ROCK: A Robust Clustering Algorithm for Categorical Attributes*

Sudipto Guha†
Stanford University
Stanford, CA 94305
sudipto@cs.stanford.edu

Rajeev Rastogi
Bell Laboratories
Murray Hill, NJ 07974
rastogi@research.bell-labs.com

Kyuseok Shim
Bell Laboratories
Murray Hill, NJ 07974
shim@research.bell-labs.com

Abstract

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. Our study shows that ROCK not only generates better quality clusters than traditional algorithms, but also exhibits good scalability properties.*

1 Introduction

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. 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.

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 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

Clustering algorithms developed in the literature can be classified into *partitional clustering* and *hierarchical clustering* [DH73, JD88]. 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

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

*This work is part of the Serendip data mining project at Bell Labs. URL: http://www.bell-labs.com/project/serendip/.

†The work was done while the author was visiting Bell Laboratories.

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[1] 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 [DH73, JD88]. 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 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 1.1: 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. A detailed description with an example of the ripple effect in centroid-based hierarchical algorithms is provided in [GRS97].

Set theoretic similarity measures such as the *Jaccard coefficient*[2] [JD88] have often been used, instead of euclidean distance, for document clustering. With the Jaccard coefficient as the distance measure between

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

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

<1, 2, 3, 4, 5>		<1, 2, 6, 7>
{1, 2, 3}	{1, 4, 5}	{1, 2, 6}
{1, 2, 4}	{2, 3, 4}	{1, 2, 7}
{1, 2, 5}	{2, 3, 5}	{1, 6, 7}
{1, 3, 4}	{2, 4, 5}	{2, 6, 7}
{1, 3, 5}	{3, 4, 5}	

Figure 1: Basket data example for Jaccard coefficient

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* [JD88]. 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 1.2: 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 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 example, group average may merge all transactions with 1 and 2 into the same cluster as illustrated in [GRS97]. ∎

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[3] 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 1.1 and 1.2. 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 1.1, 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. Similarly, in Example 1.2, we can show that links are more appropriate than distances for clustering categorical data sets (See Section 3.2).

From the above examples, it follows that unlike distances or similarities between a pair of points which are

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

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. 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 present an agglomerative hierarchical clustering algorithm ROCK (RObust Clustering using linKs) that is based on our notion of links. With real-life 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.

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 [DH73, JD88]. More recently, clustering algorithms for mining large databases have been proposed in [NH94, ZRL96, EKSX96, GRS98]. Most of these, however, are variants of either partitional (e.g., [NH94]) or centroid-based hierarchical clustering (e.g., [ZRL96, GRS98]). 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.

Recently, in [HKKM97], 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 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.

The rationale for using item clusters to cluster transactions is questionable. For example, the approach in [HKKM97] 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 1.2. 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. 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, we present possible definitions for sim for market basket databases and for data sets with categorical attributes.

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 [DH73], 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.

Categorical Data: Data sets with categorical attributes can be handled in a manner similar to how we handled market basket data previously. 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.

3.2 Links

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.

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

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. Assuming C_i has approximately $n_i^{f(\theta)}$ neighbors in C_i, we can show that $n_i^{1+2f(\theta)}$ is the expected number of links between

pairs of points in C_i. The details of this derivation of the expected number can be found in [GRS97]. 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.

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.

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. 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.

```
procedure cluster(S, k)
begin
1.  link := compute_links(S)
2.  for each s ∈ 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 ∈ q[u] ∪ 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
```

Figure 3: Clustering Algorithm

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. The expected number of *cross* links or links between pairs of points each from a different cluster can be shown to be $(n_i + n_j)^{1+2f(\theta)} - n_i^{1+2f(\theta)} - n_j^{1+2f(\theta)}$. (See [GRS97] for details.) 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,

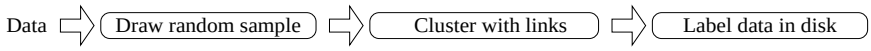

Figure 2: Overview of ROCK

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 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. A detailed description of how this for-loop works appears in [GRS97].

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 [CLR90] 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 [CW87].

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

```
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
```

Figure 4: Algorithm for 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.

4.5 Time and Space Complexity

ROCK's clustering algorithm, along with the computation of neighbor lists and links, has a worst-case time complexity of $O(n^2 + nm_m m_a + n^2 \log n)$ where m_m is the maximum number of neighbors, m_a is the average number of neighbors and n is the number of input data points. The space complexity of ROCK's clustering algorithm can also be shown to be $O(\min\{n^2, nm_m m_a\})$. Due to the lack of space, a detailed complexity analysis of ROCK is presented in

[GRS97].

4.6 Miscellaneous Issues

Due to the lack of space, we only cover the issue of labeling data on disk here. Other issues such as random sampling and handling outliers are discussed in [GRS97].

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. However, due to the lack of space, we present only a part of results with real-life data sets in this section. The results of other experiments related to scalability with synthetic data sets appear in [GRS97]. 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 [DH73, JD88]. For ROCK, in all the experiments, we used the similarity function for categorical data (as described in Section 3.1), 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 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.

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

5.1 Real-life Data Sets

We experimented with three real-life datasets whose characteristics are illustrated in Table 1. However, due to the lack of space, we do not present the description of the US mutual fund data set and experimental results for it in this paper. They can be found in [GRS97].

Congressional votes: The Congressional voting data set was obtained from the UCI Machine Learning Repository[4]. 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.

5.2 Results with Real-life Data Sets

Congressional Votes: Table 2 contains the results of running on congressional voting data, the centroid-

[4] It is at http://www.ics.uci.edu/ mlearn/MLRepository.html.

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

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 can be found in [GRS97]. 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 do not show them here. They can be found in [GRS97]. 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 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.

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

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

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.

Acknowledgments: We would like to thank Narain Gehani, Hank Korth and Avi Silberschatz for their encouragement. Without the support of Yesook Shim, it would have been impossible to complete this work.

References

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Massachusetts, 1990.

[CW87] Donald Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. In *Proc. of the 19th Annual ACM Symposium on Theory of Computing*, 1987.

[DH73] Richard O. Duda and Peter E. Hard. *Pattern Classification and Scene Analysis*. A Wiley-Interscience Publication, New York, 1973.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial database with noise. In *Int'l Conference on Knowledge Discovery in Databases and Data Mining (KDD-96)*, Portland, Oregon, August 1996.

[GRS97] Sudipto Guha, R. Rastogi, and K. Shim. Clustering algorithm for categorical attributes. Technical report, Bell Laboratories, Murray Hill, 1997.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: A clustering algorithm for large databases. In *Proc. of the ACM SIGMOD Conference on Management of Data*, May 1998.

[HKKM97] Eui-Hong Han, George Karypis, Vipin Kumar, and Bamshad Mobasher. Clustering based on association rule hypergraphs. In *1997 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, June 1997.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and effective clustering methods for spatial data mining. In *Proc. of the VLDB Conference*, Santiago, Chile, September 1994.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Montreal, Canada, June 1996.