Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications

Rakesh Agrawal **Johannes Gehrke*** **Dimitrios Gunopulos** **Prabhakar Raghavan**

IBM Almaden Research Center
650 Harry Road, San Jose, CA 95120

Abstract

Data mining applications place special requirements on clustering algorithms including: the ability to find clusters embedded in subspaces of high dimensional data, scalability, end-user comprehensibility of the results, non-presumption of any canonical data distribution, and insensitivity to the order of input records. We present CLIQUE, a clustering algorithm that satisfies each of these requirements. CLIQUE identifies dense clusters in subspaces of maximum dimensionality. It generates cluster descriptions in the form of DNF expressions that are minimized for ease of comprehension. It produces identical results irrespective of the order in which input records are presented and does not presume any specific mathematical form for data distribution. Through experiments, we show that CLIQUE efficiently finds accurate clusters in large high dimensional datasets.

1 Introduction

Clustering is a descriptive task that seeks to identify homogeneous groups of objects based on the values of their attributes (dimensions) [24] [25]. Clustering techniques have been studied extensively in statistics [3], pattern recognition [11] [19], and machine learning [9] [31]. Recent work in the database community includes CLARANS [33], Focused CLARANS [14], BIRCH [45], and DBSCAN [13].

Current clustering techniques can be broadly classified into two categories [24] [25]: *partitional* and *hierarchical*. Given a set of objects and a clustering criterion [39], partitional clustering obtains a partition of the objects into clusters such that the objects in a cluster are more similar to each other than to objects in different clusters. The popular K-means and K-medoid methods determine K cluster representatives and assign each object to the cluster with its representative closest to the object such that the sum of the distances squared between the objects and their representatives is minimized. CLARANS [33], Focused CLARANS [14], and BIRCH [45] can be viewed as extensions of this approach to work against large databases. Mode-seeking clustering methods identify clusters by searching for regions in the data space in which the object density is large. DBSCAN [13] finds dense regions that are separated by low density regions and clusters together the objects in the same dense region.

A hierarchical clustering is a nested sequence of partitions. An agglomerative, hierarchical clustering starts by placing each object in its own cluster and then merges these atomic clusters into larger and larger clusters until all objects are in a single cluster. Divisive, hierarchical clustering reverses the process by starting with all objects in cluster and subdividing into smaller pieces [24].

1.1 Desiderata from the data mining perspective

Emerging data mining applications place the following special requirements on clustering techniques, motivating the need for developing new algorithms:

Effective treatment of high dimensionality: An object (data record) typically has dozens of attributes and the domain for each attribute can be large. It is not meaningful to look for clusters in such a high dimensional space as the average density of points anywhere in the data space is likely to be quite low [6]. Compounding this problem, many dimensions or combinations of dimensions can have noise or values that are uniformly distributed. Therefore, distance functions that use all the dimensions of the data may be ineffective. Moreover, several clusters may exist in different subspaces comprised of different combinations of attributes.

Interpretability of results: Data mining applications typically require cluster descriptions that can be easily assimilated by an end-user as insight and explanations are of critical importance [15]. It is particularly important to have simple representations because most visualization techniques do not work well in high dimensional spaces.

Scalability and usability: The clustering technique should be fast and scale with the number of dimensions and the size of input. It should be insensitive to the order in which the data records are presented. Finally, it should not presume some canonical form for data distribution.

Current clustering techniques do not address all these points adequately, although considerable work has been done in addressing each point separately.

*Current Address: Department of Computer Science, University of Wisconsin, Madison, WI 53706.

The problem of high dimensionality is often tackled by requiring the user to specify the subspace (a subset of the dimensions) for cluster analysis (e.g. [23]) . However, user-identification of subspaces is quite error-prone. Another way to address high dimensionality is to apply a dimensionality reduction method to the dataset. Methods such as principal component analysis or Karhunen-Loève transformation [11] [19] optimally transform the original data space into a lower dimensional space by forming dimensions that are linear combinations of given attributes. The new space has the property that distances between points remain approximately the same as before. While these techniques may succeed in reducing the dimensionality, they have two shortcomings. First, the new dimensions can be difficult to interpret, making it hard to understand clusters in relation to the original data space. Second, these techniques are not effective in identifying clusters that may exist in different subspaces of the original data space. We further discuss these point in the Appendix and Section 4.

Clustering algorithms developed in the database community like BIRCH, CLARANS, and DBSCAN are designed to be scalable, an emphasis not present in the earlier work in the statistics and machine learning literature [33] [45]. However, these techniques were developed to discover clusters in the full dimensional space. It is not surprising therefore that they are not effective in identifying clusters that exist in the subspaces of the original data space. In Section 4, we provide experimental results with BIRCH and DBSCAN in support of this observation.

1.2 Contributions and layout of the paper

We present an algorithm, henceforth referred to as CLIQUE[1], that satisfies the above desiderata. CLIQUE automatically finds subspaces with high-density clusters. It produces identical results irrespective of the order in which the input records are presented and it does not presume any canonical distribution for input data. It generates cluster descriptions in the form of DNF expressions and strives to generate minimal descriptions for ease of comprehension. Empirical evaluation shows that CLIQUE scales linearly with the number of input records, and has good scalability as the number of dimensions (attributes) in the data or the highest dimension in which clusters are embedded is increased.

We begin by formally defining the problem of automatic subspace clustering in Section 2. Section 3 is the heart of the paper where we present CLIQUE. In Section 4, we present a performance evaluation and conclude with a summary in Section 5.

2 Subspace Clustering

Before giving a formal description of the problem of subspace clustering, we first give an intuitive explanation of our clustering model.

We are interested in automatically identifying (in general several) subspaces of a high dimensional data space that allow better clustering of the data points than the original space. Restricting our search to only subspaces of the original space, instead of using new dimensions (for example linear combinations of the original dimensions) is important because this restriction allows much simpler, comprehensible presentation of the results. Each of the original dimensions typically has a real meaning to the user, while even a simple linear combination of many dimensions may be hard to interpret [15].

We use a density based approach to clustering: a cluster is a region that has a higher density of points than its surrounding region. The problem is to automatically identify projections of the input data into a subset of the attributes with the property that these projections include regions of high density.

To approximate the density of the data points, we partition the data space and find the number of points that lie inside each cell (unit) of the partitioning. This is accomplished by partitioning each dimension into the same number of equal length intervals. This means that each unit has the same volume, and therefore the number of points inside it can be used to approximate the density of the unit.

Once the appropriate subspaces are found, the task is to find clusters in the corresponding projections. The data points are separated according to the valleys of the density function. The clusters are unions of connected high density units within a subspace. To simplify their descriptions, we constrain the clusters to be axis-parallel hyper-rectangles.

Each unit in a k-dimensional subspace can be described as a conjunction of inequalities because it is the intersection of $2k$ axis-parallel halfspaces defined by the k 1-dimensional intervals. Since each cluster is a union of such cells, it can be described with a DNF expression. A compact description is obtained by covering a cluster with a minimal number of maximal, possibly overlapping rectangles and describing the cluster as a union of these rectangles.

Subspace clustering is tolerant of missing values in input data. A data point is considered to belong to a particular subspace if the attribute values in this subspace are not missing, irrespective of the values of the rest of the attributes. This allows records with missing values to be used for clustering with more accurate results than replacing missing values with values taken from a distribution.

2.1 Problem Statement

Let $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$ be a set of bounded, totally ordered domains and $\mathcal{S} = A_1 \times A_2 \times \ldots \times A_d$ a d-dimensional numerical space. We will refer to $A_1, \ldots, A_d$ as the dimensions (attributes) of $\mathcal{S}$.

The input consists of a set of d-dimensional points $V = \{v_1, v_2, \ldots, v_m\}$ where $v_i = \langle v_{i1}, v_{i2}, \ldots, v_{id} \rangle$. The jth component of v_i is drawn from domain A_j.

We partition the data space $\mathcal{S}$ into non-overlapping rectangular *units*. The units are obtained by partitioning every dimension into ξ intervals of equal length, which is an input parameter.

Each unit u is the intersection of one interval from each attribute. It has the form $\{u_1, \ldots, u_d\}$ where $u_i = [l_i, h_i)$ is a right-open interval in the partitioning of A_i.

We say that a point $v = \langle v_1, \ldots, v_d \rangle$ is contained in a unit $u = \{u_1, \ldots, u_d\}$ if $l_i \leq v_i < h_i$ for all u_i. The *selectivity* of a unit is defined to be the fraction of total data points contained in the unit. We call a unit u *dense* if selectivity(u) is greater than τ, where the *density threshold* τ is another input parameter.

We similarly define units in all subspaces of the original d-dimensional space. Consider a projection of the data set V into $A_{t_1} \times A_{t_2} \times \ldots \times A_{t_k}$, where $k < d$ and $t_i < t_j$ if $i < j$. A unit in the subspace is the intersection of an interval from each of the k attributes.

[1]For CLustering In QUEst, the data mining research project at IBM Almaden.

A *cluster* is a maximal set of connected dense units in k-dimensions. Two k-dimensional units u_1, u_2 are *connected* if they have a common face or if there exists another k-dimensional unit u_3 such that u_1 is connected to u_3 and u_2 is connected to u_3. Units $u_1 = \{r_{t_1}, \ldots, r_{t_k}\}$ and $u_2 = \{r'_{t_1}, \ldots, r'_{t_k}\}$ have a common face if there are $k-1$ dimensions, assume dimensions $A_{t_1}, \ldots, A_{t_{k-1}}$, such that $r_{t_j} = r'_{t_j}$ and either $h_{t_k} = l'_{t_k}$ or $h'_{t_k} = l_{t_k}$.

A *region* in k dimensions is an axis-parallel rectangular k-dimensional set. We are only interested in those regions that can be expressed as unions of units; henceforth all references to a region mean such unions. A region can be expressed as a DNF expression on intervals of the domains A_i.

We say that a region R is *contained* in a cluster C if $R \cap C = R$. A region R contained in a cluster C is said to be *maximal* if no proper superset of R is contained in C.

A *minimal description* of a cluster is a non-redundant covering of the cluster with maximal regions. That is, a minimal description of a cluster C is a set $\mathcal{R}$ of maximal regions such that their union equals C but the union of any proper subset of $\mathcal{R}$ does not equal C.

The Problem: Given a set of data points and the input parameters, ξ and τ, find clusters in all subspaces of the original data space and present a minimal description of each cluster in the form of a DNF expression.

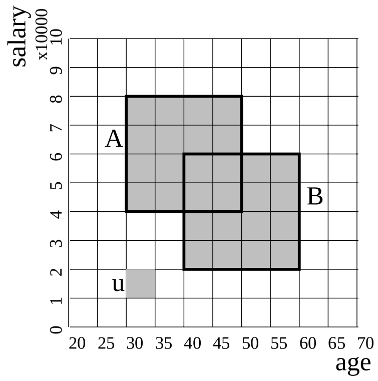

Figure 1: Illustration of definitions.

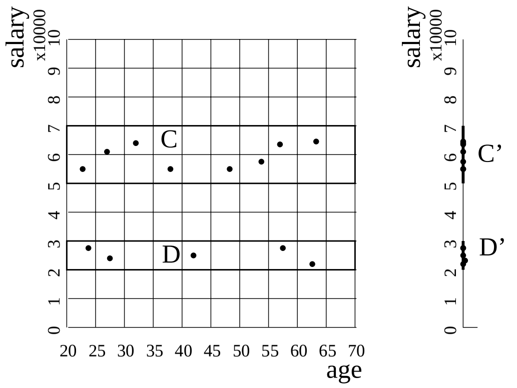

Figure 2: Identification of clusters in subspaces (projections) of the original data space.

Examples: In Figure 1, the two dimensional space (age, salary) has been partitioned by a 10×10 grid. A unit is the intersection of intervals; an example is the unit $u = (30 \leq \text{age} < 35) \wedge (1 \leq \text{salary} < 2)$. A region is a rectangular union of units. A and B are both regions: $A = (30 \leq \text{age} < 50) \wedge (4 \leq \text{salary} < 8)$ and $B = (40 \leq \text{age} < 60) \wedge (2 \leq \text{salary} < 6)$. Assuming that the dense units have been shaded, $A \cup B$ is a cluster. Note that A is a maximal region contained in this cluster, whereas $A \cap B$ is not a maximal region. The minimal description for this cluster is the DNF expression: $((30 \leq \text{age} < 50) \wedge (4 \leq \text{salary} < 8)) \vee ((40 \leq \text{age} < 60) \wedge (2 \leq \text{salary} < 6))$.

In Figure 2, assuming $\tau = 20\%$, no 2-dimensional unit is dense and there are no clusters in the original data space. If the points are projected on the salary dimension however, there are three 1-dimensional dense units. Two of these are connected, so there are two clusters in the 1-dimensional salary subspace: $C' = 5 \leq \text{salary} < 7$ and $D' = 2 \leq \text{salary} < 3$. There is no cluster in the age subspace because there is no dense unit in that subspace.

Remarks: Our model can be generalized to allow different values of ξ for different dimensions. This will require τ to be scaled to account for the difference in relative volumes when checking for the density of units in different subspaces.

Our model can also be adapted to handle categorical data. An arbitrary order is introduced in the categorical domain. The partitioning scheme admits one categorical value in each interval and also places an empty interval between two different values. Consequently, if this dimension is chosen for clustering, the clusters will have the same value in this dimension.

Related Work: A similar approach to clustering high dimensional data has been proposed by Shoshani [38]. The technique computes an approximation of the density function in a *user-specified* subspace using a grid and then uses this function to cluster the data. On the other hand, we automatically discover the interesting subspaces, and also generate minimal descriptions for the clusters.

A different technique to find rectangular clusters of high density in a projection of the data space has been proposed by Friedman [18]. This algorithm works in a top down fashion. Starting from the full space, it greedily chooses which projection should be taken and reevaluates the solution after each step in order to get closer to an optimal solution.

The subspace identification problem is related to the problem of finding quantitative association rules that also identify interesting regions of various attributes [40] [32]. However, the techniques proposed are quite different. One can also imagine adapting a tree-classifier designed for data mining (e.g. [30] [37]) for subspace clustering. In the tree-growth phase, the splitting criterion will have to be changed so that some clustering criterion (e.g. average cluster diameter) is optimized. In the tree-pruning phase, we now minimize the total description length of the clusters obtained and the data description using these clusters.

3 Algorithms

Our clustering technique, CLIQUE, consists of the following steps:

1. Identification of subspaces that contain clusters.
2. Identification of clusters.
3. Generation of minimal description for the clusters.

We discuss algorithms for each of these steps in this section.

3.1 Identification of subspaces that contain clusters

The difficulty in identifying subspaces that contain clusters lies in finding dense units in different subspaces.

3.1.1 A bottom-up algorithm to find dense units

The simplest way to identify dense units would be to create a histogram in all subspaces and count the points contained in each unit. This approach is infeasible for high dimensional data. We use a bottom-up algorithm that exploits the monotonicity of the clustering criterion with respect to dimensionality to prune the search space. This algorithm is similar to the Apriori algorithm for mining Association rules [1]. A somewhat similar bottom-up scheme was also used in [10] for determining modes in high dimensional histograms.

Lemma 1 (Monotonicity): *If a collection of points S is a cluster in a k-dimensional space, then S is also part of a cluster in any $(k-1)$-dimensional projections of this space.*
Proof A k-dimensional cluster C includes the points that fall inside a union of k-dimensional dense units. Since the units are dense, the selectivity of each one is at least τ. All the projections of any unit u in C have at least as large selectivity, because they include all points inside u, and therefore are also dense. Since the units of the cluster are connected, their projections are also connected. It follows that the projections of the points in C lie in the same cluster in any $(k-1)$-dimensional projection. □

Algorithm: The algorithm proceeds level-by-level. It first determines 1-dimensional dense units by making a pass over the data. Having determined (k–1)-dimensional dense units, the candidate k-dimensional units are determined using the candidate generation procedure given below. A pass over the data is made to find those candidate units that are dense. The algorithm terminates when no more candidates are generated.

The candidate generation procedure takes as an argument D_{k-1}, the set of all $(k-1)$-dimensional dense units. It returns a superset of the set of all k-dimensional dense units. Assume that the relation $<$ represents lexicographic ordering on attributes. First, we self-join D_{k-1}, the join condition being that units share the first k–2 dimensions. In the pseudo-code given below for this join operation, $u.a_i$ represents the ith dimension of unit u and $u.[l_i, h_i)$ represents its interval in the ith dimension.

```
insert into $C_k$
select $u_1.[l_1, h_1)$, $u_1.[l_2, h_2)$, $\ldots$,
    $u_1.[l_{k-1}, h_{k-1})$, $u_2.[l_{k-1}, h_{k-1})$
from $D_{k-1}$ $u_1$, $D_{k-1}$ $u_2$
where $u_1.a_1 = u_2.a_1$, $u_1.l_1 = u_2.l_1$, $u_1.h_1 = u_2.h_1$,
    $u_1.a_2 = u_2.a_2$, $u_1.l_2 = u_2.l_2$, $u_1.h_2 = u_2.h_2$,$\ldots$,
    $u_1.a_{k-2} = u_2.a_{k-2}$, $u_1.l_{k-2} = u_2.l_{k-2}$, $u_1.h_{k-2} = u_2.h_{k-2}$,
    $u_1.a_{k-1} < u_2.a_{k-1}$
```

We then discard those dense units from C_k which have a projection in (k–1)-dimensions that is not included in C_{k-1}. The correctness of this procedure follows from the property that for any k-dimensional dense unit, its projections in any of $k-1$ dimensions must also be dense (Lemma 1).

Scalability: The only phase of CLIQUE in which database records are accessed is the dense unit generation. During the generation of C_k, we need storage for dense units D_{k-1} and the candidate units C_k. While making a pass over the data, we need storage for C_k and at least one page to buffer the database records. Thus, the algorithm can work with databases of any size. However, memory needs to be managed carefully as the candidates may swamp the available buffer. This situation is handled by employing a scheme used in [1]. As many candidates of C_k are generated as will fit in the buffer and database is scanned to determine the selectivity of these candidates. Dense units resulting from these candidates are written to disk, while non-dense candidates are deleted. This procedure is repeated until all of C_k has been examined.

Time complexity: If a dense unit exists in k dimensions, then all of its projections in a subset of the k dimensions that is, $O(2^k)$ different combinations, are also dense. The running time of our algorithm is therefore exponential in the highest dimensionality of any dense unit. As in [1] [20], it can be shown that the candidate generation procedure produces the minimal number of candidates that can guarantee that all dense units will be found.

Let k be the highest dimensionality of any dense unit and m the number of the input points. The algorithm makes k passes over the database. It follows that the running time of our algorithm is $O(c^k + m\,k)$ for a constant c.

The number of database passes can be reduced by adapting ideas from [41] [8].

3.1.2 Making the bottom-up algorithm faster

While the procedure just described dramatically reduces the number of units that are tested for being dense, we still may have a computationally infeasible task at hand for high dimensional data. As the dimensionality of the subspaces considered increases, there is an explosion in the number of dense units, and so we need to prune the pool of candidates. The pruned set of dense units is then used to form the candidate units in the next level of the dense unit generation algorithm. The objective is to use only the dense units that lie in "interesting" subspaces.

MDL-based pruning: To decide which subspaces (and the corresponding dense units) are interesting, we apply the MDL (Minimal Description Length) principle. The basic idea underlying the MDL principle is to encode the input data under a given model and select the encoding that minimizes the code length [35].

Assume we have the subspaces $S_1, S_2, \ldots, S_n$. Our pruning technique first groups together the dense units that lie in the same subspace. Then, for each subspace, it computes the fraction of the database that is covered by the dense units in it: $x_{S_j} = \sum_{u_i \in S_j} count(u_i)$ where $count(u_i)$ is the number of points that fall inside u_i. The number x_{S_j} will be referred to as the *coverage* of subspace S_j.

Subspaces with large coverage are selected and the rest are pruned. The rationale is that if a cluster exists in k dimensions, then for every subspace of these k dimensions there exist dense units in this subspace (the projections of the dense units that cover the cluster in the original k dimensions) that cover at least the points in the cluster.

We sort the subspaces in the descending order of their coverage. We want to divide the sorted list of subspaces into two sets: the selected set I and the pruned set P. The following model is used to arrive at the cut point (Figure 3). For each set, we compute the mean of the cover fractions, and for each subspace in that set we compute the difference from the mean. The code length is the sum of the bit lengths of the numbers we have to store. If we decide to prune subspaces $S_{i+1}, \ldots S_n$, the two averages are $\mu_I(i) = \lceil(\sum_{1 \le j \le i} x_{S_j})/i\rceil$ and $\mu_P(i) = \lceil(\sum_{i+1 \le j \le n} x_{S_j})/(n-i)\rceil$. Since both $\mu_I(i)$ and $\mu_P(i)$ are integers, the number of bits required to store them is $\log_2(\mu_I(i))$ and $\log_2(\mu_P(i))$ respectively. For each subspace we have to store its difference from $\mu_I(i)$ or $\mu_P(i)$, which is another integer. The total length of the encoding is:

$$\begin{aligned} CL(i) &= \log_2(\mu_I(i)) + \sum_{1 \le j \le i} \log_2(|x_{S_j} - \mu_I(i)|) + \\ &\quad \log_2(\mu_P(i)) + \sum_{i+1 \le j \le n} \log_2(|x_{S_j} - \mu_P(i)|) \end{aligned}$$

This code length is minimized to determine the optimal cut point i.

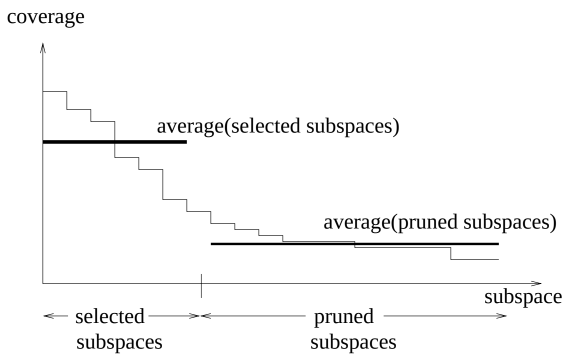

Figure 3: Partitioning of the subspaces into selected and prune sets.

Time complexity: The optimal cut will be one of the $n-1$ positions along the sorted sequence, so there are only $n-1$ sets of pruned subspaces to consider. After sorting, the optimal cut can be computed in two passes of the sorted sequence: In the first pass, we compute $\mu_I(i), \mu_P(i)$ for $1 < i < n$. These averages are used in the second pass to compute $CL(i)$ for $1 < i < n$.

Remark: The pruning of dense units in the subspaces with low coverage makes our algorithm faster, but there is a tradeoff because we may now miss some clusters. If a cluster exists in k dimensions, then all of its projections in a subset of the k dimensions are also clusters. In our bottom-up approach, all of them have to considered if we want to find the cluster in k dimensions, but some of them may be in one of the pruned subspaces.

3.2 Finding clusters

The input to the next step of CLIQUE is a set of dense units D, all in the same k-dimensional space S. The output will be a partition of D into $D^1, \ldots, D^q$, such that all units in D^i are connected and no two units $u^i \in D^i$, $u^j \in D^j$ with $i \neq j$ are connected. Each such partition is a cluster according to our definition.

The problem is equivalent to finding connected components in a graph defined as follows: Graph vertices correspond to dense units, and there is an edge between two vertices if and only if the corresponding dense units have a common face.

Units corresponding to vertices in the same connected component of the graph are connected because there is a path of units that have a common face between them, therefore they are in the same cluster. On the other hand, units corresponding to vertices in different components cannot be connected, and therefore cannot be in the same cluster.

We use a depth-first search algorithm [2] to find the connected components of the graph. We start with some unit u in D, assign it the first cluster number, and find all the units it is connected to. Then, if there still are units in D that have not yet been visited, we find one and repeat the procedure. The algorithm is given below:

```
input:  starting unit $u = \{[l_1, h_1), \ldots, [l_k, h_k)\}$
        clusternumber $n$

dfs($u, n$)
$u.num = n$
for ( $j = 1; j < k; j$++) do begin
    // examine the left neighbor of $u$ in dimension $a_j$
    $u^l = \{[l_1, h_1), \ldots, [(l_j^l), (h_j^l)), \ldots, [l_k, h_k)\}$
    if ($u^l$ is dense) and ($u^l.num$ is undefined)
        dfs($u^l, n$)
    // examine the right neighbor of $u$ in dimension $a_j$
    $u^r = \{[l_1, h_1), \ldots, [(l_j^r), (h_j^r)), \ldots, [l_k, h_k)\}$
    if ($u^r$ is dense) and ($u^r.num$ is undefined)
        dfs($u^r, n$)
end
```

Time complexity: The number of dense units for a given subspace cannot be very large, because each dense unit must have selectivity at least τ. We assume therefore that the dense units in this and subsequent steps of CLIQUE can be stored in memory.

We give asymptotic running times in terms of dense unit accesses; the dense units are stored in a main memory data structure (hash tree [1]) that allows efficient querying.

For each dense unit visited, the algorithm checks its $2k$ neighbors to find connected units. If the total number of dense units in the subspace is n, the total number of data structure accesses is $2kn$.

3.3 Generating minimal cluster descriptions

The input to this step consists of disjoint sets of connected k-dimensional units in the same subspace. Each such set is a cluster and the goal is to generate a concise description for it. To generate a minimal description of each cluster, we would want to cover all the units comprising the cluster with the minimum number of regions such that all regions contain only connected units. For a cluster C in a k-dimensional subspace S, a set $\mathcal{R}$ of regions in the same subspace S is a *cover* of C if every region $R \in \mathcal{R}$ is contained in C, and each unit in C is contained in at least one of the regions in $\mathcal{R}$.

Computing the optimal cover is known to be NP-hard, even in the 2-dimensional case [29] [34]. The optimal cover is the cover with the minimal number of rectangles. The best approximate algorithm known for the special case of finding a cover of a 2-dimensional rectilinear polygon with no holes produces a cover of size bounded by a factor of 2 times the optimal [17]. Since this algorithm only works for the 2-dimensional case, it cannot be used in our setting. For the general set cover problem, the best known algorithm for approximating the smallest set cover gives an approximation factor of $\ln n$ where n is the size of the universe being covered [16] [28].

This problem is similar to the problem of constructive solid geometry formulae in solid-modeling [44]. It is also related to the problem of covering marked boxes in a grid with rectangles in logic minimization (e.g. [22]). Some clustering algorithms in image analysis (e.g. [7] [36] [42]) also find rectangular dense regions. In these domains, datasets are in low dimensional spaces and the techniques used are computationally too expensive for large datasets of high dimensionality.

Our solution to the problem consists of two steps. We first greedily cover the cluster by a number of maximal rectangles (regions), and then discard the redundant rectangles to generate a minimal cover. We only have to consider maximal regions for the cover of a cluster; for any cover R with c regions, we can find another cover R' with c maximal regions, simply by extending each of the non-maximal regions in C.

3.3.1 Covering with maximal regions

The input to this step is a set C of connected dense units in the same k-dimensional space S. The output will be a set $\mathcal{R}$ of maximal regions such that $\mathcal{R}$ is a cover of C. We present a greedy growth algorithm for this task.

Greedy growth: We begin with an arbitrary dense unit $u_1 \in C$ and greedily grow (as described below) a maximal region R_1 that covers u_1. We add R_1 to $\mathcal{R}$. Then we find another unit $u_2 \in C$ that is not yet covered by any of the maximal regions in $\mathcal{R}$. We greedily grow a maximal region R_2 that covers u_2. We repeat this procedure until all units in C are covered by some maximal region in $\mathcal{R}$.

To obtain a maximal region covering a dense unit u, we start with u and grow it along dimension a_1, both to the left and to the right of the unit. We grow u as much as possible in both directions, using connected dense units contained in C. The result is a rectangular region. We now grow this region along dimension a_2, both to the left and to the right of the region. We again use only connected dense units from C, obtaining a possibly bigger rectangular region. This procedure is repeated for all the dimensions, yielding a maximal region covering u. The order in which dimensions are considered for growing a dense unit is randomly determined.

Figure 4 illustrates how the algorithm works. Here the dense units appear shaded. Starting from the dense unit u, first we grow along the horizontal dimension, finding rectangle A consisting of four dense units. Then A is extended in the vertical dimension. When it cannot be extended further, a maximal rectangle is obtained, in this case B. The next step is to find another maximal region starting from a dense unit not covered by B, for example w.

Time complexity: First we show that for each maximal region R, the greedy growth algorithm must perform $O(|R|)$ dense unit accesses, where $|R|$ is the number of dense units contained in R.

Let S be the subspace that R lies in, k the number of dimensions of S, and n the number of dense units in S. The greedy growth algorithm must access each unit that a region R covers to ascertain that R is indeed part of a cluster. In addition, it must access every neighbor unit of R to ascertain

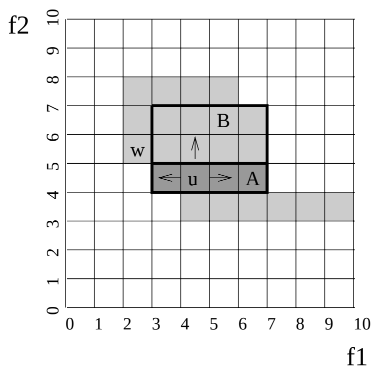

Figure 4: Illustration of the greedy growth algorithm.

that R is also maximal. The number of neighbor units is bounded by $2k|R|$, where $|R|$ is the number of dense units contained in R.

Since every new maximal region covers at least one thus far uncovered dense unit, the greedy growth algorithm will find at most $O(n)$ new regions. Every new region requires $O(|R|) = O(n)$ dense unit accesses, so the greedy growth algorithm performs a total of $O(n^2)$ dense unit accesses.

This bound is almost tight. Let S contain only one cluster (with n dense units), which is bounded by two parallel hyperplanes and a cylinder which is parallel to only one dimension. Since the hyperplanes are not parallel to any of the k dimensions, the boundary of the cluster that touches the hyperplanes consists of $O(n^{(k-1)/k})$ convex vertices, each of which must be covered by a maximal region. The size of each region is also $O(n^{(k-1)/k})$ since each region has to reach the other hyperplane. In this case the greedy growth algorithm must perform $O(n^{2(k-1)/k})$ dense unit accesses. Figure 5 shows the 2-dimensional analog for this case.

Similarly we show that there can be up to $O(n^{2(k-1)/k})$ maximal regions: we can pair every corner on one hyperplane with every corner on the other, and produce a new maximal region for each pair. The greedy growth algorithm will find $O(n)$ of these.

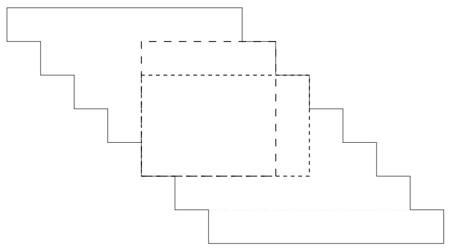

Figure 5: The worst case for the greedy growth algorithm (2-dimensional case): Assume n dense units and $n^{1/2}$ upper corners. A minimal cover must include at least one rectangle per upper corner. Since each rectangle is maximal, it must reach the lower staircase as well. This means that the circumference of the rectangle is $2n^{1/2} + 2$, and therefore its area is at least $n^{1/2}$. The sum of the sizes of the rectangles is then $O(n^{2(2-1)/2})$.

3.3.2 Minimal Cover

The last step of CLIQUE takes as input a cover for each cluster and finds a minimal cover. Minimality is defined in terms of the number of maximal regions (rectangles) required to cover the cluster.

We propose the following greedy heuristic:

Removal heuristic: Remove from the cover the smallest (in number of units) maximal region which is redundant (i.e., every unit is also contained in some other maximal region). Break ties arbitrarily. Repeat the procedure until no maximal region can be removed[2].

Time complexity: The removal heuristic is easy to implement and efficient in execution. It needs a simple scan of the sorted list of regions. The cost of sorting the regions is $O(n \log n)$ because the number of dense units n is an upper bound on the number of regions. The scan requires $|R_i|$ dense unit accesses for each region R_i. The total number of accesses for all regions is then $\sum |R_i| = O(n^2)$.

Stochastic analysis: We now present a stochastic analysis suggesting that the removal heuristic will do well when each unit is independently dense with probability p. This model is quite general: if the number of data points in each unit is a random variable drawn independently from the same (but otherwise arbitrary) distribution as all other units, and we specify any threshold τ from the domain of these random variables, then each unit is independently dense with some probability p depending on the underlying distribution and τ.

In our application, since a dense unit has high selectivity, p is likely to be small. We show now that provided p is small enough, we obtain a good approximation ratio.

Theorem 1: *Let $p = 1/2d - \epsilon$, for any fixed $\epsilon > 0$. If each unit is independently dense with probability at most p, then the expected size of the cover we obtain is within a constant factor of the optimal cover.*

Proof sketch Let c be the number of units in a cluster; our algorithm will use at most c maximal rectangles to cover it. The proof of the following Lemma is omitted here; essentially we must argue that the correlations between cluster-sizes work in our favor.

Lemma 3: *There is a constant $a > 1$ (depending only on ϵ) such that the probability that a cluster has size i is at most a^{-i}.*

To complete the proof, we bound the expected number of maximal rectangles used to cover a given cluster by $\sum_i ia^{-i}$, and the total number of maximal rectangles used to

$$\sum_{\text{clusters}} \sum_i ia^{-i}$$

Let n be the number of clusters found. Since there exists a constant θ, depending only on a, such that $\sum_i ia^{-i} \leq \theta$, we have that

$$\sum_{\text{clusters}} \sum_i ia^{-i} \leq \theta\, n$$

It follows that the expected size of the cluster covers we find is within a constant factor of the total number of clusters, and thus the size of the optimal cover. □

[2]We also considered the following *Addition heuristic*: View the cluster as empty space. Add to the cover the maximal region that will cover the maximum number of yet uncovered units in the cluster. Break ties arbitrarily. Repeat the procedure until the whole cluster is covered.

For general set cover, the addition heuristic is known to give a cover within a factor $\ln n$ of the optimum where n is the number of units to be covered [27]. Thus it would appear that the addition heuristic, since its quality of approximation matches the negative results of [16] [28], would be the obvious choice. However, its implementation in our high dimensional geometric setting is too inefficient. The implementation requires the rather complex computation of the number of uncovered units a candidate maximal region will cover. The residual uncovered regions that arise as the cover is formed can be complicated, and no efficient structures are known for efficiently maintaining the uncovered units.

4 Performance Experiments

We now empirically evaluate CLIQUE using synthetic as well as real datasets. The goals of the experiments are to assess the efficiency and accuracy of CLIQUE:

- **Efficiency:** Determine how the running time scales with:
  - Dimensionality of the data space.
  - Dimensionality of clusters.
  - Size of database.
- **Accuracy:** Test if CLIQUE recovers known clusters in some subspaces of a high dimensional data space.

The experiments were run on a 133-MHz IBM RS/6000 Model 43P workstation. The data resided in the AIX file system and was stored on a 2GB SCSI drive with sequential throughput of about 2 MB/second.

4.1 Synthetic data generation

We use the synthetic data generator from [43] to produce datasets with clusters of high density in specific subspaces. The data generator allows control over the structure and the size of datasets through parameters such as the number of records, the number of attributes, and the range of values for each attribute. The range of values was set to $[0, 100]$ for all attributes.

The clusters are hyper-rectangles in a subset of dimensions such that the average density of data points inside the hyper-rectangle is much larger than the average density in the subspace. The faces of such a cluster are parallel to the axis, therefore another way to describe the cluster is as the intersection of a set of attribute ranges.

The cluster descriptions are provided by the user. A description specifies the subspace of each hyper-rectangle and the range for each attribute in the subspace. The attribute values for a data point assigned to a cluster are generated as follows. For those attributes that define the subspace in which the cluster is embedded, the value is drawn independently at random from the uniform distribution within the range of the hyper-rectangle. For the remaining attributes, the value is drawn independently at random from the uniform distribution over the entire range of the attribute. After distributing the specified number of points equally among the specified clusters, an additional 10% points are added as random noise. Values for all the attributes of these points are drawn independently at random from the uniform distribution over the entire range of the attribute.

4.2 Synthetic data results

We first present scalability and accuracy results observed using synthetic data. The experiments were run with $\xi = 10$. All times are in seconds.

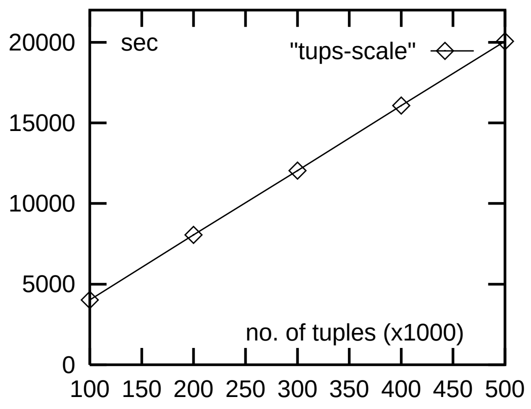

Figure 6: Scalability with the number of data records.

Database size: Figure 6 shows the scalability as the size of the database is increased from 100,000 to 500,000 records. The data space had 50 dimensions and there were 5 clusters, each in a different 5-dimensional subspace, and τ was set to 0.5%. As expected, the running time scales linearly with the size of the database because the number of passes through the database does not change.

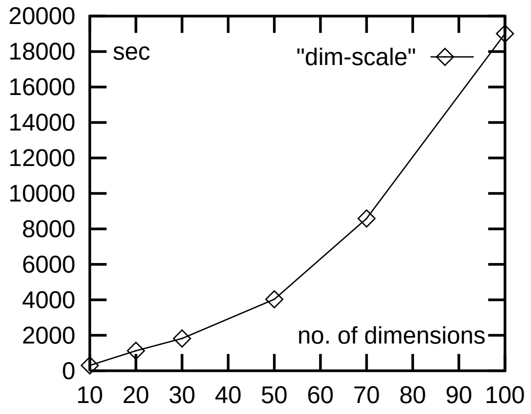

Figure 7: Scalability with the dimensionality of the data space.

Dimensionality of the data space: Figure 7 shows the scalability as the dimensionality of the data space is increased from 10 to 100. The database had 100,000 records and there where 5 clusters, each in a different 5-dimensional subspace, and τ was set to 0.5%. The curve exhibits quadratic behavior. We note that the problem of searching for interesting subspaces inherently does not scale well as the dimensionality of the data space increases. In this case, we are searching for clusters in 5 dimensions. The number of 5-dimensional subspaces of a d-dimensional space is $O(d^5)$. The algorithm performs better than the worst case because many of these dimensions are pruned during the dense unit generation phase.

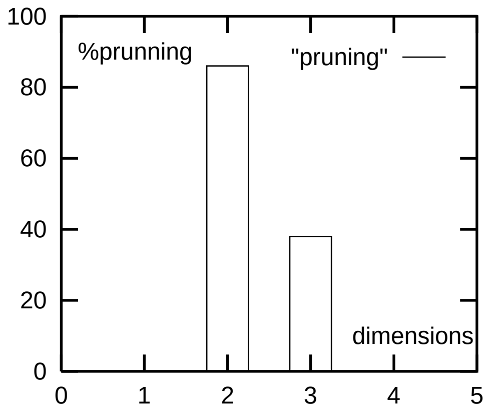

Figure 8: Number of subspaces pruned.

Figure 8 shows the percentage of subspaces pruned by MDL during an algorithm run. The input was a synthetic dataset with 50 dimensions, with 5 hidden 5-dimensional clusters, and τ was set to 0.5%. In this case, 86% of the 2-dimensional subspaces and 38% of the 3-dimensional subspaces were pruned. The result of the pruning is a much faster algorithm, though there is now a risk of missing some clusters.

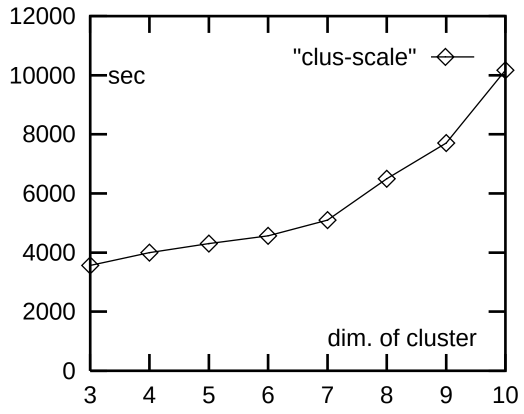

Figure 9: Scalability with the dimensionality of the hidden cluster.

Dimensionality of hidden clusters: Figure 9 shows the scalability as the highest dimensionality of the hidden clusters is increased from 3 to 10 in a 50-dimensional space. In each case, one cluster was embedded in the relevant subspace of highest dimensionality. The database had 100,000 records and τ was set at 1% for 3-dimensional clusters, 0.5% for 4-dimensional to 7-dimensional clusters and and 0.1% for 8-dimensional to 10-dimensional clusters. We selected a lower τ for the highest dimensional clusters because, as the volume of the clusters increases, the cluster density decreases. For lower dimensions however we can increase τ, and since this does not increase the number of dense units the algorithm runs at least as fast. The increase in running time reflects the time complexity of our algorithm, which is $O(mk + c^k)$ where m is the number of records, c a constant, and k the maximum dimensionality of the hidden clusters.

Accuracy: In all the above experiments, the original clusters were recovered by the algorithm. In some cases, a few extra clusters were reported, typically comprising a single dense unit with very low selectivity. This artifact is a byproduct of the data generation algorithm and the fact that τ was set low. As a result, some units had enough noise points to become dense.

4.3 Comparisons with BIRCH, DBSCAN and SVD

We ran CLIQUE, BIRCH, and DBSCAN with the same synthetic datasets[3]. The purpose of these experiments was to assess if algorithms such as BIRCH or DBSCAN designed for clustering in full dimensional space can also be used for subspace clustering. For the task of finding clusters in the full dimensional space, which was the design goal of these algorithms, CLIQUE has no advantage.

We used clusters embedded in 5-dimensional subspaces while varying the dimensionality of the space from 5 to 50. For reference, CLIQUE was able to recover all clusters in every case.

BIRCH: We provided the correct number of clusters (5) as input to the postprocessing clustering algorithm built on top of BIRCH. The output consists of cluster centers in the full dimensional space. The input datasets had 100,000 points. The input clusters were hyper-rectangles in 5-dimensional subspaces, with the values of the remaining attributes uniformly distributed. This is equivalent to a hyper-rectangle in the full data space where remaining attributes include the whole range. Therefore BIRCH successfully recovers a cluster if it reports a center approximately at the center of the equivalent hyper-rectangle in the full data space, and the number of points in the reported cluster is approximately correct.

Table 1: BIRCH experimental results.

Dim. of data	Dim. of clusters	No. of clusters	Clusters found	True clusters identified
5	5	5	5	5
10	5	5	5	5
20	5	5	3,4,5	0
30	5	5	3,4	0
40	5	5	3,4	0
50	5	5	3	0

The results summarized in Table 1 show that BIRCH can discover 5-dimensional clusters embedded in a 10-dimensional data space, but fails to do so when the dimensionality of the data space increases. This is expected because BIRCH uses a distance function that takes all dimensions into account. When the number of dimensions with uniform distribution increases, the distance function fails to distinguish the clusters.

As dimensionality of the data space increases, BIRCH does not always return 5 clusters even though 5 is given as an input parameter. For different randomly generated datasets, it returns 3, 4 or 5 clusters. The final column gives the number of correct embedded clusters that BIRCH identified.

[3]We could not run experiments with CLARANS because the code required modification to work with points in high dimensions. We expect CLARANS to show similar behavior as BIRCH in identifying clusters embedded in subspaces.

DBSCAN: DBSCAN discovers the number of clusters on its own, so we did not have to give the number of clusters as input. DBSCAN could not be run with data having more than 10 dimensions. The input datasets had 10,000 points. As in the BIRCH experiments, the clusters were in 5-dimensional subspaces. We ran DBSCAN with different input values of ϵ; we report the best results in Table 2.

DBSCAN could not discover 5-dimensional clusters in a 10-dimensional data space; it could do so when the dimensionality of the space was reduced to 7. Even in a 8-dimensional data space, it could recover only one of the 5-dimensional embedded clusters. DBSCAN uses a density based cluster definition, and even a small number of dimensions with uniform distribution can lower the density in the space enough so that no clusters are found.

Table 2: DBSCAN experimental results.

Dim. of data	Dim. of clusters	No. of clusters	Clusters found	True clusters identified
5	5	5	5	5
7	5	5	5	5
8	5	5	3	1
10	5	5	1	0

SVD: We also did Singular Value Decomposition (SVD) [11] [19] on the synthetic datasets to find if the dimensionality of the space can be reduced or if the subspaces that contain dense units can be deduced from the projections into the new space.

Table 3: SVD decomposition experimental results.

Dim. of data (d)	Dim. of clusters	No. of clusters	$r_{d/2}$	$r_{(d-5)}$	$r_{(d-1)}$
10	5	5	0.647	0.647	0.937
20	5	5	0.606	0.827	0.969
30	5	5	0.563	0.858	0.972
40	5	5	0.557	0.897	0.981
50	5	5	0.552	0.919	0.984

In Table 3, r_k gives the ratio of the sum of the k largest eigenvalues to the sum of all eigenvalues. Let $\lambda_1, \ldots, \lambda_d$ be the eigenvalues found, sorted in decreasing order. Then $r_k = \sum_{i=1}^{k} \lambda_i / \sum_{i=1}^{d} \lambda_i$. The quantity r_k indicates how much variance is retained in the new space that is defined by the k eigenvectors corresponding to the k largest eigenvalues. In our experiments the variation of the original space is such that the smallest eigenvalue is almost as large as the largest, and so we cannot achieve any dimensionality reduction. In addition, the new projections are linear combinations of all the original vectors and cannot be used to identify the subspaces that contain clusters.

4.4 Real data results

We ran CLIQUE against two datasets obtained from the insurance industry (Insur1, Insur2), another from a department store (Store), and the last from a bank (Bank). Table 4 summarizes the results of this experiment. Each run used a selectivity threshold of 1%, and every dimension was divided into 10 intervals of equal length. We show in the table the dimensionality of the original data space, the highest dimensionality of the subspace in which clusters were found, and the number of such clusters for each of the datasets. In all cases, we discovered meaningful clusters embedded in lower dimensional subspaces.

Table 4: Real data experimental results.

Dataset	Dim. of data	Dim. of clusters	No. of clusters
Insur1	9	7	2
Insur2	7	4	5
Store	24	10	4
Bank	52	9	1

5 Conclusions

We introduced the problem of automatic subspace clustering, motivated by the needs of emerging data mining applications. The solution we propose, CLIQUE, has been designed to find clusters embedded in subspaces of high dimensional data without requiring the user to guess subspaces that might have interesting clusters. CLIQUE generates cluster descriptions in the form of DNF expressions that are minimized for ease of comprehension. It is insensitive to the order of input records and does not presume some canonical data distribution. In designing CLIQUE, we combined developments from several fields including data mining, stochastic complexity, pattern recognition, and computational geometry.

Empirical evaluation shows that CLIQUE scales linearly with the size of input and has good scalability as the number of dimensions in the data or the highest dimension in which clusters are embedded is increased. CLIQUE was able to accurately discover clusters embedded in lower dimensional subspaces, although there were no clusters in the original data space. Having demonstrated the computational feasibility of automatic subspace clustering, we believe it should be considered a basic data mining operation along with other operations such as associations and sequential-patterns discovery, time-series clustering, and classification [23].

Automatic subspace clustering can be useful in other applications besides data mining. To index OLAP data, for instance, the data space is first partitioned into dense and sparse regions [12]. Data in dense regions is stored in an array whereas a tree structure is used to store sparse regions. Currently, users are required to specify dense and sparse dimensions [4]. Similarly, the precomputation techniques for range queries over OLAP data cubes [21] require identification of dense regions in sparse data cubes. CLIQUE can be used for this purpose.

In future work, we plan to address the problem of evaluating the quality of clusterings in different subspaces. One approach is to choose clusters that maximize the ratio of cluster density over expected density for clusterings with the same dimensionality. We also plan to investigate what system support can be provided to the user for selecting the model parameters, τ and ξ. Another area for future work is to try an alternative approach for finding dense units. If the user is only interested in clusters in the subspaces of highest dimensionality, we can use techniques based on recently proposed algorithms for discovering maximal itemsets [5] [26]. These techniques will allow CLIQUE to find dense units of high dimensionality without having to find all of their projections.

Acknowledgment The code for CLIQUE builds on several components that Ramakrishnan Srikant wrote for quantitative association rules. Srikant generously explained his code and modified it in places to meet CLIQUE's requirements. Roberto Bayardo provided helpful feedback to improve the presentation of the paper. Christoph Lingenfelder, Stefanos Manganaris, Hammou Messatfa, and Julio Ortega provided the datasets used in this study. We are also thankful to Prof. H.-P. Kriegel, Raymond Ng, and Raghu Ramakrishnan for providing access to DBSCAN, CLARANS, and BIRCH respectively.

6 Appendix: Dimensionality reduction

The principal component analysis or Karhunen-Loève (KL) transformation is the optimal way to project n-dimensional points to k-dimensional points such that the error of the projections (the sum of the squared distances) is minimal [11] [19]. This transformation gives a new set of orthogonal axes, each a linear combination of the original ones, sorted by the degree by which they preserve the distances of the points in the original space.

For a given set of m points in d dimensions, finding the set of axes in the KL transformation is equivalent to solving the Singular Value Decomposition problem in an $m \times d$ matrix N, each row of which represents a data point. The SVD of the matrix N is the decomposition into $N = U \times \Lambda \times V^t$, where U is an $m \times r$ matrix, Λ a diagonal $r \times r$ matrix and V a column orthonormal $d \times r$ matrix. The matrix V represents the axes of the KL-decomposition (they are also the eigenvectors of the matrix $N \times N^t$), ordered by the respective values in the matrix Λ. Note that $r \leq d$, so the new space has potentially lower dimensionality. In addition, for each small entry in the matrix Λ, the corresponding vectors may be eliminated and a lower dimensionality space obtained.

In our problem, we assume there may not be clearly defined clusters in the original space and try to find those dimensions that can be used for clustering. Clearly, two points may be far apart in a 3-dimensional space but could be quite close when a specific projection into 2 dimensions is used. The effects of such projections are what we are trying to capture. In addition, in the interest of comprehension, we do not want to use dimensions that are linear combinations of the original ones.

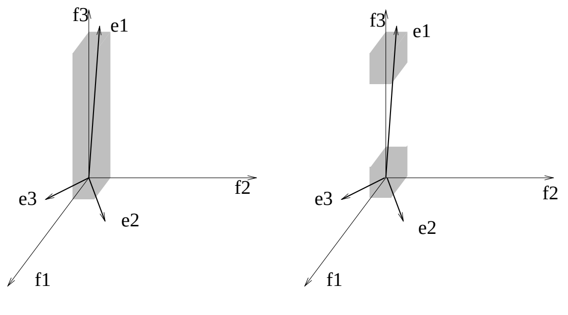

Figure 10: Examples where KL transformation is helpful.

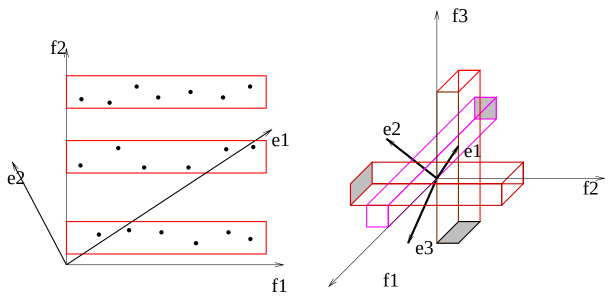

Figure 11: Examples where KL transformation is not helpful.

The following examples illustrate these points. In Figure 10, two data distributions are shown. The original axes are labeled $f1$, $f2$, $f3$. In both cases, the data points are uniformly distributed inside the shaded area. In the left case there is one cluster, in the right two. Assuming the number of points to be the same in both cases, the density of the shaded regions is different for the two sets. The eigenvectors are labeled $e1$, $e2$, $e3$, such that $e1$ corresponds to the eigenvalue with the largest magnitude and $e3$ to the eigenvalue with the smallest magnitude. The first eigenvalue is much larger than the other two, indicating that there is large variation along axis $e1$. The eigenvectors are essentially the same in both cases. Thus, it can be said that the KL transformation is quite successful in these instances. Although the transformation cannot be used by itself to find the actual clusters because it cannot distinguish between the two cases, one can argue that the clusters will be discovered after projecting the points on $e1$ and examining the distribution of the projections.

In Figure 11, the 2-dimensional data is uniformly distributed in dimension $f1$, but contains three clusters along dimension $f2$. Despite the clustering on $f2$, there is large variation along both axes. The results of the KL transformation are the eigenvectors $e1$ and $e2$ as shown. Because of the variation, the eigenvalue corresponding to eigenvector $e2$ (the second largest eigenvalue) is quite large. We have thus come up with a space of the same dimensionality. Furthermore, no projection on the new axes can be used to identify the clusters.

The right figure illustrates that clusters may exist in different subspaces. The data points are uniformly distributed inside the three 3-dimensional rectangles. The rectangles are long, skinny and not very dense. In addition they do not intersect. For reasonable selectivities, the only clusters are the projections of the rectangles on their small faces; that is, one cluster in each of the $f1 \times f2$, $f1 \times f3$ and $f2 \times f3$ subspaces. The KL decomposition does not help here because of large variation along each of the original axes. The resulting axes are $e1$, $e2$, $e3$ and the three eigenvalues are approximately equal. This means there is no 2-dimensional space which approximates the original space. A 3-dimensional space has to be used after the KL transformation for the clustering. But the density of the points in the 3-dimensional space is too low to obtain good clustering.

References

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast Discovery of Association Rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307–328. AAAI/MIT Press, 1996.

[2] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Welsley, 1974.

[3] P. Arabie and L. J. Hubert. An overview of combinatorial data analysis. In P. Arabie, L. Hubert, and G. D. Soete, editors, *Clustering and Classification*, pages 5–63. World Scientific Pub., New Jersey, 1996.

[4] Arbor Software Corporation. *Application Manager User's Guide*, Essbase Version 4.0 edition.

[5] R. Bayardo. Efficiently mining long patterns from databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Seattle, Washington, 1998.

[6] S. Berchtold, C. Bohm, D. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the 16th Symposium on Principles of Database Systems (PODS)*, pages 78–86, 1997.

[7] M. Berger and I. Regoutsos. An algorithm for point clustering and grid generation. *IEEE Transactions on Systems, Man and Cybernetics*, 21(5):1278–86, 1991.

[8] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of the ACM SIGMOD Conference on Management of Data*, May 1997.

[9] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 6, pages 153–180. AAAI/MIT Press, 1996.

[10] R. Chhikara and D. Register. A numerical classification method for partitioning of a large multidimensional mixed data set. *Technometrics*, 21:531–537, 1979.

[11] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, 1973.

[12] R. J. Earle. Method and apparatus for storing and retrieving multi-dimensional data in computer memory. U.S. Patent No. 5359724, October 1994.

[13] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of the 2nd Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[14] M. Ester, H.-P. Kriegel, and X. Xu. A database interface for clustering in large spatial databases. In *Proc. of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining*, Montreal, Canada, August 1995.

[15] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* AAAI/MIT Press, 1996.

[16] U. Feige. A threshold of ln n for approximating set cover. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 314–318, 1996.

[17] D. Franzblau. Performance guarantees on a sweep-line heuristic for covering rectilinear polygons with rectangles. *SIAM J. Disc. Math*, 2:307–321, 3 (1989).

[18] J. Friedman. Optimizing a noisy function of many variables with application to data mining. In *UW/MSR Summer Research Institute in Data Mining*, July 1997.

[19] K. Fukunaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[20] D. Gunopulos, R. Khardon, H. Mannila, and S. Saluja. Data mining, hypergraph transversals, and machine learning. In *Proc. of the 16th ACM Symp. on Principles of Database Systems*, pages 209–216, 1997.

[21] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Tucson, Arizona, May 1997.

[22] S. J. Hong. MINI: A heuristic algorithm for two-level logic minimization. In R. Newton, editor, *Selected Papers on Logic Synthesis for Integrated Circuit Design.* IEEE Press, 1987.

[23] Internationl Business Machines. *IBM Intelligent Miner User's Guide*, Version 1 Release 1, SH12-6213-00 edition, July 1996.

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

[25] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley and Sons, 1990.

[26] D.-I. Lin and Z. M. Kedem. Pincer search: A new algorithm for discovering the maximum frequent sets. In *Proc. of the 6th Int'l Conference on Extending Database Technology (EDBT)*, Valencia, Spain, 1998.

[27] L. Lovász. On the ratio of the optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[28] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 286–293, 1993.

[29] W. Masek. *Some NP-complete set covering problems.* M.S. Thesis, MIT, 1978.

[30] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. of the Fifth Int'l Conference on Extending Database Technology (EDBT)*, Avignon, France, March 1996.

[31] R. S. Michalski and R. E. Stepp. Learning from observation: Conceptual clustering. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume I, pages 331–363. Morgan Kaufmann, 1983.

[32] R. Miller and Y. Yang. Association rules over interval data. In *Proc. ACM SIGMOD International Conf. on Management of Data*, pages 452–461, 1997.

[33] R. T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of the VLDB Conference*, Santiago, Chile, September 1994.

[34] R. A. Reckhow and J. Culberson. Covering simple orthogonal polygon with a minimum number of orthogonally convex polygons. In *Proc. of the ACM 3rd Annual Computational Geometry Conference*, pages 268–277, 1987.

[35] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific Publ. Co., 1989.

[36] P. Schroeter and J. Bigun. Hierarchical image segmentation by multi-dimensional clustering and orientation-adaptive boundary refinement. *Pattern Recognition*, 25(5):695–709, May 1995.

[37] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the 22nd Int'l Conference on Very Large Databases*, Bombay, India, September 1996.

[38] A. Shoshani. Personal communication. 1997.

[39] P. Sneath and R. Sokal. *Numerical Taxonomy.* Freeman, 1973.

[40] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996.

[41] H. Toivonen. Sampling large databases for association rules. In *Proc. of the 22nd Int'l Conference on Very Large Databases*, pages 134–145, Mumbai (Bombay), India, September 1996.

[42] S. Wharton. A generalized histogram clustering for multidimensional image data. *Pattern Recognition*, 16(2):193–199, 1983.

[43] M. Zait and H. Messatfa. A comparative study of clustering methods. *Future Generation Computer Systems*, 13(2-3):149–159, November 1997.

[44] D. Zhang and A. Bowyer. CSG set-theoretic solid modelling and NC machining of blend surfaces. In *Proceedings of the Second Annual ACM Symposium on Computational Geometry*, pages 314–318, 1986.

[45] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996.