensional grid cells.

Also the choice of separators with good quality (low split density) is no assurance to find grid cells with clusters. To illustrate this we describe a short example consisting of a data set $D \subset [0,1]^d$ and each data point in D is near to a different corner of the hypercube. When $d = 20$ the size of the data set is $2^{20} = 1048676$, which is quite large. Each of the 20 axes parallel projections may look like figure 4.11(b) and contains a cluster separator of good quality. However, merging all separators would result into a grid, whose grid cells contain only one data point despite the well chosen separators.

The problem is how to find combinations of separator regions, which are large with respect to the number of combined separators and are supported by a sufficiently large number of data points. Both goals – increasing the number of combined separator regions and maximizing the number of supporting data points – contradict each other. In terms of grid cells (a combination of separator regions is a grid cell) this means to find the smallest grid cells which contain more than $minsup$ data points. It is important to note that after the discretization of the continuous vector data into separator regions this problem can be transformed into a problem of finding frequent itemsets. This is done by denoting each separator region containing one or more clusters by an item. To convert a data point x into a transaction t those items are concatenated, which correspond to separator regions including x. The noise intervals of the noise separator are not represented by items, because they mark non-relevant attributes. The set of items I is constructed from the cluster separators S_j^C and the noise separators S_j^N in the following way:

$$I = \bigcup_{S_j^C \in S^C} \{C_j^0, \ldots, C_j^{l_j}\} \ \cup \bigcup_{S_j^N \in S^N} \{N_j^1\}$$

with l_j is the number of separator regions of the cluster separator for dimension j. The transaction set is determined from the data set using the separators. Each data point $x_l \in D, l = 1, \ldots, N$ is transformed into a transaction t_l, which consists of the following items:

$$t_l = \bigcup_{S_j^C \in S^C} \left\{C_j^i : S_j^C(x_l) = i\right\} \ \cup \bigcup_{S_j^N \in S^N} \left\{N_j^1 : \text{ if } S_j^N(x_l) = 1\right\}$$

Figure 4.11(b,c) shows an illustration of the construction. The determination of the frequent itemset wrt. $minsup$ can be done with any available algorithm like apriori [6], partitioning algorithm [92], sampling based [106] or the using the FP-tree method [46]. The data generated by the transformation of separators into items differs from typical buying records in the way that the data is not sparse, that means very frequent items may occur. The handling of such data is an important research topic. New algorithms have been proposed recently, which can handle dense datasets and are able to find long frequent itemsets [2]. It is important to note that single items with very high support ($\geq 80\%$) have an strong negative impact on the efficiency of the algorithms but add no substantial new information to the result. Such items come from separators which produce very unbalanced splits of the data. While the small parts may contain potentially important information on outliers, the large part simply represents nearly the whole data set again. Since this significantly increases the size of the result of any frequent set algorithm without adding new information, such items are removed before running the frequent item set algorithm. In our experiments we used a very fast implementation of apriori from Christian Borgelt [25].

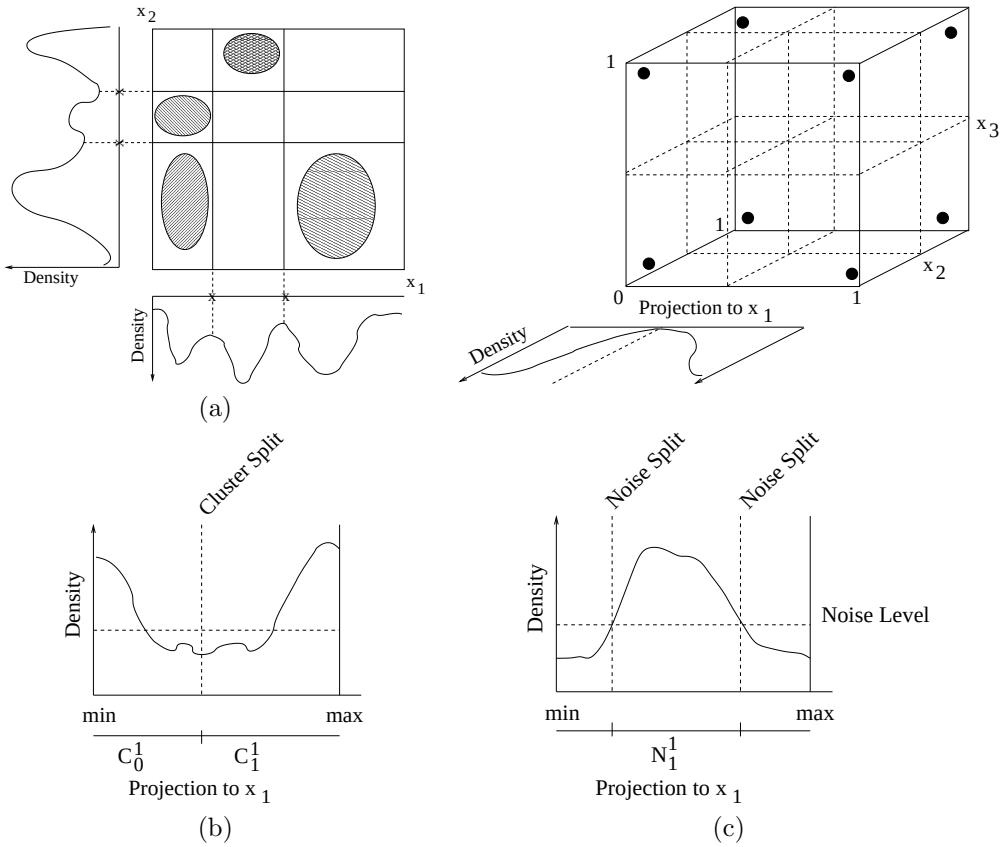


Figure 4.11: Part (a) shows a regular grid determined by separators from axes parallel one dimensional projections. It shows that clusters in the multidimensional space can not be determined by looking on one dimensional projections. Part (b) is an illustration for an example that the use of high quality separators is no assurance for finding good clusters. The example can be generalized to high dimensional spaces and each projections yields high quality separator. But the merging for all separators leads to a grid with only one data point in each cell. Part (c) illustrates the construction of the items from the separators.

The result of all algorithms is a set of frequent itemsets FI. We propose as a post-processing step to remove redundant itemsets. A redundant frequent itemset is covered by a frequent superset. The set of remaining frequent itemsets is denoted with PFI. The pruned frequent itemsets stand for grid cells which are not included by larger ones and contain a sufficiently large number of data points. The grid cells can be seen as conjunctions of separator regions from different separators, which describe projected clusters. We call this cluster descriptions partial ones since also in the pruned set similar descriptions may occur describing the same cluster.

4.3.3 Final Cluster Descriptions

To make our method more robust against parameter settings we refine the partial cluster descriptions to final ones. For example in case the *minsup*-parameter in the apriori step has been chosen to high, different partial cluster descriptions might be found for the same projected cluster. In this case each found maximal frequent itemset includes only a subset of the relevant attributes. So, after the pruning, we propose to group similar frequent itemsets.

To explain this step we introduce the geometric interpretation of the frequent itemsets. Each frequent itemset $I \in PFI$ describes a hyper box. The boxes are constructed using the split points of the separators. The data points in the box G, determined from the frequent itemset SI', are given by

$$G = \bigcap_{C_j^i \in SI'} S_j^{Ci}(F) \;\cap\; \bigcap_{N_j^1 \in SI'} S_j^{N1}(F) \;\cap\; D$$

Note that the different boxes may intersect others. On the other side it is important to note that PFI may include different frequent itemsets which describe nearly the same clustered set of points. Similarity between frequent itemsets can be measured by the similarity of the supporting data point sets. A common set similarity measure is:

$$sim(G_1, G_2) = \frac{\#(G_1 \cap G_2)}{\#(G_1 \cup G_2)}$$

with G_1 and G_2 are sets of data point ID's. Using this formula the similarity matrix between all pairs of partial cluster descriptions can be determined. Since the number of partial cluster descriptions is low the runtime of this operation is acceptable.

Using the similarity matrix the grouping of the partial cluster descriptions can be done using a standard hierarchical clustering algorithm. We used for our experiments the complete linkage algorithm from the CLUTO package [64]. The output of the algorithm is a dendrogram of the partial cluster descriptions in combination with the block diagonalized similarity matrix. Each group of similar cluster descriptions appears as block around the diagonal. The visualization can be used to find out how many groups are in the data and which cluster description should be left out. An example of the visualization for synthetic data is shown in figure 4.12.

The last issue is the generation of the final cluster descriptions. After grouping similar partial cluster descriptions we have to build final ones from the groups. For this we recall the geometric interpretation of the partial descriptions, which are hyperboxes in the data space. The intersection of all would not work very well since many data point would be excluded from the cluster, however the bounding box of all partial boxes might include unwanted data points. We decided to fuse the boxes which describe the projected clusters as crossing boxes. In logical terms, we use the disjunction of conjunctions of separator regions as final cluster description. Figure 4.13(a) shows an example for an projected cluster with its description. The relevant dimensions of a cluster are those dimensions for which a separator region appears in one of the merged partial cluster descriptions. The rest are non-relevant dimensions.

Note that also the cluster from the final cluster description may have some overlap. The block diagonalized similarity matrix is able to show which clusters have some overlap.

The overlapping nature of the boxes (and so of the clusters) makes clear that projected clustering delivers in general not a definite partitioning of the data, but a set of overlapping clusters. Examples for the overlapping nature of projected clusters are shown in figure 4.13(b), where points of the

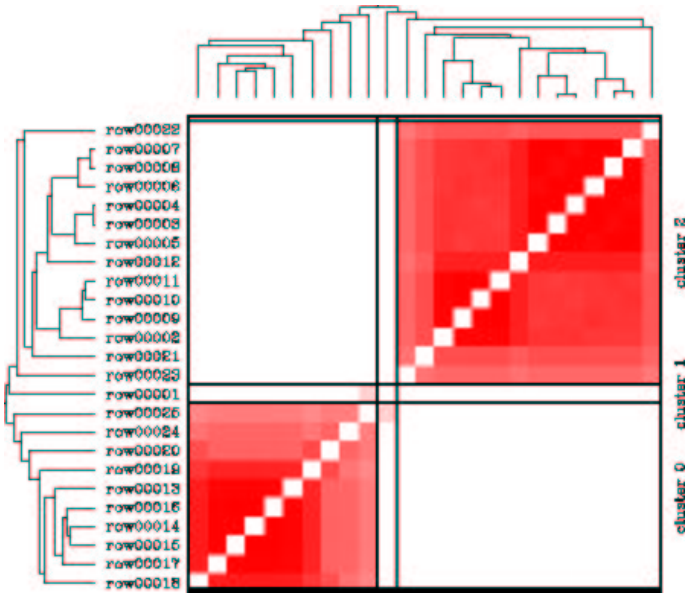


Figure 4.12: The figure shows a hierarchical clustering (complete linkage) and the block diagonalized similarity matrix of the partial cluster descriptions. The used data set has 20 dimensions and contains two projected clusters.

Table 4.3: The table summarizes the facts about the used synthetic data sets. Both data sets have 20 dimensions $(0, \ldots, 19)$. The relevant dimensions are normally distributed, while non-relevant dimensions are uniformly distributed. Shared relevant dimensions are heavy printed in bold typeface.

Data Set	Class	Relevant Dimensions	Size
DS1	Class 1	5, 18, 19	10000
	Class 2	9, 14, 15	10000
DS2	Class 1	0, **4**, 5, **6**, 8, **9**, **11**, **12**, **16**, 17	10000
	Class 2	1, **4**, **6**, **9**, **11**, **12**, 14, 15, **16**, 19	10000

cluster $S_1^{N1}(F)$ are also included in the projected cluster $S_2^{N1}(F)$. For simplicity, here the final cluster descriptions consist of only one separator region.

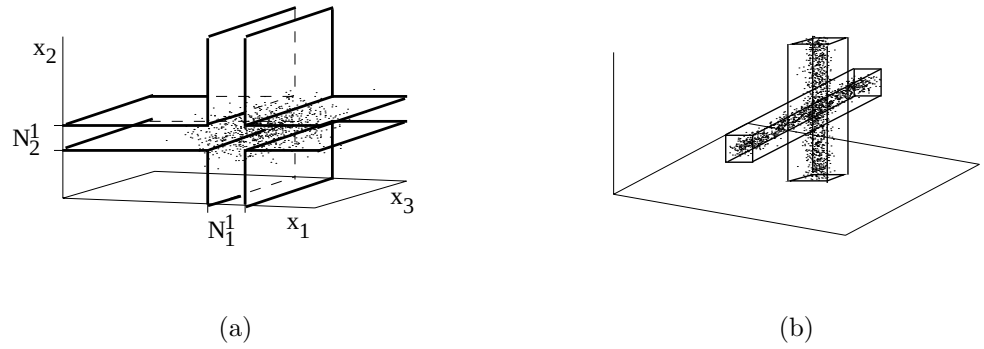
The process of finding axes parallel cluster as explained is summarized in algorithm 12.

4.3.4 Experiments

In this subsection we empirically evaluate the features of our projected clustering algorithm. In the first part we investigate the effectiveness of the method.

There are two extreme cases for projected clusters. First, the projected clusters do not share any relevant dimensions. Such projected clusters can only be separated with the noise separator, because in each one-dimensional axes-parallel projection exists at most one dense interval. In the opposite extreme case the data contains projected clusters, which share nearly all of their relevant dimensions. In such a scenario it is likely that in the projections at least two clusters overlap. If that is the case, the clusters can be separated by the cluster separator.

We demonstrate the effectiveness of our algorithm for the two cases. For this purpose we generated two 20-dimensional, synthetic data sets each containing two projected clusters. Table 4.3 summarizes the information about the data sets. The clusters of the first data set $DS1$ do not share a relevant dimension. Each cluster of this data set has three relevant dimensions. In section 4.2 we used this data set to show the weakness of ORCLUS in finding projected clusters



(a) (b)

Figure 4.13: Part (a) shows a final projected cluster description consisting of the union of N_1^1 and N_2^1. The united partial cluster descriptions are defined in the dimensions x_1 and x_2 while dimensions x_3 is a non-relevant one for this cluster. Note that the partial cluster descriptions have such a small size (length 1) only for illustration in the example. The typical length is larger. Part (b) shows a case where two projected clusters have some overlap. The points in the intersection belong to both clusters.

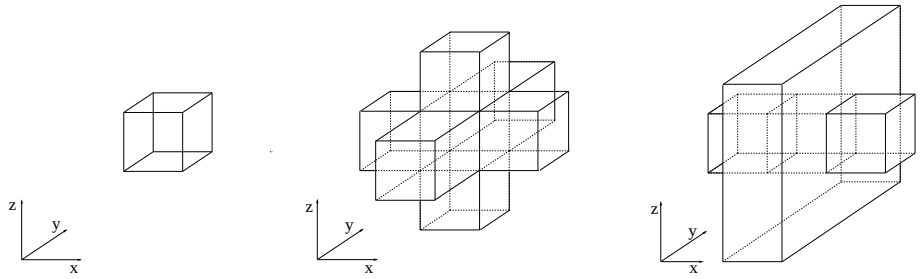
Algorithm 12 Axes Parallel Clustering

AP_clustering$(D, \xi, minsup)$

Require: $D = \{x_1, \ldots, x_N\}$ a d dimensional data set, $\xi \geq 0$ noise level, $minsup > 0$ minimal percentage of required data points in a projected cluster.

Ensure: C contains the final cluster descriptions, each with at least $minsup$ points.

$S^C \leftarrow \emptyset$, $S^N \leftarrow \emptyset$ {Init separator set}
for $j = 1$ to d **do**
  $f^{P_j(D)} \leftarrow determine\_density(P_j(D))$
  $S_j^C \leftarrow determine\_cluster\_separator(f^{P_j(D)}, \xi)$
  if a Cluster Separator was found **then**
    $S^C \leftarrow S^C \cup \{S_j^C\}$
  else
    $S_j^N \leftarrow determine\_noise\_separator(f^{P_j(D)}, \xi)$
    if a Noise Separator was found **then**
      $S^N \leftarrow S^N \cup \{S_j^N\}$
    end if
  end if
end for
if $S^C = \emptyset$ and $S^N = \emptyset$ **then**
  return(D) {No cluster found, return D as one cluster.}
end if{Determine Partial Cluster Descriptions}
$T \leftarrow generate\_transactions(D, S^C, S^N)$
$T' \leftarrow delete\_freq\_items(T, maxsup = 80\%)$
{Delete the items from the transactions with a support larger than $maxsup$.}
$FI \leftarrow determine\_freq\_itemsets(T', minsup)$
if $FI = \emptyset$ **then**
  return(D) {No cluster found, return D as one cluster.}
end if{Delete itemsets which are included by larger ones}
$PFI \leftarrow remove\_small\_freq\_itemsets(FI)$
{Determine Final Cluster Descriptions}
$SimMatrix \leftarrow determine\_similarity(PFI, T)$
$C \leftarrow group\_FI(PFI, SimMatrix)$
$label\_data(C, D)$
return(C)



(a) $x \wedge y \wedge z$, small minsup (b) $(x \wedge y) \vee (x \wedge z) \vee (y \wedge z)$, medium minsup (c) $x \vee (y \wedge z)$, large minsup

Figure 4.14: The figure shows how separator regions can be combined using conjunctions and disjunctions. The labels x, y, z in the logical terms correspond in this example to the separator regions (intervals), which are defined in the respective dimensions. Conjunctions are frequent itemsets which could be united by the logical OR-function. The parts (from a to c) sketch results for increasing *minsup*. Higher values for *minsup* result in coarser final cluster descriptions.

Table 4.4: The table shows the resulting conformation matrices from the experiments with the DS1 and DS2 data sets. The DS1 data set has been shown in section 4.2 to be difficult for ORCLUS. Part (a) shows that our algorithm can find this type of projected clusters very well. The other parts show the results for the DS2 data set with respect to different values for *minsup*. For high values the cluster description includes many false positives. This is due the large volume of the united separator regions. For small values the cluster description includes only the core part of the particular clusters. The best results are found for $minsup = 35\%$.

In/Out	C_1	C_2
Class 1	9560	9
Class 2	6	9574

(a) DS1

In/Out	C_1	C_2
Class 1	10000	1505
Class 2	1505	9999

(b) DS2, $minsup = 45\%$

In/Out	C_1	C_2
Class 1	9957	726
Class 2	227	9977

(c) DS2, $minsup = 40\%$

In/Out	C_1	C_2
Class 1	9282	91
Class 2	1	9870

(d) DS2, $minsup = 35\%$

In/Out	C_1	C_2
Class 1	6691	0
Class 2	0	9181

(e) DS2, $minsup = 30\%$

In/Out	C_1	C_2
Class 1	6691	0
Class 2	0	5113

(f) DS2, $minsup = 25\%$

with many non-relevant dimensions. As shown in table 4.4 (a) our new approach can find such projected clusters very well.

The other data set $DS2$ also contains two clusters. Here the clusters share many of their relevant dimensions. In the context of this experiment we want to explain the influence of the *minsup* parameter. This parameter determines the minimum frequency of an frequent itemset and implicitly the length of the maximal frequent itemsets (itemsets without frequent superset). A lower *minsup* allows larger frequent itemsets. However, long frequent itemsets can occur only if there are projected clusters in the data with many relevant dimensions. The extreme case is when all separator regions of relevant dimensions are covered by a single frequent itemset. So, the final cluster description consists of only one partial cluster description, which is a hyperbox with bounds in each relevant dimension. When higher values for *minsup* are chosen only smaller subsets of the full set of separator regions of relevant dimensions can become frequent. These partial cluster descriptions are similar to each other and are united (logical OR) in the following step to a final cluster description. So *minsup* parameter controls implicitly the structure of the final cluster descriptions. Low *minsup* produces hyperboxes of high dimensionality and higher *minsup* unions of hyperboxes with lower dimensionality. Figure 4.14 illustrates the different cluster descriptions.

We examine the data set $DS2$ with different values for *minsup* and show the results as confusion matrices in table 4.4 (b-f). The *minsup* parameter is varied from 45% to 25%. In all experiments the relevant dimensions were found correctly. The average length of the found partial cluster

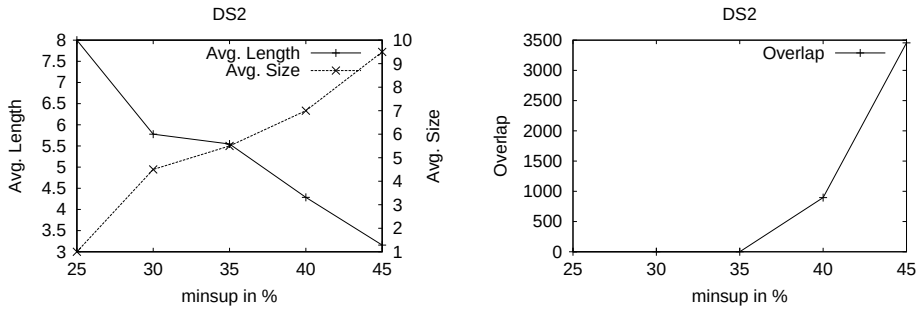


Figure 4.15: Part (a) shows the average length of the partial cluster descriptions and the average size of the grouped final cluster descriptions depending the used *minsup*. The first measure drops to one with increasing *minsup* while the second increases. This behavior exemplifies the tradeoff between the conjunctions of separator regions and disjunctions. Part (b) shows that with increasing *minsup* the overlap between the final cluster descriptions grows. If a partitioning in clusters is wanted a good choice for *minsup* is the largest value where the overlap is zero. In the case of the DS2 data set $minsup = 35\%$ is the best choice. As shown in table 4.4 for $minsup = 35\%$ also the accuracy is the best.

descriptions and the average size of the groups for the final cluster descriptions depending on *minsup* are plotted in figure 4.15 (a). The overlap (measured in data points) between the finial cluster descriptions is plotted in figure 4.15 (b). This measurement can be used as indicator for choosing a good value for *minsup* if disjunct clusterings are wanted. For the DS2 data set we used the rule, that the largest *minsup*-value with zero overlap is the best choice. As shown in table 4.4 (d) this gives also the best accuracy.

In the next experiments we examine the dependency of the algorithm's runtime from the number of data points as well as the number of dimensions. The data sets DS3 and DS4 generated for this experiments, contain also two clusters. For DS3 the dimensionality is fixed to 20 and each cluster has 10 relevant dimensions. The clusters are equally sized in each setting. The runtime of the algorithm is dominated by the first (separator finding), the second step (frequent itemset determination) and the third step (determination of similarity). The three steps perform in total a nearly constant number of scan over the data. For lower *minsup* fewer maximal itemsets are produced, which explains the slightly smaller runtime. Since the overall runtime of the algorithm is linear with respect to the number of data points, the algorithm scales to large data sets.

To investigate the ability of the algorithm to scale to large dimensionality we used the data set DS4, which also consists of two equally sized clusters, each with 25000 points. The dimensionality varies from 5 to 50. The important point is that the number of relevant dimensions is for each setting half of the full dimensionality. That means that the dimensionality of the clusters subspaces also grows with global dimensionality. In the first case we set no limit for number of used separators. This causes longer transactions and also longer frequent itemsets. Since Apriori is not designed to handle long itemsets beyond a dimensionality of 25 the runtime of this step exploded. This situation is reported in figure 4.16 (b). There is currently ongoing research on algorithms which can handle this kind on data much better and find long frequent itemsets more efficiently. For our purpose we restricted in a second experiment the number of used separators to a maximum of 20. We chose the separators with the best separation quality. This restriction limited also the length of the frequent itemsets to a size for which Apriory runs efficiently. As a consequence the resulting cluster descriptions do not contain all relevant dimensions for a cluster, but separated the clusters also very well. However, the full set of relevant dimensions for each cluster could be easily computed in a post processing step by separately determining the noise separators for each cluster. Each found noise separator marks a relevant dimension of the cluster. Since due to the restriction of the number of separator the size of the transaction set fixed size is implicitly fixed and so the growing dimensionality effects only the first step (separator finding). This explains the smaller gradient after the restriction appealed. The runtime of the algorithms is shown in figure 4.16 (c).

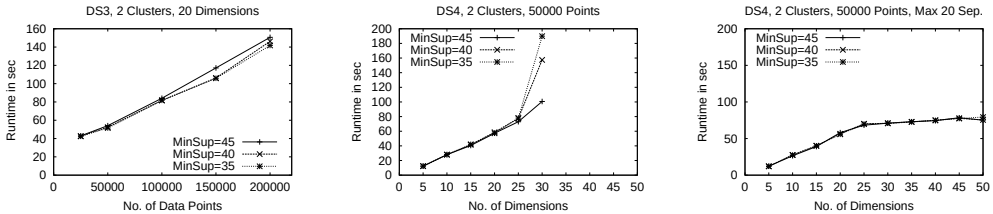


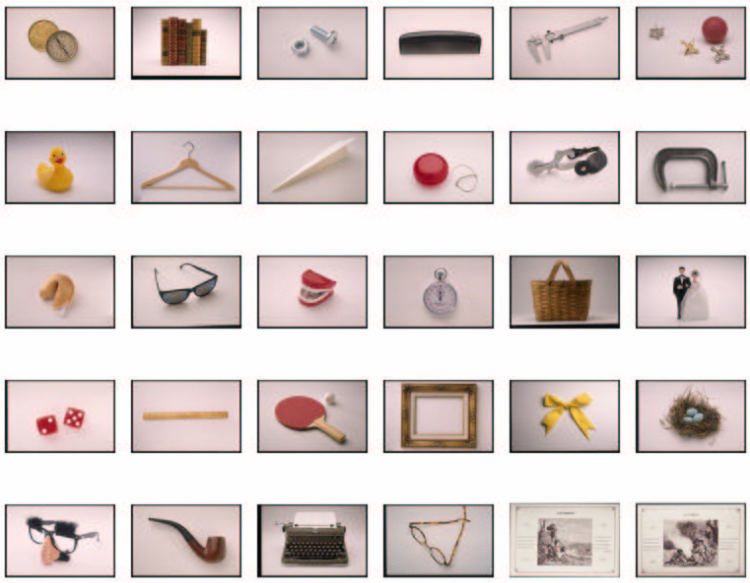
Figure 4.16: Part (a) shows the runtime depending on the number of data points. The used data DS3 contains two equally sized clusters and is 20-dimensional. The data set DS4 contains 50000 data data points and two clusters. The number of relevant dimensions is half of the full dimensionality. Part (b) shows the runtime when the number of used separators is not limited. Since the lengths of the frequent itemsets grow with the increasing number of relevant dimensions, the runtime of apriori explodes beyond 25 dimensions. Part (c) shows the runtime depending on the dimensionality when the number of separators is limited to 20, Since the resulting cluster descriptions do not contain all relevant dimensions. the full set of relevant dimensions is separately determined for each cluster in a postprocessing step.

This demonstrates that the algorithms is also highly scalable with respect to dimensionality.

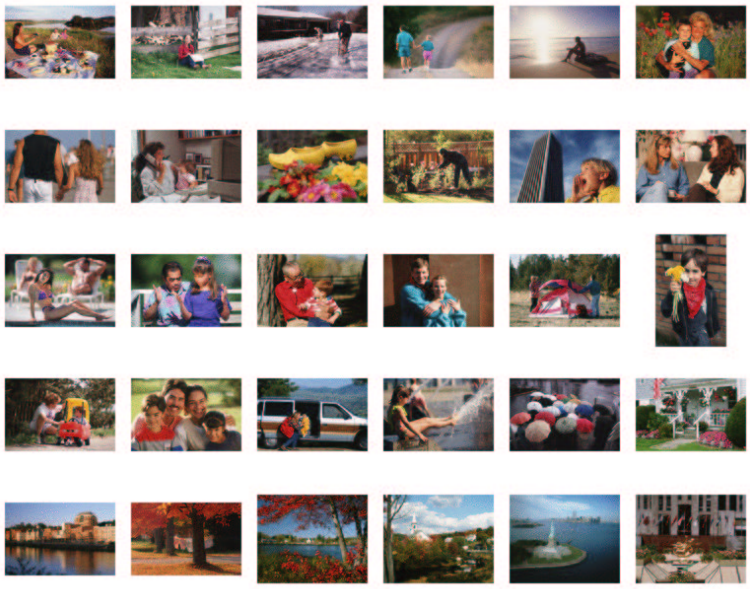
Application to real data After showing the behavior of our new algorithm we apply it to two different real data sets. The first data set is derived from a collection of 8537 images. For each image the gray scale color histogram was derived. The histograms have been wavelet-transformed into a 64-dimensional feature vectors. The image transformation is described in detail in [65]. Using our new projected clustering algorithm 2 different clusters was found. The clusters consist of 890 images (Cluster 1) and 1212 images (Cluster 2) respectively. This shows that our approach is not forced to partition the data, like PROCLUS or ORCLUS and also finds small clusters. The first cluster is very homogenous and consists of images with a white, shaded background and a special object in the foreground. There is no restriction on the foreground objects, which also could vary in their sizes. The other cluster does not seem to have a content-based interpretation. The common feature of all images is, that always at least one light area appears in the images. Figure 4.17 shows typical examples from the clusters. The full clusters as well as the full image collection could be found at http://www.informatik.uni-halle.de/∼hinnebur/diss/images/.

The second data set comes from a molecular biology simulation of a small peptide. During the simulation every 20th pico-second a snapshot of the spatial conformation of the molecule was taken. A single spatial conformation of the molecule is described by 19 dihedral angles, which were used as dimensions of a feature vector. Such short peptides do not have a stable spatial conformation and so they repeatedly fold and unfold over time. An interesting question is what are stable states of the molecule which often occur in the data. The different states were assumed to form clusters in the data. Our new projected clustering algorithm found 7 different cluster with 4 main clusters in the data. In figure 4.18 we show the hierarchical clustering and the similarity matrix of the partial cluster descriptions. As shown in the similarity matrix all clusters have some overlap, which indicates that the states could not be sharply fractionized. This goes along with the intuition that the molecule folds over intermediate steps from one state into another. Note that overlapping clusters could not be found by any other projected clustering method.

In summary we showed that our new projected cluster can find projected clusters with only a few relevant dimensions, which are likely to be missed by the best known projected clustering algorithm ORCLUS. We explained and empirically verified our new projected clustering concept based on disjunctions of conjunctions of separator regions and showed the tradeoffs in this concept. We showed that our algorithm is hilly scalable, namely linear in number dimensions as well as in the number of data points. We applied our method to image and molecular biology data and showed that our method finds clusters, which are missed by others, since our algorithms does not need to partition the whole data set and allows to find overlapping clusters. In the next section we extend our method to projected clusters with dependencies.



(a) Cluster 1



(b) Cluster 2

Figure 4.17: The clusters consist of 890 images (Cluster 1) and 1212 images (Cluster 2) respectively. This shows that our approach is not forced to partition the data, like PROCLUS or ORCLUS and also finds small clusters. The first cluster is very homogenous and consists of images with a white, shaded background and a special object in the foreground. There is no restriction on the foreground objects, which also could vary in their sizes. The other cluster does not seem to have a content-based interpretation. The common feature of all images is, that always at least one light area appears in the images.

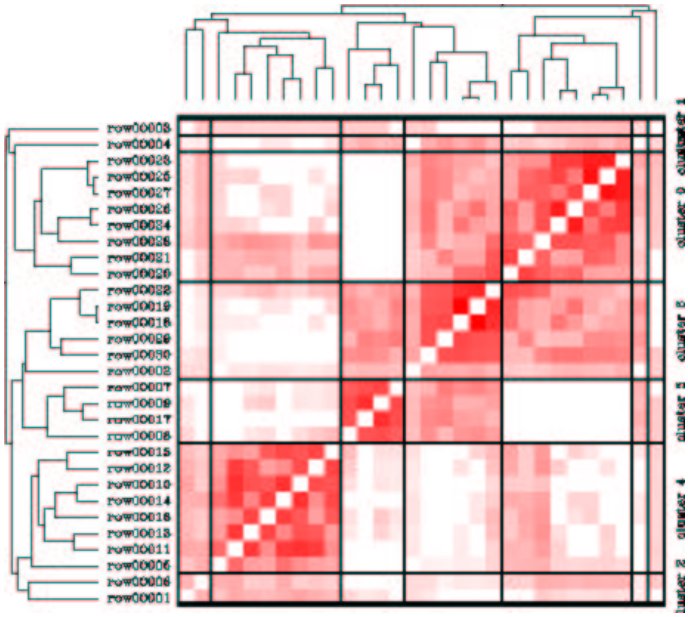


Figure 4.18: The figure shows the similarity matrix of the partial cluster descriptions for the molecular biology data. The similarity matrix shows four main clusters which overlap each other. This goes along with the intuition about the application domain that the simulated molecule has intermediate states between the stable ones. Note that overlapping clusters could not be found by any other projected clustering method.

4.3.5 Extensions to Projected Clusters with Dependencies

In the previous subsections we assumed that the data has no dependencies between the dimensions. Using this assumption we could restrict our projection pursuit in the first step (finding of separators) to axes-parallel one-dimensional projections. Now arises the question how to deal with projected clusters with dependencies between the relevant attributes.

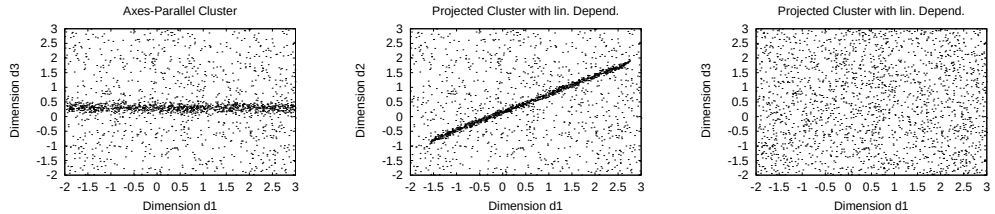
First we have to redefine the term relevant dimension. This term shall capture the contribution of a dimension to the cluster information. Unlike in case of axes-parallel projected cluster here a relevant dimension is only relevant in the context of other relevant dimensions. Figure 4.19 illustrates the difference. The parts (b-c) show two-dimensional projections of the same data, in the first case the data is projected to the dimensions (d_1, d_2) and in the second case to (d_1, d_3). In the first case the dimensions d_1 is relevant since the projected data contain a cluster, while in the second case the same dimension is not relevant. So it makes no sense to define the relevance of a dimension independently from the context (the other involved dimensions). This is due to the fact that in case of dependent attributes the relevance of a dimension can not be concluded from the marginal distribution.

As a consequence we have to redesign the first step: the finding of separators has to take multi-dimensional projections into account. Since the other steps are decoupled from the multi-dimensional representation these parts can be left unchanged.

So the problem is, how to find good separators using multi-dimensional projections. The problem can be decomposed into three sub-problems:

1. How to find useful multi-dimensional projections?
2. How to determine good separators in the projections?
3. How to rate and compare the quality of the separators?

First we look at the problem of searching the space of projections. There are two basic possibilities to do the search, namely enumeration and heuristic search. Let us first look at the enumeration



(a) (b) (c)

Figure 4.19: Unlike in the case of axes-parallel projected cluster (part a), in the case of projected clusters with dependencies a relevant dimension is only relevant in the context of other relevant dimensions. The parts (b-c) show two-dimensional projections of the same data, in the first case the data is projected to the dimensions (d_1, d_2) and in the second case to (d_1, d_3). In the first case the dimensions d_1 is relevant since the projected data contain a cluster, while in the second case the same dimension is not relevant.

of projections. Since the general space of projections is infinitely large, there have to be additional constraints to make enumeration possible. The smallest subspace of projections, which can capture dependencies is the subspace of two-dimensional axes-parallel projections with a size of $\binom{d}{2} = \frac{d(d-1)}{2}$ ($d \in \mathbb{N}$ is the number of dimensions). With this kind of projection dependencies between two attributes can be found. In general the subspace of axes-parallel projections of dimensionality d' has a size of $\binom{d}{d'}$ which grows by an exponential rate for $d' \leq d/2$. So enumeration of the general subspaces becomes quickly infeasible.

The other search methods are of heuristic nature. One possible strategy is presented in the section on projected nearest neighbor search. This strategy uses genetic search to find good low-dimensional projections ($d' = 2, 3$) and then it extends the set of dimensions in a greedy fashion.

We did not consider so far how projections are rated. What would a good projection looks like? A good projection in this context is a subspace where groups of data points can be well separated from others. Similar to the case of axes-parallel projections here we also search for separators in the subspaces. The examination tasks for the multi-dimensional projections are the same as for one-dimensional projections, namely cluster and noise separation. Clusters without dependencies are compact ellipsoids stretched along the axes, while for clusters with dependencies the shape and orientation plays an important role. So the separation of clusters with dependencies can be done using the density-based single-linkage separator, which can handle arbitrary-shaped clusters. For the noise separation task we use the noise separator assuming the noise to be uniformly distributed in the subspace. Both separator types allow to determine a separation quality which corresponds to the maximum density at the separation border between different separator regions. A good separation quality indicates a low density at the cluster border, which lowers the probability of splitting a cluster.

After building separator regions the approach described in the previous sections can be used to find cluster descriptions for the projected clusters. The difference is that the separator regions can capture dependencies between groups of relevant dimensions. With the help of this cluster information the continuous data can be transformed into discrete transactions. Partial cluster descriptions are determined by finding frequent itemsets (conjunction of separator regions). Then groups of similar partial cluster descriptions (frequent itemsets) are combined by logical *or* to final cluster descriptions (disjunction of conjunctions of separator regions). Unlike in case of axes-parallel projections here the final cluster descriptions stand for complex high dimensional regions.

The disadvantages of this approach are first the higher runtime of the separator finding step and second the uncertainty of not fully found dependencies when heuristic search is used. If the space of two-dimensional axes-parallel projections is enumerated this uncertainty could be excluded, however the whole method scales only quadratically in the number of dimensions. Runtime could be saved when heuristic search methods are used but this gain of speed comes at the risk of missing some dependencies. Here is a potential for finding reasonable tradeoffs between runtime and quality, which we will investigate in further research.

Chapter 5

Clustering & Visualization

Clustering in high-dimensional databases is an important problem and there are a number of different clustering paradigms which are applicable to high-dimensional data. In the previous chapter we developed separators for the different clustering paradigms and applied the separator framework to projected clustering. This variant of the clustering problem has the potential to be more effective in cases of high dimensional data. Since the choice of meaningful projections is often guided by domain knowledge we recognize a gap between automated algorithms which use only statistical properties and implicit heuristics for the mining process and the user's intention to influence the results by her/his domain knowledge.

Our idea presented in this chapter is to use a combination of automated cluster separators with new visualization methods for a more effective interactive clustering of the data. The starting point is the separator framework with data compression, density-based single-linkage, noise and outlier separators and the projected clustering problem. However, Choosing the contracting projections and specifying the separators to be used in building a separator tree are two difficult problems which can not be done fully automatically.

Visualization technology can help in performing these tasks. There are a large number of potential visualization techniques to be used in data mining. Examples include geometric projection techniques such as prosection matrices [39] and parallel coordinates [59, 60], icon-based techniques (e.g., [19, 85], hierarchical techniques (e.g., [74, 88, 102]), graph-based techniques (e.g., [18, 99]), pixel-oriented techniques (e.g., [1, 11, 66]), and combinations hereof ([8, 15]). In general, the visualization techniques are used in conjunction with some interaction techniques (e.g., [9, 14, 30]) and also some distortion techniques [73, 91]. The existing techniques, however, do not allow an effective support of the projection and separator finding process needed for an efficient clustering in high-dimensional space. We therefore developed a number of new visualization techniques which represent the important features of a large number of projections to allow an easy identification of the most interesting projections and a selection of the best separators. For one-dimensional projections we developed a pixel-oriented representation of the point density projections, allowing an easy identification of the best-separating projections (cf. section 5.1.3). If the number of interesting projections is large, the user may also use a different visualization which represents the largest dense regions in the projections and their separation potential by small polygonal icons (cf. section 5.1.2). The iconic representation is a reduction of the information in the pixel-oriented visualization and allows a quick overview of the data. For two-dimensional projections, we use a two-dimensional pixel representation which also allows complex density based single linkage separators to be directly specified within the visualization (cf. section 5.1.5) and a similar iconic representation. In the higher-dimensional case, due to the large number of projections, only the iconic visualization can be used (cf. section 5.1.2). All visualization techniques are integrated by using a visualization of the (extended) separator tree, which describes the projection and separator hierarchy (cf. section 5.1). To show the effectiveness of our new visualization techniques we employ the system for clustering a real data from molecular biology (cf. section 5.2). The experiments show the effectiveness of our approach.

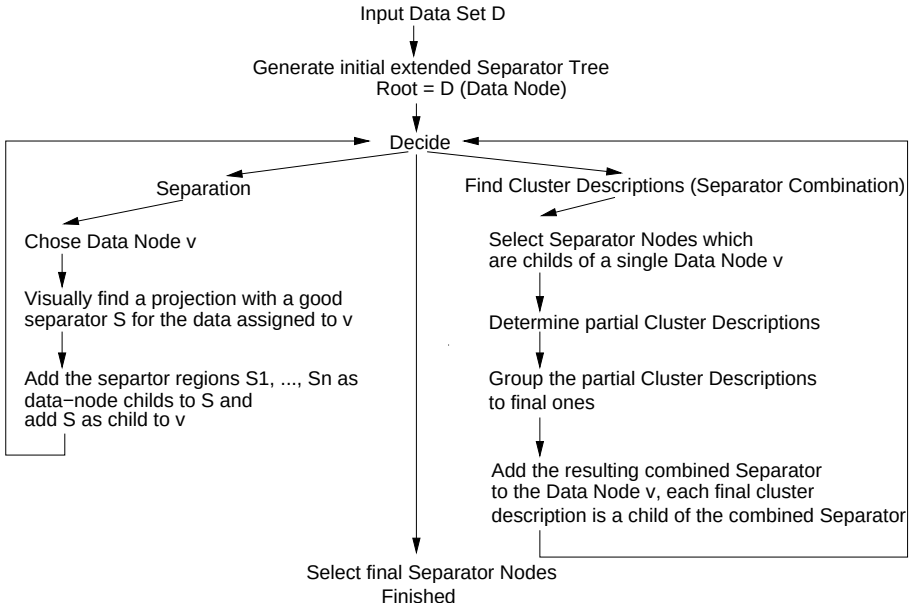


Figure 5.1: The figure shows the possible actions supported by *HD-Eye*. The left branch (separation) deals with the visual finding and determination of projections and separators. The right branch stands for the possibility of combining different separators to complex cluster descriptions. This incorporates the algorithm described in the previous section into *HD-Eye*.

5.1 The *HD-Eye* System

Let us now give a brief overview of our visual clustering tool *HD-Eye*. There are two main tasks which need visual support. The first is finding the appropriate contracting projections for partitioning the data and second, finding and specifying good separators based on these projections. Both tasks require the intuition and creativity of the human user and can not be done automatically. The main difference between the *OptiGrid* or the projected clustering algorithm from the previous section and *HD-Eye* is the visual determination of the projections and separators. By visual feedback, the human observer gets a deeper understanding of the data set and is therefore able to identify additional projections and separators corresponding to substructures which can not be found by the automated algorithms due to the non-interactive determination of the projections and separators. The visual interactive finding of separators is the left branch in figure 5.1. The right branch allows the combination of separators to find complex cluster descriptions as reported in the previous section.

The *HD-Eye* system allows to split recursively the data set. The recursive splitting lets grow the separator tree, which forms a hierarchy representing the cluster structure in an intuitive way. The simple separator tree can be extended by inserting the separators itself into the tree. An extended separator tree consists of separator and data nodes. The levels of the tree are homogenous and the nodes alternate between the levels starting a data node as root node. The input data set is assigned to the root node. The advantage of an extended separator tree is that alternative separators for the data could be stored in the same data structure. Also overlapping clusters are supported in that way. The extended separator tree is a space efficient method to store multiple simple separator tree. This allows the user to explore and to compare several alternative clusterings. Figure 5.2 shows an example for an extended separator tree and how it includes a simple separator tree.

There are two version of *HD-Eye*, the first implements only simple separator tree, while the second allows the use of extended separator trees. Figure 5.3 shows overview screenshorts of both versions. The data can be partitioned first using a set of simple one- and two-dimensional orthogonal projections (1D, 2D) and then using a set of complex separators which can handle clusters of arbitrary shape. The multiple levels of in the cluster hierarchy allow the use of different

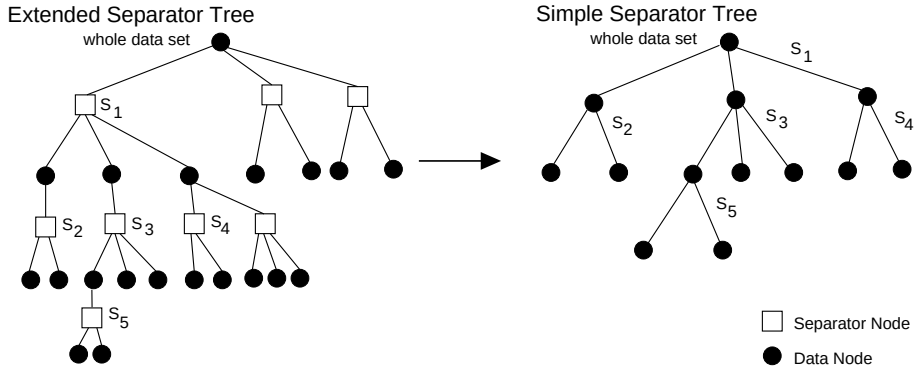


Figure 5.2: The extended separator tree allows the representation of alternative ways of separating in the same data structure. Common parts of the alternatives are shared in the extended separator tree. By fixing non-alternative separator nodes a simple separator tree can be extracted from the extended one. Non-alternative separator nodes do not share the parent data node. The extended separator tree allows the user to explore and to compare several alternative clusterings at the same time.

parameterizations for the different cluster separators. For example the noise level can be adapted to different parts of the data set. Note that the separators allow clusters of arbitrary shape and complex (linear, quadratic, and higher-order) dependencies to be found.

In the following, we describe the core of the visual mining capabilities of the HD-Eye system, namely the visual techniques for finding good projections and separators (cf. different windows in figure 5.3).

5.1.1 Visual Finding of Projections and Separators

One of the essential parts of the HD-Eye algorithm is to find projections, which are useful for separating the data into clusters. In [55] it is shown that axes-parallel projections are sufficient for detecting axes-parallel projected clusters with no linear dependencies between the attributes. In case of clusters with linear or higher-order dependencies between the dimensions (arbitrary shaped clusters) multi-dimensional projections and separators are necessary. Figure 5.4 shows an example, where the two clusters could not be separated using only the density in the marginal distributions. The two-dimensional projection to the plane (x_1, x_2) is needed.

Without prior knowledge about the dependencies or shapes of the clusters, it is hard to tune the parameters of currently available automatic cluster discovery algorithms to find the correct clustering. Visual techniques which preserve some characteristics of the data set can be of indispensable value for obtaining good separators. In contrast to dimension reduction approaches such as principal component analyses (FASTMAP [35]) or projection pursuit [58], our approach does not require that all clusters are preserved by a single projection. In the projections some cluster may overlap and may therefore not be distinguishable. For our approach, we only need projections which allow a separation of the data set into at least two separated subsets without dividing any clusters. The subsets may then be refined using other projections and are possibly partitioned further based on separators in other projections. Since we only use contracting projections, partitioning the data set in the minima of the projections does not cause any error on the clustering (cf. Lemma 12). Based on the visual representation of the projections, it is possible to find clusters with unexpected characteristics (shapes, dependencies), which are very difficult to be found by tuning the parameter settings of automatic clustering algorithms.

The most important information about a projection is whether it contains well-separated clusters. Note that well-separated clusters in the projection could result from more than one cluster in the original space. But if it is possible to partition the data into subsets the more complex structure can be detected in a recursive way using other projections which then become more meaningful. In our approach we visually code the important information (whether the projection

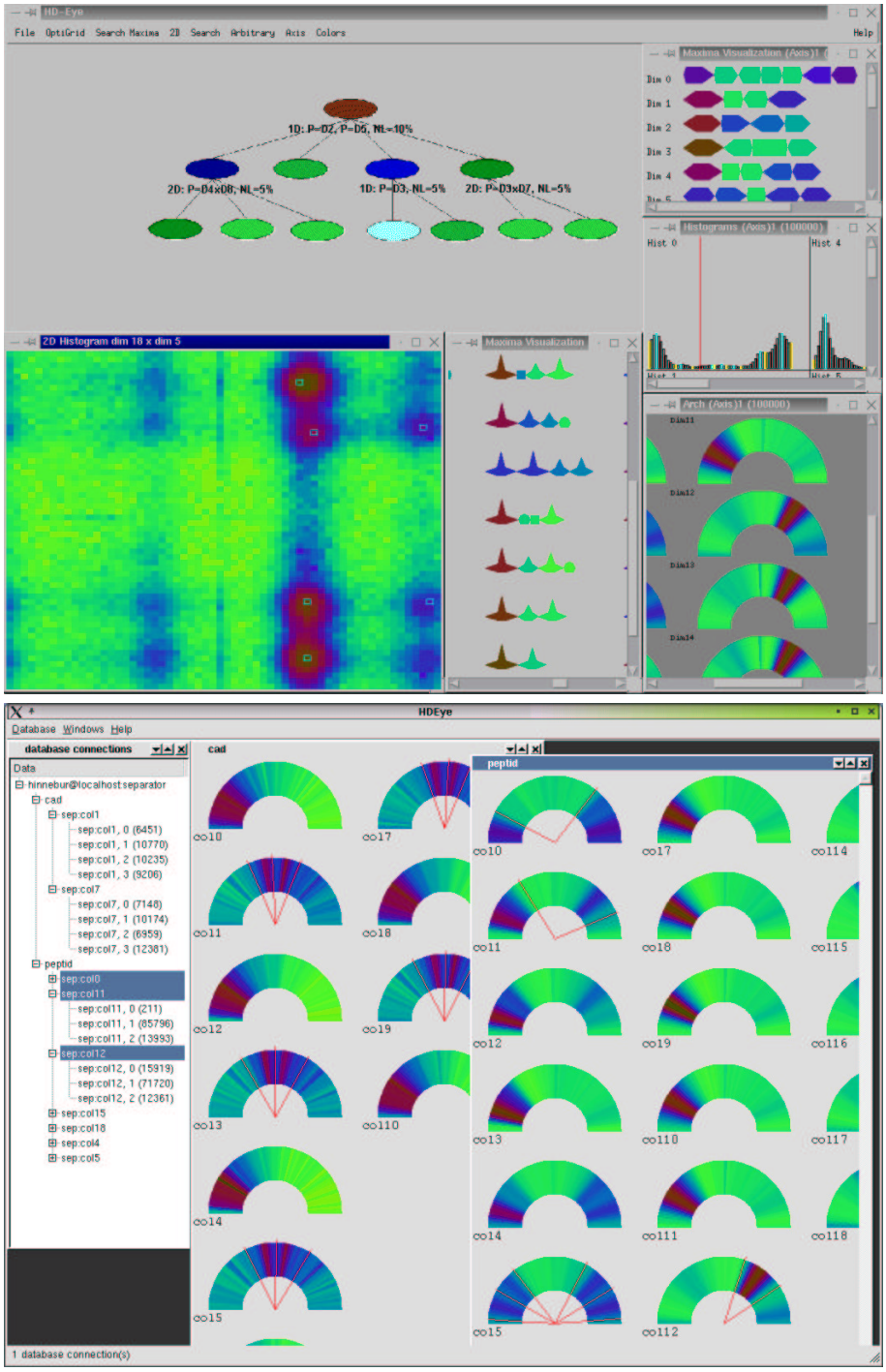


Figure 5.3: HD-Eye Screenshot Version 1 and Version 2

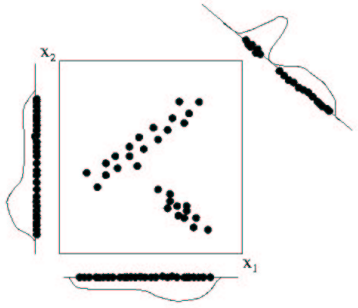


Figure 5.4: Example for the need of multi-dimensional projections. The two clusters could not be found using only the marginal distributions. The two dimensional projection to the plane (x_1, x_2) is needed.

contains well-separated clusters) in three different ways:

(a) Abstract iconic display,

(b) Color-based point density, and

(c) Curve-based point density.

Method (a) has two variants. The first variant is applicable to projections $P : \mathbb{R}^d \longrightarrow \mathbb{R}^1$ while the second variant is applicable to general contracting projections $P : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}, d' \leq d$. Methods (b) and (c) are limited to $d' \leq 2$.

In the following, we describe the three methods and discuss how the three methods are used in finding interesting projections of the form $P : \mathbb{R}^d \longrightarrow \mathbb{R}^{d'}, d' \leq 2$; and then we briefly describe how the separators may be specified directly within the visualizations.

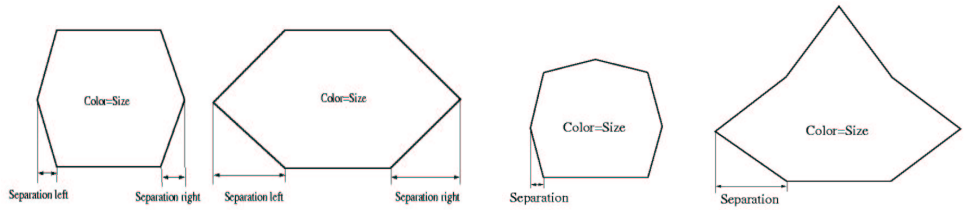
5.1.2 Abstract Iconic Visualizations of Projections

As the number of possible contracting projections is very large we need a visualization technique for displaying the main properties of the data distribution of many projections. Since we are interested in finding projections which allow a good partitioning of the data set, the main properties are in our case

- the number of separator regions defined by separating maxima of the density function of the projected data,
- the number of data items belonging to each separator region, and
- how well the regions are separated from each other.

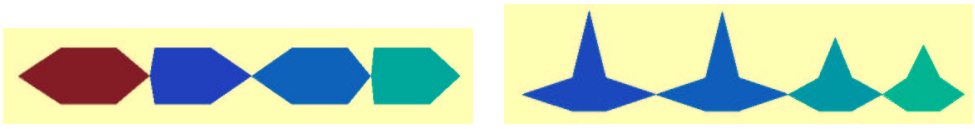
In our approach we determine these properties based on histogram information of the point density in the projected space. To make the system scalable, the histogram can be computed directly by the underlying database, so that no data points have to be read into the main memory.

The abstract iconic display method uses an icon-based technique to display the properties. The number of data points belonging to a separator region corresponds to the color of the icon. The color follows a given color table ranging from dark colors for a large number of data points to bright colors for smaller numbers. The measure of how well a separator regions is separated from the others corresponds to the shape of the icon and the degree of separation varies from sharp pikes for well-separated regions to blunt pikes for weak-separated regions (cf. figure 5.5). The icons for the separator regions of a projection are arranged in a vertical line. In the one-dimensional case (variant a), a natural order of the regions exists and the degree of separation can be measured in the two directions (left and right). The degree of separation to the left corresponds to the difference between the maximum density and the density of the border bin between the current



(a) 1 dimensional (b) multidimensional

Figure 5.5: Structure of the icons



(a) one-dimensional (b) multidimensional

Figure 5.6: Examples of the icons

regions and the next region on the left. Analogously the separation for the right side is defined. In the multidimensional case, the direction of the separation with respect to other clusters does not make sense and therefore in the visualization we order the separator regions to obtain a good contrast between the important regions and the less important ones. The degree of separation of a cluster in a projection is determined by the highest density at the border of the region, that means the smaller the maximum density at the border the better is the separation of the cluster. For the cluster detection in the projected space we employ the density-based single-linkage separator introduced in section 3.3.3 (page 30). Figure 5.5 shows the structure of the icons and figure 5.6 shows examples for visualizations resulting from a 1D and a 2D projection.

The iconic representation of the maxima of the density function can be used to identify the interesting projections and for finding good separators. Figure 5.7 shows an example with a larger number of 2D projections. The interesting projections have at least two well separated maxima (dark, sharp icons).

5.1.3 Color Density and Curve Density Visualizations of Projections

The abstract iconic visualizations of the projection are used to find interesting projections among a large number of projections. For further exploration such as testing whether the projections allow useful separators, color density and curve density plots can be used. Figure 5.8 shows one-dimensional examples for the color density and for the curve density plots. The darkness of the

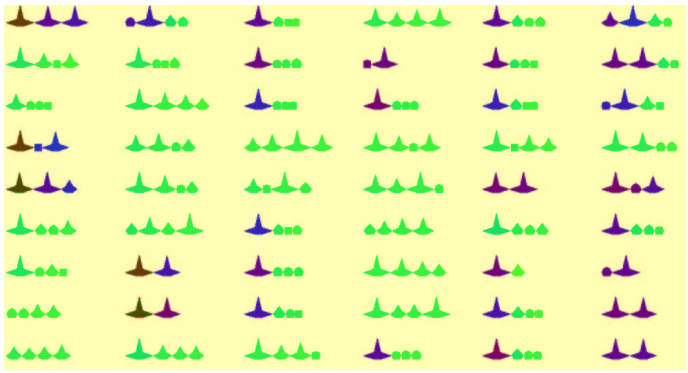
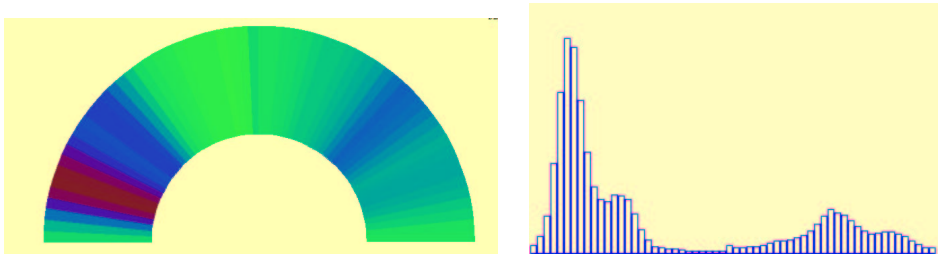


Figure 5.7: Iconic representation of a large number of projections



(a) Color (b) Curve

Figure 5.8: Examples for the one-dimensional density plots

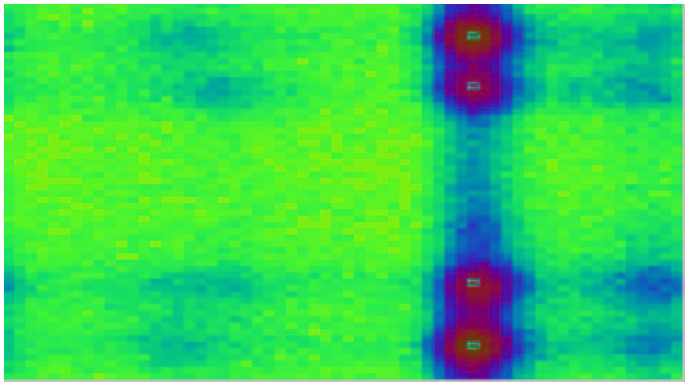


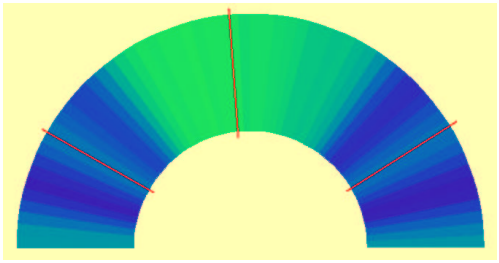
Figure 5.9: Examples for the color density plots (two-dimensional)

color increases with higher point density. The examples correspond to the example of the abstract 1D iconic visualization method (figure 5.6 (a)). We used color density plots in form of an arc instead to simple bar charts to prevent visual clutter, because long horizontally arranged bar charts could be easily mixed up (visual influence between different bars) in cases of overview displays. Figure 5.12 (b) shows an example for an overview plot with 1D color density plots.

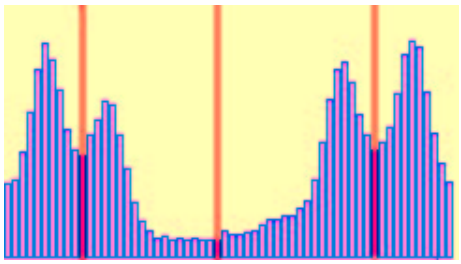
In addition to the one-dimensional plots, we also support two-dimensional color density plots. The basic idea is to code the point density of the projection by color. In figure 5.9, we show an example of a two-dimensional color density plot, which corresponds to the abstract iconic representation in figure 5.6 (b). The color density plots are also used for an intuitive specification of complex hyper-polygonal separators (cf. subsection 5.1.5).

5.1.4 Methodology for Pre-selecting Interesting Projections

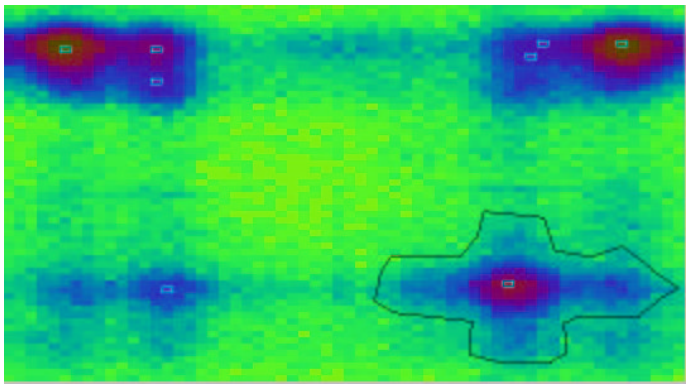
Since the number of potentially interesting projections is very large, it is important to have some procedure for the pre-selection of projections. In the HD-Eye system, we use an interactive optimization procedure. The procedure starts by proposing some projections to the user. The initial set of projections consists of all axes parallel projections, some diagonal projections, and some random combinations of the previous ones. From this set, the user can select the interesting projections. In case the presented projections are not sufficient, the user can generate other combinations from the selected ones using crossover and mutation operators which are applied to the projection matrices. In each iteration, the selected projections are preserved and potentially modified. The procedure is a sort of genetic algorithm where the fitness function is defined by the user's selections. The iteration for good projections stops when satisfying projections are found or the user is convinced that the data set can not be further partitioned.



(a) one-dimensional, color density plot



(b) one-dimensional, curve density plot



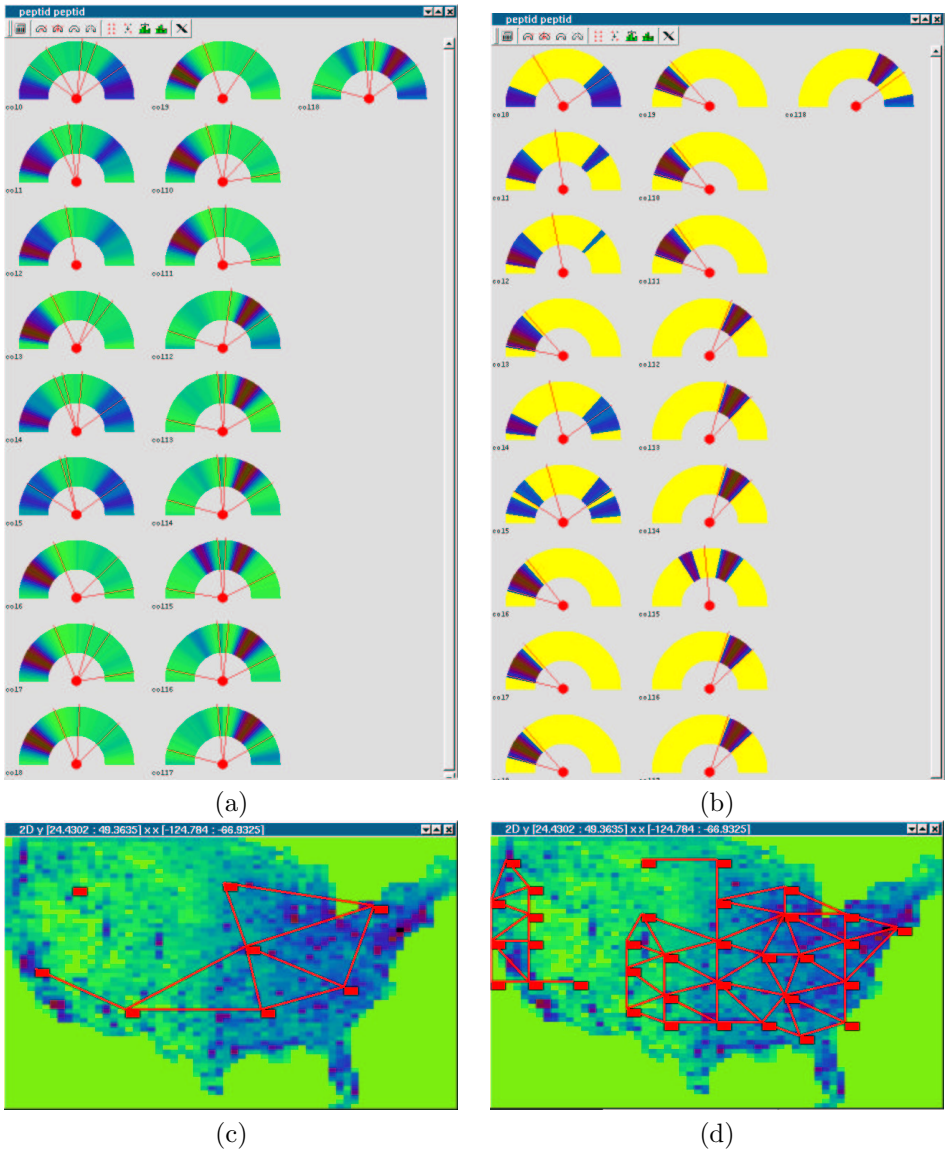
(c) two-dimensional, color density plot

Figure 5.10: Examples for Separators

5.1.5 Visual Finding of Separators

When a good projection is found which allows a separation of data points into clusters the user can use the visualization to directly specify a separator. Due to unknown characteristics of the clusters, finding separators can not be done effectively in an automatic way. In the HD-Eye system, the separators are therefore specified visually using interactive tools. A separator can be defined by simple geometric objects such as split lines or closed split polygons in the projected space. The borders of the split objects are best placed in regions of relatively low point density. Using the upper bound property (lemma 12), the point density in a projection is an approximation of the point density in the data space. For finding separators in one-dimensional projections we use the color and curve density plots with a high granularity (i.e. large number of grid cells). Split lines for the separators can be directly placed in the visualizations using the mouse. In the two-dimensional case, the split polygons can be drawn directly into the two-dimensional color density plots. Figures 5.10 (a), (b) and (c) show examples for separator lines in one-dimensional color and curve density plots and a two-dimensional color density plot.

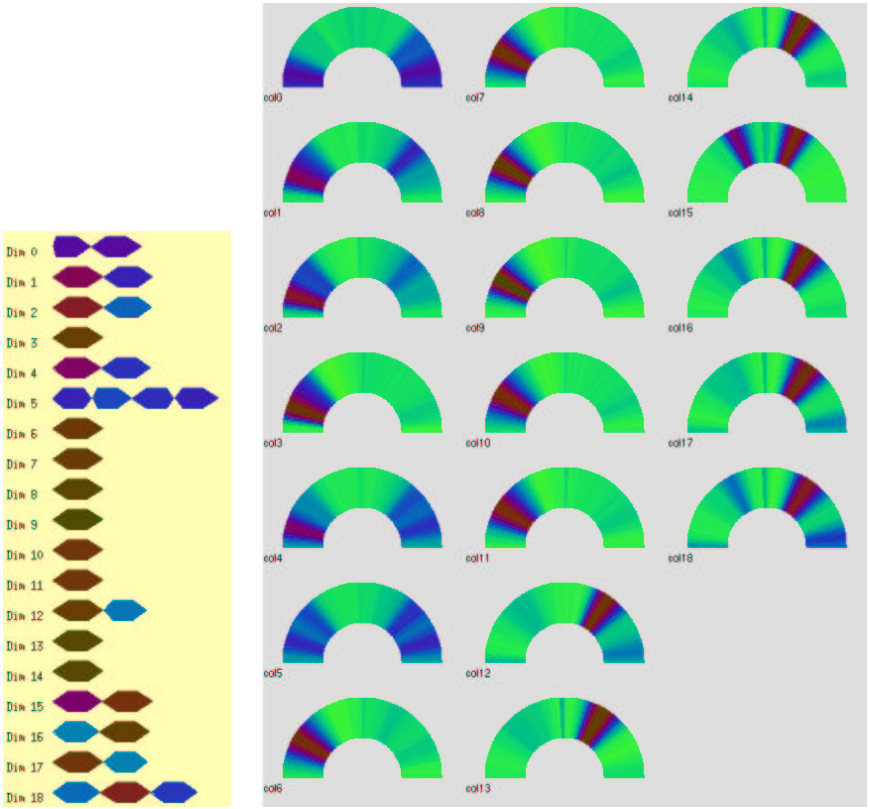
HD-Eye also allows a semi-automated way to determine separators. The automated heuristic to find separators can be tuned to find a convincing result. The user can take advantage of the visual feedback provided by the system to find meaningful values for e.g. the noise level or the number of used centroids for the density-based single-linkage separator. For instance to find a good threshold for the noise level the user can specify an initial value. The ranges below the noise threshold are drawn in yellow (a color which is not in the used color table). After determining separating minima the user can decide, based on the visual feedback provided, whether the setting produces acceptable results or some adjustments are necessary. The methodology can be used to find the appropriate number of centroids. Figure 5.11 shows examples for this feature.



(a) (b)

(c) (d)

Figure 5.11: Part (a) shows the color-based density plots of the molecular biology data with the separating minima for zero noise level. Using the visualizations the user increases the noise level up to 2%. Part (b) shows the resulting density plots where intervals below the noise level are yellow plotted. The use of the noise level removes many minima which are caused by noisy disturbances in the data. Part (c,d) show how the increase of the number of centroids gives a much better approximation. In the example, data from the US Census bureau are clustered into west coast and the eastern part.



(a) Icon groups (b) Color density plots

Figure 5.12: Separators for the Molecular Biology Data Set(19 dimensional)

5.2 Experiments

In this section, we present the results of the application of the HD-Eye system to real high-dimensional data from molecular biology. The experiments demonstrate the effectiveness of our visual techniques.

The data used for the experiment comes from a complex simulation of a very small but flexible peptide. (The time for performing the simulation took several weeks of CPU-time.) The data generated by the simulation describes the conformation of the peptide as a 19-dimensional point [31]. The simulation was done for a period of 50 nanoseconds with two snapshots taken every picosecond, resulting in about 100000 data points. The purpose of the simulation was to determine the behavior of the molecule in the conformation space. Due to the large amount of high-dimensional data, it is difficult to find, for example, the preferred conformations of the molecule. This, however, is important for applications in the pharmaceutical industry since small and flexible peptides are the basis for many medications. The flexibility of the peptides, however, is also responsible for the fact that the peptide has many intermediate non-stable conformations. The preferred conformations correspond to large clusters and intermediate conformations are observable as noise.

At first, one-dimensional projections to the coordinate axes of the data are used. The icon-based overview plot (figure 5.12(a)) shows that only the dimensions 0-2, 4, 5, 12, and 15-18 are useful for separating clusters. The color density plots (figure 5.12(b)) allow a specification of the separators.

In the second step, we use two-dimensional projections. The icon-based overview plot (figure 5.13) shows many two-dimensional projections with well separated clusters. The two-dimensional density plots (figure 5.14) offer good possibilities for polygonal separators. The plots also allow deeper insight into the correlations between the dimensions.

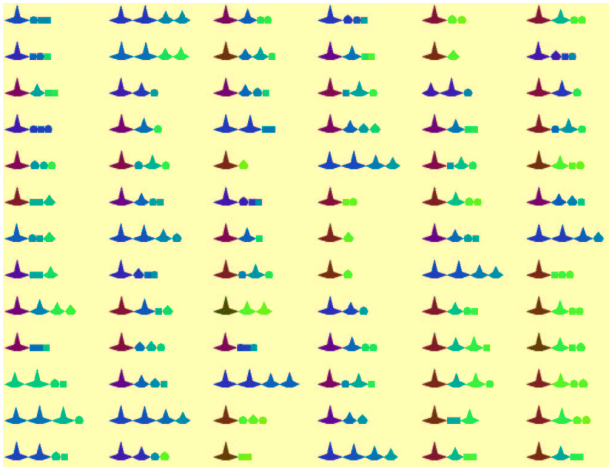


Figure 5.13: Separators for the Molecular Biology Data (19 dimensional)

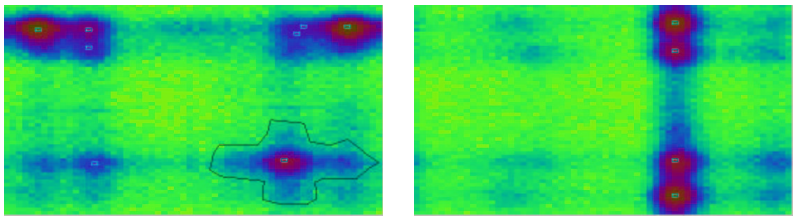
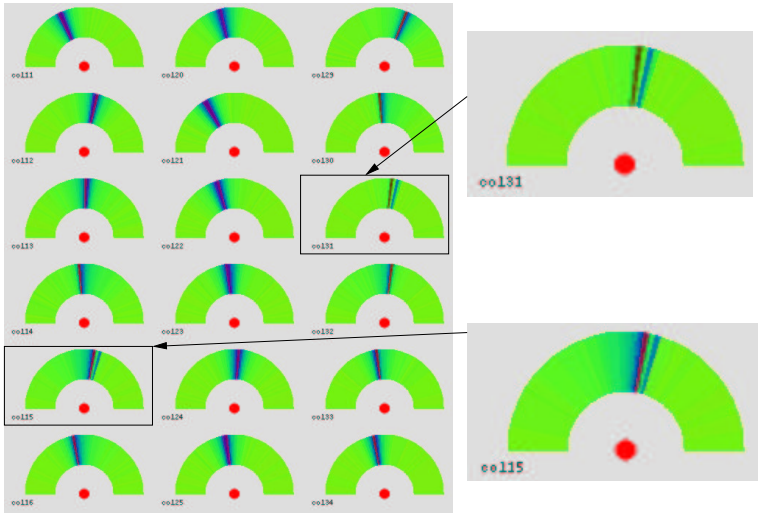
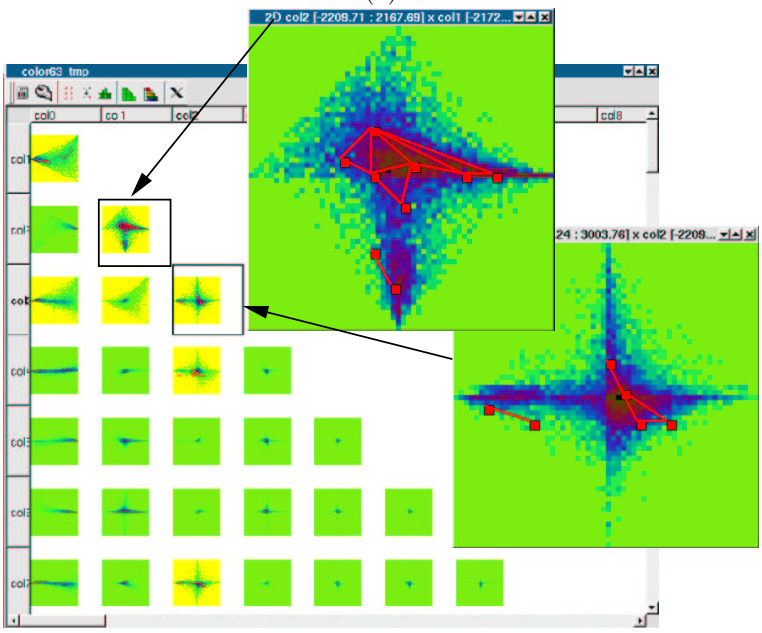


Figure 5.14: Separators for the Molecular Biology Data (19 dimensional)

In the second experiment we examined the image data set which we also used in the previous section to gain more insight into the structure of the data. First we visualized the one-dimensional projections and found two one-dimensional projections, which allow good cluster splitting (see figure 5.15 a). This explains the two cluster cluster separators found by the experiments in the previous section. Using two-dimensional projections we also found good separators (figure 5.15 b). These separators can not be found by one-dimensional projections and either by the projected clustering algorithm described in the previous section. The visualization also allowed much more insight into the structure of the data. So for example is the linear-dependend border of the data in the upper part of the left two-dimensional visualization an interesting feature which would be missed by automated algorithms.



(a)



(b)

Figure 5.15: Part (a) shows the two one-dimensional projection which is important for the projected clusters found in the previous chapter. The second part shows interesting two-dimensional projections with insights about the data. For example the linear-dependend border of the data in the upper part of the left two-dimensional visualization is an interesting feature, which would be missed by automated algorithms.

Chapter 6

Conclusions

The problem of finding valuable information and knowledge in huge masses of data is still a growing problem. This work tries to contribute to the ongoing research in data mining, which attempts to handle the knowledge discovery in large amounts of data. We focus on three aspects of clustering, a basic methodology in data mining, namely to develop a consistent framework for problems related to clustering, to the new but very difficult problem of projected clustering to handle effectively high dimensional data and the visual support of clustering to bridge the semantic gap between the user and the automated algorithms.

In the first part of the work we developed a consistent framework for clustering based on primitives (density estimators and cluster separators). One contribution of the framework is the decoupling of density estimation and clustering. This shows that the theory of density estimation can provide a sound statistical foundation for clustering. The foundation on density estimation allows to use different scaling methods for density estimation directly to scale the clustering algorithm. We showed that our new concept allows the development of new algorithms for finding density-based single-linkage clustering or local outliers with lower complexities than other known algorithms. The second contribution of our framework is the clear distinction between decisions and scaling issues. While the choice of the needed efficiency and effectivity of the used density estimator is a question of scaling the clustering algorithm, the decision, which type of separator is best applied in a given context, strongly depends on the application and the task at hand. These types of decisions can be made only by an human expert. We introduced examples which describe typical situations for the use of the particular cluster separator types. Our new concept of separator trees allows to build very complex clustering models, which also can consists of a mixture of separators of different types.

From the use of the density estimation theory we learned that clustering in high dimensional spaces is very limited, which is due the sparsity of the space. The way out is to find clusters in the projections. We developed an new notion of nearest neighbor search in high dimensional spaces, which uses only the relevant attributes (which form a projected subspace) to determine the distance between data points. New is, that the relevant attributes may change for different points. From this points we started to develop an projected clustering algorithm. Firstly, we reviewed shortly existing work about this new clustering concept and disclosed their problems. Secondly, we developed a new projected clustering algorithm, which overcomes the problems and finds clusters, which are missed by the other algorithms. New features of our algorithm are the possibility to find overlapping clusters, the algorithm is not forced to partition the data, that the found clusters can be characterized by simple cluster descriptions and finally that the relationships among the clusters can be visualized using the similarity matrix of the partial cluster descriptions. The feature that overlapping cluster are found is very important for projected clustering, because found relevant projections may correspond to different tasks or intentions, which are equally meaningful. So when different similarities can be meaningful for the same data, also the same object may belong to several clusters. Also the forced partitioning of the data should be avoided, since there could be objects which do not fit into any cluster. We also identified interesting possibilities to extend our

algorithm to find more complex projected clusters, which have dependencies between their relevant attributes.

The last part of this thesis deals with the visual support for clustering and projected clustering. We developed new visualization techniques, which help the user to disclose hidden properties of the data, allow the selection of useful projections for clustering and support the tuning of parameters for automated clustering procedures. The use of visualization is very important to bridge the semantic gap between the user's intention and the requirements of the clustering system. We showed that visualization effectively supports the tuning of parameters and so enables the detection of structure, which is missed otherwise. We implemented the ideas in a visual clustering system, called *HD-Eye*, which allows many possibilities of interaction and enables the user to influence the clustering method according to her/his domain knowledge for finding more interesting and meaningful patterns.

As future research an integration of the projected clustering algorithm into the *HD-Eye* system will be interesting. Also the development of an interface, which allows the integration of other cluster algorithms, will be helpful to extend the functionality of the tool. Another interesting extension is the development of database supported density estimators, which allow to perform the significant work of density estimation directly in the database management system. This is a way to make clustering more scalable, since significant amounts of work can be saved when the database coordinates the density estimation. This allows the extensive reuse of provisional results.

After the years of the hype related to data mining the research field had become more mature. New specific application areas of data mining appeared, like data mining of data related to the internet, data mining in biological domains, multimedia data mining and text mining. It would be a challenging task to develop clustering algorithms based on our separator framework, which are specific for the different domains. Also user studies with the visual system in the specific domains would be interesting to identify the special needs related to different scientific research communities.

Bibliography

[1] Keim D. A. and Kriegel H.-P. Visdb: Database exploration using multidimensional visualization. *Computer Graphics & Applications*, pages 40–49, 1994.

[2] Charu C. Aggarwal. Towards long pattern generation in dense databases. *SIGKDD Explorations*, 3(1):20–26, 2001.

[3] Charu C. Aggarwal, Cecilia Magdalena Procopiuc, Joel L. Wolf, Philip S. Yu, and Jong Soo Park. Fast algorithms for projected clustering. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 61–72. ACM Press, 1999.

[4] Charu C. Aggarwal and Philip S. Yu. Finding generalized projected clusters in high dimensional spaces. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, pages 70–81. ACM, 2000.

[5] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. In *SIGMOD'2001, Proc. of the ACM Intl Conf. on Management of Data*, pages 427–438. ACM Press, 2001.

[6] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. Research Report RJ 9839, IBM Almaden Research Center, San Jose, California, 1994.

[7] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, 1998, Seattle, Washington, USA*, pages 94–105. ACM Press, 1998.

[8] C. Ahlberg and B. Shneiderman. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In *Proc. ACM CHI Int. Conf. on Human Factors in Computing, Boston, MA*, pages 313–317. ACM Press, 1994.

[9] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration: An implementation and evaluation. In *Proc. ACM CHI Int. Conf. on Human Factors in Computing, Monterey, CA*, pages 619–626, 1992.

[10] S. F. Altschul, W. Gisha, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[11] M. Ankerst, D. A. Keim, and H.-P. Kriegel. Circle segments: A technique for visually exploring large multidimensional data sets. In *Proc. Visualization 1996, Hot Topic Session, San Francisco, CA*, 1996.

[12] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: Ordering points to identify the clustering structure. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 49–60. ACM Press, 1999.

[13] Mihael Ankerst, Hans-Peter Kriegel, and Thomas Seidl. A multistep approach for shape similarity search in image databases. *TKDE*, 10(6):996–1004, 1998.

[14] V. Anupam, S. Dar, T. Leibfried, and E. Petajan. Dataspace: 3-d visualization of large databases. In *Proc. Int. Symp. on Information Visualization, Atlanta, GA*, pages 82–88, 1995.

[15] D. Asimov. The grand tour: A tool for viewing multidimensional data. *SIAM Journal of Science & Stat. Comp.*, 6:128–143, 1985.

[16] Franz Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23:345 – 405, 1991.

[17] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, New York, 1994.

[18] R. A. Becker, S. G. Eick, and G. J. Wills. Visualizing network data. *IEEE Transactions on Visualizations and Graphics*, 1(1):16–28, 1995.

[19] J. Beddow. Shape coding of multidimensional data on a mircocomputer display. In *Proc. Visualization 1990, San Francisco, CA*, pages 238–246, 1990.

[20] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The x-tree : An index structure for high-dimensional data. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 28–39. Morgan Kaufmann, 1996.

[21] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer, 1999.

[22] Umeshwar Dayal Bin Zhang, Meichun Hsu. K-harmonic means - a spatial clustering algorithm with boosting. In *Temporal, Spatial, and Spatio-Temporal Data Mining, First International Workshop TSDM 2000*, pages 31–45. Springer, 2000.

[23] H. H. Bock. *Autmatic Classification*. Vandenhoeck and Ruprecht, Göttingen, 1974.

[24] Francesco Bonchi, Fosca Giannotti, Gianni Mainetto, and Dino Pedreschi. Using data mining techniques in fiscal fraud detection. In *Data Warehousing and Knowledge Discovery, First International Conference, DaWaK '99, Florence, Italy*, pages 369–376, 1999.

[25] Christian Borgelt and Rudolf Kruse. Induction of association rules: Apriori implementation. In *Accepted to the 14th Conference on Computational Statistics (Compstat 2002, Berlin, Germany)*, 2002.

[26] Paul S. Bradley, Usama M. Fayyad, and Cory A. Reina. Scaling clustering algoritms to large databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, pages 9–15. AAAI Press, 1998.

[27] Paul S. Bradley, Usama M. Fayyad, and Cory A. Reina. Scaling em (expectation maximization) clustering to large databases. Technical Report MSR-TR-98-35, Microsoft Research, 1999.

[28] M. M. Breunig, H.P. Kriegel, R. Ng, and J. Sander. Optics of: Identifying local outliers. In *In Lecture Notes in Computer Science*, volume 1704, pages 262–270. Springer, 1999.

[29] S. Breunig, H.-P. Kriegel, R. Ng, and J.Sander. Lof: Identifying density-based local outliers. In *SIGMOD'2000, Proc. of the ACM Intl Conf. on Management of Data*, pages 93–104. ACM Press, 2000.

[30] A. Buja, J. A. McDonald, J. Michalak, and W. Stuetzle. Interactive data visualization using focusing and linking. In *Proc. Visualization 1991, San Diego, CA*, pages 156–163, 1991.

[31] X. Daura, B. Jaun, D. Seebach, W. F. van Gunsteren, and A. E. Mark. Reversible peptide folding in solution by molecular dynamics simulation. *Journal of Molecular Biology*, 280:925–932, 1998.

[32] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD'96, Proc. of the 2nd Intl Conf. on Knowledge Discovery and Data Mining*, pages 226–231. AAAI Press, 1996.

[33] V. Estivill-Castro. Why so many clustering algorithms – a position paper. *Newsletter of the ACM Special Interest Group on Knowledge Discovery and Data Mining*, 4(1):65–75, June 2002.

[34] C. Faloutsos, R. Barber, M. Flickner, and J. Hafner. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[35] C. Faloutsos and K.D. Lin. Fastmap: A fast algorithm for indexing,data-mining and visualisation of traditional and multimedia datasets. In *Proc. ACM SIGMOD Int. Conf. on Managment of Data*. ACM Press, 1995.

[36] B. Fritzke. A growing neural gas network learns topologies. *Advances in Neural Information Processing Systems*, 7:625–632, 1995.

[37] B. Fritzke. The lbg-u method for vector quantization - an improvement over lbg inspired from neural networks. *Neural Processing Letters*, 5(1), 1997.

[38] K. Fukunaga and L.D. Hostler. The estimation of the gradient of a density function, with application in pattern recognation. *IEEE Trans. Info. Thy.*, 21:32–40, 1975.

[39] G. W. Furnas and A. Buja. Prosections views: Dimensional inference through sections and projections. *Journal of Computational and Graphical Statistics*, 3(4):323–353, 1994.

[40] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[41] A. Gersho and R. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1992.

[42] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: An efficient clustering algorithm for large databases. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 73–84. ACM Press, 1998.

[43] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57. ACM Press, 1984.

[44] James L. Hafner, Harpreet S. Sawhney, William Equitz, Myron Flickner, and Wayne Niblack. Efficient color histogram indexing for quadratic form distance functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(7):729–736, 1995.

[45] Maria Halkidi and Michalis Vazirgiannis. A data set oriented approach for clustering algorithm selection. In *rinciples of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001, Freiburg, Germany, September 3-5, 2001*, pages 165–179. Springer, 2001.

[46] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *SIGMOD 2000, Proceedings ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2000.

[47] W. Härdle and D. W. Scott. Smoothing in low and high dimensions by weighted averaging using rounded points. *Comput. Statist.*, 7:97–128, 1992.

[48] D. Hawkins. *Identification of outliers*. Chapman and Hall, London, 1980.

[49] Hongxing He, Warwick Graco, and Xin Yao. Application of genetic algorithm and k-nearest neighbour method in medical fraud detection. In *Simulated Evolution and Learning, Second Asia-Pacific Conference on Simulated Evolution and Learning, SEAL'98, Canberra, Australia*, pages 74–81. Lecture Notes in Computer Science, 1998.

[50] A. Hinneburg and D.A. Keim. An efficient approach to clustering in large multimedia databases with noise. In *KDD'98, Proc. of the 4th Int. Conf. on Knowledge Discovery and Data Mining*, pages 58–65. AAAI Press, 1998.

[51] A. Hinneburg, M. Wawryniuk, and D. A. Keim. Hdeye: Visual mining of high-dimensional data. *Computer Graphics & Applications Journal*, 19(5):22–31, September/October 1999.

[52] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? In *VLDB'2000, Proceedings of 26th International Conference on Very Large Data Bases, Cairo, Egypt*, pages 506–515. Morgan Kaufmann, 2000.

[53] Alexander Hinneburg and Daniel Keim. A general approach to clustering in large databases with noise. *accepted for publication in KAIS*, 2002.

[54] Alexander Hinneburg and Daniel A. Keim. Clustering methods for large databases: From the past to the future. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, page 509. ACM Press, 1999.

[55] Alexander Hinneburg and Daniel A. Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 506–517. Morgan Kaufmann, 1999.

[56] Alexander Hinneburg and Daniel A. Keim. Clustering techniques for large data sets - from the past to the future. In *Tutorial Notes for PKDD 2000 International Conference on Principles and Practize in Knowledge Discovery and Data Mining*, Lyon , France, 2000. Springer.

[57] Alexander Hinneburg, Daniel A. Keim, and Markus Wawryniuk. Hdeye: Visual mining of high-dimensional data (demo). In *SIGMOD 2002, Proceedings ACM SIGMOD International Conference on Management of Data, June 3-6, 2002, USA*. ACM Press, 2002.

[58] P. J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–474, 1985.

[59] A. Inselberg. The plane with parallel coordinates, special issue on computational geometry. *The Visual Computer*, 1:69–97, 1985.

[60] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proc. Visualization 1990, San Francisco, CA*, pages 361–370, 1990.

[61] H. V. Jagadish. A retrieval technique for similar shapes. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 29-31, 1991*, pages 208–217. ACM Press, 1991.

[62] Wen Jin, Anthony Tung, and Jiawei Han. Mining top-n local outliers in large databases. In *Proceedings of the seventh conference on Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining*, pages 293 – 298. ACM Press, 2001.

[63] Hans-Peter Kriegel Jörg Sander, Martin Ester and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, 2(2):169–194, 1997.

[64] George Karypis. Cluto, a clustering toolkit, release 2.0, 2002.

[65] D. Keim, M. Heczko, and R. Weber. Analysis of the Effectiveness-Efficiency Dependance for Image Retrieval. In *Delos Workshop Zuerich*, 2000.

[66] D. A. Keim. *Visual Support for Query Specification and Data Mining*. Shaker-Publishing Company, Aachen, Germany, 1995.

[67] Daniel A. Keim and Alexander Hinneburg. Clustering techniques for large data sets - from the past to the future. In *Tutorial Notes for ACM SIGKDD 1999 International Conference on Knowledge Discovery and Data Mining*, pages 141–181, San Diego, CA, 1999. ACM Press.

[68] Edwin M. Knorr and Raymond T. Ng. Algorithms for mining distance-based outliers in large datasets. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 392–403. Morgan Kaufmann, 1998.

[69] Edwin M. Knorr, Raymond T. Ng, and V. Tucakov. Distance-based outliers: Algorithms and applications. *VLDB Journal*, 8(3-4):237–253, 2000.

[70] T. Kohonen, K. Mkisara, O.Simula, and J. Kangas. *Artificial Networks*. Amsterdam, 1991.

[71] Flip Korn, Nikolaos Sidiropoulos, Christos Faloutsos, Eliot Siegel, and Zenon Protopapas. Fast nearest neighbor search in medical image databases. In *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 215–226, 1996.

[72] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, 1992.

[73] Pirolli P. Lamping J., Rao R. A focus + context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proc. ACM CHI Conf. on Human Factors in Computing (CHI95)*, pages 401–408. ACM Press, 1995.

[74] J. LeBlanc, M. O. Ward, and N. Wittels. Exploring n-dimensional databases. In *Proc. Visualization 1990, San Francisco, CA*, pages 230–237, 1990.

[75] Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer. *IEEE Trans. Comm.*, COM-28:84–95, 1980.

[76] J. MacQueen. Some methods for classification and analysis of multivariate observations. *5th Berkley Symp. Math. Statist. Prob.*, 1:281–297, 1967.

[77] T. Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. In *ICANN'93: International Conference on Artificial Neural Networks*, pages 427–434, Amsterdam, 1993. Springer.

[78] T. Martinetz and K. J. Schulten. *A Neural-Gas Network Learns Topologies*, pages 397–402. Elsevier Science Publishers, North-Holland, 1991.

[79] T. Martinetz and K. J. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[80] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.

[81] Rajiv Mehrotra and James E. Gary. Feature-index-based similar shape retrieval. In *Proc. of the 3rd Working Conf. on Visual Database Systems*, pages 46–65, 1995.

[82] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 144–155. Morgan Kaufmann, 1994.

[83] S. Openshaw. *Census User Handbook*. Pearson Professsional Ltd, Cambridge, 1995.

[84] G. P. Patil, M. T. Boswell, S. W. Joshi, M. V. Ratnaparkhi, and J. J. J. Roux. *Dictionary and Classified Bibliography of Statistical Distribution in Scientific Work, Vol. 1, Discrete Models*. International Co-operative Publ. House, Burtonville, MD, 1985.

[85] R. M. Pickett and G. G. Grinstein. Iconographic displays for visualizing multidimensional data. In *Proc. IEEE Conf. on Systems, Man and Cybernetics, Piscataway, NJ*, pages 514–519. IEEE Press, 1988.

[86] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *"Simple Monte Carlo Integration" and "Adaptive and Recursive Monte Carlo Methods." 7.6 and 7.8 in Numerical Recipes in FORTRAN: The Art of Scientific Computing, 2nd ed.* Cambridge University Press, 295-299 and 306-319, 1992.

[87] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. In *SIGMOD'2000, Proc. of the ACM Int. Conf. on Management of Data*, pages 427–438. ACM Press, 2000.

[88] G. Robertson, S. Card, and J. Mackinlay. Cone trees: Animated 3d visualizations of hierarchical information. In *Proc. ACM CHI Int. Conf. on Human Factors in Computing*, pages 189–194, 1991.

[89] R. Rojas. *Neural Networks A Systematic Introduction*. Springer, Berlin, 1996.

[90] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Publishing Company.

[91] Brown M. Sarkar M. Graphical fisheye views. *Communications of the ACM*, 37(12):73–84, 1994.

[92] Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An efficient algorithm for mining association rules in large databases. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*, pages 432–444. Morgan Kaufmann, 1995.

[93] Erich Schikuta. Grid-clustering: An efficient hierarchical clustering method for very large data sets. In *In Proc. 13th Int. Conf. on Pattern Recognition, volume 2*, pages 101–105, Vienna, Austria, October 1996. IEEE Computer Society Press.

[94] P. Schnell. A method to find point-groups. *Biometrika*, 6:47–48, 1964.

[95] D. Scott and J. Thompson. Probability density estimation in higher dimensions. In *Proceedings of the Fifteenth Symposium on the Interface*, pages 173–179. North Holland-Elsevier Science Publishers, 1983.

[96] D. W. Scott. Average shifted histograms: Effective nonparametric density estimators in several dimensions. *Ann. Statist.*, 13:1024–1040, 1985.

[97] D. W. Scott and S. J. Sheather. Kernel density estimation with binned data. *Comm. Statist. Theory Meth.*, 14:1353–1359, 1985.

[98] D.W. Scott. *Multivariate Density Estimation*. Wiley and Sons, 1992.

[99] S.Eick and G. J. Wills. Navigating large networks with hierarchies. In *Proc. Visualization 1993, San Jose, CA*, pages 204–210, 1993.

[100] Thomas Seidl and Hans-Peter Kriegel. Efficient user-adaptable similarity search in large multimedia databases. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 506–515, 1997.

[101] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. Wavecluster: A multi-resolution clustering approach for very large spatial databases. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 428–439. Morgan Kaufmann, 1998.

[102] B. Shneiderman. Tree visualization with treemaps: A 2d space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[103] R. Sibson. Slink: an optimally efficient algorithm for the single-linkage cluster method. *The Computer Journal*, 16(1):30–34, 1973.

[104] B. W. Silverman. Kernel density estimation using the fast fourier transformation. *Appl. Stat.*, 31:93–99, 1982.

[105] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

[106] Hannu Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 134–145. Morgan Kaufmann, 1996.

[107] T. Wallace and P. Wintz. An efficient three-dimensional aircraft recognition algorithm using normalized fourier descriptors. *Computer Graphics and Image Processing*, 13:99–126, 1980.

[108] M. P. Wand and M. C. Jones. *Kernel Smoothing*. Chapman & Hall, 1995.

[109] Wei Wang, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 186–195. Morgan Kaufmann, 1997.

[110] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 194–205, 1998.

[111] D. Wishart. Mode analysis: A generalisation of nearest neighbor, which reducing chaining effects. pages 282–312. A. J. Cole, 1969.

[112] Xiaowei Xu, Martin Ester, Hans-Peter Kriegel, and Jörg Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 324–331. IEEE Computer Society, 1998.

[113] Bin Zhang. Generalized k-harmonic means – boosting in unsupervised learning. Technical Report HPL-2000-137, HP Research Labs, 2000.

[114] Bin Zhang, Meichun Hsu, and Umeshwar Dayal. K-harmonic means - a data clustering algorithm. Technical Report HPL-1999-124, HP Research Labs, 1999.

[115] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 103–114. ACM Press, 1996.

[116] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Fast density estimation using cf-kernel for very large databases. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 312–316. ACM, 1999.