# 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 $k$.*

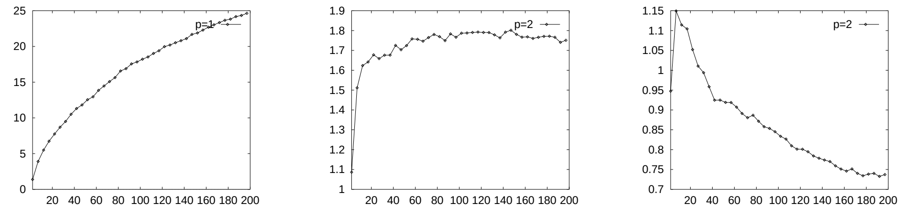
**Proof:** If $C$ is the expected difference between the maximum and minimum of two randomly drawn points, then the same value for $n$ points drawn from the same distribution must be in the range $[C, (n-1)\cdot C]$. ■

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 ← 0
  for i ← 0 to no_iter do
    p ← generate_random_projection( d_tar )
    p.quality ← C(p, x_q, D, dist)
    if p_best.quality < p.quality then
      p_best ← p
    end if
  end for
  return( p_best )
```

**Algorithm 7** Genetic Optimization

```
genetic_search (x_q, d_tar, D, C, dist, no_iter)
  population ← ∅, pop_size ← 100, elite ← 10, child ← 80
  for i := 0 to pop_size do
    p ← generate_random_projection( d_tar )
    p.quality ← C(p, x_q, D, dist)
    population.insert(p)
  end for
  for i ← 0 to no_iter do
    new_pop ← ∅
    insert the elite best projection into new_pop
    for j ← elite to elite + child do
      {projections with high quality have higher probability to be selected for cross-over}
      parent1 ← randomly select a projection from old_pop
      parent2 ← randomly select a projection from old_pop
      child ← gen. a new proj. by comb. parent1, parent2
      child.quality ← 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 ← 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 $dist_t$:

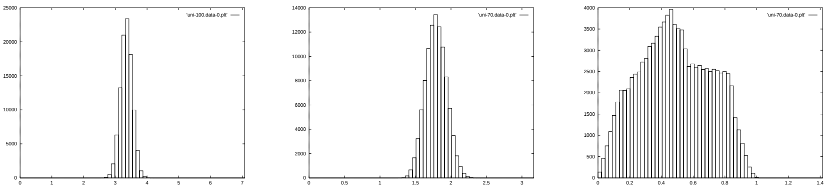
$$\Phi(dist_t) = P[dist(x_q, x) < dist_t], x \in D, dist_t \in \mathbb{R}$$

The corresponding probability density is

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

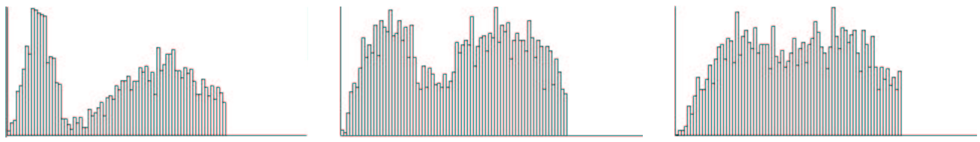
Note that $\Phi(dist_t)$ is not continuous and therefore we can only estimate the probability density $f(dist_t)$. In this subsection, we use simple histograms showing the distances of the data points from random query points.

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.

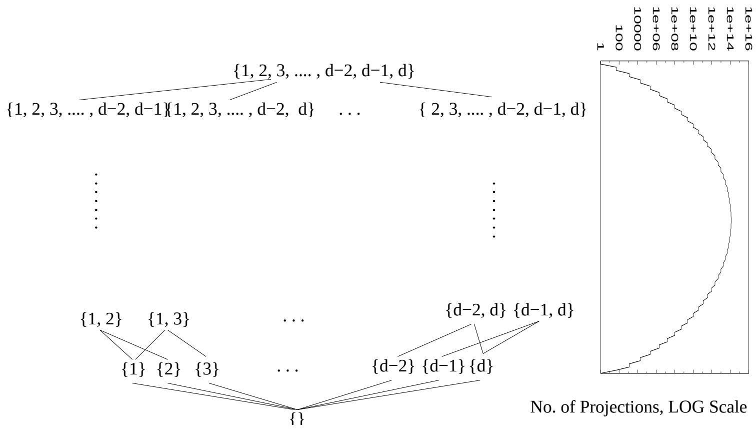



**Figure 4.4:** The space of axes parallel projections forms a lattices of subsets. The plot on the right side shows the number of projections of a fixed dimensionality $d'$, which is like $\binom{d}{d'}$. The plot comes from a 50 dimensional space, but the proportions are similar in other spaces.

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.

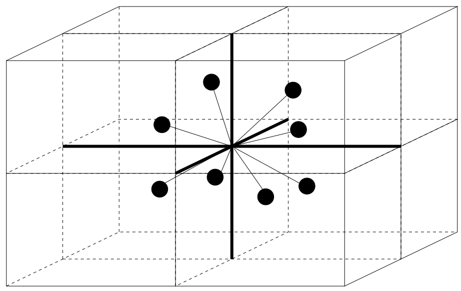


**Figure 4.5:** Example Scenario for a Normal Distribution, $d = 3$

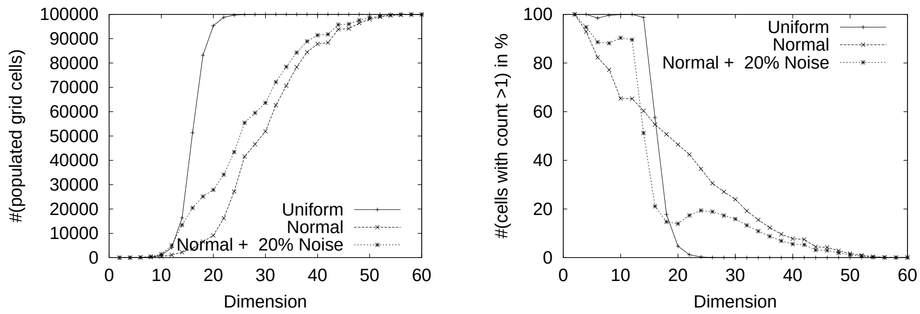
The problem is that the number of data points can not grow exponentially with the dimension, and therefore, in high-dimensional space it is generally impossible to determine the distribution of the data with sufficient statistical significance. (The only thing which can be verified easily is that the 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 ($\sigma$). 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.

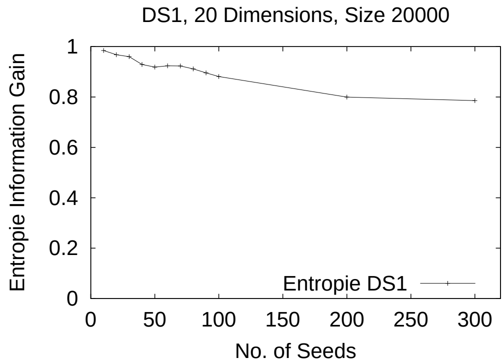



**Figure 4.7:** The diagram shows the entropy depending on the number of randomly chosen seeds for data set DS1. The data set consists of 20000 data points with 20 dimensions and contains two equally sized projected clusters. Each axes-parallel projected cluster has 3 relevant and 17 non-relevant dimensions. 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.

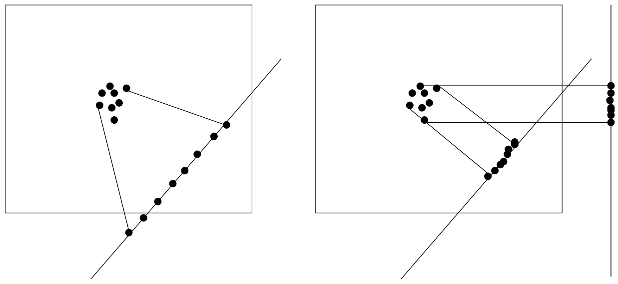
To elaborate the splitting issue we simulated the initialization of ORCLUS [7] by picking seeds randomly from the data as described in the paper [4]. We generated a 20-dimensional data set (to which is later referred as data set DS1) with two axes-parallel projected clusters of same size, each with three 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 $P : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ 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. ■

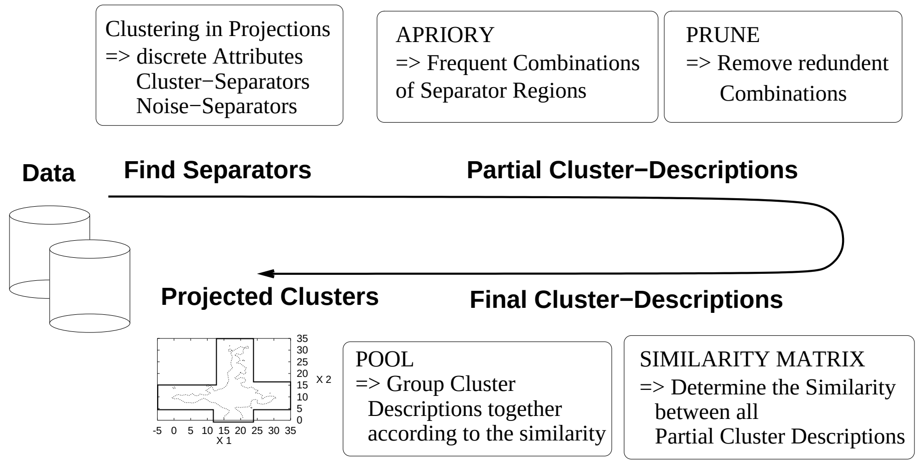



**Figure 4.9:** The first step is to find noise and cluster separators in projections of the data. This can be seen as a transformation of the continuous vector data into discrete transaction. A transaction consists of the separator regions which include the original data point. Second, the transactions 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 $\epsilon$ 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 $\xi$ 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.

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

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 $\xi$ the noise threshold.

**Ensure:** $M_{sep}$ contains only separating minima.

**for all** $x_i \in M$ **do**
  $x_i.lMax \leftarrow$ Determine the maximum density in the left interval $[x_{i-1}, x_i]$
  $x_{min}.rMax \leftarrow$ 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 $[x_{last}, max]$.$\}$
**end for**
$i \leftarrow 1$; $M_{del} \leftarrow \emptyset$
**while** $x_i \in M$ and $x_i.lMax < \xi$ **do**
  $M_{del} \leftarrow M_{del} \cup \{x_i\}$; $i \leftarrow i + 1$
**end while**
$left \leftarrow 0$; $right \leftarrow 0$
**for** $i$ to $last$ **do**
  **if** $x_i.lMax \geq \xi$ **then**
    $left \leftarrow i$
  **end if**
  **if** $x_i.rMax \geq \xi$ **then**
    $right \leftarrow i$
  **end if**
  **if** $left \neq 0$ and $right \neq 0$ **then**
    $x_{remain} \leftarrow x_{remain} \in \{x_i : left \leq i \leq right\}$ and $f^{P(F)}(x_{remain}) \leq f^{P(F)}(x_i)$.
    $M_{del} \leftarrow M_{del} \cup \{x_i : left \leq i \leq right\} - \{x_{remain}\}$
    $left \leftarrow 0$; $right \leftarrow 0$
  **end if**
**end for**
**if** $left \neq 0$ **then**
  $M_{del} \leftarrow M_{del} \cup \{x_i : left \leq i \leq last\}$
**end if**
$M_{sep} \leftarrow M - M_{del}$
**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**$(D, \xi)$

Determine histograms for all projections $P_1, \ldots, P_d$<br>
$S^C \leftarrow \emptyset$, $S^N \leftarrow \emptyset$ {Sets of Separators (Cluster, Noise)}<br>
**for all** histograms $h_1, \ldots, h_d$ **do**<br>
    $S^C \leftarrow S^C \cup$ findClusterSeparator$(h_i, \xi)$<br>
    **if** no Cluster Separator Found **then**<br>
        $S^N \leftarrow S^N \cup$ findNoiseSeparator$(h_i, \xi)$<br>
        **if** no Noise Separator found **then**<br>
            mark the current dimension as non-relevant<br>
        **end if**<br>
    **end if**<br>
**end for**<br>
**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

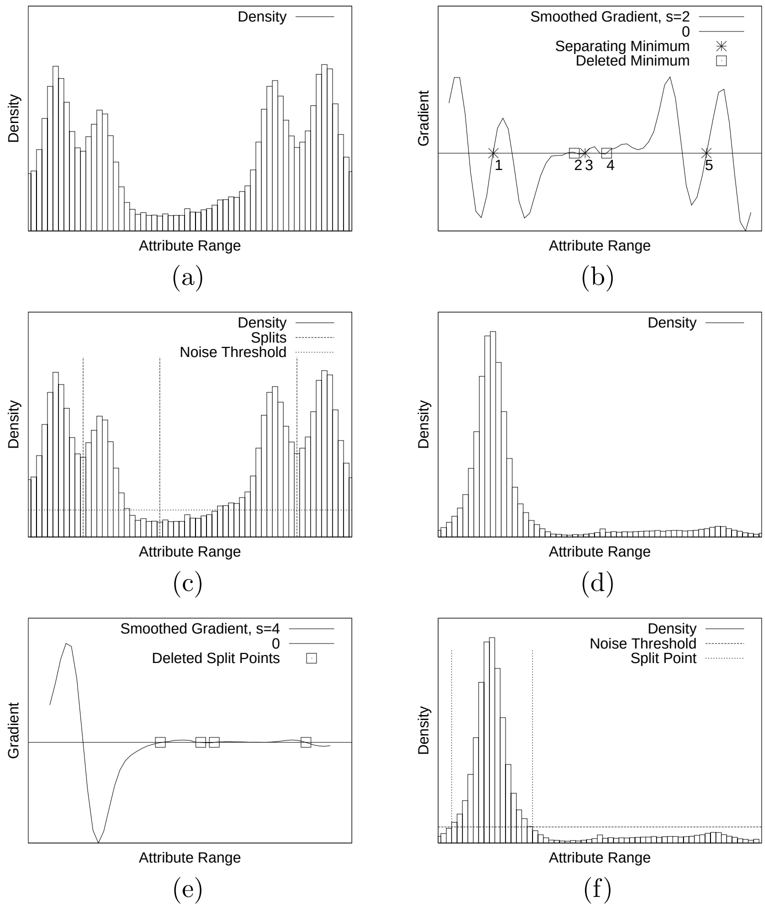



**Figure 4.10:** Part (a) shows a density function (histogram) of a one dimensional projection (source: attribute 5 of the molecular biology data). Part (b) shows the smoothed gradient (s=2) with the zero points for the minimums. Since not all minimums separates clusters, two of $\{2, 3, 4\}$ have to 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:

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

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 $x_l \in D, l = 1, \dots, N$ 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].

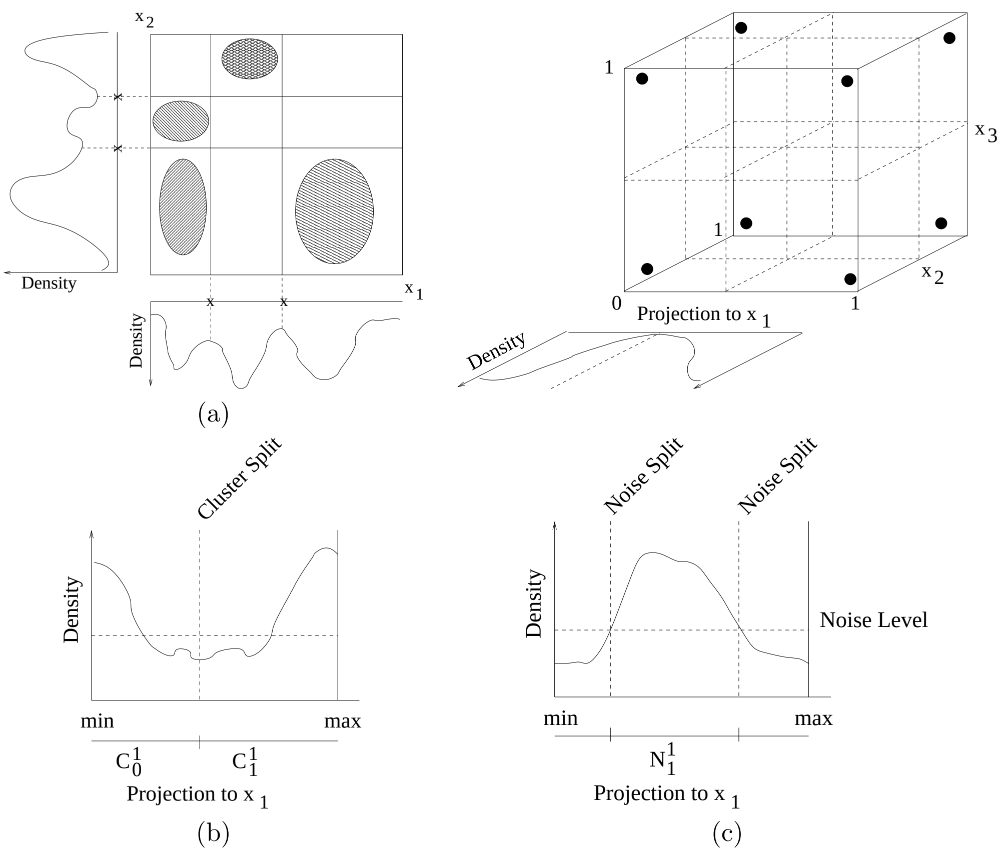



**Figure 4.11:** Part (a) shows a regular grid determined by separators from axes parallel one dimensional projections. It shows that clusters in the multidimensional space can not be determined by looking on one dimensional projections. Part (b) is an illustration for an example that the use of high 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

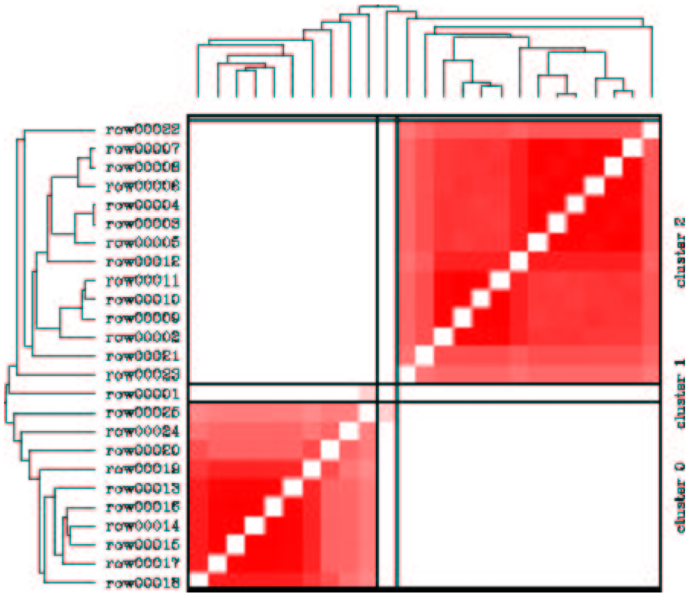


**Figure 4.12:** The figure shows a hierarchical clustering (complete linkage) and the block diagonalized similarity matrix of the partial cluster descriptions. The used data set has 20 dimensions and contains two projected clusters.

**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.

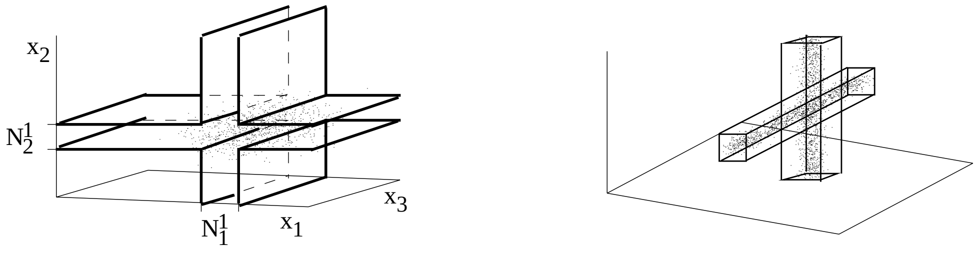
The process of finding axes parallel cluster as explained is summarized in algorithm 12.

### 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

---

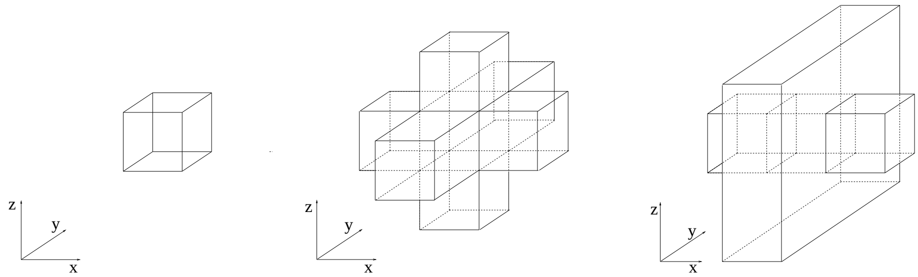
**AP_clustering**($D, \xi, minsup$)

**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, $minsup = 30\%$

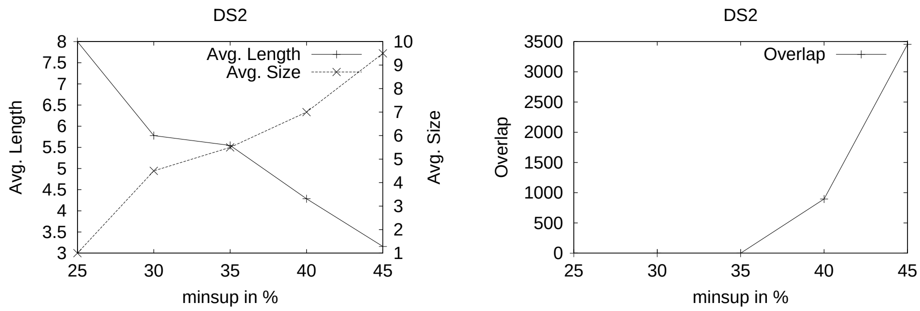
| In/Out | $C_1$ | $C_2$ |
|---|---|---|
| Class 1 | 6691 | 0 |
| Class 2 | 0 | 5113 |

(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



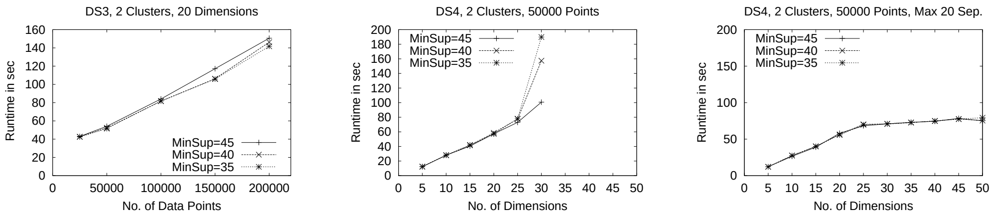

(a) (b)

**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).




(a) (b) (c)

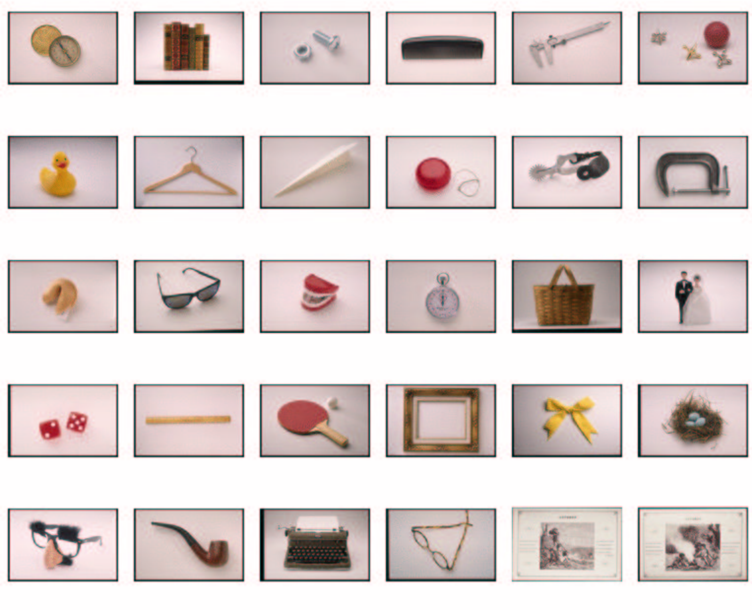
**Figure 4.16:** Part (a) shows the runtime depending on the number of data points. The used data DS3 contains two equally sized clusters and is 20-dimensional. The data set DS4 contains 50000 data data points and two clusters. The number of relevant dimensions is half of the full dimensionality. Part (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.

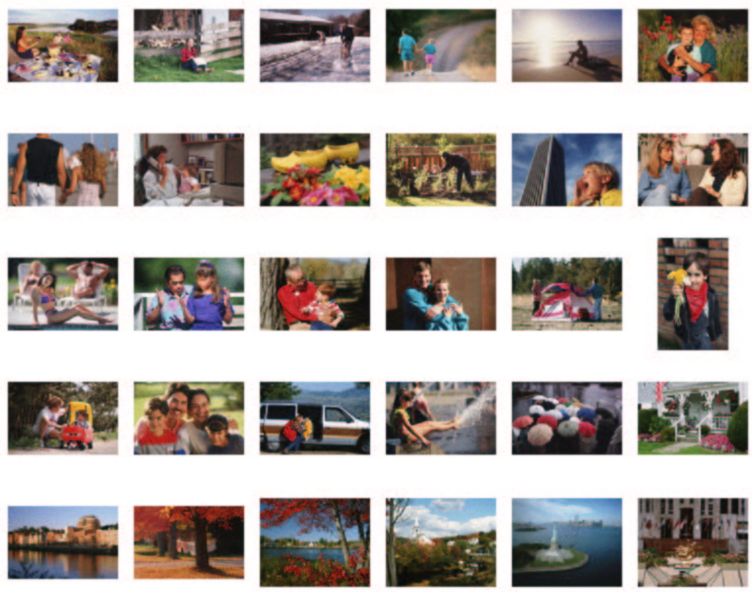
**Application to real data** After showing the behavior of our new algorithm we apply it to two different real data sets. The first data set is derived from a collection of 8537 images. For each image the gray scale color histogram was derived. The histograms have been wavelet-transformed into a 64-dimensional 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.

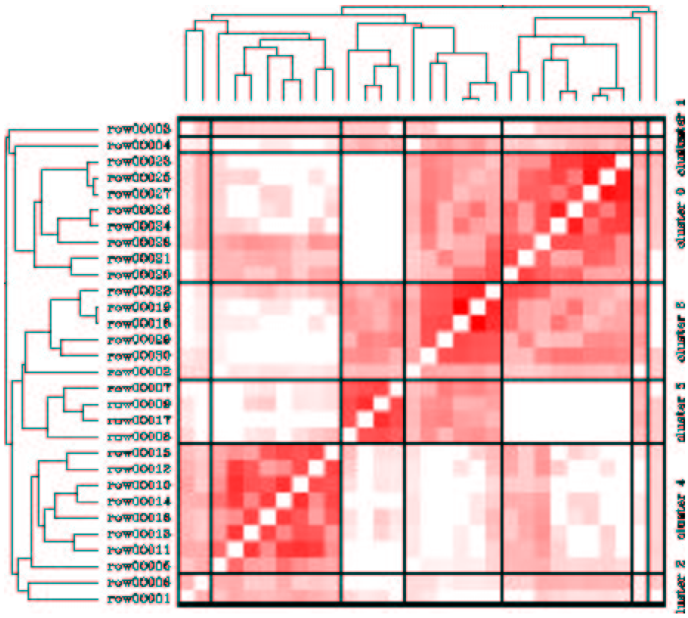


**Figure 4.18:** The figure shows the similarity matrix of the partial cluster descriptions for the molecular biology data. The similarity matrix shows four main clusters which overlap each other. This goes along with the intuition about the application domain that the simulated molecule has intermediate 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.

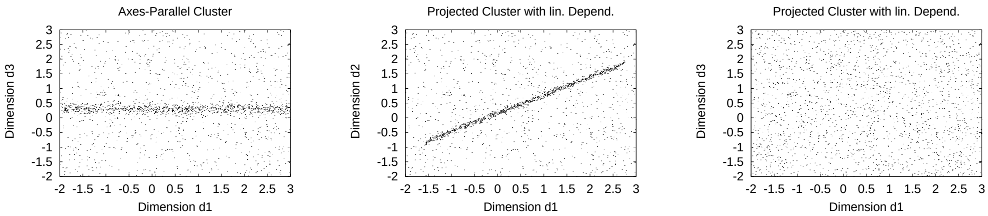
So the problem is, how to find good separators using multi-dimensional projections. The problem can be decomposed into three sub-problems:

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.