Fast Algorithms for Projected Clustering

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@watson.ibm.com

Cecilia Procopiuc
Duke University
Durham, NC 27706
magda@cs.duke.edu

Joel L. Wolf, Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
{ jlw, psyu }@watson.ibm.com

Jong Soo Park
Sungshin Women's University
Seoul, Korea
jpark@cs.sungshin.ac.kr

Abstract

The clustering problem is well known in the database literature for its numerous applications in problems such as customer segmentation, classification and trend analysis. Unfortunately, all known algorithms tend to break down in high dimensional spaces because of the inherent sparsity of the points. In such high dimensional spaces not all dimensions may be relevant to a given cluster. One way of handling this is to pick the closely correlated dimensions and find clusters in the corresponding subspace. Traditional feature selection algorithms attempt to achieve this. The weakness of this approach is that in typical high dimensional data mining applications different sets of points may cluster better for different subsets of dimensions. The number of dimensions in each such cluster-specific subspace may also vary. Hence, it may be impossible to find a single small subset of dimensions for all the clusters. We therefore discuss a generalization of the clustering problem, referred to as the *projected clustering problem*, in which the subsets of dimensions selected are specific to the clusters themselves. We develop an algorithmic framework for solving the projected clustering problem, and test its performance on synthetic data.

1 Introduction

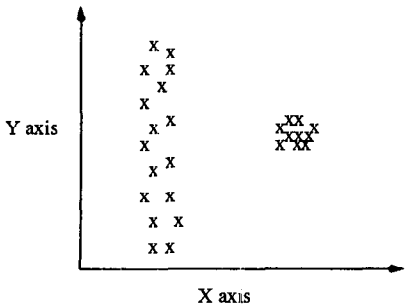
The clustering problem has been discussed extensively in the database literature as a tool for similarity search, customer segmentation, pattern recognition, trend analysis and classification. Various methods have been studied in considerable detail by both the statistics and database communities [3, 4, 7, 8, 9, 13, 21, 26]. Detailed surveys on clustering methods can be found in [6, 17, 18, 20, 25].

The problem of clustering data points is defined as follows: Given a set of points in multidimensional space, find a partition of the points into *clusters* so that the points within each cluster are close to one another. (There may also be a group of outlier points.) Some algorithms assume that the number of clusters is prespecified as a user parameter. Various objective functions may be used in order to make a quantitative determination as to how well the points are clustered. Alternatively, distribution based methods [15, 24] may be used in order to find clusters of arbitrary shape.

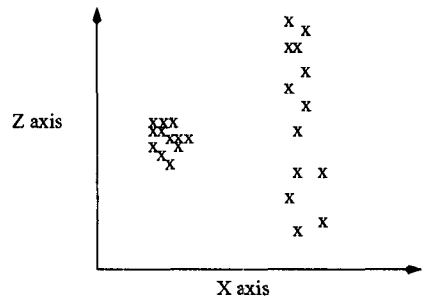
Most clustering algorithms do not work efficiently in higher dimensional spaces because of the inherent sparsity of the data [1, 22]. In high dimensional applications, it is likely that for any given pair of points there exist at least a few dimensions on which the points are far apart from one another. So a clustering algorithm is often preceded by feature selection (see for example [19]). The goal is to find the particular dimensions on which the points in the data are correlated. Pruning away the remaining dimensions reduces the noise in the data. The problem of using traditional feature selection algorithms is that picking certain dimensions in advance can lead to a loss of information. Furthermore, in many real data examples, some points are correlated with respect to a given set of dimensions and others are correlated with respect to different dimensions. Thus it may not always be feasible to prune off too many dimensions without at the same time incurring a substantial loss of information. We demonstrate this with the help of an example.

In Figure 1 we have illustrated two different projected cross sections for a set of points in 3-dimensional space. There are two patterns in the data. The first pattern corresponds to a set of points that are close to one another in the x-y plane, while the second pattern corresponds to a set of points that are close to one another in the x-z plane. We would like to have some way of discovering such patterns. Note that traditional feature selection does not work in this case, as each dimension is relevant to at least one of the clusters. At the same time, clustering in the full dimensional space will not discover the two patterns, since each of them is spread out along one of the dimensions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SIGMOD '99 Philadelphia PA
Copyright ACM 1999 1-58113-084-8/99/05...$5.00



(a) Cross Section on X-Y axis



(b) Cross Section on X-Z axis

Figure 1: Difficulties Associated with Feature Preselection

In this context we now define what we call a projected cluster. Consider a set of data points in some multidimensional space. A *projected cluster* is a subset $\mathcal{C}$ of data points together with a subset $\mathcal{D}$ of dimensions such that the points in $\mathcal{C}$ are closely clustered in the subspace of dimensions $\mathcal{D}$. In Figure 1, two clusters exist in two different projected subpaces. Cluster 1 exists in projected x-y space, while cluster 2 exists in projected x-z space.

In this paper we focus on a method to find clusters in small projected subspaces for data of high dimensionality. We call our algorithm PROCLUS to denote the fact that it is a PROjected CLUStering algorithm. We assume that the number k of clusters to be found is an input parameter. The output of the algorithm will be twofold:

- a $(k+1)$-way partition $\{\mathcal{C}_1, ..., \mathcal{C}_k, \mathcal{O}\}$ of the data, so that the points in each partition element except the last form a cluster. (The points in the last partition element are the *outliers*, which by definition do not cluster well.)
- a possibly different subset $\mathcal{D}_i$ of dimensions for each cluster $\mathcal{C}_i$, $1 \leq i \leq k$, so that the points in $\mathcal{C}_i$ are correlated with respect to these dimensions. (The dimensions for the outlier set $\mathcal{O}$ can be assumed to be the empty set.) For different clusters, the cardinality of the corresponding set $\mathcal{D}_i$ can be different.

In addition to the number of clusters k the algorithm takes as input the average number of dimensions l in a cluster. The two parameters can be varied independently of one another. (The only restriction is that the total number of dimensions $k \cdot l$ must be integral.)

1.1 Contributions of this paper

The contributions of this paper are as follows:

(1) We discuss the concept of projected clustering for finding clusters in multidimensional spaces. Thus, we compute clusters based not only on points but also on dimensions. For data in a large number of dimensions this can result in a significant improvement in the quality of the clustering.

(2) We propose an algorithm for the projected clustering problem which uses the so-called *medoid* technique described in [21] to find the appropriate sets of clusters and dimensions. The algorithm uses a locality analysis in order to find the set of dimensions associated with each medoid.

The fact that different points may cluster better for different subsets of dimensions has been observed for the first time by Agrawal et. al. in [1]. This paper presents an effective method for finding regions of greater density in high dimensional data in a way which has good scalability and usability. The work in [1] illustrates the merits of looking at different subspaces for different clusters as opposed to a full dimensional clustering. The algorithm, called CLIQUE, works from lower to higher dimensionality subspaces and discovers "dense" regions in each subspace. More precisely, each dimension is divided into a number of intervals ξ. For a given set of dimensions, a cross-product of such intervals (one on each dimension in the set) is called a *unit* in the respective subspace. Units are *dense* if the number of points they contain is above a certain threshold τ. Both ξ and τ are user parameters. The algorithm discovers all dense units in each k-dimensional subspace by building from the dense units in $(k-1)$-dimensional subspaces, and then "connects" these axis-parallel units to form the reported rectangular regions. Although such an approach can discover interesting characteristics of the data, it does not produce a clustering in the accepted definition of the word, since the points are not partitioned into disjoint groups. Rather, there is a large overlap among the reported dense regions, due to the fact that for a given dense region all its projections on lower dimensionality subspaces are also dense and get reported.

While both CLIQUE and PROCLUS aim to discover interesting correlations among data in various subspaces of the original high dimensional space, their output is significantly different. CLIQUE is successful in exploring dense regions in all subspaces of some desired dimensionality. For many applications in customer segmentation and trend analysis, a partition of the points is required. Furthermore, partitions provide clearer interpretability of the results, as compared to reporting dense regions with very high overlap. In such cases, PROCLUS is preferable to CLIQUE.

The remainder of this paper is organized as follows. Section 2 describes our clustering algorithm in detail. In Section 3 we provide a theoretical analysis of the robustness of PROCLUS. Empirical results based on synthetic data are presented in Section 4. Section 5 contains conclusions and areas of future work.

1.2 Definitions and Notations

In order to describe our algorithm we introduce a few notations and definitions. Let N denote the total number of data points, and d denote the dimensionality of the data space. Let $\mathcal{C} = \{x_1, x_2, \ldots, x_t\}$ be the set of points in a cluster. The *centroid* of a cluster is the algebraic average of all the points in the cluster. Thus, the centroid of the cluster $\mathcal{C}$ is given by $\overline{x}_{\mathcal{C}} = \sum_{i=1}^{t} x_i/t$. Given a specific distance function $d(\cdot,\cdot)$, we define the *radius* of a cluster to be the average distance of a point from the centroid of the cluster: $r_{\mathcal{C}} = \sum_{i=1}^{t} d(\overline{x}_{\mathcal{C}}, x_i)/t$.

Various distance functions have been used in full dimensional clustering algorithms, depending on the particular problem being solved. Two such well known functions are the *Manhattan* distance and the *euclidean* distance. The Manhattan distance between two points $x_1 = (x_{1,1}, \ldots, x_{1,d})$ and $x_2 = (x_{2,1}, \ldots, x_{2,d})$ is given by $d_1'(x_1, x_2) = \sum_{i=1}^{d} |x_{1,i} - x_{2,i}|$, and the euclidean distance is given by $d_2'(x_1, x_2) = \sqrt{\sum_{i=1}^{d} (x_{1,i} - x_{2,i})^2}$. Both distance functions are derived from *norms*. In general, the distance corresponding to the so-called L_p norm is given by $d_p'(x_1, x_2) = (\sum_{i=1}^{d} |x_{1,i} - x_{2,i}|^p)^{1/p}$. Thus the Manhattan distance corresponds to the L_1 norm and the euclidean distance to the L_2 norm.

In this paper we will use a variant of the Manhattan distance, called *Manhattan segmental distance*, that is defined relative to some set of dimensions $\mathcal{D}$. Specifically, for any two points $x_1 = (x_{1,1}, \ldots, x_{1,d})$ and $x_2 = (x_{2,1}, \ldots, x_{2,d})$, and for any set of dimensions $\mathcal{D}$, $|\mathcal{D}| \leq d$, the Manhattan segmental distance between x_1 and x_2 relative to $\mathcal{D}$ is given by $d_{\mathcal{D}}(x_1, x_2) = (\sum_{i \in \mathcal{D}} |x_{1,i} - x_{2,i}|)/|\mathcal{D}|$. Employing the Manhattan segmental distance as opposed to the traditional Manhattan distance is useful when comparing points in two different clusters that have varying number of dimensions, because the number of dimensions has been normalized away. There is no comparably easy normalized variant for the euclidean metric. For many applications, the Manhattan segmental distance often has physical significance. One potential application is collaborative filtering [10], where customers need to be partitioned into groups with similar interests for target marketing. Here one needs to be able to handle a large number of dimensions (for different products or product categories) with an objective function representing the average difference of preferences on the different objects.

2 The Clustering Algorithm

The problem of finding projected clusters is two-fold: we must locate the cluster centers and find the appropriate set of dimensions in which each cluster exists. In the full dimensionality setting, the problem of finding cluster centers has been extensively investigated, both in the database and in the computational geometry communities. A well known general approach is the so-called K-Medoids method (see, for example, [18] for a detailed discussion), which uses points in the original data set to serve as surrogate centers for clusters during their creation. Such points are referred to as *medoids*. One method which uses the K-Medoids approach, called CLARANS, was proposed by Ng and Han [21] for clustering in full dimensional space. We combine the greedy method of [14] with the local search approach of the CLARANS algorithm [21] to generate possible sets of medoids, and use some original ideas in order to find the appropriate dimensions for the associated clusters. The overall pseudocode for the algorithm is given in Figure 2.

The algorithm proceeds in three phases: an initialization phase, an iterative phase, and a cluster refinement phase. The general approach is to find the best set of medoids by a hill climbing process similar to the one used in CLARANS, but generalized to deal with projected clustering. By "hill climbing" we refer to the process of successively improving a set of medoids, which serve as the anchor points for the different clusters. The purpose of the initialization phase is to reduce the set of points on which we do the hill climbing, while at the same time trying to select representative points from each cluster in this set. The second phase represents the hill climbing process that we use in order to find a good set of medoids. We also compute a set of dimensions corresponding to each medoid so that the points assigned to the medoid best form a cluster in the subspace determined by those dimensions. The assignment of points to medoids is based on Manhattan segmental distances relative to these sets of dimensions. Thus, we search not just in the space of possible medoids but also in the space of possible dimensions associated with each medoid. Finally, we do a cluster refinement phase in which we use one pass over the data in order to improve the quality of the clustering.

We detail each phase in the following.

2.1 Initialization Phase

We call a set of k medoids *piercing* if each point is drawn from a different cluster. Clearly, finding such a set together with appropriate sets of dimensions is the key objective of our algorithm. The initialization phase is geared towards finding a small enough superset of a piercing set, so that it is possible to efficiently perform hill-climbing on it, as opposed to the entire database of points.

In full dimensional algorithms, one of the techniques for finding a piercing set of medoids is based on a greedy method. In this process medoids are picked iteratively, so that the current medoid is well separated from the medoids that have been chosen so far. The greedy technique has been proposed in [14] and is illustrated in Figure 3. In full dimensionality, if there are no outliers and if the clusters are well enough separated, this method always returns a piercing set of medoids. However, it does not guarantee a piercing set for the projected clustering problem.

In our algorithm we will use the greedy technique in order to find a good *superset* of a piercing set of medoids. In other words, if we wish to find k clusters in the data, we will pick a set of points of cardinality a few times larger than k. We will perform two successive steps of subset selection in order to choose our superset. In the first step, we choose a random sample of data points of size proportional to the number of clusters we wish to generate. (In Figure 2, we denote this size by $A \cdot k$, where A is a constant.) We shall denote this sample by $\mathcal{S}$. In the second step, we apply the above-mentioned greedy technique to $\mathcal{S}$ in order to obtain an even smaller final set of points on which to apply the hill climbing technique during the next phase. In our algorithm, the final set of points has size $B \cdot k$, where B is a small constant. We shall denote this set by $\mathcal{M}$. The reasons for choosing this two-step method are as follows:

(1) The greedy technique tends to pick many outliers due to its distance based approach. On the other hand, the set $\mathcal{S}$ probably contains only a very small number of outliers, and the greedy algorithm is likely to pick some representatives from each cluster.

(2) The reduction to the sample set $\mathcal{S}$ significantly reduces the running time of the initialization phase.

2.2 Iterative Phase

We start by choosing a random set of k medoids from $\mathcal{M}$ and progressively improve the quality of medoids by iteratively replacing the bad medoids in the current set with new points from $\mathcal{M}$. The hill climbing technique can be viewed as a restricted search on the complete

Algorithm PROCLUS(*No. of Clusters: k, Avg. Dimensions: l*)
{ $\mathcal{C}_i$ is the ith cluster }
{ $\mathcal{D}_i$ is the set of dimensions associated with cluster $\mathcal{C}_i$ }
{ $M_{current}$ is the set of medoids in current iteration }
{ M_{best} is the best set of medoids found so far }
{ $\mathcal{N}$ is the final set of medoids with associated dimensions }
{ A, B are constant integers }
begin

{*1. Initialization Phase*}
$\mathcal{S}$ = random sample of size $A \cdot k$
$\mathcal{M}$ = GREEDY($\mathcal{S}$, $B \cdot k$)

{*2. Iterative Phase*}
$BestObjective = \infty$
$M_{current}$ = Random set of medoids $\{m_1, m_2, \ldots m_k\} \subset \mathcal{M}$
repeat
{ Approximate the optimal set of dimensions }
for each medoid $m_i \in M_{current}$ **do**
begin
Let δ_i be distance to nearest medoid from m_i
$\mathcal{L}_i$ = Points in sphere centered at m_i with radius δ_i
end;
$\mathcal{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_k\}$
$(\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k)$ = FindDimensions(k,l,$\mathcal{L}$)
{ Form the clusters }
$(\mathcal{C}_1, \ldots, \mathcal{C}_k)$ = AssignPoints($\mathcal{D}_1, \ldots \mathcal{D}_k$)
$ObjectiveFunction = EvaluateClusters(\mathcal{C}_1, \ldots \mathcal{C}_k, \mathcal{D}_1 \ldots \mathcal{D}_k)$
if $ObjectiveFunction < BestObjective$ **then**
begin
$BestObjective = ObjectiveFunction$
$M_{best} = M_{current}$
Compute the bad medoids in M_{best}
end
Compute $M_{current}$ by replacing the bad medoids in
M_{best} with random points from $\mathcal{M}$
until (*termination_criterion*)

{*3. Refinement Phase*}
$\mathcal{L} = \{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$
$(\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k)$ = FindDimensions(k,l,$\mathcal{L}$)
$(\mathcal{C}_1, \ldots, \mathcal{C}_k)$ = AssignPoints($\mathcal{D}_1, \ldots \mathcal{D}_k$)
$\mathcal{N} = (M_{best}, \mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k)$
return($\mathcal{N}$)
end

Figure 2: The Clustering Algorithm

```
Algorithm Greedy(Set of points: S, Number of medoids: k)
{ d(·,·) is the distance function }
begin
M = {m_1}   { m_1 is a random point of S }
{ compute distance between each point and medoid m_1 }
for each x ∈ S \ M
    dist(x) = d(x, m_1)
for i = 2 to k
begin
    { choose medoid m_i to be far from previous medoids }
    let m_i ∈ S \ M be s.t. dist(m_i) = max{dist(x) | x ∈ S \ M}
    M = M ∪ {m_i}
    { compute distance of each point to closest medoid }
    for each x ∈ S \ M
        dist(x) = min{dist(x), d(x, m_i)}
end
return M
end
```

Figure 3: The Greedy Algorithm

graph with vertex set defined by all sets of medoids of cardinality k. The current node in the graph, denoted M_{best} in Figure 2, represents the best set of medoids found so far. The algorithm tries to advance to another node in the graph as follows: It first determines the bad medoids in M_{best} using a criterion we discuss at the end of this subsection. Then it replaces the bad medoids with random points from $\mathcal{M}$ to obtain another vertex in the graph, denoted $M_{current}$. If the clustering determined by $M_{current}$ is better than the one determined by M_{best}, the algorithm sets $M_{best} = M_{current}$. Otherwise, it sets $M_{current}$ to another vertex of the graph, obtained by again replacing the bad medoids in M_{best} with random points of $\mathcal{M}$. If M_{best} does not change after a certain number of vertices have been tried, the hill climbing phase terminates and M_{best} is reported.

In what follows, we detail how we evaluate the clustering determined by a given set of k medoids. This implies solving two problems: finding the appropriate set of dimensions for each medoid in the set, and forming the cluster corresponding to each medoid.

Finding Dimensions: Given a set of k medoids $M = \{m_1, ..., m_k\}$, PROCLUS evaluates the locality of the space near them in order to find the dimensions that matter most. More exactly, for each medoid m_i let δ_i be the minimum distance from any *other* medoid to m_i, i.e. $\delta_i = min_{j \neq i} d(m_i, m_j)$. For each i, we define the *locality* $\mathcal{L}_i$ to be the set of points that are within distance δ_i from m_i. (Note that the sets $\mathcal{L}_1, ..., \mathcal{L}_k$ need not necessarily be disjoint, nor cover the entire set of data points). We then compute the average distance along each dimension from the points in $\mathcal{L}_i$ to m_i. Let $X_{i,j}$ denote this average distance along dimension j. To each medoid m_i we wish to associate those dimensions j for which the values $X_{i,j}$ are as small as possible relative to statistical expectation, subject to the restriction that the total number of dimensions associated to medoids must be equal to $k \cdot l$. We add the additional constraint that the number of dimensions associated with a medoid must be at least 2. Corresponding to each medoid i we compute the mean $Y_i = (\sum_{j=1}^{d} X_{i,j})/d$ and the standard deviation $\sigma_i = \sqrt{\frac{\sum_j (X_{i,j} - Y_i)^2}{d-1}}$ of the values $X_{i,j}$. Note that Y_i represents in fact the average of the Manhattan segmental distances between the points in $\mathcal{L}_i$ and m_i relative to the entire space. Thus the value $Z_{i,j} = \frac{X_{i,j} - Y_i}{\sigma_i}$ indicates how the j-dimensional average distance associated with the medoid m_i is related to the average Manhattan segmental distance associated with the same medoid. A negative value of $Z_{i,j}$ indicates that along dimension j the points in $\mathcal{L}_i$ are more closely correlated to the medoid m_i. We want to pick the smallest values $Z_{i,j}$ so that a total of $k \cdot l$ values are chosen, and at least 2 values are chosen for any fixed i. This problem is equivalent to a so-called *separable convex resource allocation problem*, and can be solved exactly by a greedy algorithm [16]. Specifically, we sort all the $Z_{i,j}$ values in increasing order, preallocate the 2 smallest for each i (giving a total of $2k$ values), and then greedily pick the remaining lowest $k \cdot (l-2)$ values. (There are other algorithms for solving this problem exactly that are even faster from a complexity point of view. We employ a greedy algorithm here because it is sufficiently fast in light of the typically expected values of k and l. However, see [16] for further details.) With each medoid i we associate those dimensions j whose corresponding $Z_{i,j}$ value was chosen by the above algorithm. We denote the sets of dimensions thus found by $\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k$. This is illustrated in Figure 4.

Forming Clusters: Given the medoids and their associated sets of dimensions, we assign the points to the medoids using a single pass over the database. For each i, we compute the Manhattan segmental distance relative to $\mathcal{D}_i$ between the point and the medoid m_i, and assign the point to the closest medoid. See Figure 5.

We evaluate the quality of a set of medoids as the average Manhattan segmental distance from the points to the centroids of the clusters to which they belong (see Figure 6). Note that the centroid of a cluster will typically differ from the medoid. We also determine the *bad* medoids as follows: The medoid of the cluster with the least number of points is bad. In addition, the medoid of any cluster with less than $(N/k) \cdot minDeviation$ points is bad, where $minDeviation$ is a constant smaller than 1 (in most experiments, we choose $minDeviation = 0.1$).

We make the assumption that if a medoid forms a cluster with less than $(N/k) \cdot minDeviation$ points (where $minDeviation$ is usually 0.1), it is *likely* that the medoid is either an outlier, or it belongs to a cluster

that is pierced by at least one other medoid in the set. Conversely, we assume that an outlier is *likely* to form a cluster with very few points. We also assume that if the current set of medoids contains two or more medoids from the same natural cluster, one of these medoids (which is the most "central") is *likely* to form a cluster containing most of the points in the natural cluster, while the remaining medoids that pierce that cluster will form clusters with few points.

2.3 Refinement Phase

After the best set of medoids is found, we do one more pass over the data to improve the quality of the clustering. Let $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ be the clusters corresponding to these medoids, formed during the Iterative Phase. We discard the dimensions associated with each medoid and compute new ones by a procedure similar to that in the previous subsection. The only difference is that in order to analyze the dimensions associated with each medoid, we use the distribution of the points in the clusters at the end of the iterative phase, as opposed to the localities of the medoids. In other words, we use $\mathcal{C}_i$ instead of $\mathcal{L}_i$. Once the new dimensions have been computed, we reassign the points to the medoids relative to these new sets of dimensions. The process is illustrated in Figure 2.

Outliers are also handled during this last pass over the data. For each medoid m_i and new set of dimensions $\mathcal{D}_i$, we find the smallest Manhattan segmental distance Δ_i to any of the other $(k-1)$ medoids with respect to the set of dimensions $\mathcal{D}_i$:

$\Delta_i = \min_{j \neq i} d_{\mathcal{D}_i}(m_i, m_j)$

We also refer to Δ_i as the sphere of influence of the medoid m_i. A point is an outlier if its segmental distance to each medoid m_i, relative to the set of dimensions $\mathcal{D}_i$, exceeds Δ_i.

3 Analyzing the Robustness of PROCLUS

To ensure good accuracy of the output, PROCLUS must be able to achieve two essential results: find a piercing set of medoids, and associate the correct set of dimensions to each medoid. In our discussion of the *Initialization Phase*, we gave some insight into why we expect the set $\mathcal{M}$ to contain a piercing set of medoids. In the following, we will discuss some issues related to the robustness of the procedure for finding dimensions. It is important to note that since the locality of a medoid is used in order to determine the set of dimensions corresponding to it, a sufficient number of points must exists in the locality in order to have a robust algorithm. The total number of points in the localities of all the medoids is also useful in order to estimate the number of dimensions for a given cluster. To give some insight into how the localities of medoids

Algorithm FindDimensions($k, l, \mathcal{L}$)
begin
{ d is the total number of dimensions }
{ $X_{i,j}$ is the average distance from the points in $\mathcal{L}_i$ to medoid m_i, along dimension j}
for each medoid i **do**
  begin
  $Y_i = \frac{\sum_{j=1}^{d} X_{i,j}}{d}$
  $\mathcal{D}_i = \emptyset$
  $\sigma_i = \sqrt{\frac{\sum_{j=1}^{d} (X_{i,j} - Y_i)^2}{d-1}}$
  for each dimension j **do** $Z_{i,j} = (X_{i,j} - Y_i)/\sigma_i$
  end
Pick the $k \cdot l$ numbers with the least (most negative) values of $Z_{i,j}$ subject to the constraint that there are at least 2 dimensions for each cluster
if $Z_{i,j}$ is picked **then** add dimension j to $\mathcal{D}_i$
return($\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k$)
end

Figure 4: Finding the Dimensions

Algorithm AssignPoints($\mathcal{D}_1, \mathcal{D}_2, \ldots \mathcal{D}_k$)
begin
for each $i \in \{1, \ldots, k\}$ **do** $\mathcal{C}_i = \phi$
for each data point p **do**
  begin
  Let $d_{\mathcal{D}_i}(p, m_i)$ be Manhattan segmental distance of point p from medoid m_i relative to dimensions $\mathcal{D}_i$;
  Find i with lowest value of $d_{\mathcal{D}_i}(p, m_i)$ and add p to $\mathcal{C}_i$;
  end;
return ($\mathcal{C}_1, \ldots, \mathcal{C}_k$)
end;

Figure 5: Assigning Points to the Various Clusters

Algorithm EvaluateClusters($\mathcal{C}_1, \ldots, \mathcal{C}_k, \mathcal{D}_1, \ldots \mathcal{D}_k$)
begin
for each $\mathcal{C}_i$ **do**
  begin
  for each dimension $j \in \mathcal{D}_i$ **do**
    begin
    $Y_{i,j}$ = Average distance of points in $\mathcal{C}_i$ to *centroid* of $\mathcal{C}_i$ along dimension j
    end
  $w_i = \frac{\sum_j Y_{i,j}}{|\mathcal{D}_i|}$
  end
return($\frac{\sum_{i=1}^{k} |\mathcal{C}_i| \cdot w_i}{N}$)
end

Figure 6: Evaluating the Clusters

look like, suppose first that the medoids are chosen randomly from the entire set of points, rather than by the more elaborate procedure in PROCLUS. We prove the following.

Theorem 3.1 *Let k be the number of medoids and N be the total number of data points. Then, for a random set of k medoids $\{m_1, \ldots, m_k\}$, the expected number of points in $\mathcal{L}_i$ for the medoid m_i is N/k.*

Proof: Let $d_1^i, d_2^i, \ldots d_N^i$ denote the distances of the N points from medoid m_i. The problem is equivalent to the following standard result in order statistics (see [2] for details):

Given a set of N values $\{d_1^i, d_2^i, \ldots, d_N^i\}$, suppose we choose $k-1$ of them randomly. Then, the expected number of values in the set that are smaller than the $k-1$ chosen values is equal to N/k.

The $k-1$ randomly chosen values correspond to the distances from the $k-1$ other medoids to medoid m_i. ∎

The above result shows that, if the medoids were chosen at random from the entire data set, the expected number of points in each locality would be sufficient to ensure the robustness of the $FindDimensions(\cdot)$ procedure. Since our method for choosing the medoids is not random, but rather biased towards ensuring that the medoids are as far away from each other as possible (i.e. their localities have large radii), we expect the localities of the medoids to contain at least N/k points each.

4 Empirical Results

The simulations were performed on a 233-MHz IBM RS/6000 computer with 128M of memory, running AIX 4.1.4. The data was stored on a 2GB SCSI drive. We report results obtained for synthetic data. We evaluate the accuracy of PROCLUS on synthetic data and determine how the running time scales with:

- size of database.
- dimensionality of the data space.
- average dimensionality of clusters.

We also investigate the cases in which CLIQUE can be used to return a partition of the data set. For those cases, we compare its accuracy and running time to those of PROCLUS.

4.1 Synthetic Data Generation

In order to generate the data we used a method similar to that discussed by Zhang et. al. [26]. However, we added generalizations to the data generation process in order to take into account the possibility of different clusters occurring in different subspaces. The points have coordinates in the range $[0, 100]$ and are either cluster points or outliers. The maximum percentage of outliers is a simulation parameter and was chosen to be $F_{outlier} = 5\%$. Outliers were distributed uniformly at random throughout the entire space.

In order to generate cluster points the program takes as input parameters the number of clusters k and a Poisson parameter μ that determines the number of dimensions in each cluster, as we explain below. The algorithm proceeds by defining so-called *anchor points* around which the clusters will be distributed, as well as the dimensions associated with each such anchor point. Then, it determines how many points will be associated with each cluster and finally it generates the cluster points. We explain these steps in more detail below.

The anchor points of clusters are obtained by generating k uniformly distributed points in the d-dimensional space. We shall denote the anchor point for the ith cluster by c_i.

The number of dimensions associated with a cluster is given by the realization of a Poisson random variable with mean μ, with the additional restriction that this number must be at least 2 and at most d. Once the number of dimensions d_i for the cluster i is generated, the dimensions for each cluster are chosen using the following technique: The dimensions in the first cluster are chosen randomly. The dimensions for the ith cluster are then generated inductively by choosing $\min\{d_{i-1}, d_i/2\}$ dimensions from the $(i-1)$st cluster and generating the other dimensions randomly. This iterative technique is intended to model the fact that different clusters frequently share subsets of correlated dimensions.

To decide the number of points in each cluster, we generate k exponential random variables with mean 1 and then assign to each cluster a number of points proportional to these realizations. More exactly, let $r_1, r_2, \ldots r_k$ be the realizations of the k random variables, and let $N_c = N \cdot (1 - F_{outlier})$ be the number of cluster points. Then, the number of points in cluster i is given by $N_c \cdot \frac{r_i}{\sum_{i=1}^{k} r_i}$.

Finally, the points for a given cluster i are generated as follows: The coordinates of the points on the non-cluster dimensions are generated uniformly at random. For a cluster dimension j, the coordinates of the points projected onto dimension j follow a normal distribution with mean at the respective coordinate of the anchor point, and variance determined randomly in the following manner: Fix a *spread parameter* r and choose a scale factor $s_{ij} \in [1, s]$ uniformly at random, where s is user defined. Then the variance of the normal distribution on dimension j is $(s_{ij} \cdot r)^2$. For our data generation we chose $r = s = 2$.

Input	Dimensions	Points
A	3, 4, 7, 9, 14, 16, 17	21391
B	3, 4, 7, 12, 13, 14, 17	23278
C	4, 6, 11, 13, 14, 17, 19	18245
D	4, 7, 9, 13, 14, 16, 17	15728
E	3, 4, 9, 12, 14, 16, 17	16357
Outliers	-	5000

Found	Dimensions	Points
1	4, 6, 11, 13, 14, 17, 19	18701
2	3, 4, 7, 9, 14, 16, 17	21915
3	3, 4, 7, 12, 13, 14, 17	23975
4	4, 7, 9, 13, 14, 16, 17	16018
5	3, 4, 9, 12, 14, 16, 17	16995
Outliers	-	2396

Table 1: PROCLUS: Dimensions of the Input Clusters (Top) and Output Clusters (Bottom) for Case 1

Input	Dimensions	Points
A	2, 3, 4, 9, 11, 14, 18	21391
B	2, 3, 7	23278
C	2, 12	18245
D	2, 3, 4, 12, 13, 17	15728
E	2, 4	16357
Outliers	-	5001

Found	Dimensions	Points
1	2, 3, 7	22051
2	2, 4	16800
3	2, 3, 4, 12, 13, 17	15387
4	2, 12	18970
5	2, 3, 4, 9, 11, 14, 18	21498
Outliers	-	5294

Table 2: PROCLUS: Dimensions of the Input Clusters (Top) and Output Clusters (Bottom) for Case 2

4.2 Accuracy Results

To test how accurately the algorithm performs we compute the *Confusion Matrix* defined as follows: entry (i, j) is equal to the number of data points assigned to output cluster i, that were generated as part of input cluster j. The last row and column of the matrix represent output outliers, respectively input outliers, and their entries are similarly defined. Obviously, we want each row to have one entry that is much larger than the others, which indicates a clear correspondence between the input and output clusters. In the tables below, the input clusters are denoted by letters, while the output clusters are denoted by numbers. Another significant result is the set of dimensions computed for each output cluster, as compared to the set of dimensions of the corresponding input cluster.

We divided the experiments in two classes. First, we used input files for which all clusters had been generated in the same number of dimensions, but in different subspaces (Case 1). Then, we used input files containing clusters generated in different number of dimensions (Case 2). We report below the results for one experiment in each class. We obtained similar quality in all the other experiments we performed. Both files had $N = 100,000$ data points in a 20-dimensional space, with $k = 5$. The first input file had $l = 7$ (i.e. all input clusters were generated in some 7-dimensional subspace), while the second file had $l = 4$, and the clusters were generated as follows: two clusters were generated in different 2-dimensional subspaces, one in a 3-dimensional subspace, one in a 6-dimensional subspace, and one in a 7-dimensional subspace.

In both cases PROCLUS discovers output clusters in which the majority of points comes from one input cluster, as shown in Tables 3 and 4. In other words, it recognizes the natural clustering of the points. We note that for both files the output clusters pick some of the original outliers and report them as cluster points. This is not necessarily an error, since the outliers were randomly placed throughout the entire space, and it is probable that some of them have actually been placed inside clusters. The output clusters in Table 4 also have some small number of points that should have been assigned to other clusters. For example, the 267 points in row 1 and column C should have been assigned to cluster 4, because they were generated as part of input cluster C, and output cluster 4 has a clear correspondence to cluster C. However, the percentage of misplaced points is very small so that it does not influence the correspondence between input and output clusters, nor would it significantly alter the result of any data mining application based on this clustering. Moreover, there is a perfect correspondence between the sets of dimensions of the output clusters and their corresponding input clusters, as illustrated by Tables 1 and 2. This is important for applications that require not only a good partitioning of the data, but also additional information as to what dimensions (or attributes) are relevant for each partition.

As we mentioned before, CLIQUE does not guarantee that the result it returns represents a partitioning of the points. To quantify how different its output is from an actual partitioning, we compute the average overlap as follows:

Input / Output	A	B	C	D	E	Out.
1	0	0	18245	0	0	456
2	21391	0	0	0	0	523
3	1	23278	0	101	0	697
4	0	0	0	15728	0	290
5	0	0	0	0	16357	638
Outliers	0	0	0	0	0	2396

Table 3: PROCLUS: Confusion Matrix (same number of dimensions) for Case 1

Input / Output	A	B	C	D	E	Out.
1	0	20992	267	416	18	358
2	34	0	0	0	16097	669
3	0	9	1	15309	10	58
4	0	2256	16536	0	0	178
5	21357	0	0	2	10	129
Outliers	0	21	1441	1	222	3609

Table 4: PROCLUS: Confusion Matrix (different number of dimensions) for Case 2

$$overlap = \frac{\sum_{i=1}^{q} |\mathcal{C}_i|}{|\cup_{i=1}^{q} \mathcal{C}_i|},$$

where q is the number of output clusters. Thus, an overlap of 1 means that on the average each point that is not an outlier is assigned to only one cluster, and so the result can be considered a partitioning. On the other hand, a large overlap means that many of the points are assigned to more than one output cluster, so the result cannot be considered a reasonable approximation for a partitioning. In the experiments below we try to determine the cases in which CLIQUE is likely to generate an output with small overlap. For those cases we compare the results of CLIQUE and PROCLUS in terms of quality and running time, to decide which method is preferable. One problem we have encountered during these experiments is that on the average half of the cluster points are considered outliers by CLIQUE. This is a consequence of the density-based approach of the algorithm, since lower-density areas in a cluster can cause some of its points to be thrown away. Another reason is the fact that clusters are considered to be axis-parallel regions. Such a region generally offers a low coverage of the corresponding input cluster, especially as the dimensionality of the cluster increases. Hence, a significant percentage of relevant data points are erroneously considered outliers by CLIQUE. Of course, this percentage can be lowered by tuning the input parameters ξ and τ appropriately. This leads to a tradeoff between quality of output and running time. Moreover, the density threshold of a unit must take into account both the number of intervals on a dimension and the dimensionality of the space. Hence, variation of one input parameter must be correlated with the variation of the other parameter. No obvious method is indicated in [1] for how to choose the two parameters.

For files in which clusters exist in different number of dimensions CLIQUE reported a large number of output clusters, most of which were projections of a higher dimensional cluster. As a result, the average overlap was also large. It is unclear how one can differentiate between, for example, a 2-dimensional output cluster corresponding to a 2-dimensional input cluster, and the 2-dimensional projection of a 6-dimensional cluster. In this case, CLIQUE cannot be used to obtain a good approximation for a partitioning.

Below, we discuss the results we obtained with CLIQUE for input files in which all clusters exist in the same number of dimensions. As in [1], we set ξ (the number of intervals on each dimension) to 10, and we try various values for the density threshold τ. We present the results obtained on an input file with $l = 7$, the same for which we reported the PROCLUS results above. However, the issues we discuss were noted on other input files and for different values of l, as well. For

Input / Output	A	B	C	D	E	Out.
2	11128	0	0	0	0	0
15	19510	0	0	0	0	0
31	0	0	0	101	0	0
32	0	0	0	111	0	0
47	0	0	0	12849	0	0

Table 5: CLIQUE: Matching between Input and Output Clusters (small snapshot)

$\tau = 0.5$ and $\tau = 0.8$, the average overlap was 1, but the percentage of cluster points discovered by CLIQUE was low (42.7%, respectively 30.7%). We then experimented with lower values for τ, more exactly $\tau = 0.2$ and $\tau = 0.1$, expecting the percentage of cluster points to increase. However, because of the low density, CLIQUE reported output clusters in 8 dimensions (one dimension more than they were generated), and the percentage of cluster points decreased to 21.2% for $\tau = 0.1$. Two of the original input clusters were entirely missed, and all their points declared outliers. Of the remaining three input clusters, at least 50% of the points in each one were thrown away as outliers, and two of these input clusters were split into four output clusters. We finally ran CLIQUE with $\tau = 0.1$ and set it to find clusters only in 7 dimensions, using an option provided by the program. It reported 48 output clusters, with a percentage of cluster points equal to 74.6%. The average overlap was 3.63, which means that on the average, an input point had been assigned to at least 3 output clusters. We present the results of this last experiment in Table 5. Due to lack of space, we do not provide the entire Confusion Matrix, but only a small snapshot that reflects both "good" and "bad" output clusters discovered by CLIQUE. We conclude that, while there are cases in which CLIQUE returns a partitioning of the points, PROCLUS is still useful because of its better accuracy.

4.3 Scalability Results

In what follows we will say that two input files are *similar* if the following parameters are identical for both files: number of points N, dimensionality of the space d, number of clusters k, and average dimensionality of a cluster l.

As noticed in the previous subsection, the output of CLIQUE could only be interpreted as an (approximate) partitioning of the points when all clusters exist in the same number of dimensions. Hence, we compare the running times of CLIQUE and PROCLUS only on such files. However, we also tested PROCLUS on similar files in which clusters exist in different number of dimensions, and found no significant difference between the respective running times. Because of the random nature of PROCLUS, each running time reported in this section is averaged over three similar input files. We want to mention that in each run the quality of the results returned by PROCLUS was similar to that presented in the previous subsection.

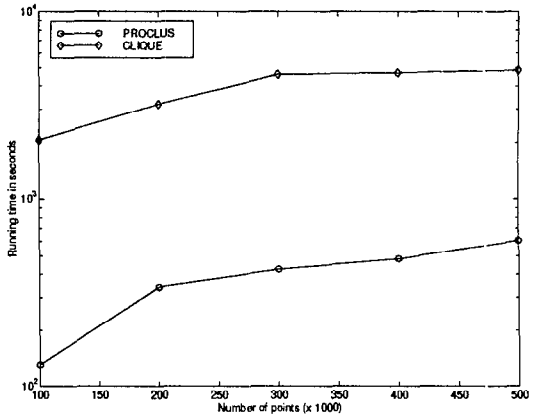


Figure 7: Scalability with number of points

Number of points: All data files on which we tested contained 5 clusters, each existing in some 5-dimensional subspace. The data space was 20-dimensional. We ran CLIQUE with $\xi = 10$ and $\tau = 0.5$. Figure 7 shows that PROCLUS scales linearly with the number of input points, while outperforming CLIQUE by a factor of approximately 10. The graph has logarithmic scale along the y coordinate.

Average dimensionality of the clusters: All files on which we tested had $N = 100,000$ points and contained 5 clusters. The data space was 20-dimensional. We ran CLIQUE with $\xi = 10$ and $\tau = 0.5$ for files in which the dimensionality of clusters was 4, 5 or 6, and with $\tau = 0.1$ for dimensionality of clusters equal to 7 and 8. We selected a lower τ for the higher dimensional clusters because, as the volume of the clusters increases, the cluster density decreases. This corresponds to the approach used for the experiments in [1].

Figure 8 shows that the two algorithms have a different type of dependency on the average cluster dimensionality l. The results we obtained for CLIQUE are consistent with those reported in [1], where an exponential dependency on l is proven. On the other hand, the running time of PROCLUS is only slightly influenced by l. This happens because the main contribution of l to the running time is during the computation of segmental distances, which takes $O(N \cdot$

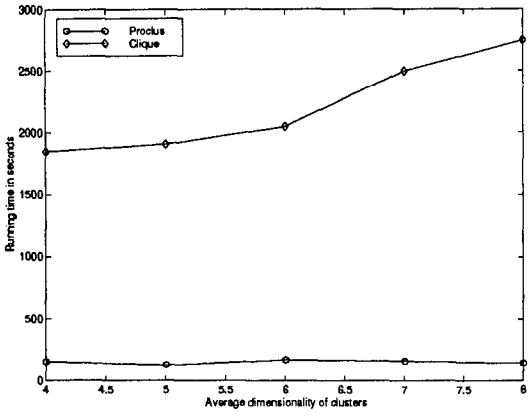


Figure 8: Scalability with average dimensionality

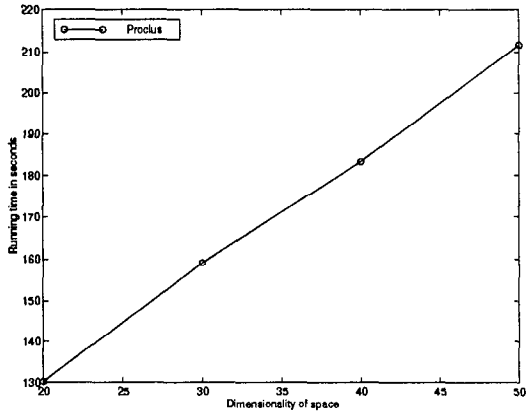


Figure 9: Scalability with dimensionality of the space

$k \cdot l$) for each iteration. Since we are also computing distances in the full dimensional space in time $O(N \cdot k \cdot d)$, the running time of an iteration is dominated by this second term and only slightly influenced by a change in l.

This very good behavior of PROCLUS with respect to l is important for the situations in which it is not clear what value should be chosen for parameter l. Because the running time is so small (about 150 seconds for each point shown in the graph), it is easy to simply run the algorithm a few times and try different values for l.

Dimensionality of the space: We tested on files with $N = 100,000$ points that contained 5 clusters, each existing in a 5-dimensional space. The sensitivity with respect to the dimensionality of the space is illustrated in Figure 9. As expected, PROCLUS scales linearly with the dimensionality of the entire space.

5 Conclusions

We have proposed a new concept, called *projected clustering*, for discovering interesting patterns in subspaces of high dimensional data spaces. This is a generalization of feature selection, in that it allows the selection of different sets of dimensions for different subsets of the data. While feature selection algorithms do not work on all types of data, projected clustering is general enough to allow us to deal with different correlations among various subsets of the input points.

We have also provided a projected clustering algorithm called PROCLUS that returns a *partition* of the data points into clusters, together with the sets of dimensions on which points in each cluster are correlated. The CLIQUE algorithm, which was previously proposed for a variant of this problem, successfully discovers patterns in subspaces of the data space, but its output does not guarantee a partition of the points. Such a partition is often desired in classification and trend analysis problems for better interpretability of results. We conclude that for these applications PROCLUS is the method of choice.

6 Acknowledgements

We would like to thank Dimitrios Gunopulos for providing us with the CLIQUE code.

References

[1] R. Agrawal, J. Gehrke, D. Gunopolos, P. Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998.

[2] D. Hand, Order Statistics. John Wiley and Sons, New York, 1981.

[3] M. Berger, I. Rigoutsos. An Algorithm for Point Clustering and Grid Generation. *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, 5:1278-1286, 1991.

[4] M. R. Brito, E. Chavez, A. Quiroz, J. Yukich. Connectivity of the Mutual k-Nearest-Neighbor Graph for Clustering and Outlier Detection. *Statistics and Probability Letters*, 35 (1997) pages 33-42.

[5] P. Cheeseman, J. Kelly, S. Matthew. AutoClass: A Bayesian Classification System. *Proceedings of the 5th International Conference on Machine Learning*, Morgan Kaufmann, June 1988.

[6] R. Dubes, A. Jain. *Clustering Methodologies in Exploratory Data Analysis.* Advances in Computers, Edited by M. Yovits, Vol. 19, Academic Press, New York, 1980.

[7] M. Ester, H.-P. Kriegel, X. Xu. A Database Interface for Clustering in Large Spatial Databases. Proceedings of the first International Conference on Knowledge Discovery and Data Mining, 1995.

[8] M. Ester, H.-P. Kriegel and X. Xu, Knowledge Discovery in Large Spatial Databases: Focusing Techniques for Efficient Class Identification. *Proceedings of the Fourth International Symposium on Large Spatial Databases,* Portland, Maine, U.S.A. 1995.

[9] M. Ester, H.-P. Kriegel, J. Sander, X. Xu. A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proceedings of the 2nd International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1995.

[10] U. Shardanand, P. Maes. Social information filtering: algorithms for automating "word of mouth". *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pages 210-217, 1995.

[11] D. Fisher. Knowledge Acquisition via Incremental Conceptual Clustering. *Machine Learning* 2(2), 1987.

[12] D. Fisher. Optimization and Simplification of Hierarchical Clusters. *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, August 1995.

[13] D. Gibson, J. Kleinberg, P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. *Proceedings of the 24th VLDB Conference*, pp. 311-322, 1998.

[14] T. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science,* Vol. 38, pp. 293-306, 1985.

[15] S. Guha, R. Rastogi, K. Shim. CURE: An Efficient Clustering Algorithm for Large Databases. *Proceedings of the 1998 ACM SIGMOD Conference*, pp. 73-84, 1998.

[16] T. Ibaraki, N. Katoh. Resource Allocation Problems: Algorithmic Approaches. *MIT Press,*, Cambridge, Massachusetts, 1988.

[17] A. Jain, R. Dubes. Algorithms for Clustering Data. *Prentice Hall,* Englewood Cliffs, New Jersey, 1998.

[18] L. Kaufman, P. Rousseeuw. Finding Groups in Data - An Introduction to Cluster Analysis. Wiley Series in Probability and Mathematical Statistics, 1990.

[19] R. Kohavi, D. Sommerfield. Feature Subset Selection Using the Wrapper Method: Overfitting and Dynamic Search Space Topology. *Proceedings of the First International Conference on Knowledge Discovery and Data Mining, 1995.*

[20] R. Lee. *Clustering Analysis and its applications.* Advances in Information Systems Science, edited by J. Toum, Vol. 8, pp. 169-292, Plenum Press, New York, 1981.

[21] R. Ng, J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *Proceedings of the 20th VLDB Conference*, 1994, pp. 144-155.

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

[23] S. Wharton. A Generalized Histogram Clustering for Multidimensional Image Data. *Pattern Recognition*, Vol. 16, No. 2: pp. 193-199, 1983.

[24] X. Xu, M. Ester, H.-P. Kriegel, J. Sander. A Distribution-Based Clustering Algorithm for Mining in Large Spatial Databases. *Proceedings of the Fourteenth International Conference on Data Engineering*, 1998, pp. 324-331.

[25] M. Zait, H. Messatfa. A Comparative Study of Clustering Methods. *FGCS Journal, Special Issue on Data Mining*, 1997.

[26] T. Zhang, R. Ramakrishnan, M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data,* Montreal, Canada, June 1996.