Pergamon

Information Systems Vol. 25, No. 5, pp. 345–366, 2000
© 2000 Elsevier Science Ltd. All rights reserved
Printed in Great Britain
0306-4379/00 \$20.00 + 0.00

ROCK: A ROBUST CLUSTERING ALGORITHM FOR CATEGORICAL ATTRIBUTES[†]

Sudipto Guha[1], Rajeev Rastogi[2], and Kyuseok Shim[3]

[1]Stanford University, Stanford, CA 94305, USA
[2]Bell Laboratories, Murray Hill, NJ 07974, USA
[3]Korea Advanced Institute of Science and Technology and Advanced Information Technology Research Center, Taejon 305-701, Korea

(Received 18 January 2000; in final revised form 31 May 2000)

Abstract — Clustering, in data mining, is useful to discover distribution patterns in the underlying data. Clustering algorithms usually employ a distance *metric* based (e.g., euclidean) similarity measure in order to partition the database such that data points in the same partition are more similar than points in different partitions. In this paper, we study clustering algorithms for data with boolean and categorical attributes. We show that traditional clustering algorithms that use distances between points for clustering are not appropriate for boolean and categorical attributes. Instead, we propose a novel concept of *links* to measure the similarity/proximity between a pair of data points. We develop a *robust* hierarchical clustering algorithm ROCK that employs links and not distances when merging clusters. Our methods naturally extend to *non-metric* similarity measures that are relevant in situations where a domain expert/similarity table is the only source of knowledge. In addition to presenting detailed complexity results for ROCK, we also conduct an experimental study with real-life as well as synthetic data sets to demonstrate the effectiveness of our techniques. For data with categorical attributes, our findings indicate that ROCK not only generates better quality clusters than traditional algorithms, but it also exhibits good scalability properties. ©2000 Elsevier Science Ltd. All rights reserved

Key words: Data Mining, Knowledge Discovery, Clustering Algorithms

1. INTRODUCTION

The problem of *data mining* or *knowledge discovery* has become increasingly important in recent years. There is an enormous wealth of information embedded in large data warehouses maintained by retailers, telecom service providers and credit card companies that contain information related to customer purchases and customer calls. Corporations could benefit immensely in the areas of marketing, advertising and sales if interesting and previously unknown customer buying and calling patterns can be discovered from the large volumes of data.

Clustering is a useful technique for grouping data points such that points within a single group/cluster have similar characteristics (or are close to each other), while points in different groups are dissimilar. For example, consider a market basket database containing one transaction per customer, each transaction containing the set of items purchased by the customer. The transaction data can be used to cluster the customers such that customers with similar buying patterns are in a single cluster. For example, one cluster may consist of predominantly married customers with infants who buy diapers, baby food, toys etc. (in addition to necessities like milk, sugar and butter), while another may consist of high-income customers that buy imported products like French and Italian wine, Swiss cheese and Belgian chocolate. The clusters can then be used to characterize the different customer groups, and these characterizations can be used in targeted marketing and advertising such that specific products are directed towards specific customer groups. The characterizations can also be used to predict buying patterns of new customers based on their profiles. For example, it may be possible to conclude that high-income customers buy imported foods, and then mail customized catalogs for imported foods to only these high-income customers.

The above market basket database containing transactions is actually an example of a scenario in which attributes of data points are non-numeric. Transactions in the database can be viewed

[†]Recommended by Felipe Carino

as records with *boolean* attributes, each attribute corresponding to a single item. Further, in the record for a transaction, the attribute corresponding to an item is True if and only if the transaction contains the item; otherwise, it is False. Boolean attributes themselves are a special case of *categorical* attributes. The domain of categorical attributes is not limited to simply True and False values, but could be any arbitrary finite set of values. An example of a categorical attribute is color whose domain includes values such as brown, black, white, etc. Clustering in the presence of such categorical attributes is the focus of this paper.

1.1. Shortcomings of Traditional Clustering Algorithms

Given n data points in a d-dimensional space, a clustering algorithm partitions the data points into k clusters such that the data points in a cluster are more similar to each other than data points in different clusters. Clustering algorithms developed in the literature can be classified into *partitional clustering* and *hierarchical clustering* [4, 9]. Partitional clustering algorithms, as the name suggests, divide the point space into k clusters that optimize a certain criterion function. The most commonly used criterion function for metric spaces is

$$E = \sum_{i=1}^{k} \sum_{\vec{x} \in C_i} d(\vec{x}, \vec{m}_i)$$

In the above equation, $\vec{m}_i$ is the centroid of cluster C_i while $d(\vec{x}, \vec{m}_i)$ is the euclidean distance[†] between $\vec{x}$ and $\vec{m}_i$. Thus, intuitively, the criterion function E attempts to minimize the distance of every point from the mean of the cluster to which the point belongs. A common approach is to minimize the criterion function using an iterative, hill-climbing technique. For example, starting with k initial partitions, data points are moved from one cluster to another to improve the value of the criterion function.

While the use of the above criterion function could yield satisfactory results for numeric attributes, it is not appropriate for data sets with categorical attributes. For example, consider a market basket database. Typically, the number of items, and thus the number of attributes in such a database is very large (a few thousand) while the size of an average transaction is much smaller (less than a hundred). Furthermore, customers with similar buying patterns and belonging to a single cluster, may buy a small subset of items from a much larger set that defines the cluster. For instance, consider the cluster defined by the set of imported items like French wine, Swiss cheese, Italian pasta sauce, Belgian beer etc. Every transaction in the cluster does not contain all of the above items, but some subset of them. Thus, it is quite possible that a pair of transactions in a cluster have few items in common, but are *linked* by a number of other transactions in the cluster, that have substantial items in common with the two transactions.

The above situation is further exacerbated by the fact that the set of items that define clusters may not have uniform sizes. A cluster involving all the common items such as diapers, baby food and toys will typically involve a large number of items and customer transactions, while the cluster defined by imported products will be much smaller. In the larger cluster, since transactions are spread out over a larger number of items, most transaction pairs will have few items in common and consequently, a smaller percentage of transaction pairs will have a sizable number of items in common. Thus, distances of transactions from the mean in the larger cluster will be much higher. Since the criterion function is defined in terms of distance from the mean, splitting the larger cluster reduces its value, and thus minimizing the criterion function favors splitting large clusters. However, this is not desirable since the large cluster is split even though transactions in the cluster are well connected and strongly linked.

Hierarchical clustering algorithms, too, may be unsuitable for clustering data sets containing categorical attributes. For instance, consider the centroid-based *agglomerative* hierarchical clustering algorithm [4, 9]. In this algorithm, initially, each point is treated as a separate cluster. Pairs of clusters whose centroids or means are the closest are then successively merged until the

[†]The euclidean distance between two points $(x_1, x_2, \ldots, x_d)$ and $(y_1, y_2, \ldots, y_d)$ is $(\sum_{i=1}^{d}(x_i - y_i)^2)^{\frac{1}{2}}$.

desired number of clusters remain. For categorical attributes, however, distances between centroids of clusters is a poor estimate of the similarity between them as is illustrated by the following example.

Example 1 Consider a market basket database containing the following 4 transactions over items 1, 2, 3, 4, 5 and 6 – (a) {1, 2, 3, 5}, (b) {2, 3, 4, 5}, (c) {1, 4}, and (d) {6}. The transactions can be viewed as points with boolean (0/1) attributes corresponding to the items 1, 2, 3, 4, 5 and 6. The four points thus become (1,1,1,0,1,0), (0,1,1,1,1,0), (1,0,0,1,0,0) and (0,0,0,0,0,1). Using euclidean distance to measure the closeness between points/clusters, the distance between the first two points is $\sqrt{2}$, which is the smallest distance between pairs of points. As a result, they are merged by the centroid-based hierarchical algorithm. The centroid of the new merged cluster is (0.5,1,1,0.5,1,0). In the next step, the third and fourth points are merged since the distance between them is $\sqrt{3}$ which is less than the distance between the centroid of the merged cluster from each of them – $\sqrt{3.5}$ and $\sqrt{4.5}$, respectively. However, this corresponds to merging transactions {1, 4} and {6} that don't have a single item in common. Thus, using distances between the centroids of clusters when making decisions about the clusters to merge could cause points belonging to different clusters to be assigned to a single cluster. □

Once points belonging to different clusters are merged, the situation gets progressively worse as the clustering progresses. What typically happens is a ripple effect – as the cluster size grows, the number of attributes appearing in the mean go up, and their value in the mean decreases. This makes it very difficult to distinguish the difference between two points that differ on few attributes, or two points that differ on every attribute by small amounts. An example will make this issue very clear. Consider the means of two clusters $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 0, 0, 0)$ and $(0, 0, 0, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, with roughly the same number of points. Even though, the two clusters have no attributes in common, the euclidean distance between the two means is less than the distance of the point $(1, 1, 1, 0, 0, 0)$ to the mean of the first cluster. Obviously, this is undesirable since the point shares common attributes with the first cluster. An oblivious method based on distance will merge the two clusters and will generate a new cluster with mean $(\frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6})$. Interestingly, the distance of the point $(1, 1, 1, 0, 0, 0)$ to the new cluster is even larger than the original distance of the point to the first of the merged clusters. In effect, what is happening is that the center of the cluster is spreading over more and more attributes. As this tendency starts, it now becomes closer to other centers which also span a large number of attributes. Thus, these centers tend to spread out in all the attribute values and lose the information about the points in the cluster that they represent. This is why a small ripple soon spreads out to fill all the attributes. This is exactly the behavior that we observed when we ran the centroid-based hierarchical algorithm on a real-life data set (see Section 5).

Set theoretic similarity measures such as the *Jaccard coefficient*[†] [9] have often been used, instead of euclidean distance, for document clustering. With the Jaccard coefficient as the distance measure between clusters, centroid-based hierarchical clustering schemes cannot be used since the similarity measure is non-metric, and defined for only points in the cluster and not for its centroid. Thus, we have to use either the *minimum spanning tree* (MST) hierarchical clustering algorithm or hierarchical clustering with *group average* [9]. The MST algorithm merges, at each step, the pair of clusters containing the most similar pair of points while the group average algorithm merges the ones for which the average similarity between pairs of points in the clusters is the highest. The MST algorithm is known to be very sensitive to outliers while the group average algorithm has a tendency to split large clusters (since, as mentioned earlier, the average similarity between two subclusters of a large cluster is small). Furthermore, the Jaccard coefficient is a measure of the similarity between only the two points in question – it thus, does not reflect the properties of the neighborhood of the points. Consequently, the Jaccard coefficient fails to capture the natural clustering of "not so well-separated" data sets with categorical attributes and this is illustrated further in the following example.

Example 2 Consider a market basket database over items $1, 2, \ldots, 8, 9$. Consider the 2 transaction clusters shown in Figure 1. The first cluster is defined by 5 items while the second cluster is defined

[†]The Jaccard coefficient for similarity between transactions T_1 and T_2 is $\frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$.

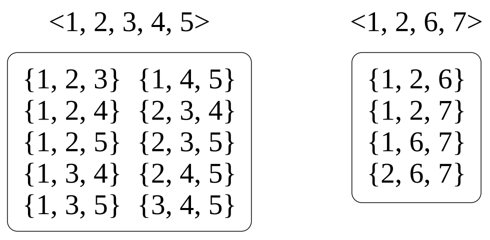

Fig. 1: Basket Data Example for Jaccard Coefficient

by 4 items. These items are shown at the top of each of the two clusters. Note that items 1 and 2 are common to both clusters. Each cluster contains transactions of size 3, one for every subset (of size 3) of the set of items that define the cluster. The Jaccard coefficient between an arbitrary pair of transactions belonging to the first cluster ranges from 0.2 (e.g., {1,2,3} and {3,4,5}) to 0.5 (e.g., {1,2,3} and {1,2,4}). Note that even though {1,2,3} and {1,2,7} share common items and have a high Jaccard coefficient of 0.5, they belong to different clusters. In contrast, {1,2,3} and {3,4,5} have a lower Jaccard coefficient of 0.2, but belong to the same cluster.

The MST algorithm may first merge transactions {1,2,3} and {1,2,7} since the Jaccard coefficient for them has the maximum value of 0.5. Once this happens, the cluster may subsequently merge with transactions from both clusters like {1,3,4} and {1,6,7} since these are very similar to transactions in the merged cluster. This is not surprising since the MST algorithm is known to be fragile when clusters are not well-separated.

The use of group average for merging clusters ameliorates some of the problems with the MST algorithm. However, it may still fail to discover the correct clusters. For instance, similar to MST, it may first merge a pair of transactions containing items 1 and 2, and belonging to different clusters. Note that the group average of the Jaccard coefficient between the new cluster and every other transaction containing both 1 and 2 is still maximum, that is, 0.5. Consequently, every transaction containing both 1 and 2 may get merged together into a single cluster in subsequent steps. Thus, in the final clustering, transactions {1,2,3} and {1,2,7} from the two different clusters may be assigned to the same cluster. □

1.2. Our Contributions

In this paper, we present a novel concept of clustering that is based on *links* between data points, instead of distances based on the L_p[†] metric or the Jaccard coefficient. For domains with discrete non-numeric attributes, the unsuitability of the L_p distance metrics and the Jaccard coefficient as an estimate of the similarity between clusters is evident from Examples 1 and 2. The situation with these distance metrics further worsens as the number of attributes/dimensions increase.

The notion of *links* between data points helps us overcome the problems with L_p distances and the Jaccard coefficient. Let a pair of points be *neighbors* if their similarity exceeds a certain threshold. The similarity value for pairs of points can be based on L_p distances, the Jaccard coefficient or any other *non-metric* similarity function obtained from a domain expert/similarity table. The number of links between a pair of points is then the number of *common* neighbors for the points. Points belonging to a single cluster will in general have a large number of common neighbors, and consequently more links. Thus, during clustering, merging clusters/points with the most number of links first will result in better and more meaningful clusters.

Specifically, in Example 1, suppose we defined a pair of transactions to be neighbors if they contained at least one item in common. In that case, transactions {1, 4} and {6} would have no links between them and thus would not be merged. Thus, links are more appropriate than distances for clustering categorical data sets. Similarly, in Example 2, let us assume that a pair of transactions are considered to be neighbors if they have at least two items in common. Then, any pair of transactions containing both 1 and 2 and in the same cluster (e.g., {1, 2, 3} and

[†] $L_p = (\sum_1^d |x_i - y_i|^p)^{1/p}$, $1 \le p \le \infty$ and d is the dimensionality of the data points.

{1, 2, 4}) has 5 common neighbors (due to {1, 2, 5}, {1, 2, 6}, {1, 2, 7}, {1, 3, 4} and {2, 3, 4}) while a pair of transactions containing 1 and 2, but in different clusters (e.g., {1, 2, 3} and {1, 2, 6}) has only 3 neighbors in common (due to {1, 2, 4}, {1, 2, 5} and {1, 2, 7}). Thus, even though popular similarity measures like L_p distances and the Jaccard coefficient would consider the transaction pairs ({1, 2, 3}, {1, 2, 4}) and ({1, 2, 3}, {1, 2, 6}) equally similar, our link-based approach would consider the former pair (with 5 links) belonging to the same cluster more similar than the latter pair (with 3 links) in different clusters. As a result, clustering algorithms that use links when making decisions about clusters to merge would favor merging the first transaction pair over the second, and are thus less fragile.

From the above examples, it follows that unlike distances or similarities between a pair of points which are local properties involving only the two points in question, the link concept incorporates global information about the other points in the neighborhood of the two points. The larger the number of links between a pair of points, the greater is the likelihood that they belong to the same cluster. Thus, clustering using links injects global knowledge into the clustering process and is thus more robust. For example, even though a pair of clusters are not well-separated and have a few points that are quite similar, these points will not be coalesced into a single cluster since they will have very few common neighbors and thus very few links. To the best of our knowledge, we are not aware of any work that so elegantly and succinctly captures, in a relationship involving a pair of data points, information about their neighbors.

In this paper, we first discuss recent work on clustering algorithms for data mining in Section 2. In Section 3, we present a new criterion function based on our notion of links and show that maximizing the value of this criterion function results in desirable clusters for data with categorical attributes. We then present an agglomerative hierarchical clustering algorithm ROCK (RObust Clustering using linKs) that iteratively merges clusters so as to try and maximize the criterion function in Section 4. We also present complexity results for ROCK. Following this, in Section 5, with real-life as well as synthetic data sets, we show that the quality of clusters generated by ROCK are far superior to the clusters produced by the traditional centroid-based hierarchical clustering algorithm. One of the real-life data sets we consider is a time-series database, thus demonstrating the utility of ROCK as a tool for also clustering time-series data. In addition, we show that ROCK can be used to cluster large data sets and scales well with database size. Finally, in Section 6, we summarize our research results.

2. RELATED WORK

Clustering has been extensively studied by researchers in psychology, statistics, biology and so on. Surveys of clustering algorithms can be found in [4, 9]. More recently, clustering algorithms for mining large databases have been proposed in [11, 6, 14, 5, 7]. Most of these, however, are variants of either partitional (e.g., [11]) or centroid-based hierarchical clustering (e.g., [14, 7]). As a result, as pointed out in Section 1.1, these algorithms are more suitable for clustering numeric data rather than data sets with categorical attributes. For instance, CLARANS [11] employs a randomized search to find the k best cluster *medoids*. BIRCH, proposed in [14], first preclusters data and then uses a centroid-based hierarchical algorithm to cluster the partial clusters. The CURE algorithm [7] uses a combination of random sampling and partition clustering to handle large databases. In addition, its hierarchical clustering algorithm represents each cluster by a certain number of points that are generated by selecting *well scattered* points and then shrinking them toward the cluster centroid by a specified fraction. DBSCAN, a density-based algorithm proposed in [5], grows clusters by including the dense neighborhoods of points already in the cluster. This approach, however, may be prone to errors if clusters are not well-separated.

Recently, in [8], the authors address the problem of clustering related customer transactions in a market basket database. Frequent itemsets used to generate association rules are used to construct a weighted hypergraph. Each frequent itemset is a hyperedge in the weighted hypergraph and the weight of the hyperedge is computed as the average of the confidences for all possible association rules that can be generated from the itemset. Then, a hypergraph partitioning algorithm from [10] is used to partition the items such that the sum of the weights of hyperedges that are cut due to the partitioning is minimized. The result is a clustering of items (not transactions) that occur together in the transactions. Finally, the item clusters are used as the description of the cluster and a scoring metric is used to assign customer transactions to the best item cluster. For example, a transaction T may be assigned to the item cluster C_i for which the ratio $\frac{|T \cap C_i|}{|C_i|}$ is the highest.

The rationale for using item clusters to cluster transactions is questionable. For example, the approach in [8] makes the assumption that itemsets that define clusters are disjoint and have no overlap among them. This may not be true in practice since transactions in different clusters may have a few common items. For instance, consider the market basket database in Example 2. With minimum support set to 2 transactions, the hypergraph partitioning algorithm generates two item clusters of which one is $\{7\}$ and the other contains the remaining items (since 7 has the least hyperedges to other items). However, this results in transactions $\{1, 2, 6\}$ and $\{3, 4, 5\}$ being assigned to the same cluster since both have the highest score with respect to the big item cluster.

3. CLUSTERING PARADIGM

In this section, we present our new clustering model that is based on the notions of *neighbors* and *links*. We also discuss the criterion function that we would like to optimize under our new clustering paradigm.

3.1. Neighbors

Simply put, a point's *neighbors* are those points that are considerably similar to it. Let $sim(p_i, p_j)$ be a *similarity function* that is normalized and captures the closeness between the pair of points p_i and p_j. The function *sim* could be one of the well-known distance metrics (e.g., L_1, L_2) or it could even be *non-metric* (e.g., a distance/similarity function provided by a domain expert). We assume that *sim* assumes values between 0 and 1, with larger values indicating that the points are more similar. Given a threshold θ between 0 and 1, a pair of points p_i, p_j are defined to be *neighbors* if the following holds:

$$sim(p_i, p_j) \geq \theta$$

In the above equation, θ is a user-defined parameter that can be used to control how close a pair of points must be in order to be considered neighbors. Thus, higher values of θ correspond to a higher threshold for the similarity between a pair of points before they are considered neighbors. Assuming that *sim* is 1 for identical points and 0 for totally dissimilar points, a value of 1 for θ constrains a point to be a neighbor to only other identical points. On the other hand, a value of 0 for θ permits any arbitrary pair of points to be neighbors. Depending on the desired closeness, an appropriate value of θ may be chosen by the user.

In the following subsections, we present possible definitions for *sim* for market basket databases and for data sets with categorical attributes.

3.1.1. Market Basket Data

The database consists of a set of transactions, each of which is a set of items. A possible definition based on the Jaccard coefficient [4], for $sim(T_1, T_2)$, the similarity between the two transactions T_1 and T_2, is the following:

$$sim(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$$

where $|T_i|$ is the number of items in T_i. The more items that the two transactions T_1 and T_2 have in common, that is, the larger $|T_1 \cap T_2|$ is, the more similar they are. Dividing by $|T_1 \cup T_2|$ is the scaling factor which ensures that θ is between 0 and 1. Thus, the above equation computes the relative closeness based on the items appearing in both transactions T_1 and T_2.

The above definition of a neighbor rules out subsets of a transaction that are very small in size. A typical example is that of a store where milk is bought by everyone. A transaction with only milk will not be considered very similar to other bigger transactions that contain milk. Also, note that for a pair of transactions T_1 and T_2, *sim* can take at most $\min\{|T_1|, |T_2|\} + 1$ values. Thus, there are at most $\min\{|T_1|, |T_2|\} + 1$ distinct similarity levels between the two transactions. As a result, if most transactions have uniform sizes, then there aren't too many possible values for *sim* for the transactions in the database, and this could simplify the choice of an appropriate value for the parameter θ.

3.1.2. Categorical Data

Data sets with categorical attributes can be handled in a manner similar to how we handled market basket data in the previous subsection. Categorical data typically is of fixed dimension and is more structured than market basket data. However, it is still possible that in certain records, values may be *missing* for certain attributes, as is the case for some of the real-life data sets we consider in Section 5.

We propose to handle categorical attributes with missing values by modeling each record with categorical attributes as a transaction. Corresponding to every attribute A and value v in its domain, we introduce an item $A.v$. A transaction T_i for a record contains $A.v$ if and only if the value of attribute A in the record is v. Note that if the value for an attribute is missing in the record, then the corresponding transaction does not contain items for the attribute. Thus, in the proposal, we simply ignore missing values. The similarity function proposed in the previous subsection can then be used to compute similarities between records by determining the similarity between the corresponding transactions.

Obviously, the above suggested method for dealing with missing values is one of several possible ways to handle them, and may not work well across all domains. For instance, in time-series data, each data point consists of a sequence of time slot, value pairs. We can conceptualize time-series data as a categorical dataset. Each data point can be viewed as a record with every time slot corresponding to a single categorical attribute. The values that are possible in the time slot then constitute the domain of the categorical attribute. Missing values for attributes can frequently result since two individual time-series could be sampled at different times. For example, for young mutual funds that began a year ago, prices for time periods preceding the last year do not exist.

In this case, for two records, in order to compute the similarity between them, we are only interested in considering attributes that have values in both records. This way, if two records are identical for the attributes that do not contain missing values, then we will conclude that the similarity between them is high even though for a number of other attributes, one of the records may have a missing value. Thus, for a pair of records, the transaction for each record only contains items that correspond to attributes for which values are not missing in either record. The similarity between the transactions can then be computed as described earlier in Section 3.1.1. Note that the same record may correspond to different transactions when computing its similarity with respect to different records (depending on the missing values for attributes in the different records).

3.2. Links

Clustering points based on only the closeness or similarity between them is not strong enough to distinguish two "not so well-separated" clusters because it is possible for points in different clusters to be neighbors. In this situation, even if a pair of points p_i and p_j in different clusters are neighbors, it is very unlikely that the pairs have a large number of *common* neighbors, that is, points that are neighbors to both p_i and p_j. This observation motivates the following definition of *links* below that builds on the notion of closeness between points to determine more effectively when close points actually belong to the same cluster.

Let us define $link(p_i, p_j)$ to be the number of *common* neighbors between p_i and p_j. From the definition of links, it follows that if $link(p_i, p_j)$ is large, then it is more probable that p_i and p_j belong to the same cluster. In our framework, we exploit this property of links when making decisions about points to merge into a single cluster. Most existing work only uses the similarity measure between points when clustering them – at each step, points that are the most similar are merged into a single cluster. Since the similarity measure between a pair of points only takes into account characteristics of the points themselves, it is a more *local* approach to clustering. This approach is susceptible to errors since as we mentioned earlier, two distinct clusters may have a few points or outliers that could be very close – relying simply on the similarities between points to make clustering decisions could cause the two clusters to be merged.

The link-based approach adopts a *global* approach to the clustering problem. It captures the global knowledge of neighboring data points into the relationship between individual pairs of points. Thus, since the ROCK clustering algorithm utilizes the information about links between points when making decisions on the points to be merged into a single cluster, it is very robust.

The notion of links between a pair of points, in effect, is the number of distinct paths of length 2 between points p_i and p_j such that every pair of consecutive points on the path are neighbors. Alternative definitions for links, based on paths of length 3 or more, are certainly possible; however, we do not consider these for the following reasons. First and most important, computing paths of length 2 is computationally a lot more efficient than computing paths of higher lengths. Second, points connected by paths of length 2 represent more tightly connected points than points connected by paths with larger lengths. Finally, paths of length 2 constitute the simplest and most cost-efficient way of capturing the knowledge about the mutual neighborhood of points – the additional information gained as a result of considering longer paths may not be as valuable.

Our link-based approach can correctly identify the overlapping clusters in Figure 1. This is because for each transaction, the transaction that it has the most links with is a transaction in its own cluster. For instance, let $\theta = 0.5$ and $sim(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}$. Transaction $\{1,2,6\}$ has 5 links with transaction $\{1,2,7\}$ in its own cluster (due to $\{1,2,3\}$, $\{1,2,4\}$, $\{1,2,5\}$, $\{1,6,7\}$ and $\{2,6,7\}$) and only 3 links with transaction $\{1,2,3\}$ in the other cluster (due to $\{1,2,4\}$, $\{1,2,5\}$ and $\{1,2,7\}$). Similarly, transaction $\{1,6,7\}$ has 2 links with every transaction in the smaller cluster (e.g., $\{1,2,6\}$) and 0 links with every other transaction in the bigger cluster. Thus, even though the clusters contain common items, with $\theta = 0.5$, our link-based approach would generate the correct clusters shown in Figure 1.

3.3. Criterion Function

For a clustering method, an important question is the following: "is it possible to characterize the best clusters ?". If one could mathematically characterize the "best clusters", then this would aid in the development of algorithms that attempt to find these good clusters. In this subsection, we present a *criterion function* – the best clusters are the ones that maximize the value of the criterion function.

Since we are interested in each cluster to have a high degree of connectivity, we would like to maximize the sum of $link(p_q, p_r)$ for data point pairs p_q, p_r belonging to a single cluster and at the same time, minimize the sum of $link(p_q, p_s)$ for p_q, p_s in different clusters. This leads us to the following criterion function that we would like to maximize for the k clusters.

$$E_l = \sum_{i=1}^{k} n_i * \sum_{p_q, p_r \in C_i} \frac{link(p_q, p_r)}{n_i^{1+2f(\theta)}}$$

where C_i denotes cluster i of size n_i. The rationale for the above criterion function E_l is as follows. It may seem that since one of our goals was to maximize $link(p_q, p_r)$ for all pairs of points p_q, p_r, a simple criterion function like $\sum_{i=1}^{k} \sum_{p_q, p_r \in C_i} link(p_q, p_r)$ that simply sums up the links between pairs of points in the same cluster, ought to work fine. However, even though this criterion function will ensure that points with a large number of links between them are assigned to the same cluster, it does not prevent a clustering in which all points are assigned to a single cluster. Thus, it does not force points with few links between them to be split between different clusters.

In order to remedy the above problem, in the criterion function E_l, we divide the total number of links involving pairs of points in cluster C_i by the expected total number of links in C_i, and then weigh this quantity by n_i, the number of points in C_i. Our estimate for the total number of links in cluster C_i is $n_i^{1+2f(\theta)}$, where $f(\theta)$ is a function that is dependent on the data set as well

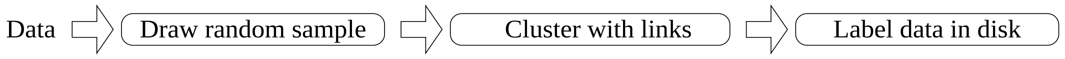

Fig. 2: Overview of ROCK

as the kind of clusters we are interested in, and has the following important property: each point belonging to cluster C_i has approximately $n_i^{f(\theta)}$ neighbors in C_i. If such a function f does exist, then since we can assume that points outside C_i result in a very small number of links to the points in C_i, each point in cluster C_i contributes $n_i^{2f(\theta)}$ links - one for each pair of its neighbors. Thus, we obtain $n_i^{1+2f(\theta)}$ as the expected number of links between pairs of points in C_i. Dividing by the expected number of links in E_l prevents points with very few links between them from being put in the same cluster since assigning them to the same cluster would cause the expected number of links for the cluster to increase more than the actual number of links and the result would be a smaller value for the criterion function.

Of course, it may not be easy to determine an accurate value for function $f(\theta)$. However, we have found that if clusters are fairly well-defined, even an inaccurate but reasonable estimate for $f(\theta)$ can work well in practice (see Section 5). Furthermore, in E_l, every cluster is normalized by $n_i^{1+2f(\theta)}$. Thus, errors in the estimation of $f(\theta)$ affect all the clusters similarly, and does not penalize one cluster excessively over other clusters.

For the market basket data case, one possibility for $f(\theta)$ is $\frac{1-\theta}{1+\theta}$. This can be informally derived under the simplifying assumptions that transactions are of approximately the same size (say t) and are uniformly distributed amongst the (say m) items purchased by customers in cluster C_i. For some constant $c \leq 1$, the number of transactions in the cluster is approximately $\binom{mc}{t}$ and the number of transactions whose similarity to a particular transaction T_i exceeds θ is approximately $\binom{mc}{\frac{(1-\theta)t}{1+\theta}}$ (these are all the transactions that have at least $\frac{2\theta t}{1+\theta}$ items in common with T_i). Thus, the number of neighbors for a transaction in C_i is approximately $n_i^{\frac{1-\theta}{1+\theta}}$ and $f(\theta) = \frac{1-\theta}{1+\theta}$. Intuitively, this makes sense, because when $\theta = 1$, a transaction has only itself as a neighbor and since $f(\theta) = 0$, the expected number of links is in C_i is n_i; on the other hand, when $\theta = 0$, every other transaction in C_i is a neighbor to a transaction and in this case, $f(\theta) = 1$ and the expected number of links in C_i appropriately becomes n_i^3.

In the following section, we adapt standard hierarchical clustering so that it attempts to maximize our link-based criterion function.

4. THE ROCK CLUSTERING ALGORITHM

In this section, we describe the ROCK (RObust Clustering using linKs) clustering algorithm which belongs to the class of agglomerative hierarchical clustering algorithms. We begin by presenting an overview of ROCK, and reserve the details and complexity results for subsequent subsections.

4.1. Overview of ROCK

The steps involved in clustering using ROCK are described in Figure 2. After drawing a random sample from the database, a hierarchical clustering algorithm that employs links is applied to the sampled points. Finally, the clusters involving only the sampled points are used to assign the remaining data points on disk to the appropriate clusters. In the following subsections, we first describe the steps performed by ROCK in greater detail.

4.2. Goodness Measure

In Section 3.3, we presented the criterion function which can be used to estimate the "goodness" of clusters. The best clustering of points were those that resulted in the highest values for the criterion function. Since our goal is to find a clustering that maximizes the criterion function, we use a measure similar to the criterion function in order to determine the best pair of clusters to merge at each step of ROCK's hierarchical clustering algorithm. For a pair of clusters C_i, C_j, let $link[C_i, C_j]$ store the number of cross links between clusters C_i and C_j, that is, $\sum_{p_q \in C_i, p_r \in C_j} link(p_q, p_r)$. Then, we define the *goodness measure* $g(C_i, C_j)$ for merging clusters C_i, C_j as follows.

$$g(C_i, C_j) = \frac{link[C_i, C_j]}{(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}}$$

The pair of clusters for which the above goodness measure is maximum is the best pair of clusters to be merged at any given step. It seems intuitive that pairs of clusters with a large number of cross links are, in general, good candidates for merging. However, using only the number of cross links between pairs of clusters as an indicator of the goodness of merging them may not be appropriate. This naive approach may work well for well-separated clusters, but in case of outliers or clusters with points that are neighbors, a large cluster may swallow other clusters and thus, points from different clusters may be merged into a single cluster. This is because a large cluster typically would have a larger number of cross links with other clusters.

In order to remedy the problem, as we did in section 3.3, we divide the number of cross links between clusters by the expected number of cross links between them. Thus, if every point in C_i has $n_i^{f(\theta)}$ neighbors, then the expected number of links involving only points in the cluster is approximately $n_i^{1+2f(\theta)}$. Since for large clusters, we can assume that points outside the cluster contribute minimally to the number of links between pairs of points in the cluster, the expected number of links between points within the cluster is approximately $n_i^{1+2f(\theta)}$. As a result, it follows that if two fairly large clusters with sizes n_i and n_j are merged, the number of links between pairs of points in the merged cluster is $(n_i + n_j)^{1+2f(\theta)}$, while the number of links in each of the clusters (before merging) were $n_i^{1+2f(\theta)}$ and $n_j^{1+2f(\theta)}$, respectively. Thus, the expected number of *cross* links or links between pairs of points each from a different cluster, becomes $(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}$. We use this normalization factor in the above goodness measure as a heuristic to steer us in the direction of clusters with large values for the criterion function.

4.3. Clustering Algorithm

ROCK's hierarchical clustering algorithm is presented in Figure 3. It accepts as input the set S of n sampled points to be clustered (that are drawn randomly from the original data set), and the number of desired clusters k. The procedure begins by computing the number of links between pairs of points in Step 1 (schemes for this are described in the next subsection). Initially, each point is a separate cluster. For each cluster i, we build a local heap $q[i]$ and maintain the heap during the execution of the algorithm. $q[i]$ contains every cluster j such that $link[i, j]$ is non-zero. The clusters j in $q[i]$ are ordered in the decreasing order of the goodness measure with respect to i, $g(i, j)$.

In addition to the local heaps $q[i]$ for each cluster i, the algorithm also maintains an additional global heap Q that contains all the clusters. Furthermore, the clusters in Q are ordered in the decreasing order of their best goodness measures. Thus, $g(j, \max(q[j]))$ is used to order the various clusters j in Q, where $\max(q[j])$, the max element in $q[j]$, is the best cluster to merge with cluster j. At each step, the max cluster j in Q and the max cluster in $q[j]$ are the best pair of clusters to be merged.

The while-loop in Step 5 iterates until only k clusters remain in the global heap Q. In addition, it also stops clustering if the number of links between every pair of the remaining clusters becomes zero. In each step of the while-loop, the max cluster u is extracted from Q by extract_max and $q[u]$ is used to determine the best cluster v for it. Since clusters u and v will be merged, entries for u and v are no longer required and can be deleted from Q. Clusters u and v are then merged in Step 9 to create a cluster w containing $|u| + |v|$ points. There are two tasks that need to be carried out once clusters u and v are merged: (1) for every cluster that contains u or v in its local

```
procedure cluster($S, k$)
begin
1.  $link$ := compute_links($S$)
2.  for each $s \in S$ do
3.      $q[s]$ := build_local_heap($link, s$)
4.  $Q$ := build_global_heap($S, q$)
5.  while size($Q$) $> k$ do {
6.      $u$ := extract_max($Q$)
7.      $v$ := max($q[u]$)
8.      delete($Q, v$)
9.      $w$ := merge($u, v$)
10.     for each $x \in q[u] \cup q[v]$ do {
11.         $link[x, w] := link[x, u] + link[x, v]$
12.         delete($q[x], u$); delete($q[x], v$)
13.         insert($q[x], w, g(x, w)$); insert($q[w], x, g(x, w)$)
14.         update($Q, x, q[x]$)
15.     }
16.     insert($Q, w, q[w]$)
17.     deallocate($q[u]$); deallocate($q[v]$)
18. }
end
```

Fig. 3: Clustering Algorithm

heap, the elements u and v need to be replaced with the new merged cluster w and the local heap needs to be updated, and (2) a new local heap for w needs to be created.

Both these tasks are carried out in the for-loop of Step 10–15. The number of links between clusters x and w is simply the sum of the number of links between x and u, and x and v. This is used to compute $g(x, w)$, the new goodness measure for the pair of clusters x and w, and the two clusters are inserted into each other's local heaps. Note that $q[w]$ can only contain clusters that were previously either in $q[u]$ or $q[v]$ since these are the only clusters that have non-zero links with cluster w. Also, note that, as a result of merging clusters u and v, it is possible that the cluster u or v was previously the best to be merged with x and now w becomes the best one for being merged. Furthermore, it is also possible that neither u nor v was the best cluster to merge with x, but now w is a better cluster to merge with x. For such cases, whenever the max cluster in the local heap for x changes, the algorithm needs to relocate x in Q to reflect information relating to the new best cluster for x (see Step 14). The procedure also needs to make sure that Q contains the best cluster to be merged for the new cluster w.

4.4. Computation of Links

One way of viewing the problem of computing links between every pair of points is to consider an n x n adjacency matrix A in which entry $A[i, j]$ is 1 or 0 depending on whether or not points i and j, respectively, are neighbors. The number of links between a pair of points i and j can be obtained by multiplying row i with column j (that is, $\sum_{l=1}^{n} A[i, l] * A[l, j]$). Thus, the problem of computing the number of links for all pairs of points is simply that of multiplying the adjacency matrix A with itself, in other words, A x A. The time complexity of the naive algorithm to compute the square of a matrix is $O(n^3)$. However the problem of calculating the square of a matrix is a well studied problem and well-known algorithms such as Strassen's algorithm [3] runs in time $O(N^{2.81})$. The best complexity possible currently is $O(N^{2.37})$ due to the algorithm by Coppersfield and Winograd [2].

We expect that, on an average, the number of neighbors for each point will be small compared

```
procedure compute_links($S$)
begin
1.  Compute $nbrlist[i]$ for every point $i$ in S
2.  Set $link[i, j]$ to be zero for all $i, j$
3.  for $i := 1$ to $n$ do {
4.      $N := nbrlist[i]$
5.      for $j := 1$ to $|N| - 1$ do
6.          for  $l := j + 1$ to $|N|$ do
7.              $link[N[j], N[l]] := link[N[j], N[l]] + 1$
8.  }
end
```

Fig. 4: Algorithm for Computing Links

to the number of input points n, causing the adjacency matrix A to be sparse. For such sparse matrices, the algorithm in Figure 4 provides a more efficient way of computing links.

For every point, after computing a list of its neighbors, the algorithm considers all pairs of its neighbors. For each pair, the point contributes one link. If the process is repeated for every point and the link count is incremented for each pair of neighbors, then at the end, the link counts for all pairs of points will be obtained. If m_i is the size of the neighbor list for point i, then for point i, we have to increase the link count by one in m_i^2 entries. Thus, the complexity of the algorithm is $\sum_i m_i^2$ which is $O(nm_m m_a)$, where m_a and m_m are the average and maximum number of neighbors for a point, respectively. In the worst case, the value of m_m can be n in which case the complexity of the algorithm becomes $O(m_a n^2)$. In practice, we expect m_m to be reasonably close to m_a and thus, for these cases, the complexity of the algorithm reduces to $O(m_a^2 n)$ on average. For market basket data, when transactions are uniformly distributed amongst the attributes, we showed that the expected value for the number of neighbors per point is $n^{f(\theta)}$, where $f(\theta) = \frac{1-\theta}{1+\theta}$. Assuming $\theta = 0.5$, m_a is approximately $n^{\frac{1}{3}}$, which is much smaller than $\sqrt{n}$. This results in a time complexity of $O(n^2)$ for computing the links. Note that the list of neighbors for every point can be computed in $O(n^2)$ time. In our experiments, we found that values of θ larger than 0.5 generally resulted in good clustering. For these larger θ values, the overhead of computing links can be expected to be low in practice.

4.5. Time and Space Complexity

Computation of Links As shown in the previous section, it is possible to compute links among pairs of points in $O(n^{2.37})$ using standard matrix multiplication techniques, or alternatively in $O(n^2 m_a)$ time for average number of neighbors m_a. The space requirement for the link computation is at most $n(n+1)/2$, when every pair of points are linked. However, in general, not every pair of points will have links between them and we expect the storage requirements to be much smaller. We can shown this to be $O(\min\{nm_m m_a, n^2\})$ where m_m is the maximum number of neighbors for a point. This is because a point i can have links to at most $\min\{n, m_m m_i\}$ other points.

Clustering Algorithm The time to build each local heap initially is $O(n)$ (a heap for a set of n input clusters can be built in time that is linear in the number of clusters [3]). The global heap also has at most n clusters initially, and can be constructed in $O(n)$ time. We next examine the complexities of the steps in the while-loop which is executed $O(n)$ times. The inner for-loop dominates the complexity of the while-loop. Since the size of each local queue can be n in the worst case, and the new merged cluster w may need to be inserted in $O(n)$ local queues, the time complexity of the for-loop becomes $O(n \log n)$, and that of the while-loop is $O(n^2 \log n)$ in the worst case. Due to the above analysis, ROCK's clustering algorithm, along with computation of neighbor lists and links, has a worst-case time complexity of $O(n^2 + nm_m m_a + n^2 \log n)$.

The space complexity of the algorithm depends on the initial size of the local heaps. The reason for this is that when two clusters are merged, their local heaps are deleted and the size of the new cluster's local heap can be no more than the sum of the sizes of the local heaps of the merged clusters. Since each local heap only contains those clusters to which it has non-zero links, the space complexity of ROCK's clustering algorithm is the same as that of link computation, that is, $O(\min\{n^2, nm_m m_a\})$.

4.6. Miscellaneous Issues

Random Sampling In case the database is large, random sampling enables ROCK to reduce the number of points to be considered and ensures that the input data set fits in main-memory. Consequently, significant improvements in execution times for ROCK can be realized. With an appropriate sample size, the quality of the clustering is not sacrificed. On the contrary, random sampling can aid clustering by filtering outliers. Efficient algorithms for selecting random samples from a database can be found in [13], and we do not discuss them here. Also, an analysis of the appropriate sample size for good quality clustering can be found in [7]. Note that the salient feature of ROCK is not sampling but the clustering algorithm that utilizes links instead of distances.

Handling Outliers In ROCK, outliers can be handled fairly effectively. The first pruning occurs when we choose a value for θ, and by definition outliers are relatively isolated from the rest of the points. This immediately allows us to discard the points with very few or no neighbors because they will never participate in the clustering. This is the most significant part where outliers are eliminated.

However in some situations, outliers may be present as small groups of points that are loosely connected to the rest of the dataset. This immediately suggests to us that these clusters will persist as small clusters for the most part of clustering. These will only participate in clustering once the number of clusters remaining is actually close to the number of clusters in the data. So we stop the clustering at a point such that the number of remaining clusters is a small multiple of the expected number of clusters. We then weed out the clusters that have very little support.

Labeling Data on Disk In the final labeling phase, ROCK assigns the remaining data points residing on disk to the clusters generated using the sampled points. This is performed as follows. First, a fraction of points from each cluster i is obtained; let L_i denote this set of points from cluster i and used for labeling. Then, the original data set is read from disk, and each point p is assigned to the cluster i such that p has the maximum neighbors in L_i (after normalization). In other words, if point p has N_i neighbors in set L_i, then p is assigned to the cluster i for which $\frac{N_i}{(|L_i|+1)^{f(\theta)}}$ is maximum. Note that $(|L_i| + 1)^{f(\theta)}$ is the expected number of neighbors for p in set L_i. Thus, labeling each point p requires at most $\sum_{i=1}^{k} |L_i|$ operations to determine the points in L_i that are neighbors of p.

5. EXPERIMENTAL RESULTS

To get a better feel for how ROCK performs in practice, we ran ROCK on real-life as well as synthetic data sets. We use real-life data sets to compare the quality of clustering due to ROCK with the clusters generated by a traditional centroid-based hierarchical clustering algorithm [4, 9]. The synthetic data sets, on the other hand, are used primarily to demonstrate the scalability properties of ROCK. For ROCK, in all the experiments, we used the similarity function for categorical data (as described in Section 3.1.2), and $f(\theta) = \frac{1-\theta}{1+\theta}$.

In the traditional algorithm, we handle categorical attributes by converting them to boolean attributes with 0/1 values. For every categorical attribute, we define a new attribute for every value in its domain. The new attribute is 1 if and only if the value for the original categorical attribute is equal to the value corresponding to the boolean attribute. Otherwise, it is 0. We use euclidean distance as the distance measure between the centroids of clusters. Also, outlier handling

Data Set	No of Records	No of Attributes	Missing Values	Note
Congressional Votes	435	16	Yes (very few)	168 Republicans and 267 Democrats
Mushroom	8124	22	Yes (very few)	4208 edible and 3916 poisonous
U.S. Mutual Fund	795	548	Yes	Jan 4, 1993 - Mar 3, 1995

Table 1: Data Sets

is performed even in the traditional hierarchical algorithm by eliminating clusters with only one point when the number of clusters reduces to $\frac{1}{3}$ of the original number.

Our experimental results with both real-life as well as synthetic data sets demonstrate the effectiveness of our link-based approach for clustering categorical as well as time-series data. All experiments were performed on a Sun Ultra-2/200 machine with 512 MB of RAM and running Solaris 2.5.

5.1. Real-Life Data Sets

We experimented with three real-life datasets whose characteristics are illustrated in Table 1.

Congressional Votes The Congressional voting data set was obtained from the UCI Machine Learning Repository (http://www.ics.uci.edu/ mlearn/MLRepository.html). It is the United States Congressional Voting Records in 1984. Each record corresponds to one Congress man's votes on 16 issues (e.g., education spending, crime). All attributes are boolean with Yes (that is, 1) and No (that is, 0) values, and very few contain missing values. A classification label of Republican or Democrat is provided with each data record. The data set contains records for 168 Republicans and 267 Democrats.

Mushroom The mushroom data set was also obtained from the UCI Machine Learning Repository. Each data record contains information that describes the physical characteristics (e.g., color, odor, size, shape) of a single mushroom. A record also contains a poisonous or edible label for the mushroom. All attributes are categorical attributes; for instance, the values that the size attribute takes are narrow and broad, while the values of shape can be bell, flat, conical or convex, and odor is one of spicy, almond, foul, fishy, pungent etc. The mushroom database has the largest number of records (that is, 8124) among the real-life data sets we used in our experiments. The number of edible and poisonous mushrooms in the data set are 4208 and 3916, respectively.

US Mutual Funds We ran ROCK on a time-series database of the closing prices of U.S. mutual funds that were collected from the MIT AI Laboratories' Experimental Stock Market Data Server[†] The funds represented in this dataset include bond funds, income funds, asset allocation funds, balanced funds, equity income funds, foreign stock funds, growth stock funds, aggressive growth stock funds and small company growth funds. The closing prices for each fund are for business dates only. Some of the mutual funds that were launched later than Jan 4, 1993 do not have a price for the entire range of dates from Jan 4, 1993 until Mar 3, 1995. Thus, there are many missing values for a certain number of mutual funds in our data set.

While the congressional voting and mushroom data sets have categorical attributes, the mutual fund data set has real values that correspond to the closing prices for each date. We adopt the following approach in order to generate the similarity value for an arbitrary pair of mutual funds. For a mutual fund, we map the real values for each date to one of three categorical values, Up, Down and No, based on changes to its closing price compared to the previous business date. As the names for the categorical values suggest, Up, Down and No correspond to a positive, negative and no change to the price relative to the previous price. The similarity function is then defined for the categorical data as described in Section 3.1.2.

[†]The web server does not exist any more. However, the data we used was collected a few years ago for [1] and [12] when the web site was available at http://www.ai.mit.edu/stocks/mf.html.

Traditional Hierarchical Clustering Algorithm		
Cluster No	No of Republicans	No of Democrats
1	157	52
2	11	215

ROCK		
Cluster No	No of Republicans	No of Democrats
1	144	22
2	5	201

Table 2: Clustering Result for Congressional Voting Data

Note that it is possible to employ alternate techniques from the one we have outlined above for the similarity function. For example, in [1, 12], the authors propose methods for generating a similarity or dissimilarity value for a pair of time-series sequences based on a very general similarity model with amplitude scaling, outlier removal and translation. The similarity values produced by their technique can be directly used in ROCK to determine neighbors and links for the various time-series sequences. For the purpose of our experiments in this paper, however, we do not use the similarity model from [1][†], but instead use our simpler model that transforms real values to Up, Down and No. Our experimental results indicate that even with our simpler similarity model, we find some very interesting clusters.

5.2. Results with Real-Life Data Sets

Congressional Votes Table 2 contains the results of running on congressional voting data, the centroid-based hierarchical algorithm and ROCK with θ set to 0.73. As the table illustrates, ROCK and the traditional algorithm, both identify two clusters one containing a large number of republicans and the other containing a majority of democrats. However, in the cluster for republicans found by the traditional algorithm, around 25% of the members are democrats, while with ROCK, only 12% are democrats. The improvement in the quality of clustering can be attributed to both our outlier removal scheme as well as the usage of links by ROCK. Note that due to the elimination of outliers, the sum of the sizes of our clusters does not equal to the size of the input data set.

The frequent values of categorical attributes for the two clusters are described in Table 7 in the appendix. We found that only on 3 issues did a majority of Republicans and Democrats cast the same vote. However, on 12 of the remaining 13 issues, the majority of the Democrats voted differently from the majority of the Republicans. Furthermore, on each of the 12 issues, the Yes/No vote had sizable support in their respective clusters. Since on majority of the attributes the records in each cluster have similar values that are different from the values for the attributes in the other cluster, we can consider the two clusters to be well-separated. Furthermore, there isn't a significant difference in the sizes of the two clusters. These two characteristics of the voting data set allow the traditional algorithm to discover the clusters easily.

Mushroom Table 3 describes the result of clustering the mushroom database using the traditional algorithm and ROCK. We set the number of desired clusters for both algorithms to be 20. We set θ for ROCK to be 0.8. ROCK found 21 clusters instead of 20 – no pair of clusters among the 21 clusters had links between them and so ROCK could not proceed further. As the results in the table indicate, all except one (Cluster 15) of the clusters discovered by ROCK are pure clusters in the sense that mushrooms in every cluster were either all poisonous or all edible. Furthermore, there is a wide variance among the sizes of the clusters – 3 clusters have sizes above 1000 while 9 of the 21 clusters have a size less than 100. Furthermore, the sizes of the largest and smallest cluster are 1728 and 8, respectively. We also generated the characteristics of the clusters shown in Table 3, but due to lack of space, we show the characteristics of only the five largest clusters among them in tables 8 and 9 in the appendix. We found that, in general, records in different clusters could be identical with respect to some attribute values. Thus, every pair of clusters generally have

[†]The reason is that we did not have access to the code for generating the similarity values.

Traditional Hierarchical Algorithm					
Cluster No	No of Edible	No of Poisonous	Cluster No	No of Edible	No of Poisonous
1	666	478	11	120	144
2	283	318	12	128	140
3	201	188	13	144	163
4	164	227	14	198	163
5	194	125	15	131	211
6	207	150	16	201	156
7	233	238	17	151	140
8	181	139	18	190	122
9	135	78	19	175	150
10	172	217	20	168	206

ROCK					
Cluster No	No of Edible	No of Poisonous	Cluster No	No of Edible	No of Poisonous
1	96	0	12	48	0
2	0	256	13	0	288
3	704	0	14	192	0
4	96	0	15	32	72
5	768	0	16	0	1728
6	0	192	17	288	0
7	1728	0	18	0	8
8	0	32	19	192	0
9	0	1296	20	16	0
10	0	8	21	0	36
11	48	0			

Table 3: Clustering Result for Mushroom Data

some common values for the attributes and thus clusters are not well-separated. An interesting exception was the odor attribute which had values none, anise or almond for edible mushrooms, while for poisonous mushrooms, the values for the odor attribute were either foul, fishy or spicy.

As expected, the quality of the clusters generated by the traditional algorithm were very poor. This is because clusters are not well-separated and there is a wide variance in the sizes of clusters. As a result, with the traditional algorithm, we observed that cluster centers tend to spread out in all the attribute values and lose information about points in the cluster that they represent. Thus, as discussed earlier in Section 1, distances between centroids of clusters become a poor estimate of the similarity between them.

As shown in Table 3, points belonging to different clusters are merged into a single cluster and large clusters are split into smaller ones. None of the clusters generated by the traditional algorithm are pure. Also, every cluster contains a sizable number of both poisonous and edible mushrooms. Furthermore, the sizes of clusters detected by traditional hierarchical clustering are fairly uniform. More than 90% of the clusters have sizes between 200 and 400, and only 1 cluster has more than 1000 mushrooms. This confirms that our notion of links finds more meaningful clusters for mushroom data.

US Mutual Funds For the mutual funds data set, we could not run the traditional algorithm because the sizes of records vary significantly. The reason for this is that a number of young mutual funds started after Jan 4, 1993 and as a result, a large number of values for them are missing from the data set. This makes it difficult to use the traditional algorithm since it is unclear as to how to treat the missing values in the context of traditional hierarchical clustering.

The result of ROCK with $\theta = 0.8$ is presented in Table 4. The mutual fund data set is not very amenable to clustering and contains a number of outliers, that is, clusters with only a single mutual fund. The reason for this is that even though some funds are in the same group, they sometimes do not perform similarly because the fund managers maintain different portfolios. Despite this, ROCK found 24 clusters of size 2 (that is, containing exactly two mutual funds) that we do not present here due to lack of space. However, we do want to point out that these clusters with size two were also very interesting. For example, one of the clusters with size 2 contained Harbor International Fund and Ivy International Fund. This is because even though the two mutual funds

Cluster Name	Number of Funds	Ticker Symbol	Note
Bonds 1	4	BTFTX BTFIX BTTTX BTMTX	Coupon Bonds
Bonds 2	10	CPTNX FRGVX VWESX FGOVX PRCIX et al	–
Bonds 3	24	FMUIX SCTFX PRXCX PRFHX VLHYX et. al	Municipal Bonds
Bonds 4	15	FTFIX FRHIX PHTBX FHIGX FMBDX et. al	Municipal Bonds
Bonds 5	5	USGNX SGNMX VFITX OPIGX PHGBX	–
Bonds 6	3	VFLTX SWCAX FFLIX	Municipal Bonds
Bonds 7	26	WPGVX DRBDX VUSTX SGZTX PRULX et. al	Income
Financial Service	3	FIDSX FSFSX FSRBX	–
Precious Metals	10	FDPMX LEXMX VGPMX STIVX USERX et.al	Gold
International 1	4	FSIGX INIFX PRFEX USIFX	–
International 2	4	PRASX FSEAX SCOPX	Asia
International 3	6	TEMWX TEPLX TEMFX ANWPX AEPGX	–
Balanced	5	RPBAX SCBAX PREIX VTSMX OPVSX	–
Growth 1	8	DTSGX AVLFX PESPX PNOPX ACEMX et al.	–
Growth 2	107	STCSX SOPFX USAUX PBHGX VEIPX et al.	–
Growth 3	70	VWNDX SLASX ANEFX FKGRX FISEX et al.	–

Table 4: Mutual Funds Clusters

belong to different mutual fund companies, the portfolio managers of both funds were the same for the period of the data set. Interestingly, the same two mutual funds were also found to be similar using techniques to detect similar time sequences in [1]. This indicates to us that even our simple transformation of the mutual fund data to a form containing Up, Down and No when input to ROCK is capable of generating some interesting results that were obtained using very powerful and general tools for data mining. Some of the other interesting clusters of size 2 included a pair of Japan funds, a pair of European funds, a pair that invest in Energy and two that invested in Emerging Markets.

In Table 4, we present the 16 clusters whose size exceeded 3. For each cluster, the first column contains the name for the cluster that is based on the group of funds that the cluster belongs to, and column 3 contains the ticker symbols for the first few funds in the cluster. In the final column, whenever possible, we assign a category to each cluster based on the types of investments made by the the funds in the cluster (e.g., Bonds, Gold, Asia). The Financial Service cluster has 3 funds – Fidelity Select Financial Services (FIDSX), Invesco Strategic Financial Services (FSFSX) and Fidelity Select Regional Banks (FSRBX) that invest primarily in banks, brokerages and financial institutions. The cluster named International 2 contains funds that invest in South-east Asia and the Pacific rim region; they are T. Rowe Price New Asia (PRASX), Fidelity Southeast Asia (FSEAX), and Scudder Pacific Opportunities (SCOPX). The Precious Metals cluster includes mutual funds that invest mainly in Gold.

Thus, our results with the mutual funds data goes on to prove that ROCK can also be used to cluster time-series data. In addition, it can be employed to determine interesting distributions in the underlying data even when there are a large number of outliers that do not belong to any of the clusters, as well as when the data contains a sizable number of missing values. Furthermore, a nice and desirable characteristic of our technique is that it does not merge a pair of clusters if there are no links between them. Thus, the desired number of clusters input to ROCK is just a hint – ROCK may discover more than the specified number of clusters (if there are no links between clusters) or fewer (in case certain clusters are determined to be outliers and eliminated).

5.3. Synthetic Data Set

The synthetic data set is a market basket database containing 114586 transactions. Of these, 5456 (or roughly, 5%) are outliers, while the others belong to one of 10 clusters with sizes varying between 5000 and 15000 (see Table 5). Each cluster is defined by a set of items – the number of items that define each cluster is as shown in the last row of Table 5. Roughly 40% of the items that define a cluster are common with items for other clusters, the remaining 60% being exclusive to the cluster. A transaction for a cluster is generated by randomly selecting items from the set of items that define the cluster (outliers are generated by randomly selecting from among the items

Cluster No.	1	2	3	4	5	6	7	8	9	10	Outliers
No. of Transactions	9736	13029	14832	10893	13022	7391	8564	11973	14279	5411	5456
No. of Items	19	20	19	19	22	19	19	21	22	19	116

Table 5: Synthetic Data Set

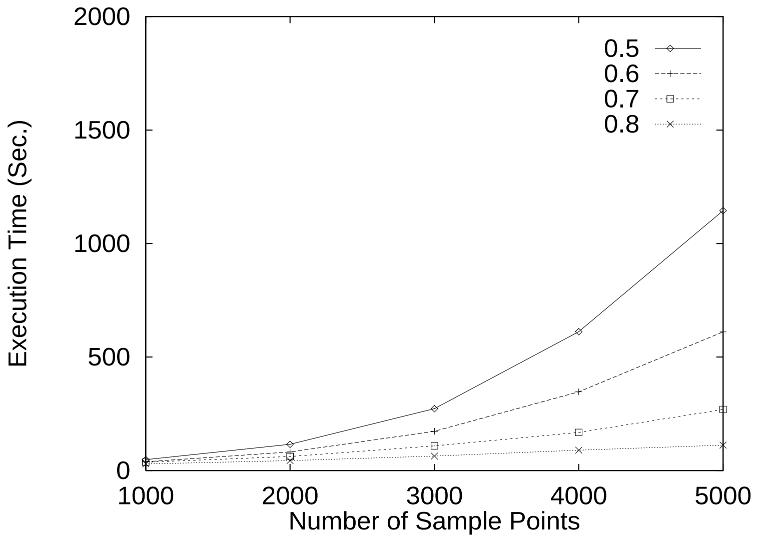

Fig. 5: Scalability of ROCK with Respect to Size of Random Sample

for all the clusters). The transaction size parameter has a normal distribution with an average value of 15. Due to the normal distribution, 98% of transactions have sizes between 11 and 19.

5.4. Results with Synthetic Data Set

Scalability for Large Databases Since we use the combination of random sampling and labeling to handle large data sets, the size of the database has minimal impact on the execution time of ROCK. However, the random sample size has a significant effect on ROCK's performance. Thus, in our scale-up experiment, we study the effects of the random sample size on running time. In our running times, we do not include the time for the final labeling phase.

Figure 5 plots ROCK's execution time on the synthetic data set as the random sample size is varied for four different settings of θ. The graph illustrates that the computational complexity of ROCK is roughly quadratic with respect to the sample size. Furthermore, for a given sample size, the performance of ROCK improves as θ is increased. The reason for this is that as θ is increased, each transaction has fewer neighbors and this makes the computation of links more efficient.

Quality of Clustering Since for each transaction in our synthetic data set, we know the cluster to which it belongs, we can easily compute the number of transactions misclassified in a clustering and use this as an assessment of its quality. Table 6 describes the number of transactions misclassified by ROCK for our synthetic data set with θ values of 0.5 and 0.6 and a range of sample sizes. As the table illustrates, ROCK with random sampling finds the original clusters very accurately when θ is either 0.5 or 0.6. The table also shows that the quality of clustering improves as the random sample size increases.

Note that the quality of clustering is better with $\theta = 0.5$ than with $\theta = 0.6$. The main reason for this is that the random sample sizes we consider range from being less than 1% of the database size to about 4.5%. In addition, transaction sizes can be as small as 11, while the number of items defining each cluster is approximately 20. Finally, a high percentage (roughly 40%) of items in a cluster are also present in other clusters. Thus, a smaller similarity threshold is required to ensure that a larger number of transaction pairs from the same cluster are neighbors.

Sample Size	1000	2000	3000	4000	5000
$\theta = 0.5$	37	0	0	0	0
$\theta = 0.6$	8123	1051	384	104	8

Table 6: Number of Misclassified Transactions

6. CONCLUDING REMARKS

In this paper, we proposed a new concept of links to measure the similarity/proximity between a pair of data points with categorical attributes. We also developed a robust hierarchical clustering algorithm ROCK that employs links and not distances for merging clusters. Our methods naturally extend to *non-metric* similarity measures that are relevant in situations where a domain expert/similarity table is the only source of knowledge.

The results of our experimental study with real-life data sets are very encouraging. For example, with the mushroom data set, ROCK discovered almost pure clusters containing either only edible or only poisonous mushrooms. Furthermore, there were significant differences in the sizes of the clusters found. In contrast, the quality of clusters found by the traditional centroid-based hierarchical algorithm was very poor. Not only did it generate uniform sized clusters, but also most clusters contained a sizable number of both edible and poisonous mushrooms. With the mutual funds data set, we could find, using our link-based approach, groups of mutual funds that have similar performance. This demonstrates the utility of ROCK as a tool for also clustering time-series data. Finally, for our large synthetic data set, we found that the combination of random sampling and labeling enables ROCK's performance to scale quite well for large databases.

Acknowledgements — We would like to thank Narain Gehani, Hank Korth, Rajeev Motwani and Avi Silberschatz for their encouragement. Without the support of Yesook Shim, it would have been impossible to complete this work. The work of Kyuseok Shim was partially supported by the Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Center (AITrc).

REFERENCES

[1] R. Agrawal, K.-I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the VLDB Conference*, Zürich, Switzerland, pp. 490–501 (1995).

[2] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing* (1987).

[3] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to Algorithms.* The MIT Press, Massachusetts (1990).

[4] R.O. Duda and P. E. Hard. *Pattern Classification and Scene Analysis.* A Wiley-Interscience Publication, New York (1973).

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial database with noise. In *International Conference on Knowledge Discovery in Databases and Data Mining (KDD-96)*, Portland, Oregon, pp. 226–231, AAAI Press (1996).

[6] M. Ester, H.-P. Kriegel, and X. Xu. A database interface for clustering in large spatial databases. In *International Conference on Knowledge Discovery in Databases and Data Mining (KDD-95)*, Montreal, Canada, pp. 94–99, AAAI Press (1995).

[7] S. Guha, R. Rastogi, and K. Shim. CURE: A clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 73–84 (1998).

[8] E.-H. Han, G. Karypis, V. Kumar, and B. Mobasher. Clustering based on association rule hypergraphs. In *1997 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, pp. 9–13 (1997).

[9] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data.* Prentice Hall, Englewood Cliffs, New Jersey (1988).

[10] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar. Multilevel hypergraph partitioning: application in VLSI domain. In *Proceedings of the ACM/IEEE Design Automation Conference*, Montreal, Canada (1997).

[11] R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the VLDB Conference*, Santiago, Chile, pp. 144–155 (1994).

[12] K. Shim, R. Srikant, and R. Agrawal. High-dimensional similarity joins. In *IEEE 13th International Conference on Data Engineering*, Birmingham, UK, pp. 301–311 (1997).

[13] J. Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software*, **11**(1):37–57 (1985).

[14] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, pp. 103–114 (1996).

APPENDIX A: CHARACTERISTICS OF CLUSTERS FOR REAL-LIFE DATA SETS

Cluster 1 (Republicans)	Cluster 2 (Democrats)
(immigration,y,0.51)	(immigration,y,0.51)
(export-administration-act-south-africa,y,0.55)	(export-administration-act-south-africa,y,0.7)
(synfuels-corporation-cutback,n,0.77)	(synfuels-corporation-cutback,n,0.56)
(adoption-of-the-budget-resolution,n,0.87)	(adoption-of-the-budget-resolution,y,0.94)
(physician-fee-freeze,y,0.92)	(physician-fee-freeze,n,0.96)
(el-salvador-aid,y,0.99)	(el-salvador-aid,n,0.92)
(religious-groups-in-schools,y,0.93)	(religious-groups-in-schools,n,0.67)
(anti-satellite-test-ban,n,0.84)	(anti-satellite-test-ban,y,0.89)
(aid-to-nicaraguan-contras,n,0.9)	(aid-to-nicaraguan-contras,y,0.97)
(mx-missile,n,0.93)	(mx-missile,y,0.86)
(education-spending,y,0.86)	(education-spending,n,0.9)
(crime,y,0.98)	(crime,n,0.73)
(duty-free-exports,n,0.89)	(duty-free-exports,y,0.68)
(handicapped-infants,n,0.85)	(handicapped-infants,y,0.65)
(superfund-right-to-sue,y,0.9)	(superfund-right-to-sue,n,0.79)
(water-project-cost-sharing,y,0.51)	

Table 7: Characteristics of Clusters in Congressional Voting Data

Cluster 3
(cap-shape,convex,0.5) (cap-shape,bell,0.36) (cap-shape,flat,0.14)
(cap-surface,smooth,0.36) (cap-surface,scaly,0.64)
(cap-color,brown,0.14) (cap-color,white,0.36) (cap-color,yellow,0.5)
(bruises,bruises,1) (odor,anise,0.5) (odor,almond,0.5)
(gill-attachment,free,1) (gill-spacing,close,1) (gill-size,broad,1)
(gill-color,black,0.18) (gill-color,brown,0.27) (gill-color,gray,0.18)
(gill-color,pink,0.091) (gill-color,white,0.27)
(stalk-shape,enlarging,1) (stalk-root,rooted,0.27) (stalk-root,club,0.73)
(stalk-surface-above-ring,smooth,1)
(stalk-surface-below-ring,smooth,0.73) (stalk-surface-below-ring,scaly,0.27)
(stalk-color-above-ring,white,1) (stalk-color-below-ring,white,1)
(veil-type,partial,1) (veil-color,white,1)
(ring-number,one,1) (ring-type,pendant,1)
(spore-print-color,black,0.5) (spore-print-color,brown,0.5)
(population,scattered,0.5) (population,numerous,0.36) (population,solitary,0.14)
(habitat,grasses,0.5) (habitat,meadows,0.36) (habitat,paths,0.14)

Cluster 5
(cap-shape,convex,0.5) (cap-shape,flat,0.5)
(cap-surface,smooth,0.5) (cap-surface,fibrous,0.5)
(cap-color,white,0.33) (cap-color,brown,0.33) (cap-color,gray,0.33)
(bruises,no,1) (odor,none,1)
(gill-attachment,free,1) (gill-spacing,crowded,1) (gill-size,broad,1)
(gill-color,black,0.25) (gill-color,brown,0.25) (gill-color,pink,0.25) (gill-color,chocolate,0.25)
(stalk-shape,tapering,1) (stalk-root,equal,1)
(stalk-surface-above-ring,smooth,0.5) (stalk-surface-above-ring,ibrous,0.5)
(stalk-surface-below-ring,ibrous,0.5) (stalk-surface-below-ring,smooth,0.5)
(stalk-color-above-ring,white,1) (stalk-color-below-ring,white,1)
(veil-type,partial,1) (veil-color,white,1)
(ring-number,one,1) (ring-type,evanescent,1)
(spore-print-color,black,0.5) (spore-print-color,brown,0.5)
(population,scattered,0.5) (population,abundant,0.5)
(habitat,grasses,1)

Cluster 7
(cap-shape,convex,0.5) (cap-shape,flat,0.5)
(cap-surface,fibrous,0.5) (cap-surface,scaly,0.5)
(cap-color,brown,0.33) (cap-color,red,0.33) (cap-color,gray,0.33)
(bruises,bruises,1) (odor,none,1)
(gill-attachment,free,1) (gill-spacing,close,1) (gill-size,broad,1)
(gill-color,brown,0.25) (gill-color,pink,0.25) (gill-color,white,0.25) (gill-color,purple,0.25)
(stalk-shape,tapering,1) (stalk-root,bulbous,1)
(stalk-surface-above-ring,smooth,1)
(stalk-surface-below-ring,smooth,1)
(stalk-color-above-ring,gray,0.33) (stalk-color-above-ring,pink,0.33) (stalk-color-above-ring,white,0.33)
(stalk-color-below-ring,pink,0.33) (stalk-color-below-ring,gray,0.33) (stalk-color-below-ring,white,0.33)
(veil-type,partial,1) (veil-color,white,1)
(ring-number,one,1) (ring-type,pendant,1)
(spore-print-color,black,0.5) (spore-print-color,brown,0.5)
(population,solitary,0.5) (population,several,0.5)
(habitat,woods,1)

Table 8: Characteristics of Large Clusters in Mushroom Data: Edible

Cluster 9
(cap-shape,convex,0.5) (cap-shape,flat,0.5)
(cap-surface,scaly,0.5) (cap-surface,fibrous,0.5)
(cap-color,yellow,0.5) (cap-color,gray,0.5)
(bruises,no,1) (odor,foul,1)
(gill-attachment,free,1) (gill-spacing,close,1) (gill-size,broad,1)
(gill-color,gray,0.33) (gill-color,pink,0.33) (gill-color,chocolate,0.33)
(stalk-shape,enlarging,1) (stalk-root,bulbous,1)
(stalk-surface-above-ring,silky,1) (stalk-surface-below-ring,silky,1)
(stalk-color-above-ring,pink,0.33) (stalk-color-above-ring,brown,0.33) (stalk-color-above-ring,buff,0.33)
(stalk-color-below-ring,pink,0.33) (stalk-color-below-ring,brown,0.33) (stalk-color-below-ring,buff,0.33)
(veil-type,partial,1) (veil-color,white,1)
(ring-number,one,1) (ring-type,large,1)
(spore-print-color,chocolate,1)
(population,several,0.5) (population,solitary,0.5)
(habitat,grasses,0.33) (habitat,woods,0.33) (habitat,paths,0.33)

Cluster 16
(cap-shape,convex,0.33) (cap-shape,flat,0.33) (cap-shape,knobbed,0.33)
(cap-surface,smooth,0.5) (cap-surface,scaly,0.5)
(cap-color,brown,0.5) (cap-color,red,0.5)
(bruises,no,1) (odor,fishy,0.3) (odor,foul,0.33) (odor,spicy,0.33)
(gill-attachment,free,1) (gill-spacing,close,1) (gill-size,narrow,1)
(gill-color,buff,1)
(stalk-shape,tapering,1)
(stalk-surface-above-ring,smooth,0.5) (stalk-surface-above-ring,silky,0.5)
(stalk-surface-below-ring,smooth,0.5) (stalk-surface-below-ring,silky,0.5)
(stalk-color-above-ring,white,0.5) (stalk-color-above-ring,pink,0.5)
(stalk-color-below-ring,white,0.5) (stalk-color-below-ring,pink,0.5)
(veil-type,partial,1) (veil-color,white,1)
(ring-number,one,1) (ring-type,evanescent,1)
(spore-print-color,white,1)
(population,several,1)
(habitat,woods,0.33) (habitat,paths,0.33) (habitat,leaves,0.33)

Table 9: Characteristics of Large Clusters in Mushroom Data: Poisonous