A Cohesion-based Clustering Technique for Categorical Data

Aida Nemalhabib

A Thesis

In

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science

Concordia University
Montreal, Quebec, Canada.

February 2006

Your file Votre référence
ISBN: 0-494-14331-2
Our file Notre référence
ISBN: 0-494-14331-2

ABSTRACT

A Cohesion-based Clustering Technique for Categorical Data

Aida Nemalhabib

Clustering is a technique which aims to partition a given dataset of objects into groups of similar objects. In this work, we consider categorical data, which are unordered unlike numerical data. This makes clustering such data a more challenging task. We propose a clustering technique for categorical data, which uses a novel similarity function, called *cohesion,* to measure the degree to which objects "stick" to clusters. We have implemented this technique, to which we refer as CLUC (**CLU**stering with **C**ohesion). To evaluate CLUC, we compared its results with those produced by well-known clustering algorithms. The results of our extensive experiments on real and synthetic datasets show that CLUC generates high quality clusters which conform better to clusterings by human experts. For some well-known real datasets, CLUC even discovers clusterings identical to those provided by experts. Our results also indicate that CLUC is order insensitive in general and is scalable when the dataset grows in size (the number of objects) and/or dimensions (attributes).

ACKNOWLEDGEMENTS

I wish to express my gratitude and appreciation to my supervisor, Dr. Nematollaah Shiri who provided guidance and support throughout this research project. Special thanks go to colleagues in the database lab at Concordia University for sharing time and thought, especially A. Kiani for many helpful discussions. I acknowledge O. El Demerdash for his valuable guidance and supports during this research. I would also like to thank Halina Monkiewicz, Graduate Program Advisor and Hirut Adugna, Secretary of the department, who were very kind and helpful during my study.

Finally, I would like to give my special thanks to my parents for their endless love and encouragement. Without their support, I could not reach this far.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 Introduction

We begin this chapter by an introduction to clustering, which is a central problem in data mining, statistics, and machine learning. We then motivate our work and list our contributions. We close this chapter by an outline of this thesis report.

1.1 Data Clustering

We are living in the era of information and technology. Everyday the volume of data used in scientific, engineering, and commercial applications grows enormously. For many years in the past, stored data have been processed and analyzed in order to get some quantitative statistical information such as mean, correlation, regression, etc. It was recognized that there are other "hidden" patterns and knowledge in such data collections, which could not be discovered by traditional statistical techniques. These patterns are sometimes much more valuable than the data itself for decision support, medical diagnosis, business management, etc. To discover patterns in datasets, new techniques are required to be devised. Data mining refers to discovering knowledge from databases and aims to extract novel and potentially useful information from data. Experts in different research areas such as data mining and machine learning have developed numerous mining tools and techniques for this purpose.

One of the remarkable mining techniques is clustering which aims to discover distribution of patterns in a dataset. A clustering algorithm rearranges a set of objects in the input dataset into groups, called clusters, such that objects in each cluster are similar to each other and are dissimilar to objects in other clusters. Clustering can be considered

as a data-modeling problem since its focus is on finding meaningful patterns and relationships among objects in a dataset. Clustering is also referred to as unsupervised learning task as it groups unlabeled data with some or no prior information about the dataset and predefined classes.

Clustering has a wide range of applications. For example, in an insurance company, clustering can be used to group similar customers according to their income, age, types of policies purchased, and claim reports. Biologists use clustering algorithms to classify plants, animals, etc., according to their various characteristics or to partition genes according to their protein functionality [28]. In medicine, useful taxonomies are obtained by clustering diseases and their symptoms. Web analysis [27], information retrieval [18], text mining [23], image segmentation [15], speech and character recognition [6] are some other examples of applications of clustering.

1.2 Motivation

Clustering is a process which generates groups of similar objects. Selecting an appropriate distance or similarity measure is a vital issue in clustering algorithms. Most well-known distance and similarity functions are proposed and applied to numerical spaces in which objects are described by numerical attributes. Hence, most existing clustering algorithms in the literature are developed to work on numerical datasets. In contrast to numerical datasets, categorical datasets only include attributes with nominal values, which are unordered. For these datasets, defining a suitable distance/similarity function is a much trickier task, and accordingly, clustering categorical datasets is more

challenging than numerical data. Table 1.1 illustrates a sample categorical data in which the objects are described by their color and size.

Table 1.1 Sample categorical dataset

Object1 = { Color : blue, size : large }
Object2 = { Color : red, size : medium }
Object3 = { Color :yellow, size : small }
Object4 = { Color : black, size : Xsmall }
Object5 = { Color : white, size : Xlarge }
Object6 = { Color : green, size : XXlarge }

Almost every clustering algorithm works with some user-defined input parameters or values. The quality of a clustering produced depends very much on these input values, and hence, selecting suitable input values is crucial for effectiveness of the algorithms. In some clustering algorithms, the number k of clusters is one of these input parameters which should be provided by the user. Finding an appropriate value for k in itself is a non-trivial task for the following reasons:

1. Selecting a small value for k may cause losing some information about the data.
2. Partitioning data into many clusters may make the model less comprehensible.
3. Sometimes, selecting a value for k may generate unsound results, i.e., it may not correspond to the knowledge of a human expert or user.

For example, in the clustering shown in Figure 1.1, all the objects are identical. Clustering them in more than one group does not make sense, i.e., according to the definition of clustering, objects in each cluster should be similar to each other and dissimilar to objects in other groups.

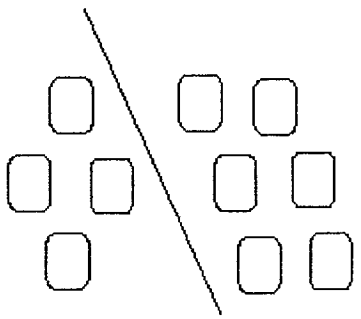

Figure 1.1 Example of an unsound clustering

As another example, consider the dataset shown in Table 1.1. In this table, each object has characteristics (attribute values) which are different from the other objects, and they are all distinct. Therefore, to have a "quality" clustering, each object should represent a separate cluster, which means the correct value for k in this case is 6, and any other value would result in unsound clusters.

Some clustering algorithms may require other input parameters. In most cases, users do not know "good" values of these parameters. This could have serious impacts on a clustering result. For example, users may just know that the input value should be a positive integer but they may have no idea about the range of this value. Therefore, it is desirable to develop a clustering algorithm in which the input parameters are meaningful to users and easy to set.

We remark that clustering is a subjective task, i.e., the same dataset can be clustered in different ways depending on the user's point of view. Consider the objects shown in Figure 1.2.

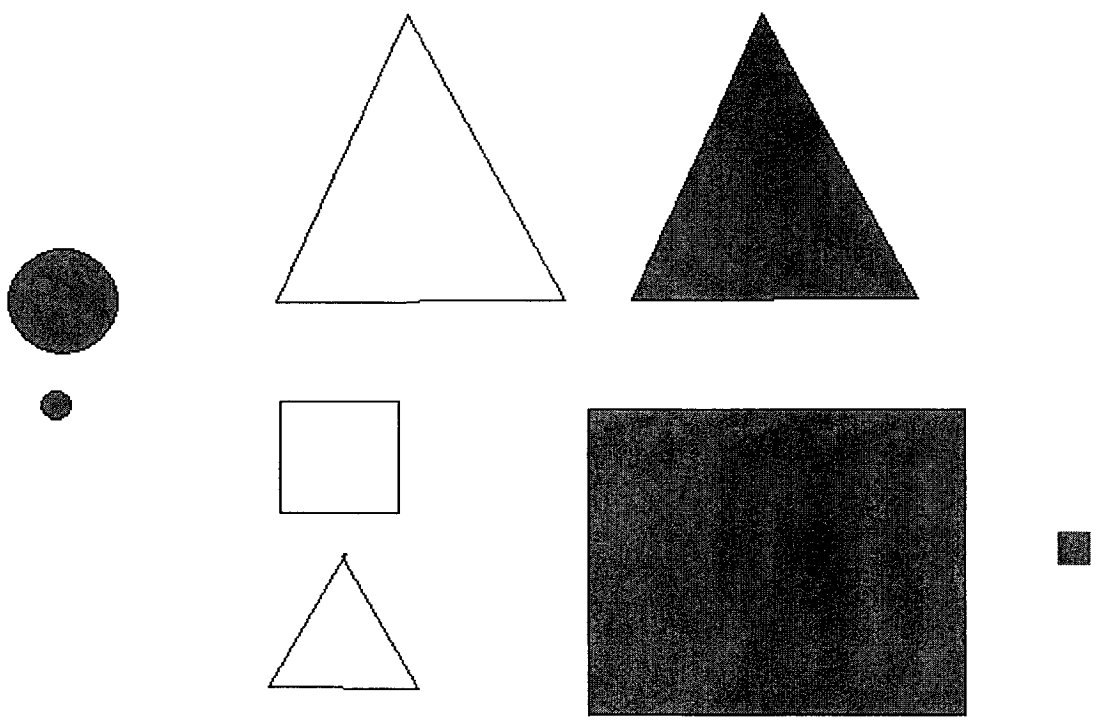

Figure 1.2 Geometric objects

This figure shows eight geometric objects of three shapes (circle, triangle, and square), with two different colors (black and white), in three sizes (small, medium, and large). Not every single clustering of these objects is necessarily desirable to every user. Figures 1.3 and 1.4 show two clustering of these objects, among others.

In Figure 1.3, clustering is done based on a single attribute, color. This is a high level clustering in which the number of clusters is small, two in this case. Black objects are grouped in cluster 1 while the white objects are assigned to cluster 2.

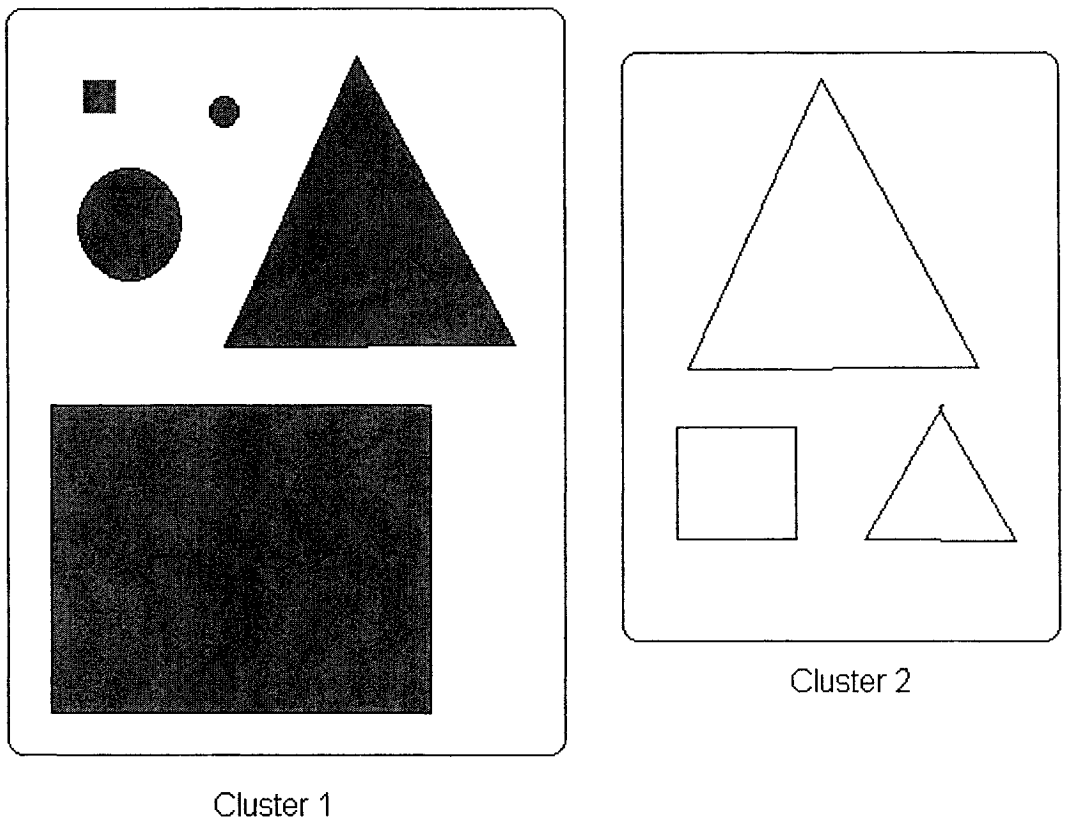

Figure 1.3 High level clustering

In Figure 1.4, there are two attributes used for describing the clusters. Cluster 1 contains black squares. Cluster 2 includes objects which are white and medium size. Objects in cluster 3 are black circles, and those in cluster 4 are large triangles. The clustering shown in Figure 1.4 is more detailed than the one in Figure 1.3, but it is still "meaningful". These are just two examples of many other clusterings that could be considered for this collection.

In general, we can say that different users may prefer different clusterings, depending on the application at hand. For instance, a user may prefer high level clustering in which fewer clusters are generated, while another user may prefer more clusters, identified by more attributes.

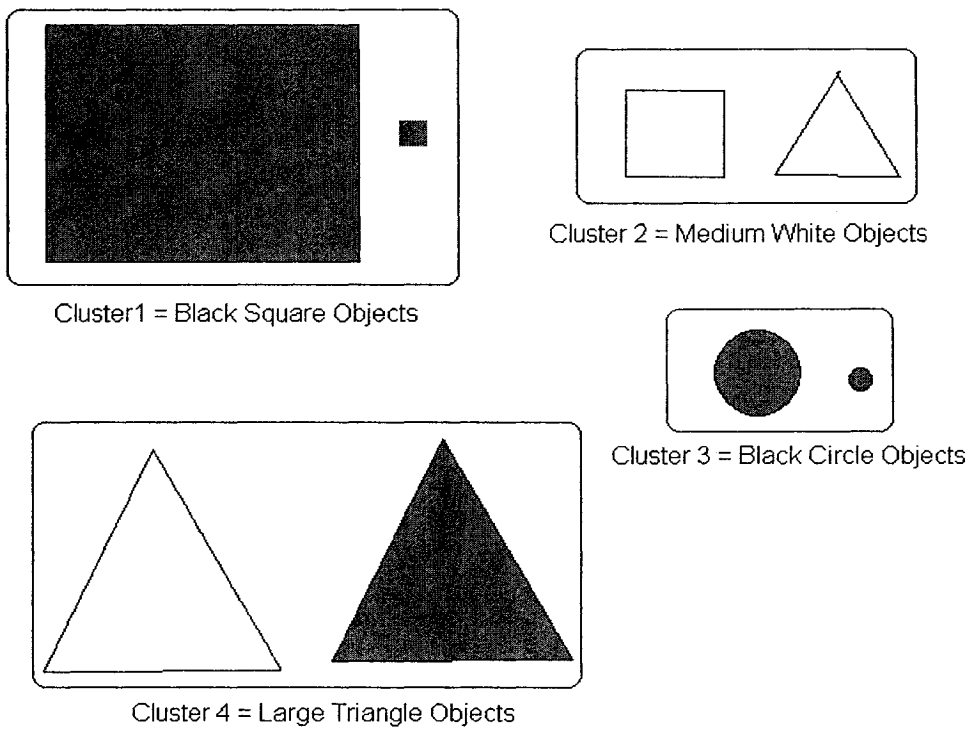

Figure 1.4 Low level clustering

There are some features for a good clustering algorithm, such as [13]:

1. **Scalability.** The clustering algorithm should be applicable to large datasets which do not fit into the main memory. It is desired that execution time increases linearly when the number of objects in the datasets increases.

2. **Ability to handle high dimensional datasets.** Some algorithms are only suitable for low (say two or three) dimensional datasets. Since some datasets such as categorical are high dimensional, it is vital to develop clustering algorithms which can be used for such data.
3. **Ability to deal with different types of data.** Datasets could be of various types. An ideal clustering technique should handle different types of datasets and not just specific ones.
4. **Ability to identify clusters of any shape.** Some algorithms which use Euclidean and Manhattan distance functions, tend to generate only spherical clusters. A clustering algorithm should detect clusters of arbitrary shape. This is important when using numerical or spatial datasets.
5. **Robust with respect to outliers or noise**. Since in practice, datasets include noise, it is important that a clustering algorithm be able to distinguish noise or outliers from real data and produce high quality results, when data include noise.
6. **Order independence.** The order of objects in the input dataset should not affect a clustering result.
7. **Few input parameters. T**he quality of a clustering result depends on the input parameters. Having many input parameters makes it difficult to set and control the quality of clusters.
8. **Interpretability of generated result.** It is necessary for clustering algorithm to generate results which are meaningful and interpretable by the user.

Our motivation in this work is to develop a clustering algorithm with the above properties (except 3 and 4) for categorical data and the ability to perform clustering according to user's expectation, expressed as the degree of similarity of objects.

1.3 Thesis Contributions

The main contributions of this thesis are as follow:

1. Introducing a new similarity measure, called *cohesion*, for categorical datasets.
2. Developing a clustering algorithm, called CLUC, based on the notion of cohesion. Our proposed algorithm satisfies the aforementioned desired properties and can discover natural clusters in the input dataset. The only input parameter to our algorithm, cohesion, is meaningful to the user as its captures the degree of similarity of objects in each cluster.
3. Performing extensive experiments both on real and synthetic datasets for evaluating efficiency and scalability of the proposed algorithm.
4. Comparing the results generated by CLUC with those generated by other clustering algorithms for categorical data.
5. Evaluating efficiency of CLUC for high dimensional datasets.
6. Demonstrating usability and interpretability of generated results.

1.4 Outline of the Thesis

The rest of this report is organized into six chapters. Chapter 2 provides a background. It reviews data types, similarity and distance measurements, and describes a general classification of clustering algorithms. Chapter 3 provides a survey of categorical

clustering approaches. Chapter 4 introduces the CLUC algorithm, which we propose for clustering categorical datasets, based on a novel similarity measure, called cohesion. Chapter 5 describes a design and implementation of CLUC. Chapter 6 presents the experiments and results generated by CLUC on some real and synthetic datasets. We also compare the results produced by CLUC and other well-known techniques. We also study the scalability of CLUC for higher dimensional and large datasets. Chapter 7 includes concluding remarks, and lists our contributions in this thesis. Limitation of CLUC and possible future works are also discussed.

CHAPTER 2 Background

In this chapter, we first state the clustering problem. We then introduce different data types, similarity and distance measurements. We also provide a classification of clustering algorithms.

2.1 The Clustering Problem

Clustering is a process which partitions a set of objects $D = \{X_1, X_2, ..., X_n\}$ characterized by a set of attributes $A = \{A_1, A_2, ..., A_m\}$ into groups based on a similarity notion. Suppose $C = \{C_1, C_2, ...C_k\}$ is a clustering of D. Then C should satisfy the following conditions:

1. $\forall C_i \in C, C_i \neq \varnothing$.
2. $\forall i, 1 \leq i \leq n, \exists j, 1 \leq j \leq k$, $X_i \in C_j$.
3. $\forall i \neq j,\ 1 \leq i, j \leq k, C_i \bigcap C_j = \varnothing$, where k is the number of clusters in C.

The first condition says no partition in C is empty. Condition 2 and 3 says each object belongs to exactly one cluster. Note that condition 3 is necessary only for crisp clustering. In fuzzy clustering, each object belongs to all clusters but with different degrees of membership. This thesis focuses only on crisp clustering.

Each object X_i in D is distinguished from the others by a set of characteristics known as attributes (also called as features or fields). For example, in a hospital dataset, each patient is identified by his/her name, gender, age, address, phone, file number, etc.

Each attribute has a domain from which it takes its values. For instance, the domain of an attribute can be numerical (age) or categorical (sex).

Suppose objects in D are described by m attributes. The domain of each attribute A_i is denoted by D_i. We refer to $D_1 \times D_2 \times ... \times D_m$ as the space on which D is defined; in other words, every dataset D on the above space is a subset of $D_1 \times D_2 \times ... \times D_m$. Each element X in D is called an object, a point, data, individual, record, tuple, or transaction, and is represented as $X = (x_1, x_2, ..., x_m)$, where $x_i \in D_i$, for $1 \leq i \leq m$. The number of objects in D is called *size* of D and is shown by $|D|$, and the number m of its attributes is called dimension of D.

2.2 Categorization of Data Types

Selecting a suitable clustering algorithm mainly depends on the type of dataset. In each dataset, attributes get their values from various types of domain. In general, these values are classified as follows [13]:

1- **Nominal values.** These values are taken from unordered domains. For any pair of values x_1, x_2 from a nominal space, we have either $x_1 = x_2$ or $x_1 \neq x_2$. The sets consisting of the names of weekdays or months, or names of colors are examples of nominal values. Both categorical and binary values are of this type.

2- **Ordinal values.** The domain of ordinal values is a total ordered space. That is, for any pair of values x_1, x_2 from an ordinal space, we have either $x_1 \leq x_2$ or $x_2 \leq x_1$, however measuring the distance between them does not make much sense. An

example of an ordinal set is G=$\{A^+, A, A^-, B^+, B, B^-, ..., F\}$ used for grades. We should mention that no mathematical operation can be performed on ordinal values.

3- **Interval-scaled values.** The domains of these values are ordered and the distance between any pair of such values is also meaningful. These domains do not include "natural" zero. For instance, two temperatures values $25°C$ and $10°C$ are interval-scaled values because we can say that their difference is $15°C$. However, the Zero value in temperature domain is arbitrary and not a natural one, since when we change the measurement scales, this value will change as well, i.e., $0°C = 32°F$.

4- **Ratio-scaled values.** A ratio-scaled space is the same as an interval-scaled space but it has a fixed absolute zero value. For a pair of value x_1, x_2, both their distance and their relative size or magnitude exists. Physical measurement such as height and weight are of this type.

Variables that are nominal or ordinal are discrete, while interval and ratio-scaled variables are continuous. In the following section, we will discuss different proximity measurements used for various data types.

2.3 Proximity Measure

After determining the data type for objects in a dataset, an important issue in clustering algorithms is finding a measurement which determines resemblance of pairs of points, pairs of clusters, or a point and a cluster. This proximity measure can be either a similarity or a distance function. There are many ways to define similarity or distance functions depending on the type of data. Let Ω be the space from which the input dataset

is taken. A distance function *dis* defined on Ω should satisfy the following conditions [28]:

1. $dis: \Omega \times \Omega \rightarrow R^{+}$
2. Commutative: $dis(x, y) = dis(y, x)$, $\forall x, y \in \Omega$.

A distance function is called a *metric distance* if it also satisfies the following two additional conditions:

1. Reflexivity : $dis(x, y) = 0$ iff $x = y$.
2. Triangle inequality: $dis(x, y) \leq dis(x, z) + dis(z, y)$, for all $x, y, z \in \Omega$.

Many popular metric and non metric distance functions have been defined and used in different contexts. They are more applicable to numeric spaces where the domains of attributes are continuous. Below we list some common distance functions defined on m-dimensional numerical spaces such as Ω, where every point $X \in \Omega$ is an m-tuple $(x_1, x_2, ..., x_m)$:

- Euclidean [28]: $dis(X, Y) = \left(\sum_{i=1}^{m} | x_i - y_i |^2 \right)^{1/2}$.

The Euclidean distance is the shortest distance between two points (Figure 2.1).

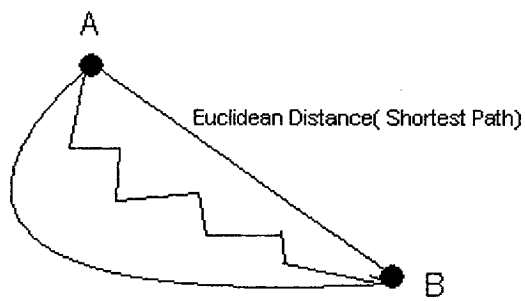

Figure 2.1 Euclidean distance between A and B

- Manhattan (City-block) Distance [28]: $dis(X,Y) = \sum_{i=1}^{m} |x_i - y_i|$.

This distance measures the length of the grid-like path to go from one point A to point B. If we present the space using grid length 1, one should pass the units only vertically or horizontally to reach from A to B. In Figure 2.2, two Manhattan distances between points A and B are shown in bold, which are equal.

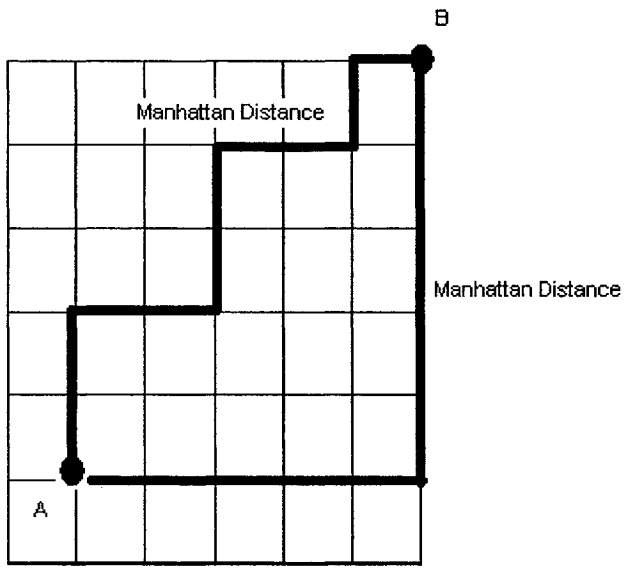

Figure 2.2 Manhattan distance between A and B

- Chebyshev [28]: $dis(X,Y) = \max_{1 \le i \le m} |x_i - y_i|$.

This gives the maximum distance between two points in one attribute (or dimension).

- Minkowski [28]: $dis(X,Y) = \left(\sum_{i=1}^{m} | x_i - y_i |^{\lambda} \right)^{1/\lambda}$.

This is a generalization of the aforementioned distance functions. For λ=2, it is the Euclidean distance between two points. When λ=1, it is the Manhattan distance, and for $\lambda = \infty$, it represents Chebyshev distance.

A function *sim* on a space Ω is called a similarity function if [28]:

1. *sim*: $\Omega \times \Omega \rightarrow [0,1]$. Value 1 returned by this function shows the highest similarity between the input pair of objects, while 0 indicates they are completely dissimilar.
2. Commutative: $sim(x, y) = sim(y, x)$, $\forall x, y \in \Omega$.
3. $sim(x, y) = 1$ iff $x = y$, i.e., each object has the maximum similarity with itself or with an identical object.

Commutative property in both similarity and distance functions makes them independent of the direction of comparison. Similarity functions are more commonly used when data are described by nominal values such as categorical or binary values. Suppose D is an m-dimensional dataset and V is the union of all its attribute domains, i.e., $V = D_1 \cup D_2 \cup ... \cup D_m$. To distinguish values from different domains, we tag each attribute value a with its attribute name or ID, e.g., $A_i.a$. For any pair of objects $X, Y \in D$, the number of matched and mismatched attribute values can be shown using an association table, an example of which is given in Table 2.1.

Table 2.1 Association table for objects X and Y

	$a \in X$	$a \notin X$
$a \in Y$	n_{11}	n_{01}
$a \notin Y$	n_{10}	n_{00}

In this table, n_{11} is the number of attribute values which exist both in X and Y, n_{10} is the number of values that exit only in X, n_{01} is the number of values that exist only in Y, and n_{00} is the number of values which exist neither in X nor in Y. If we consider X and Y as a set of attribute values, then $n_{11} = |X \cap Y|$, $n_{10} = |X - Y|$, $n_{01} = |Y - X|$, and $n_{00} = |V - (X \cup Y)|$.

Using association tables, many similarity metrics are defined for objects with nominal attribute values, some of which are as follows:

- Jaccard Coefficient[20,28]: $sim(X,Y) = \frac{n_{11}}{(n_{11} + n_{01} + n_{10})} = \frac{|X \cap Y|}{|X \cup Y|}$

- Simple Matching Coefficient [4]:

$$sim(X,Y) = \frac{(n_{11} + n_{00})}{(n_{11} + n_{10} + n_{01} + n_{00})} = \frac{|X \cap Y| + |V - (X \cup Y)|}{|V|}$$

- Dice Coefficient[20]: $sim(X,Y) = \frac{2 \times n_{11}}{2 \times n_{11} + n_{10} + n_{01}} = \frac{2 \times |X \cap Y|}{|X| + |Y|}$

- Cosine Coefficient[20]: $sim(X,Y) = \frac{n_{11}}{\sqrt{(n_{11} + n_{10}) \times (n_{11} + n_{01})}} = \frac{|X \cap Y|}{\sqrt{|X| \times |Y|}}$

- Overlap Coefficient[20]: $sim(X,Y) = \frac{|n_{11}|}{\min(n_{10}, n_{01})} = \frac{|X \cap Y|}{\min(|X|, |Y|)}$.

Many clustering algorithms which work on categorical datasets use the above formulas as their proximity measurements.

In clustering, it is common to measure proximity between an object x and a Cluster C or between a pair of clusters C_1 and C_2. There are many functions proposed for measuring the distance between an object x and a cluster C, for instance, the followings [9]:

- Max proximity : $dis(x,C) = \max_{\forall y \in C}(d(x,y))$
- Min proximity : $dis(x,C) = \min_{\forall y \in C}(d(x,y))$
- Average proximity: $dis(x,C) = \sum_{y \in C} \frac{dis(x,y)}{N}$, where N is the size of C.

Some proximity measures used for clusters C_1 and C_2 are defined as follows [9]:

- Max proximity: $dis(C_1,C_2) = \max_{\forall x \in C_1, y \in C_2}(d(x,y))$
- Min proximity: $dis(C_1,C_2) = \min_{\forall x \in C_1, y \in C_2}(d(x,y))$
- Median proximity: $dis(C_1,C_2) = dis(x,y)$, where x and y are the median points of C_1 and C_2, respectively.
- Average proximity: $dis(C_1,C_2) = dis(x,y)$, where x and y are the average points of C_1 and C_2, respectively.

2.4 Classification of Clustering Approaches

In addition to specifying the data type and determining an appropriate proximity measure, it is essential to find a procedure with which one can find "natural" clusters in a dataset. There have been numerous studies for this which resulted in several clustering algorithms. Selecting a suitable algorithm mainly depends on the type of dataset and the application at hand. In general, clustering algorithms can be classified as follows [13]:

- Partition-based
- Hierarchical (agglomerative or divisive)
- Density-based

- Grid-based
- Model-based

In what follows, we give a brief description for these clustering algorithms.

2.4.1 Partition-based

The basic idea behind a partition-based algorithm is to partition a dataset into a number of clusters specified by users. These are greedy algorithms which form clusters by optimizing locally or globally an objective criterion function, for instance maximizing the similarity between objects in a cluster, or minimizing their distance. Example of well-known partition-based algorithms is k-mean algorithm [10].

In general, partition-based clustering algorithms consist of three steps: initialization, partitioning, and refinement. In the initialization phase, k points are selected as clusters' representatives. Then the remaining points are assigned to these k clusters according to the optimization of a critical function. When a point is assigned to a cluster, the representative point of that cluster must be updated. Selecting the first k points can be done in various ways; for example, they can be chosen randomly.

After the initial partitioning, the dataset is scanned again and the points are reassigned to the best cluster according to the criterion function. The refinement phase is repeated until no more object is reallocated.

Partition-based clustering algorithms have some drawbacks including:

- Selecting the number k of clusters is problematic. There is no efficient way to find a suitable k.

- The convergence of the algorithm depends on the initialization part, i.e., selecting the k initial points.
- They can be sensitive to outliers or noise. Even a point which is far from all points is forced to be assigned to one of the clusters.

2.4.2 Hierarchical

This type of clustering algorithms partitions data into nested clusters. A sequence of clusters is generated gradually which can be shown by a dendogram, example of which is illustrated in Figure 2.3. Hierarchical clustering algorithms produce different sizes of clustering in a single execution. Examples of the widely-used algorithms of this type include CURE [12], BRICH [30], and ROCK [11].

There are two types of hierarchical algorithms, agglomerative and divisive. They differ in the way they construct nested clusters. Agglomerative is a bottom-up algorithm in which clusters are formed in a series of successive merging of points. It starts by considering each point as a singleton cluster. At each level, the most similar/closest clusters are merged together and form a new single cluster. This iterative process goes on until we get the desired number of clusters or certain conditions hold. Since often the number of final clusters is initially unknown to the user, instead of specifying the number of clusters, the user can decide a threshold for distance or similarity. This user-defined threshold will be used for terminating the algorithm. If the distance/similarity between all pairs of clusters is greater/smaller than that threshold, the clustering process terminates; otherwise the process of merging clusters continues.

Some of the agglomerative techniques are: simple linkage, average linkage, and complete linkage. They differ in the way they measure the distance between a pair of clusters which is used for merging them.

In contrast to agglomerative algorithms, divisive clustering is a top-down process. At the beginning, all the points are in one cluster. Then at each level, some of these clusters may be further divided into smaller ones. Figure 2.3 shows the agglomerative and divisive algorithms.

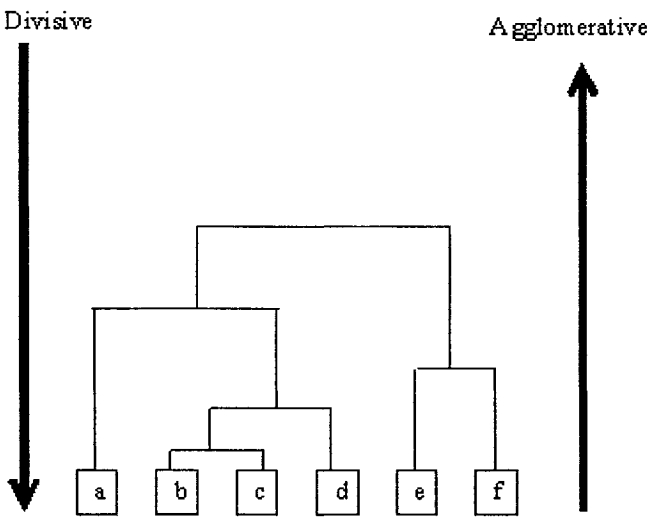

Figure 2.3 Hierarchical clustering of points

Hierarchical clustering algorithms have the following drawbacks:

- They keep the similarity/proximity matrix in memory. This makes the algorithm not to be scalable for large datasets. They are more suitable when a similarity matrix is available instead of data points.

- Their time complexity is at least $O(n^2)$, where n is the number of points in the dataset.
- Since the time complexity is high, sampling may be used to decrease the complexity, which affects the quality of the results.
- These algorithms never undo the results of the clustering at the previous level. So if a point is assigned to a wrong cluster, there is no way to reassign it to a proper cluster.

2.4.3 Density-based

Density-based algorithms consider clusters as high-density regions which are separated from each other by lower-density regions called noise. Each dense region must contain a minimum number of objects defined by users. Some of the density-based clustering algorithms, such as DBSCAN [8], classify objects in a given dataset into two groups: core and border. Core objects have at least *MinPts* neighbors within a neighborhood *Eps*. An object which is not core is a border. Clusters are generated by merging neighboring core objects.

Such algorithms are suitable for discovering clusters with any arbitrary shape, e.g., spatial datasets. Moreover, they are capable of distinguishing clusters in a noisy dataset. However, a generated result is affected highly by the input parameters such as *MinPts* and *Eps*.

2.4.4 Grid-Based

Grid-based algorithms use grid data structure for clustering datasets. They partition data space into finite number of cells (grids or units). All the operations for

clustering are done on these grids. As in Density-based, cells which contain more than a certain number of objects are called dense. Clusters are generated using dense grid cells.

STING [27] and CLIQUE [3] are examples of grid-based algorithms which are applicable to numerical datasets. CLICK [24] is a grid-based algorithm applicable to categorical datasets. As in density-based algorithm, grid-based algorithms are capable to generate arbitrary shape clusters and can distinguish noise.

3.4.5 Model-Based

A model-based clustering algorithm finds clusters based on statistical probability models. In such an approach, it is assumed that objects $X_1, X_2, ..., X_n$ in dataset D follow some probability distribution, typically Gaussian distribution. Model-based algorithms consider a finite mixture model for clusters and then try to find the best fit of a given model to the data. The most advantage of these algorithms is the ability to find clusters with any shape. They can also be utilized for datasets with different kinds of attributes. An example of model-based algorithm is expectation-Maximization (EM) algorithm [7].
In the following chapter, we study categorical clustering algorithms and provide a classification of them.

Chapter 3 Related Work

In this chapter, we review clustering algorithms proposed for categorical datasets, in which objects are described only by categorical values from nominal types. For this, we use the following information to compare them.

- Proximity measure used for objects or clusters.
- Input parameters.
- How they create clusters.
- How they define quality of clusters.
- Computational complexity.
- Detecting noise.
- Handling large datasets.
- Affect of input order of the dataset.

3.1 K-mode

Huang [16] proposed a partition-based clustering algorithm, called K-mode, as an expansion of the K-mean [17] algorithm for categorical attributes. In this algorithm, the distance function between two points $X = (x_1, x_2, ..., x_m)$ and $Y = (y_1, y_2, ..., y_m)$ is defined either as:

$$d(X,Y) = \sum_{i=1}^{m} \delta(x_i, y_i)$$

or :

$$d_{\chi^2}(X,Y)=\sum_{i=1}^{m}\frac{(n_{x_i}+n_{y_i})}{n_{x_i}n_{y_i}}\delta(x_i,y_i) \qquad (3.1)$$

where

$$\delta(x_i,y_i)=\begin{cases}0 & (x_i=y_i)\\ 1 & (x_i\neq y_i)\end{cases}$$

and n_{x_i} and n_{y_i} are frequencies of x_i and y_i values in the dataset. Since in Equation 3.1 more importance is given to rare attribute values, it is mostly used to discovering under-represented object clusters [16].

For a dataset D, the best clustering $C=\{C_1,C_2,...,C_k\}$ is obtained by minimizing the following cost function E:

$$E=\sum_{j=1}^{k}\sum_{i=1}^{n}P_{i,j}d(X_i,Q_j)$$

where each $P_{i,j}$ is one of the elements in the partition matrix as defined in [16], X_i belongs to the dataset, n is size of the dataset, k is the number of clusters, and each Q_j is the *Mode* object for cluster C_j , $1\leq j\leq k$. The mode object $Q=(q_1,q_2,\ldots,q_m)$ is the representative in a cluster C which minimizes the following distance function:

$$dis(Q,C)=\sum_{X_i\in C}dis(X_i,Q)$$

The clustering procedure in K-mode is as follow:

1- Get the number k of clusters from the user.

2- Select k mode objects as representative for k clusters.

3- Allocate all remaining objects in the dataset to the clusters. Each object will be allocated to a cluster with nearest mode object to it. Update the mode object in the cluster after each allocation.

4- Scan the dataset again and find the best cluster for each object. If the best cluster is different from the one to which the object belongs, reallocate the object to best cluster and update the mode object in both previous and new clusters.

5- Repeat step 4 until no reallocation is required.

Table 3.1 lists the advantages and weakness of this algorithm.

Table 3.1 List of major cons & pros of K-mode

Pros	**Cons**
1. It can handle large datasets. 2. It is easy to program. 3. The computational time is linear in the size of the dataset.	1. Selecting k is problematic. 2. Convergence depends on the selection of k initial Mode vectors. 3. It is sensitive to the input order of data. 4. It can not detect noise. Noise will affect the quality of produced clusters. 5. It may scan the dataset more than 2 or 3 times.

K-mode requires only one input from the user which is the number of clusters. Time complexity of K-mode is $O(nkL)$, where n is the size of dataset, k is the number of clusters and L is the number of iterations.

3.2 ROCK

Guha et al. proposed ROCK (**Ro**bust **C**lustering using lin**k**s) [11] as a clustering algorithm for categorical and boolean datasets. It is an agglomerative hierarchical algorithm which merges clusters based on *links* instead of other distance metrics or similarity functions. For two distinct points p and q, $link(p,q)$ is the number of common neighbor points between them. According to their definition, two points p and q are called neighbor if for a similarity function *sim*, we have that:

$$sim(p,q) \geq \theta$$

where θ is a threshold defined by the user between 0 and 1. The function *sim* can be any similarity measure defined for categorical and nominal values such as Jaccard coefficient.

ROCK produces best clustering by maximizing the following objective function:

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

where k denotes the number of desired clusters, C_i is cluster i with size n_i, and $f(\theta)$ is a function estimated by the user with the following property: for each point in cluster C_i, there should exist nearly $n_i^{f(\theta)}$ neighbors in C_i.

ROCK performs clustering in three phases. In the first phase, it draws a random sample from the whole dataset. Sampling improves scalability of ROCK. In the next phase, the sample data is clustered using a link-based hierarchical algorithm. At each step of hierarchical clustering, each pair of clusters which have the highest number of links

between their points are merged. For two clusters C_i and C_j, the goodness measurement $g(C_i, C_j)$ for merging clusters is defined as follows:

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

Where $link(C_i, C_j) = \sum_{p_q \in C_i, p_r \in C_j} link(p_q, p_r)$, and n_i and n_j are sizes for clusters C_i and C_j, respectively.

In the final phase, the remaining points in the dataset are assigned to the cluster that is the closest to them. Table 3.2 lists the advantages and weakness of this algorithm.

Table 3.2 List of major cons & pros of ROCK

Pros	**Cons**
1. It can handle noise. 2. It is insensitive to input order of data.	1. It uses sampling for handling large datasets. Using sampling will decrease the quality of generated clusters. 2. The estimation of the accurate value for function $f(\theta)$ is problematic for the user.

Rock takes three inputs from the user: number of clusters, similarity threshold θ, and $f(\theta)$. The time complexity for ROCK is $O(nm_m m_\alpha + n^2 \log n)$ where n is the size of the dataset, m_m is the maximum number of neighbors for a point, and m_α is the average number of neighbors.

3.3 LIMBO

Another hierarchical clustering algorithm for categorical datasets is LIMBO (scaLable InforMation BOttleneck) proposed by Andritsos et al. [1]. In general, LIMBO clusters data in three phases. In the first phase, it produces a compact summary model of the input data. This model contains sufficient statistical information about the dataset and is kept in a B-tree like structure, called Distributional Cluster Feature tree (or DCF tree, for short). Using this summary model, LIMBO can handle large datasets, because it just keeps statistical information about the clusters instead of keeping all the data in memory. Figure 3.1 presents an instance of a DCF tree.

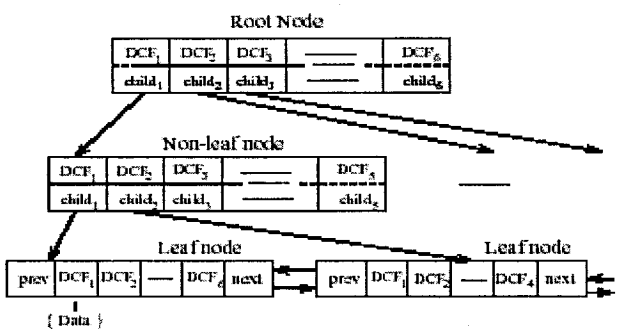

Figure 3.1 A DCF tree with a branching factor 6 [1]

DCF tree can be built in two ways: either it bounds the memory used ($LIMBO_S$), or its accuracy is controlled by a user defined threshold ($LIMBO_\Phi$). After constructing a DCF tree, LIMBO uses, in the second phase, an agglomerative hierarchical clustering algorithm, called Agglomerative Information Bottleneck (AIB) [25] to cluster the leaf nodes in the DCF tree. This algorithm applies a novel distance measure between clusters based on mutual information from information theory. The distance between two clusters C_i and C_j is the amount of information loss between them defined as follows:

$$\delta I(C_i, C_j) = [p(C_i) + p(C_j)] D_{JS}[p(V \mid C_i), p(V \mid C_j)] \quad (3.2)$$

Where D_{JS} is the Jensen-Shannon (JS) divergence defined in [1], $p(C_i)$ is the probability function for clusters C_i, and $p(V \mid C_i)$ is the conditional probability distribution for set of values V in cluster C_i. Note that V is the union of all the attribute domains for dataset D. Equation 3.2 also measures the increment of uncertainty when two clusters are merged.

At each level, two clusters which have the minimum information loss will be merged together. The objective in AIB is minimizing the average relative entropy defined as :

$$\sum_{c \in C_k, t \in D} p(t,c) D_{KL}[p(a \mid t) \| p(a \mid c)]$$

where D_{KL} is relative entropy, or Kullback-Leibler (KL) divergence, which measures the difference between two probability distributions.

After clustering DCF tree leaf nodes, the whole dataset is scanned again, in the final phase, and each object is assigned to the cluster which is closest to it.

Table 3.3 lists the advantages and weakness of this algorithm.

Table 3.3 List of major cons & pros of LIMBO

Pros	**Cons**
1. It can handle large datasets using DCF tree model of datasets. 2. It scans dataset two times. One for creating DCF tree and one for final process.	1. It needs the number of clusters as input which is problematic. 2. DCF tree is sensitive to the input order of dataset. So the input order of data will affect the quality of generated clusters.

LIMBO takes two parameters. As $LIMBO_S$, it needs number k of clusters and the amounts of available main memory. As $LIMBO_\Phi$, it needs number k of clusters and a threshold Φ for controlling the accuracy of DCF tree.

The computational complexity is the total of the following costs:

1- The cost of building a DCF tree includes two parts: first is the I/O cost for reading the dataset once. The other part is the cost for adding nodes to the DCF tree which is $O(nhdB + dUB^2)$, where n is the size of the dataset, h is the height of the tree, d is the number of attributes, B is the maximum branching factor of the tree, and U is the number of non-leaf nodes.

2- Cost of clustering leaf nodes in DCF tree: $O(l^2 d^2 \log l)$, where l is the number of entries at the leaves of DCF tree and d is the total number of attributes.

3- Cost of assigning objects to clusters: which is the cost of one I/O cost for reading the data + $O(kdn)$, where k is the number of clusters, d is the total number of attributes, and n is the size of the dataset.

3.4 COOLCAT

Barbara et. al [4] proposed COOLCAT (**C**lustering using **COOL**ing) which is an incremental clustering algorithm that employs information theory for generating clusters. Instead of using distance or similarity function, it uses entropy, which is a measurement of uncertainty, i.e., the more the entropy, the less the certainty with which we can predict the value of a variable.

The authors indicate that more similarity between objects in a group results in less entropy in the group. They used this idea to generate the clusters. The best clustering

$C = \{C_1, C_2, ..., C_K\}$ for a given dataset D is produced by minimizing the expected entropy in the whole system which is defined as follow:

$$E(C) = \sum_{i=1}^{k} \frac{|C_i|}{|D|} E(C_i)$$

where k is the number of clusters and C_i is a cluster in C, $E(C_i)$ represents the entropy of C_i. As an incremental clustering algorithm, COOLCAT read the objects sequentially from the dataset and their effect on the expected entropy of the system is determined.

COOLCAT gets the number of clusters as an input. It has two main steps: initialization and incremental steps. To start, it selects k initial clusters from a random sample from the dataset. These k singleton clusters are chosen to maximize the minimum entropy between pairs of chosen objects.

In the incremental step, collections of objects are read from the disk into the memory. Then each object read is assigned to the cluster to which the entropy is minimized.

COOLCAT is sensitive to the order of objects in the input data. For reducing this effect, after reading batch of points and assigning them to more suitable clusters, a fraction m of points which are considered as worse fit point to clusters, are identified as bad-clustered and they are reallocated to the clusters again. The worst fit points are the m points with the lowest fitting probability values. (For more details, see [4]).

Table 3.4 lists the advantages and drawbacks of this algorithm.

Table 3.4 List of major cons & pros for COOLCAT

Pros	Cons
1. It is scalable to large dataset. 2. It does not need to keep all the points in the memory at the same time. Only the representation of clusters is kept in the memory. 3. It scans dataset just once.	1. It is good for well-separated clusters. 2. It needs the number k of clusters as an input. 3. It can not handle noise. 4. The quality of generated results depends to the sampling data.

The input parameters for COOLCAT are the size of initial sample, number of clusters, and selecting value for m. The computational complexity is $O(n)$, where n is the size of the dataset.

3.5 CLICK

Peters et. al. introduced CLICK [24] which is an integration of grid-based and density-based approaches. As CLIQUE [3] which works on numerical datasets, CLICK is able to find subspace clusters but for categorical datasets. The main idea underlying this algorithm is to detect dense spaces or subspaces in a dataset. To define a space or a subspace on a dataset D with m attribute domains $D_1, D_2, ..., D_m$, the authors define the notion interval for attributes. According to their definition, a set $S_i \subset D_i$ is called an interval on D_i. The Cartesian product of k intervals, on different attribute domains, is referred to as k-interval. For $k<m$ and $k=m$, the k-interval is called *subspace* and *space*, respectively. *Dense* subspaces or spaces are those which include the number of records greater than an expected value. CLICK works on the subspaces and spaces which are generated from strongly connected attribute values. The two attribute values a and b are

strongly connected if the number of records that include both values exceeds an expected value.

In CLICK, each cluster should satisfy the following conditions:

1. It should be a dense space or subspace.
2. Any pair of attribute values from different intervals in a cluster should be strongly connected to each other.
3. All the intervals are maximal, i.e. no more attribute values can be added to any of the intervals which are strongly connected to other values in other intervals.

CLICK performs clustering in three steps: Pre-Processing, Clique-Detection, and Post-Processing. In Pre-Processing, the dataset is scanned once and all pairs of strongly connected attribute values are found. These values form the vertices of a k-partite graph. Edges in this graph are the links between two values strongly connected to each other.

In the next step, all the maximal k-partite cliques are detected. A maximal k-partite clique is a maximal subset of attribute values from k-partite graph in which every pair of attribute values from different domains are strongly connected to each other. The maximal k-partite cliques are the candidate clusters for next step.

In Post-Processing step, those cliques which are dense are selected as final clusters. Since some of the clusters may have overlapping, the clusters which are often co-supported by the same tuples are merged together. This is a maximal frequent set mining problem. (For more details, see [24]).

As a density-based algorithm, CLICK does not need the number of clusters in advance but it takes two other inputs from the user: α applied for specifying density and

MinSup which is used as the minimum support for merging clusters. Both these parameters are positive numbers with no upper bound being specified. That is, the user has no idea for choosing "right" values for these inputs. Table 3.5 lists the advantages and weakness of this algorithm.

Table 3.5 List of major cons & pros of CLICK

Pros	Cons
1. It does not need the number of clusters. 2. It is able to generate subclusters. 3. It works well for high dimensional datasets.	1. No upper bound is specified for input parameter. So it is unclear how to set them.

3.6 CLOPE

Yang et al. [29] proposed CLOPE (**C**lustering with s**LOPE**) as a clustering algorithm especially designed for transactional data. As in LargeItem algorithm [26], CLOPE optimizes a global criterion function iteratively for partitioning the input dataset. Using a global criterion function is more effective and faster than a local one since the latter measures pair-wise similarity.

The principal idea of this algorithm is to increase the overlapping of transactions in each cluster. For this purpose, for each cluster C_j, authors define size, width, and height as follows:

$$Size(C_j) = S(C_j) = \sum_{t_i \in C_j} |t_i|$$

$$Width(C_j) = W(C_j) = \text{The number of distinct items in cluster } C_j$$

$$Height(C_j) = \frac{S(C_j)}{W(C_j)}$$

To cluster the dataset, they proposed the following global criterion function to be maximized:

$$Profit_r(C) = \frac{\sum_{i=1}^{k} \frac{S(C_i)}{W(C_i)^r} \times |C_i|}{\sum_{i=1}^{k} |C_i|}$$

In the above formula, $|C_i|$ is the number of objects or transactions in cluster C_i and $r \in R^+$ is a user-defined parameter, called *repulsion*, which is used to control the tightness of a cluster.

CLOPE performs clustering in two phases: initialization and refinement. In the initialization phase, the dataset is scanned once. Records are read into memory one by one and assigned to an existing cluster or to a new cluster that maximizes the global function. In the refinement phase, the dataset is scanned iteratively to refine the clusters. In this process, transactions may move to other clusters if necessary. The refinement phase terminates if in one scan no records are moved or no new cluster is generated. In this algorithm, the number of clusters is generated automatically depending on the given input *r*. Table 3.6 lists the advantages and weakness of this algorithm.

Table 3.6 List of major cons & pros of CLOPE

Pros	Cons
1. It can handle large datasets. 2. It works well for high dimensional datasets. 3. It is easy to implement. 4. It is not very sensitive to the order of tuples in the input.	1. It is not clear what value r of repulsion gives good clustering and the upper bound of r is unknown.

This algorithm has only one input parameter, which is the repulsion r. The computational complexity is $O(nkAL)$, where n is the size of the dataset, k is the number of clusters, A is the average length of a transaction, L is the number of iterations. According to [29], CLOPE needs a few scans of the dataset to perform clustering.

CHAPTER 4 The CLUC Algorithm

In this chapter, we present our algorithm CLUC (**CLU**stering with **C**ohesion) for clustering categorical data. First, we introduce some background and notations. Then we define a novel measurement called *cohesion*, which is used by CLUC. We conclude this chapter by an example illustrating how CLUC works.

4.1 Background and Notations

A set in which items can occur more than once is called a *bag* and is represented as a collection of item-multiplicity pairs. Suppose B is a bag which contains items $i_1, i_2 \ldots, i_n$. Also suppose each item i_j appears in B with frequency x_j. Then B can be presented as:

$$B = \{i_1 : x_1,\ i_2 : x_2, \ldots,\ i_n : x_n\}$$

For example A = {a,b,c,b,d,a}is a bag, which can be shown as A= {a:2, b:2, c:1, d:1}.

Given a bag $B = \{i_1 : x_1, i_2 : x_2, \ldots, i_n : x_n\}$, the size of B, denoted by $|B|$, is:

$$|B| = \sum_{i=1}^{n} x_i$$

For example, if A= {a:2, b:2, c:1, d:1}, then |A| = 2+2+1+1=6

Note sets can be considered as bags in which every value has 1 as its frequency.

The following operations on bags generalize the corresponding operations from standard set theory.

- **Bag Intersection**

For bags A and B, the bag intersection $\cap_B$ is defined as:

$$A \cap_B B = \{x:k \mid x:m \in A, x:n \in B,\ k = \min(m,n)\}$$

The bag intersection is commutative, i.e., $A \cap_B B = B \cap_B A$.

For example, if A={a:2, b:3, c:2} and B={a:5, b:1,d:4}, then $A \cap_B B$ = {a:2, b:1}. By default, the frequency of every element not present in a bag is 0.

- **Bag Union**

For bags A and B, their bag union, denoted by $\cup_B$, is the collection of items in A or B with their frequencies retained, formally:

$$A \cup_B B = \{x:k \mid x:m \in A,\ x:n \in B,\ k = m+n\}$$

This operator is commutative, i.e. , $A \cup_B B = B \cup_B A$.

For example, if A={a:2, b:3, c:2} and B={a:5, b:1}, then $A \cup_B B$ = {a:7, b:4, c:2}.

- **Bag Overlap**

In our work, we define a new operator on bags, called bag *Overlap*. For a pair of bags A and B, the *overlap* of A and B, denoted by $\cap_O$, is the bag of items which exist in both A and B with the frequencies retained. In symbols:

$$A \cap_O B = \{x:k \mid x:m \in A,\ x:n \in B, m \neq 0, n \neq 0, k = m+n\}$$

Note that this operation is also commutative, i.e, $A \cap_O B = B \cap_O A$.

For example, the overlap of A={a:2, b:3 , c:4} and B={a:5, b:1, d:2, e:1 } is {a:7, b:4}.

4.2 Cohesion-based Similarity

As mentioned before, each object in a dataset is described by a set of attribute values, so we can consider each object as a set. For example, in a dataset D with attributes color, size, and shape, T={ color: red, size: medium, shape : circle} is an instance of D which is represented as a set. Notice that the symbol ":" is overloaded; for bags it is used to assign a frequency to an item but here it is used to assign a value to an attribute. Presenting objects as sets are suitable when the dataset is categorical. Consider two objects T_1 and T_2 from a categorical dataset D. In Figure 4.1, we show different relations between these two objects using Vnn diagram.

In Figure 4.1(a) two objects have no common values and are completely dissimilar (disjoint) from each other. For example, T_1 ={color: red , size: big, shape: square} and T_2={color: white , size: small, shape: triangle} are disjoint.

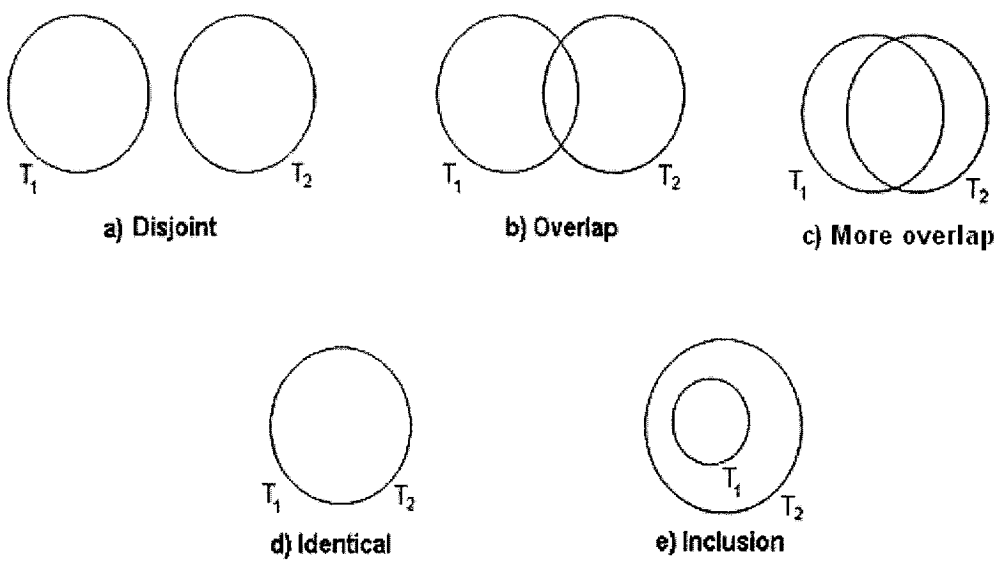

Figure 4.1 Different relations between objects T_1 and T_2

In Figure 4.1(b), T_1 and T_2 have a few common values, so the similarity between them is weak. For example, objects T_1 ={color: red , size: big, shape: square} and T_2={color: red , size: small, shape: triangle} have only identical colors.

In Figure 4.1(c), T_1 and T_2 have more common values with each other, so the similarity between them is stronger. For example, objects T_1 ={color: red , size: big, shape: square} and T_2= {color: red , size: big, shape: triangle} have color and size in common.

In Figure 4.1(d), T_1 and T_2 have identical values for all attributes, and hence fully coincide. For example, $T_1 = T_2$={color: red , size: big, shape: square}.

As it can be seen in the above figures, objects "stick" together according to their common attribute values. When the number of common values increases, the cohesion among objects increases as well. The point to note here is that not only the number of common attributes but also the size of objects affects their similarity. Consider object T_1 and T_2 in Figure 4.1(e). It is clear from this figure that although T_2 contains all the values from T_1, there are some values in T_2 which T_1 does not have. Different-size objects are common in transactional data, such as market-basket data. Two objects are more similar to each other if they have more common values and their sizes are closer.

A well-known similarity measure for two objects S_1, S_2, which considers both common values and the size is Dice coefficient defined as [20]:

$$Dice(S_1, S_2) = \frac{2 \times |S_1 \cap S_2|}{|S_1| + |S_2|} \qquad (4.1)$$

We adopt and extend the Dice coefficient in Equation 4.1 and use in our work as a measure of similarity between two bags. For bags B_1 and B_2, we define their Dice coefficient as follow:

$$CDice(B_1, B_2) = \frac{|B_1 \cap_o B_2|}{|B_1 \cup_B B_2|}$$

Our extended Dice coefficient above for bags is a similarity function because it satisfies the following conditions:

1. $0 \leq CDice(B_1, B_2) \leq 1$. When B_1 and B_2 are completely different from each other, then they have no intersection and their similarity is zero. In contrast, when B_1 and B_2 are identical, then their similarity is 1, the highest value.
2. *CDice* is commutative.
3. $CDice(B, B) = 1$.

Many existing similarity measures give the proximity between two sets only. We introduce a novel measurement, called *Cohesion*, which is a similarity measurement between a set and a collection of sets. Suppose S is a set and Q is a set of n sets $Q = \{S_1, S_2, ..., S_n\}$. Then the cohesion between S and Q is defined as follow:

$$\text{Cohesion}(S, Q) = \frac{|S \cap_O T|}{|S \cup_B T|}$$

where T is the bag union of all the sets in Q:

$$T = S_1 \cup_B S_2 \cup_B ... \cup_B S_n.$$

Cohesion can also be expressed as the follows:

$$Cohesion(S,Q)=\frac{\sum_{a_i\in S \text{ and } frequency_T(a_i)>0} (frequency_T(a_i)+1)}{\sum_{S_j\in Q}|S_j|+|S|}$$

where $frequency_T(a_i)$ is the number of occurrences of a_i in T, and $|S_j|$ is the size of S_j. We can use cohesion for measuring proximity of an object to a cluster. Cohesion shows the degree with which an object "sticks" to a cluster.

Example Consider clusters C_1= {{e,h,i},{a,h,i},{b,h,d}} and C_2={{b,g}, {g}, {c,g,h}}, and the object T={a,b,g,h,i}. Let Q_1 and Q_2 be respectively the bag union of objects in C_1 and C_2, i.e., Q_1={a:1, b:1, d:1, e:1, h:3, i:2} and Q_2={b:1, c:1, h:1, g:3}. Thus,

$$Cohesion(T,C_1)=\frac{|T\cap_O Q_1|}{|T\cup_B Q_1|}=\frac{(2+2+4+3)}{14}=0.78$$

and

$$Cohesion(T,C_2)=\frac{|T\cap_O Q_2|}{|T\cup_B Q_2|}=\frac{(2+4+2)}{11}=0.72$$

According to the calculation above, T is inclined more to C_1 than to C_2, that is, T is more similar to objects in C_1 than objects in C_2.

4.3 Our Proposed Algorithm (CLUC)

We apply the above formulation of cohesion and develop a clustering algorithm for categorical dataset [22]. Given a categorical dataset D, we propose a clustering algorithm, called CLUC, that partitions D in such a way that objects in each cluster have the highest cohesion value to the cluster that includes them and this value is not less than the cohesion supplied by the user. Since each object has the highest cohesion with itself,

to avoid generating clusters of identical objects only, it is necessary that a minimum expected cohesion α in [0, 1] be provided by the user.

As in some earlier method, e.g. [21], our algorithm works similar to K-mode algorithm, however, it finds the number of clusters automatically. It consists of two phases: *initialization* and *refinement*. In the first phase, all objects from D are read into the main memory one by one. For each object T, we search for a cluster to which T has the maximum cohesion greater than the threshold α. If such a cluster C_i is found, then T will be assigned to C_i. If no such a cluster is found, a new cluster will be created to which T is assigned. In this case, we refer to T as the "*creator*" of the new cluster. Objects which are creator of new clusters will be marked as "creators". We record for each object, the cluster to which it is assigned.

In the refinement phase, we scan D again. For each object T, we search for the best candidate cluster to which T has the highest cohesion greater than α. If the best cluster C_j is different from C_i (the one to which T is already assigned) we move T to C_j and delete C_i if it becomes empty after moving T. If no such cluster is found and T was not a creator, then we create a new cluster. The refinement phase is repeated until no object changes its clusters and no new cluster is generated. In our experiments, we observed that sometimes the cohesion of an object O, which has created a new cluster C_i, becomes lower than the cohesion α after inclusion of more objects to C_i. If object O generates another new cluster in the refinement phase, all the existing objects in C_i will move to the new generated cluster, this will be repeated in subsequent refinement phases, resulting in increased number of the refinement phases. To control this situation, as a run-

time measure, we restrict each object to be creator of at most one cluster. Objects which are creators are just allowed to move, but not to create new clusters. These steps are formalized in the following algorithm, in which *cluster(T)* returns the cluster to which T is assigned.

Algorithm CLUC(D, α)

Input: Dataset D and the cohesion factor α.

Output: Clustering of D

/* phase 1 –initialization */

1. read the next object T from D;
2. **while** not end of D
3. find a cluster C_i such that $Co(T,C_i) \geq \alpha$ and $Co(T,C_i) \geq Co(T,C_j)$ for all $j \neq i$
4. **if** C_i is found , **then** assign T to C_i;
5. **else** create a new cluster C_n, assign T to C_n, and mark T as "creator";
6. read the next object T from D;
7. **endwhile**

/* phase 2 – Refinement */

8. **repeat**
9. moved = false;
10. read the next object T from D;
11. **while** not end of D
12. find a cluster C_i such that $Co(T,C_i) \geq \alpha$ and $Co(T,C_i) \geq Co(T,C_j)$ for all $j \neq i$
13. C=*cluster(T)*
14. **if** $C_i \neq$ *cluster(T)* **then**

move T to C_i; moved = true and delete C if it is empty

15. **else if** no cluster was found and T was not "creator" **then**

    create a new cluster C_k;

    move T to C_k;

    moved=true;

16. read the next object T from D;

17. **endwhile**

18. **until** moved=false

4.4 Memory Requirement for CLUC

In CLUC, it is not required to keep the whole dataset in the main memory. Only clusters information is kept in the main memory, and each time, it reads and uses one object to perform its task. For each cluster C, it is necessary to record the number of objects it includes, the total sum of the sizes of all objects in C, and a hash table that stores the attribute values in the cluster with their corresponding frequencies. In terms of the size of the hash table, we need 4 bytes for each pair (an attribute value and its frequency) in the hash table. We also keep track of each transaction id and its corresponding cluster id. Thus, the total memory needed is $4I*k+4*|D|$, where I is the total number of attribute values and k is the total number of clusters. In the worst case, when no two objects are identical, we use $4*|D| *(I+1)$ bytes.

4.5. Meaning of Cohesion α

An important point to note about cohesion value α is that it indicates the percentage of attributes that are expected to contribute to a clustering representation, which is used to describe the result. Suppose for a dataset D with m attributes, a user is looking for a clustering in which each cluster is described by f ($f \leq m$) attributes. This can be obtained by taking $\alpha = \frac{f}{m}$. For example, for a dataset D with 16 attributes, for generating clusters to be described by 12 attributes, we use $\alpha = 0.75$ as the cohesion value. The relationship between cohesion and the collection of attributes that represents the clustering result is important as it makes the input parameter α meaningful. This distinguishes our algorithm from previous methods, some of which require more input parameters, which are not in general easy for the user to set, or need some prior knowledge of the dataset.

4.6. Illustrative Example

To illustrate the computation of CLUC, we give an illustrative example here. To this end, we use the same dataset shown in Figure 1.2. Table 4.1 shows the objects in D.

Table 4.1 Dataset D

I_1 = {square , black, small}
I_2 = {triangle, black, big}
I_3 = {triangle, white, big}
I_4 = {square , black , big}
I_5 = {circle, black, medium}
I_6 = {circle, black, small}
I_7 = {square ,white, medium}
I_8 = {triangle , white, medium}

Each object is described by three categorical attributes: shape, color, and size. For having smaller number of clusters (high-level clustering) a small value for α should be considered. Let α=0.30.

- **Initialization Phase :**

1. I_1 is read . Since no cluster exits, it is assigned to a new cluster C_1 and is marked as the creator.
2. I_2 is read . Cohesion (I_2, C_1)=0.33, so I_2 is assigned to C_1.
3. I_3 is read. Cohesion (I_3, C_1)=0.44, so I_3 is assigned to C_1.
4. I_4 is read. Cohesion (I_4, C_1)=0.66, so I_4 is assigned to C_1.
5. I_5 is read. Cohesion (I_5,C_1)=0.26. At this point, I_5 creates a new cluster C_2 and it is marked as its creator.
6. I_6 is read. Cohesion (I_6,C_1)=0.4. Cohesion (I_6,C_2)=0.66. I_6 is assigned to C_2.
7. I_7 is read. Cohesion (I_7,C_1)=0.33. Cohesion (I_7 ,C_2)=0.22. I_7 is assigned to C_1.
8. I_8 is read. Cohesion (I_8,C_1)=0.44. Cohesion (I_8, C_2)=0.22. I_8 is assigned to C_1.

- **Refinement Phase**

After initialization, we scan the dataset again for refinement.

1. I_1 is read. Cohesion (I_1, C_1)=0.33. Cohesion (I_6, C_2)=0.55. I_1 moves to C_2.
2. I_2 is read. Cohesion (I_2,C_1)=0.5. Cohesion (I_2 , C_2)=0.33. I_2 stays in C_1.
3. I_3 is read. Cohesion (I_3,C_1)=0.6. Cohesion (I_3, C_2)=0. I_3 stays in C_1.
4. I_4 is read. Cohesion (I_4,C_1)=0.46. Cohesion (I_4,C_2)=0.5. I_4 moves to C_2.
5. I_5 is read. Cohesion (I_5,C_1)=0.33. Cohesion (I_5, C_2)=0.5. I_5 stays in C_2.

6. I_6 is read. Cohesion (I_6, C_1)=0.13. Cohesion (I_6, C_2)=0.66. I_6 stays in C_2.

7. I_7 is read. Cohesion (I_7, C_1)=0.41. Cohesion (I_7 , C_2)=0.33. I_7 stays in C_1.

8. I_8 is read. Cohesion (I_8, C_1)=0.66. Cohesion (I_8, C_2)=0.13. I_8 stays in C_1.

Since 2 movement happened here, we need to scan the dataset again :

1. I_1 is read. Cohesion (I_1, C_1)=0.26. Cohesion (I_6, C_2)=0.66. I_1 stays in C_2.

2. I_2 is read. Cohesion (I_2, C_1)=0.41. Cohesion (I_2 , C_2)=0.46. I_2 moves to C_1.

3. I_3 is read. Cohesion (I_3, C_1)=0.55. Cohesion (I_3, C_2)=0.27. I_3 stays in C_1.

4. I_4 is read. Cohesion (I_4, C_1)=0.33. Cohesion (I_4, C_2)=0.6 . I_4 stays in C_2.

5. I_5 is read. Cohesion (I_5, C_1)=0.25. Cohesion (I_5, C_2)=0.46. I_5 stays in C_1.

6. I_6 is read. Cohesion (I_6, C_1)=0. Cohesion (I_6, C_2)=0.6. I_6 stays in C_2.

7. I_7 is read. Cohesion (I_7, C_1)=0.55. Cohesion (I_7 , C_2)=0.27. I_7 stays in C_1.

8. I_8 is read. Cohesion (I_8, C_1)=0.77. Cohesion (I_8, C_2)=0.22. I_8 stays C_2.

Since one object was moved, we scan the dataset again.

1. I_1 is read. Cohesion (I_1, C_1)=0.16. Cohesion (I_6, C_2)=0.6. I_1 stays in C_2.

2. I_2 is read. Cohesion (I_2, C_1)=0.41. Cohesion (I_2 , C_2)=0.46. I_2 moves to C_1.

3. I_3 is read. Cohesion (I_3, C_1)=0.55. Cohesion (I_3, C_2)=0.27. I_3 stays in C_1.

4. I_4 is read. Cohesion (I_4, C_1)=0.33. Cohesion (I_4, C_2)=0.6 . I_4 stays in C_2.

5. I_5 is read. Cohesion (I_5, C_1)=0.25. Cohesion (I_5, C_2)=0.46. I_5 stays in C_1.

6. I_6 is read. Cohesion (I_6, C_1)=0. Cohesion (I_6, C_2)=0.6. I_6 stays in C_2.

7. I_7 is read. Cohesion (I_7, C_1)=0.55. Cohesion (I_7 , C_2)=0.27. I_7 stays in C_1.

8. I_8 is read. Cohesion (I_8, C_1)=0.77. Cohesion (I_8, C_2)=0.22. I_8 stays C_2.

Since no object moved between clusters, the clustering process stops here. So with α=0.30, CLUC generates 2 clusters shown in Figure 4.2.

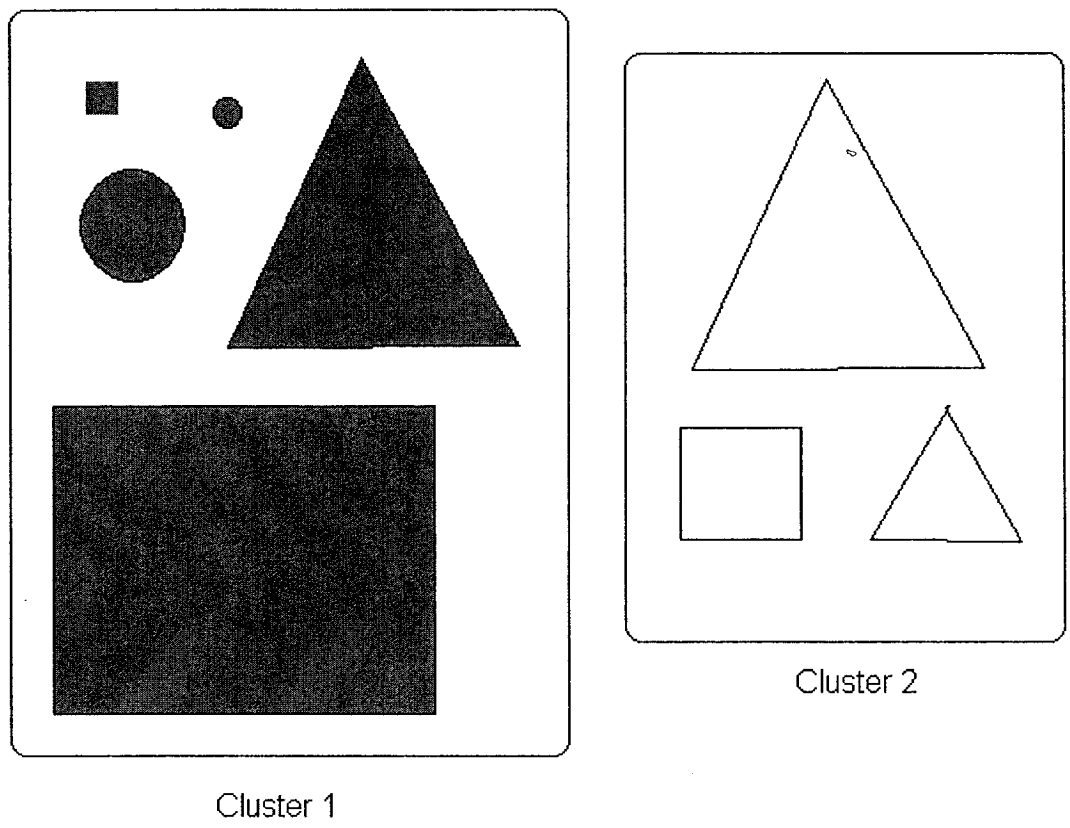

Figure 4.2 Clustering result for α=0.3

As it can be seen in this figure, the clustering is done according to the color of objects. One cluster contains black objects while the other holds white ones. Selecting 0.30 for α means that objects in each cluster should be similar to each other in 0.30*3 = 0.9 attributes which is almost one attribute. In cluster 1, all the objects are recognized by color=black and in cluster 2 by color=white. This example shows that CLUC generates natural and meaningful clusters.

In Figure 4.3, clustering is done with α=0.50. As can be seen in this figure, all the clusters are described with two attributes $(0.50*3=1.5\cong 2)$.

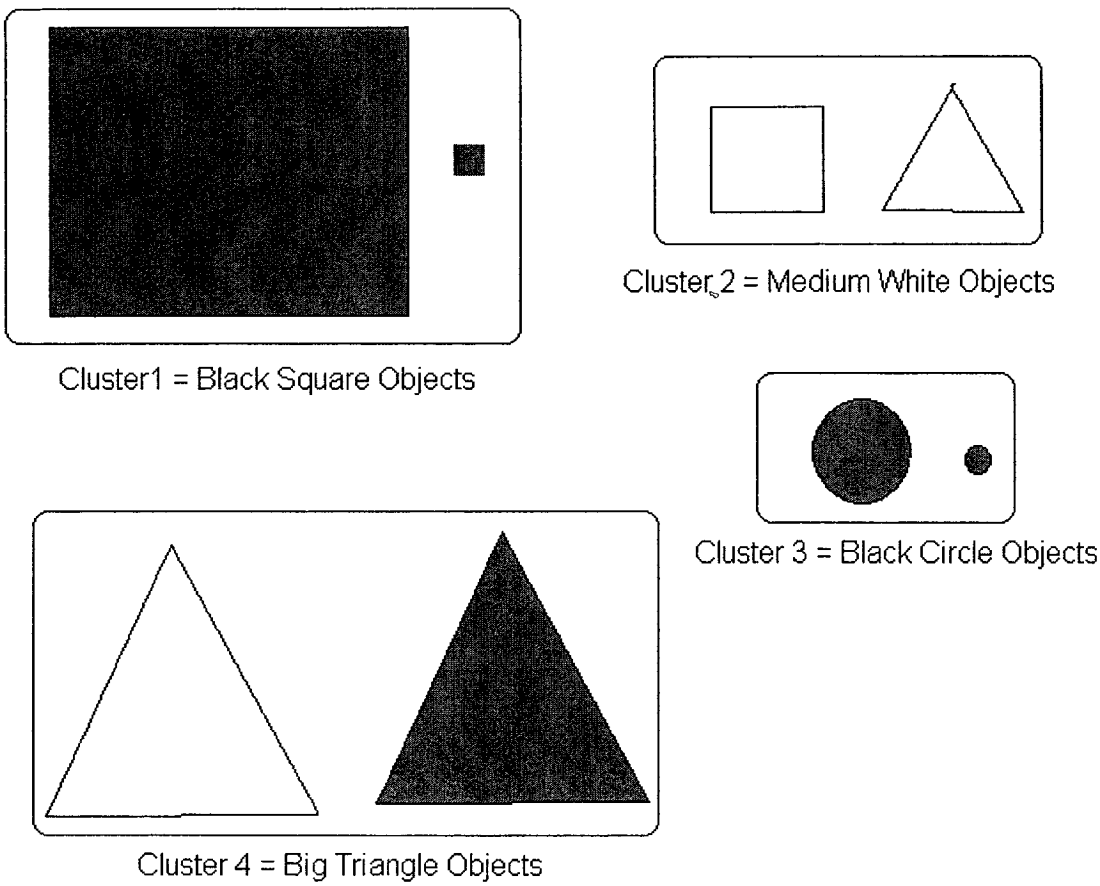

Figure 4.3 Clustering result for α=0.5

CHAPTER 5 Design and Implementation of CLUC

This chapter explains our design and implementation of CLUC. The system architecture, internal structure and organization are described in the design section. Each system component is sufficiently described in the implementation part.

5.1 Design

CLUC includes six main modules. Figure 5.1 presents the top-level structure for our software.

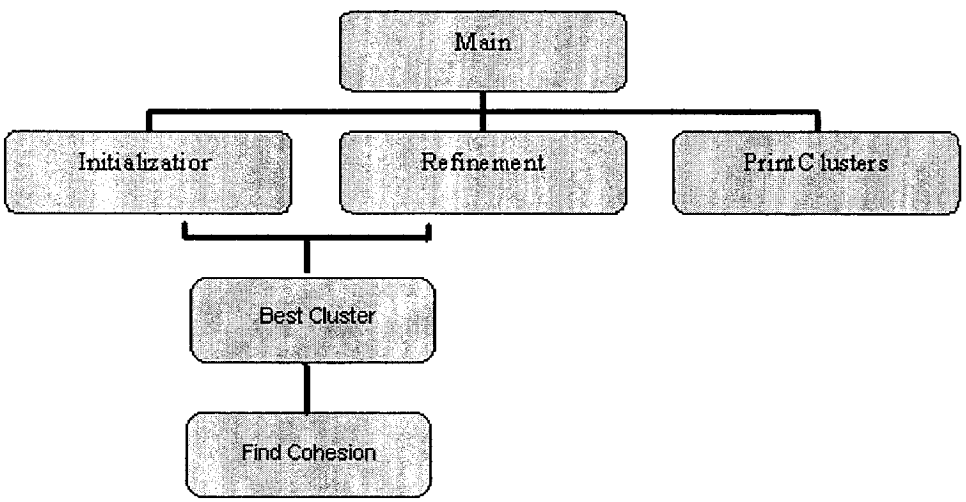

Figure 5.1 The CLUC design

Main Module - This module provides an interface between the user and the system. It gets the inputs (cohesion and the name of the dataset) from the user. It uses initialization

and refinement modules for clustering the dataset and then uses "Print Clusters" module to produce the output files.

Initialization Module - This module scans the dataset and generates the initial clusters. It generates clusters with the specified minimum cohesion received from main module. This module uses the "Best Cluster" module to find the best cluster for an object.

Refinement Module - This module takes the generated cluster obtained by the Initialization module as input and refines it. It does the refinement by iteratively scanning the dataset and relocating each point to the best candidate cluster to which it has the highest cohesion. As in the Initialization module, this module uses the "Best cluster" module.

Print Cluster Module - This module generates two output files: one which describes the information for each cluster and the other which shows the transaction number and its related clusters.

Best Cluster Module. This module has interfaces with initialization and refinement module. It takes a transaction from these modules, computes the cohesion of the transaction to each existing cluster or create a new one using "Find Cohesion" module. It then finds the best cluster for the transaction and returns the cluster ID to the module it was called from.

Find Cohesion Module - This module has only interface with "Best Cluster" module. It gets a transaction and a cluster ID from "Best Cluster" module and computes the cohesion of the transaction to the cluster.

In the following section, we describe an implementation of CLUC.

5.2 Implementation

We used an object-oriented approach to develop CLUC and implemented it with C++ under Unix. The program prompts the user to enter the name of the dataset and the value for cohesion, which should be between 0 and 1. The dataset is a text file with .txt extension. A sample of input file is shown in Figure 5.2.

```
1a,2d,3f,4d,5f,6y
1q,2s,3d,4f,5g,6h
1z,2f,3d,4f,5u,6k
1a,2c,3d,4g,5g,6h
1q,2x,3c,4v,5b,6n
1q,2s,3e,4r,5t,6y
```

Figure 5.2 Sample input dataset

In this figure, each line represents an object described by 6 attributes. Note that each attribute value is tagged with its attribute ID, e.g., 1a is the value of the first attribute. Attribute values are separated from each other by ",".

After clustering, the description of generated clusters will be written to a file (output.txt), the relation between each transaction and the clusters will be printed on another text file (assignment.txt), and the number of clusters are shown on the screen to the user. Figures 5.3 and 5.4 show samples of output files.

```
Transaction 1 assigned to Cluster 2
Transaction 2 assigend to Cluster 3
Transaction 3 assigned to Clustter 1
Transaction 4 assigned to Cluster 2
```

Figure 5.3 Sample assignment file

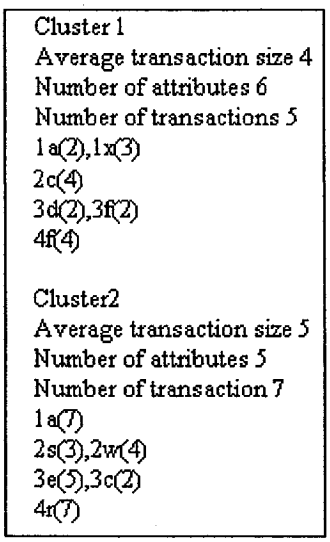

Figure 5.4 Sample output file

The following classes are defined in the implementation:

Class InFile – This is a class to work with input files.

Class OutFile – This class works with output files.

Class Transaction – Each record in the dataset is a transaction object.

Class Hash Table – This is used to work with hash tables and supports the following operations on each hash table object:

- Add key
- Delete key
- Get number of key
- Get the total frequency of keys
- Find a key and return its frequency

Class Cluster – This class is used to work on cluster and implements the following operations:

- Add Transaction
- Delete Transaction
- Get cluster ID
- Get number of transactions
- Get number of attributes
- Get total frequency of attributes

The two important points in our implementation are the following:

- **Data structure for clusters**

For each cluster the following information is needed to be kept in memory:

1. attribute values
2. cluster ID
3. number of transactions
4. total sum of transaction sizes in the cluster.

Since for calculating cohesion, we need to have the frequency of the attribute values in the cluster, a hash table structure is used to speed up maintenance of the attribute values. When a transaction is added to a cluster, its attribute values should also be added to the hash table in that cluster. If the attribute value already exists, its frequency is incremented by 1; otherwise a new entry will be added to the hash table for that attribute value with frequency equal to 1.

- **Allocation of a record to a cluster**

CLUC assigns each record to a cluster. To this end, a one-dimensional array is utilized in the implementation. Each element in the array corresponds to one record in the dataset.

When the ith record is assigned to the jth cluster, the ith element in the array is set to j (cluster id).

CHAPTER 6 Experiments and Results

This chapter presents the experimental evaluation of CLUC algorithm. We evaluated CLUC in two aspects, quality and scalability. For the quality, we compared the results generated by some popular categorical clustering techniques. For this, we considered ROCK and LIMBO, which are hierarchical algorithms, and K-mode which is a partition-based algorithm. We selected COOLCAT as an incremental clustering algorithm, and chose CLICK which is a combination of grid-based and density-based algorithms. We also considered CLOPE which is the most similar categorical clustering algorithm to CLUC. A main difference is the objective function we use. For comparing the quality, we used the same input datasets for all the algorithms we considered. To evaluate scalability, we had no access to large real datasets, instead we run CLUC on various synthetic datasets of different size. We made use of DataGen [32] software to create large synthetic datasets with high dimensions.

6.1 Datasets

We performed extensive experiments on both real-life and synthetic datasets to evaluate CLUC. For quality comparison, we used four well-known real datasets including Mushroom, Congressional Voting, Soybean, and Zoo, described in the following sections, downloaded from the Machine Learning Repository website [31]. To study scalability of CLUC, we considered five large synthetic datasets which we generated.

6.1.1 Mushroom Dataset

This dataset contains information about 23 kinds of mushrooms. Objects in this dataset are classified by experts into edible, poisonous, and unknown edibility. According to [31] unknown mushroom in this dataset are combined with poisonous group and labeled as poisonous. Each attribute describes one of the physical characteristics of the mushroom such as cap-color, cap-shape, odor, etc. Totally, there are 8124 objects in the dataset described using 23 nominal attributes. It should be noted that one of these attributes represents mushroom's type (class name attribute) which is not used in our clustering, for the obvious reason. Only one attribute (stalk-root) has missing values in 2480 records. Table 6.1 shows the attributes and their values. There are 4208 edible and 3916 poisonous mushrooms in the dataset, according to the classification by human expert.

Table 6.1 Description of Mushroom dataset

Attribute Number	Attribute Name	Attribute Values
1	cap-shape	bell=b,conical=c,convex=x,flat=f,knobbed=k, sunken=s
2	cap-surface	fibrous=f,grooves=g,scaly=y,smooth=s
3	cap-color	brown=n,buff=b,cinnamon=c,gray=g,green=r,pink= p, purple=u,red=e,white=w,yellow=y
4	bruises	bruises=t,no=f
5	odor	almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6	gill-attachment	attached=a,descending=d,free=f,notched=n
7	gill-spacing	close=c,crowded=w,distant=d
8	gill-size	broad=b,narrow=n
9	gill-color	black=k,brown=n,buff=b,chocolate=h,gray=g,white =w, green=r,orange=o,pink=p,purple=u,red=e, yellow=y
10	stalk-shape	enlarging=e,tapering=t
11	stalk-root	bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z, rooted=r
12	stalk-surface-above-ring	ibrous=f,scaly=y,silky=k,smooth=s
13	stalk-surface-below-ring	ibrous=f,scaly=y,silky=k,smooth=s
14	stalk-color-above-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15	talk-color-below-ring	brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16	veil-type	partial=p,universal=u
17	veil-color	brown=n,orange=o,white=w,yellow=y
18	ring-number	none=n,one=o,two=t
19	ring-type	cobwebby=c,evanescent=e,flaring=f,large=l, none=n, pendant=p,sheathing=s,zone=z
20	spore-print-color	black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21	population	abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22	habitat	grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d
23	Class Name	edible, poisonous

6.1.2 Congressional Votes Dataset

This dataset includes one of the United States Congressional Voting Records in 1984. Each record contains the votes for one congress man. The votes are on 16 issues such as duty free export, immigration, etc. The answer to each vote is yes, no, or the value is missing. A missing value in this dataset means that the answer to that vote is neither yes nor no. So in our clustering, we consider each missing value as a particular vote, different from yes or no. The total number of tuples in this dataset is 435, where 168 votes are from Republicans and 267 votes from Democrats. Table 6.2 describes attribute names and values in this dataset. The first attribute is the "class name", which could be either democrat or republic. This attribute is not used in the clustering process for the obvious reason. There are 288 missing values in this dataset.

Table 6.2 Description of Congressional Votes dataset

Attribute number	Attribute Name
1	Class Name
2	handicapped-infants
3	water-project-cost-sharing
4	adoption-of-the-budget-resolution
5	physician-fee-freeze
6	el-Salvador-aid
7	religious-groups-in-schools
8	anti-satellite-test-ban
9	aid-to-Nicaraguan-contras
10	mix-missile
11	immigration
12	synfuels-corporation-cutback
13	education-spending
14	superfund-right-to-sue
15	crime
16	duty-free-exports
17	export-administration-act-south-Africa

6.1.3 Soybean Disease Dataset

This dataset is taken from [31] and contains information about soybean diseases. It includes 47 tuples classified into 4 groups: Diaporthe Stem Canker (D1), Charcoal Rot (D2), Rhizoctonia Root Rot (D3), and Phytophthora Rot (D4) with the following distribution: 10 records for D1, 10 records for D2, 10 records for D3, 17 records for D4. Each disease is described by 35 attributes without missing values. Table 6.3 describes the soybean disease dataset. The last attribute, called "class name" (the 36^{th} attribute) is not used in the clustering algorithm.

Table 6.3 Description of Soybean dataset

Attribute Number	Attribute Name	Attribute value
1	date	april,may,june,july,august,september,october
2	plant-stand	normal,lt-normal
3	precip	lt-norm,norm,gt-norm
4	temp	lt-norm,norm,gt-norm
5	hail	yes,no
6	crop-hist	diff-lst-year,same-lst-yr,same-lst-two-yrs,same-lst-sev-yrs
7	area-damaged	scattered,low-areas,upper-areas,whole-field
8	severity	minor,pot-severe,severe
9	seed-tmt	none,fungicide,other
10	germination	90-100%,80-89%,lt-80%
11	plant-growth	norm,abnorm
12	leaves	norm,abnorm
13	leafspots-halo	absent,yellow-halos,no-yellow-halos
14	leafspots-marg	w-s-marg,no-w-s-marg,dna
15	leafspot-size	lt-1/8,gt-1/8,dna
16	leaf-shread	absent,present
17	leaf-malf	absent,present
18	leaf-mild	absent,upper-surf,lower-surf
19	stem	norm,abnorm

20	lodging	yes,no
21	stem-cankers	absent,below-soil,above-soil,above-sec-nde
22	canker-lesion	dna,brown,dk-brown-blk,tan
23	fruiting-bodies	absent,present
24	external decay	absent,firm-and-dry,watery
25	mycelium	absent,present
26	int-discolor	none,brown,black
27	sclerotia	absent,present
28	fruit-pods	norm,diseased,few-present,dna
29	fruit spots	absent,colored,brown-w/blk-specks,distort,dna
30	seed	norm,abnorm
31	mold-growth	absent,present
32	seed-discolor	absent,present
33	seed-size	norm,lt-norm
34	shriveling	absent,present
35	roots	norm,rotted,galls-cysts
36	Class name	D1,D2,D3,D4

6.1.4 Zoo Dataset

This dataset, taken from [31], contains information about 101 animals. There are 18 attributes in the dataset, one of which is the name of the animal, 15 attributes have Boolean values, and 2 are numeric. The attributes describe the characteristics of each animal, for example, being aquatic, having feather, number of legs, etc. One of the numeric attributes is the *number of legs*. Although numeric, we treated such attribute as categorical, since we can not apply numeric proximity measure on it. For example, for three animals with 2,4,8 legs respectively, we can not say that the similarity between two animals with 2 and 4 legs is higher than the similar between two animals with 2 and 8 legs. Another numeric attribute is the last attribute, called class (the 18^{th} attribute here

defined by type) and is not used in the clustering procedure. Table 6.4 describes the attribute names and their values.

Animals are classified into 7 groups. Table 6.5 indicates which animal belongs to which class. There are no missing values in this dataset.

Table 6.4 Description of Zoo dataset

Attribute Number	**Attibute name**	**Attribute vlaue**
1	animal name	Unique for each instance
2	hair	Boolean
3	feathers	Boolean
4	eggs	Boolean
5	milk	Boolean
6	airborne	Boolean
7	aquatic	Boolean
8	predator	Boolean
9	toothed	Boolean
10	backbone	Boolean
11	breathes	Boolean
12	venomous	Boolean
13	fins	Boolean
14	legs	Numeric (set of values: {0,2,4,5,6,8})
15	tail	Boolean
16	domestic	Boolean
17	catsize	Boolean
18	Type	Numeric (set of values: {1,2,...,7})

Table 6.5 Class distribution for Zoo dataset

Class Number	Number of data	Class distribution
1	41	aardvark, antelope, bear, boar, buffalo, calf, cavy, cheetah, deer, dolphin, elephant, fruitbat, giraffe, girl, goat, gorilla, hamster, hare, leopard, lion, lynx, mink, mole, mongoose,pony, opossum, oryx, platypus, polecat, porpoise, puma, pussycat, raccoon, reindeer,seal, sealion, squirrel, vampire, vole, wallaby,wolf
2	20	chicken, crow, dove, duck, flamingo, gull, hawk,kiwi, lark, ostrich, parakeet, penguin, pheasant,rhea, skimmer, skua, sparrow, swan, vulture, wren
3	5	pitviper, seasnake, slowworm, tortoise, tuatara
4	13	bass, carp, catfish, chub, dogfish, haddock, herring, pike, piranha, seahorse, sole, stingray, tuna
5	4	frog, frog, newt, toad
6	8	flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp
7	10	clam, crab, crayfish, lobster, octopus,scorpion, seawasp, slug, starfish, worm

6.1.5 Synthetic Datasets

To evaluate scalability of CLUC, we need large and high dimensional datasets in terms of both number of records and attributes. To this end, we generated synthetic categorical datasets using the DataGen program obtained from [31]. This program generates datasets according to the specifications given by the user such as number of tuples, number of attributes, attribute domain size, etc. In our experiments, we produced five datasets with different specifications described in Table 6.6

Dataset D1 is generated with 500,000 tuples, 12 attributes with 20 to 40 different values each. Datasets D2, D3, D4, and D5 include 100,000 tuples and 10, 20, 30, 40 numbers of attributes respectively with 10 to 15 different values each.

Table 6.6 Description of synthetic datasets

Datasets	No. of Tuples	No. of Attributes	Attribute Domain Values	Total Size in MB
D1	500,000	12	20-40	26
D2	100,000	10	10-15	5
D3	100,000	20	10-15	10
D4	100,000	30	10-15	14
D5	100,000	40	10-15	19

6.2 Cluster Quality Measurements

The goal of a clustering method is to generate high quality clusterings. Following the ideas proposed in [1], we define *quality* based on the number of misclassified objects. Suppose D is a given dataset for which a clustering is defined by experts. Let $G = \{G_1, G_2, ..., G_h\}$ be a clustering of D generated by a clustering algorithm. The number of clusters in G may be different from the number of clusters in the clustering $C = \{C_1, C_2, .., C_k\}$ defined by a human expert.

We define a mapping $\xi : G \to C$ such that for all $i \in \{1, 2, ..., h\}$, there exists a $j \in \{1, 2, ...k\}$, $\xi(G_i) = C_j$, where $|G_i \cap C_j| \geq |G_i \cap C_p|$, for all $p \in \{1, 2, ..., k\}$.

Misclassification error E_c is defined as:

$$E_c = 1 - (\sum_{i=1}^{k} |G_i \cap \xi(G_i)| / n) \quad (3)$$

where n is the total number of objects in the dataset D [1]. E_c shows the percentage of data assigned to wrong clusters, so a lower value for E_c indicates a higher quality clustering. The following example illustrates this formula.

Example : Suppose $D=\{O_1,O_2,O_3,O_4,O_5,O_6,O_7,O_8,O_9,O_{10}\}$. Let C be the following clustering of D defined by a human expert:

$$C= \{C_1,C_2\},\ C_1=\{O_1,O_3,O_4,O_7,O_8\},\ C_2= \{O_2,O_5,O_6,O_9,O_{10}\}.$$

Now suppose $G=\{G_1,G_2,G_3\}$ is a clustering generated by an algorithm where :

$$G_1=\{O_2,O_3,O_4\},\ G_2=\{O_5,O_7,O_8,O_1\},\ G_3=\{O_6,O_9,O_{10}\}$$

In this case, the best mapping ξ from G to C will be:

$$\xi(G_1) = C_1,\ \xi(G_2) = C_2,\ \xi(G_3) = C_2$$

Since there are two objects O_2 and O_1 misclassified in G, we have that $E_c = 0.2$.

Suppose for a given dataset D, two clusterings C_1 and C_2 are generated with the same number of clusters. Clustering C_1 is preferred to C_2 if C_1 has a lower misclassification error. Misclassification error is not an appropriate measure for evaluating two clusterings with different number of clusters, since it does not reflect compactness of a clustering in terms of the number of generated clusters. For example, in an extreme case of clustering in which the size of each cluster generated is 1, the misclassification error is zero, however this clustering need not be a desirable and useful one.

The authors in [14] use class entropy to evaluate clusterings with various sizes as follows. Let $G=\{G_1,G_2,...,G_h\}$ be a clustering of D and $C=\{C_1,C_2,...,C_k\}$ is a

clustering of D defined by an expert. Then for each cluster C_j, $1 \le j \le k$, a class entropy is defined as:

$$EC_j = -\sum_{j=1}^{h} \frac{n(C_j, G_i)}{n(C_j)} \log \frac{n(C_j, G_i)}{n(C_j)}$$

where $n(C_j, G_i)$ is the number of objects in G_i with label C_j, and $n(C_j)$ is the number of objects with label C_j in the dataset D. The overall class entropy EC for clustering G is defined as :

$$EC = \frac{1}{\sum_{j=1}^{k} n(C_j)} \sum_{j=1}^{k} n(C_j) EC_j$$

When the number of clusters increases, the overall class entropy increases as well. So, for two clustering G_1 and G_2 with the same misclassification error, the one with less number of clusters is preferred. The lower the overall class entropy the better the clustering is.

6.2.1 Experimental Results on Mushroom

We ran CLUC on the Mushroom dataset with different inputs cohesion α. With $\alpha = 0.3$, CLUC partitions objects into two clusters: cluster 1 with 816 poison and 4122 edible, and cluster 2 with 3100 poison and 86 edible. In cluster 1, the numbers of edible objects is almost 5 times more than the poisonous ones and in cluster 2, the number of poisonous is 36 times more than the edible ones. In this case, we have that E_c=0.11, and the number of clusters generated is the same as the number of classes defined by experts on Mushroom. For $\alpha = 0.35$, CLUC generated three clusters on

Mushroom dataset. This clustering is also meaningful, since mushrooms are known to belong to three kinds: edible, poisonous, and unknown, but the unknown mushrooms have been given poisonous label in this dataset. By choosing 0.73 for α, CLUC generated 24 pure clusters which is very close to the 23 species of mushrooms in this dataset. It is interesting and useful for the human expert to verify the clustering suggested by CLUC, compared to the 23 clusters. This means CLUC could potentially be a useful tool for further enhance human expertise. For $\alpha = 1$, the number of clusters is equal to the size of this dataset. This also indicates that there were no duplicated objects in the dataset.

For quality evaluation, we compared the result generated by CLUC with those produced by two groups of algorithms, namely those which need the number of clusters as input, and those which find the number of clusters automatically. For the first group, we considered LIMBO, ROCK, and COOLCAT. As mentioned in Section 2, LIMBO and ROCK are hierarchical algorithms, while COOLCAT is an incremental partition-based algorithm. In these algorithms, the number of clusters is specified by the user. For the second group, we considered CLOPE and CLICK which find the number of clusters automatically. CLICK is a combination of grid-based and density-based algorithms. We should mention that CLOPE is the most similar algorithm to our CLUC.

First, we compared the quality of generated clusters by CLUC with those generated by LIMBO, COOLCAT and ROCK. For comparison to be fair and useful, we compared those which produced the same number of clusters. Since the number of predefined clusters on the Mushroom dataset is two (edible and poisonous), we run LIMBO, COOLCAT and ROCK with k=2. For CLUC, we ran it with different values for α, and found out that for $\alpha = 0.3$, CLUC generates two clusters. Table 6.7 shows the

results of our experiments with all these algorithms on the Mushroom dataset. We should mention that the results for LIMBO, COOLCAT and ROCK are taken from [1]. In this table, the input parameters for these algorithms were as follow: LIMBO clustered this dataset with specifying the size of available memory to be equal to 128KB. In ROCK, Jaccard coefficient was used for similarity function, and the neighborhood radius parameter θ was assigned to 0.8. COOLCAT was run on this dataset with 1000 sample points.

By comparing the results in this table, we can see that CLUC generates better quality clusters compared to COOLCAT and ROCK. Although CLUC generates the same clustering as LIMBO, our algorihtm is superior since it finds the correct number of clusters automatically.

Table 6.7 Results from different algorithms on Mushroom

Algorithm	**Two Clusters**	**E_C**
CLUC ($\alpha = 0.3$)	Automatic	0.11
$LIMBO_S$ (128 KB)	User-defined	0.11
COOLCAT(s=1000)	User-defined	0.27
ROCK(θ=0.8)	User-defined	0.43

We also compare CLUC to CLOPE and CLICK, both of which find the number of clusters automatically. Table 6.8 shows the results of these algorithms when they all generate pure clusters, i.e., E_c = 0. As can be seen from this table, CLUC was able to generate 24 clusters; however, CLOPE and CLICK generated 30 and 213 clusters,

respectively. Since CLUC generates the minimum number of pure clusters, its overall class entropy is less than the others and hence preferred.

Table 6.8 Clustering on Mushroom ($E_C = 0$)

Algorithm	**Number of Clusters**
CLUC(α=0.73)	24
CLOPE (r= 3.1)	30
CLICK (α =0.4, minsup=0.5)	213

6.2.2 Experimental Results on Congressional Votes

The Congressional Vote dataset has 16 boolean attributes. The potential number of objects that can be described by these attributes is $2^{16} = 65536$. Comparing this with the size of the dataset, which is 435, shows that the dataset is sparse. For this reason we set the cohesion value to 0.30 when running CLUC on this dataset.

Table 6.9 describes the results of clustering Congressional Vote dataset using CLUC, LIMBO, COOLCAT, and ROCK. In this table, the results for LIMBO, COOLCAT, and ROCK are taken from [1]. For LIMBO, the memory bound value given as the input was set to 128KB. In COOLCAT, clustering was performed by taking 435 objects as the sample of data. In ROCK, θ was set to 0.7 and Jaccard coefficient was used as the similarity function. As Table 6.9 illustrates, CLUC generates better clustering than COOLCAT and ROCK in terms of quality. Both CLUC and LIMBO produced high quality clusters but CLUC is superior to LIMBO since it generates the exact number of desired clusters automatically.

Table 6.9 Results from different algorithms on Congressional Votes

Algorithm	Two Clusters	E_C
CLUC ($\alpha = 0.30$)	Automatic	0.13
LIMBO $_S$(128 KB)	User-defined	0.13
COOLCAT(S = 435)	User-defined	0.15
ROCK($\theta = 0.7$)	User-defined	0.16

6.2.3 Experimental Results on Soybean

Table 6.10 contains the results of running CLUC and CLOPE on the Soybean dataset. For CLUC, the cohesion value α used was 0.73 and the repulsion value r used in CLOPE was 2.8. The number of clusters in Soybean specified by human expert is 4. As it can be seen from this table, CLUC generated exactly 4 clusters, but CLOPE produced five.

Table 6.10 Clustering on Soybean ($E_C = 0$)

Algorithm	Number of Clusters
CLUC (α =0.73)	4
CLOPE (r = 2.8)	5

6.2.4 Experimental Results on Zoo

We also ran CLUC with different cohesion values on the Zoo dataset. Using α=0.45 as the cohesion, CLUC partitioned the animals into two groups: mammals and non-mammals. This partitioning is the one that would be naturally given by a human

expert. In order to generate 7 clusters, which is the number of classes defined also by experts, CLUC was run with different minimum cohesion. Running CLUC with α =0.67 generated 7 clusters with $E = 0.1$ misclassification error. For α =0.70, CLUC was able to find 10 clusters with $E_c = 0.06$ as the classification error. We analyzed the resulting clusters, and noted the following interesting cluster C among others:

C= {slug, worm, flea, gnat, honeybee, housefly, ladybird, moth, termite, wasp}, explained as follows.

According to a classification by experts, flea, gnat, honeybee, housefly, ladybird, moth, termite, and wasp are from the same group called group 6, but slug and worm belong to another group which is group 7. Tables 6.11 and 6.12 compare the similarity between slug and worm with animals in groups 6 and 7, respectively.

Table 6.11 Comparing slug and worm with animals in group 6

Animal Name	Hair	Feather	Eggs	Milk	Airbone	Aquatic	Predator	Toothed	Backbone	Breaths	Venomous	Fins	Legs	Tail	Domestic	Catsize
flea	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0	0
gnat	0	0	1	0	1	0	0	0	0	1	0	0	6	0	0	0
honeybee	1	0	1	0	1	0	0	0	0	1	1	0	6	0	1	0
housefly	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0	0
ladybird	0	0	1	0	1	0	1	0	0	1	0	0	6	0	0	0
moth	1	0	1	0	1	0	0	0	0	1	0	0	6	0	0	0
termite	0	0	1	0	0	0	0	0	0	1	0	0	6	0	0	0
wasp	1	0	1	0	1	0	0	0	0	1	1	0	6	0	0	0
worm	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
slug	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0

Table 6.12 Comparing slug and worm with animals in group 7

Animal Name	Hair	Feather	Eggs	Milk	Airbone	Aquatic	Predator	Toothed	Backbone	Breaths	Venomous	Fins	Legs	Tail	Domestic	Catsize
clam	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
crab	0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0
crayfish	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0
lobster	0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0
octopus	0	0	1	0	0	1	1	0	0	0	0	0	8	0	0	1
scorpion	0	0	0	0	0	0	1	0	0	1	1	0	8	1	0	0
seawasp	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0
starfish	0	0	1	0	0	1	1	0	0	0	0	0	5	0	0	0
slug	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
worm	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0

As Tables 6.11 and 6.12 indicate, worm and slug share exactly 10 attributes with animals in group 6, while they have 8 common attributes with animals in group 7. In this case, the misclassification error is high because of specific classification done by experts on this dataset.

6.3 Scalability

To investigate scalability of CLUC, we performed experiments on different large and high dimensional synthetic datasets, since large real datasets were not available. We first studied how the number of objects affects the execution time. For this, we considered D1 (Table 6.6) and three arbitrary subsets of it, called D12, D13, D14 with 10,000, 50,000, and 100,000 objects respectively. Figure 6.1 shows our performance results.

As it can be seen in the figure, increasing the number of objects increases the execution time linearly. This indicates that CLUC is scalable when the input dataset

grows in size. Also note that when the value of cohesion (α) increases, the execution time increases accordingly, as there could be more clusters to generate.

To study the impact of the number of attributes on the execution time, we used four other synthetic datasets: D2, D3, D4 and D5. We also select three subsets of each with different number of objects to study changes both in the number of attributes and in the number of objects at the same time. The results are presented in Figure 6.2. As can be seen, the increase in execution time is linear, indicating that CLUC is scalable on high-dimensional datasets. In all our experiments, CLUC generated exactly 10 clusters which correspond to the number of rules used to generate these datasets. In addition, all these generated clusters were completely pure, with no misclassification error.

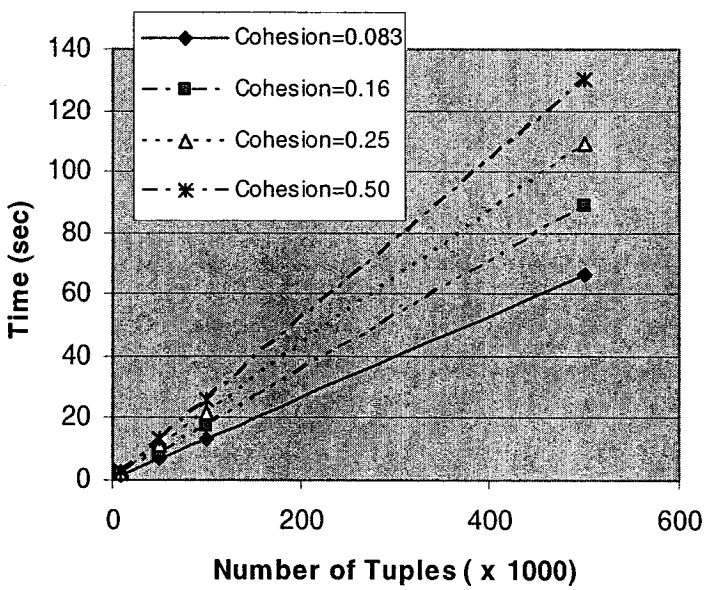

Figure 6.1 Execution time vs. number of objects

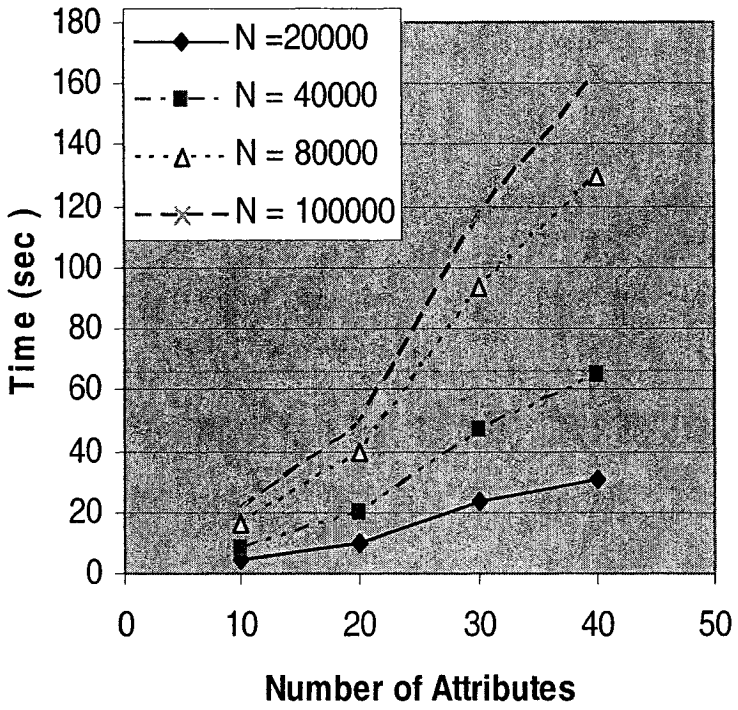

Figure 6.2 Execution time vs. number of attributes

6.4 Order Dependency

We have conducted several experiments to study the impact of the order of objects in the input dataset on performance and the quality of clustering generated by CLUC.

Taking $\alpha = 0.75$, we run CLUC 100 times using 100 different input orders of the Soybean dataset. We observed that in 98 cases, CLUC generated four clusters, and only in 2 cases the number of produced clusters was five. In 85 cases, pure clusters were generated, and in 15 cases, only one object was misclassified. We also observed that iterations in the refinement phase vary when working on different orders of the input.

We ran CLUC 50 times on the Mushroom dataset for $\alpha = 0.73$ using different orders of objects. The orders of objects were created randomly. In 8 cases, CLUC generated pure clusters, and in 42 cases, the misclassification error was 0.01 ($E_C = 0.01$). Table 6.13 shows the number of resulting clusters in these 50 experiments.

We also run CLUC 20 times on Mushroom for $\alpha = 0.30$. Two clusters were generated in 14 cases, and in the rest, the number of resulting clusters was 3. In all these cases, the misclassification value was 0.11.

Our experiments on the Soybean and Mushroom datasets indicate CLUC is almost insensitive to the particular order of the input dataset, making it a robust technique.

Table 6.13 Number of generated clusters with CLUC on Mushroom

Cases	**Number of clusters**
9	24
29	23
9	22
3	25

ROCK and CLICK are not sensitive to the order of input data. The authors in [29] claim that CLOPE is not very sensitive to the input order. We believe that in LIMBO, the input order affects the DCF tree and consequently it will affect the generated result. K-mode is a partition-based algorithm so the clustering result depends to the order of records in the dataset.

6.5 Interpreting Clusters

One important aspect in a clustering algorithm is to generate interpretable results. As mentioned before, cohesion implies the minimum percentage of number of attributes that contribute to cluster representations. For the Soybean dataset, which has 35 attributes, we considered $\alpha = 0.75$. This means, we expect that at least 26 attribute values (35* 0.75=26.25) describe each cluster and more than half of the objects in each cluster share those 26 values. If the number of these attributes is more than 26, then we will select the 26 attribute values with highest frequency. Table 6.14 shows the Soybean clusters representation. As it can be seen, only some of the attributes are selected to describe the clusters.

Table 6.14 Interpreting Soybean clusters

Clusters	Representative Attributes
Cluster1	plant-stand = normal , precip = gt-norm , temp = norm ,plant-growth = abnorm , leaves = abnorm ,leafspots-halo=absent ,leafspots-marg = dnaleafspot-size = dna , leaf-shread =absent , leaf-malf = absent, leaf-mild = absent, stem = abnorm , stem-cankers = above-sec-nde ,fruiting-bodies =present , external decay = firm-and-dry ,mycelium = absent, int-discolor = none, sclerotia =absent, fruit-pods = norm, fruit spots = dna, seed = norm,mold-growth = absent, seed-discolor = absent,seed-size = norm , shriveling = absent, roots = norm,
Cluster2	plant-stand = normal, precip = lt-norm,severity = pot-severe, plant-growth = abnorm,leaves = abnorm, leafspots-halo = absent, leafspots-marg = dna, leafspot-size = dna,leaf-shread = absent, leaf-malf = absent,leaf-mild =absent, stem = abnorm,stem-cankers = absent, canker-lesion = tan, fruiting-bodies = absent, external decay = absent, mycelium = absent, int-discolor = black, sclerotia =present, fruit-pods = norm,fruit spots = dna , seed = norm,mold-growth = absent, seed-discolor =

	absent,seed-size =norm, shriveling = absent, roots = norm,
Cluster3	Precip = gt-norm, temp = lt-norm ,area-damaged = low-areas , plant-growth =abnorm, leaves = norm, leafspots-halo = absent,leafspots-marg = dna, leafspot-size = dna, leaf-shread = absent, leaf-malf = absent, leaf-mild = absent, stem = abnorm, stem-cankers = below-soil, canker-lesion = brown, fruiting-bodies =absent, external decay = firm-and-dry, int-discolor = none, sclerotia = absent, fruit-pods = dna , fruit spots = dna, seed = norm , mold-growth = absent,seed-discolor = absent , seed-size = norm,shriveling = absent, roots = norm,
Cluster4	plant-stand = lt-normal, precip = gt-norm,area-damaged = low-areas, plant-growth = abnorm,leaves = abnorm , leafspots-halo = absent, leafspots-marg = dna, leafspot-size = dna,leaf-shread = absent, leaf-malf = absent,leaf-mild =absent , stem = abnorm,lodging = yes, canker-lesion = dna, fruiting-bodies =absent, mycelium = absent, int-discolor = none, sclerotia = absent, fruit-pods = dna, fruit spots = dna, seed = norm, mold-growth = absent, seed-discolor = absent, seed-size = norm,shriveling = absent, roots = rotted,

6.6 The Convergence Issue

To study convergence of CLUC and the average number of scans to terminate CLUC, we performed several experiments. To this end, we ran CLUC on Mushroom, Congressional Vote, Soybean, and the Zoo datasets with different values for α.Tables 6.15, 6.16, 6.17, and 6.18 show the results obtained on these datasets, respectively. As can be seen in Tables 6.17 and 6.18, the average number of scans for Soybean and Zoo is 3. However, in some cases on the Mushroom and Congressional Vote datasets, the number of scans increased to more than 6 times. In these cases, the clusters are not well-separated. Furthermore, we noted that objects move between two clusters which had

overlaps in some dimensions. This was a main reason for increasing the number of iterations in the refinement phase in CLUC. For large dataset, unless we have priori knowledge about the shape and overlapping of clusters, CLUC may take longer to converge. There are ways to overcome or reduce this. One way is the idea of "creator" objects for a cluster which we already introduced. As mentioned before, a creator object can not generates more than one cluster.

Table 6.15 Results from clustering of Mushroom by CLUC

Cohesion α	Number of Scans	Number of Clusters
0.1	1	1
0.2	1	1
0.3	6	2
0.4	7	7
0.5	11	11
0.6	3	17
0.7	3	21
0.8	8	56
0.9	16	828
1	2	8124

Table 6.16 Results from clustering of Congressional Votes by CLUC

Cohesion α	Number of Scans	Number of clusters
0.1	2	4
0.2	2	4
0.3	3	5
0.4	10	8
0.5	7	12
0.6	10	20
0.7	10	44
0.8	9	91
0.9	7	210
1	2	342

Table 6.17 Results from clustering of Soybean by CLUC

Cohesion α	Number of Scans	Number of clusters
0.1	2	1
0.2	2	1
0.3	2	1
0.4	2	1
0.5	2	1
0.6	2	1
0.7	2	3
0.8	3	7
0.9	3	23
1	2	47

Table 6.18 Results from clustering of ZOO by CLUC

Cohesion α	Number of Scans	Number of clusters
0.1	2	1
0.2	2	1
0.3	2	1
0.5	3	3
0.6	4	5
0.7	4	9
0.7	4	13
0.8	3	17
0.9	2	58
1	2	101

6.7 Handling and Detecting Noise

Since CLUC is based on cohesion, an object which does not have the enough similarity to the objects in existing clusters will generate a cluster by itself. In all the above experiments, at the end of the algorithm, those clusters which have one or two objects can be considered as noise and ignored. This of course is user and application dependent.

CHAPTER 7

Conclusions and Future Work

Our objective in this dissertation was to develop a clustering algorithm for categorical datasets . To this end, we first defined a novel similarity measurement called cohesion for cateogrical data. Unlike other pairwise similarity measures, cohesion is a proximity measure between a point and clusters.

We developed CLUC, a novel algorithm for clustering categorical data, based on cohesion which in a way controls the degree to which objects stick together in a cluster. Advantages of CLUC is that it does not need the number of clusters in advance, and that it is able to find natural number of clusters automatically. CLUC takes just one input defined by the user. This is the cohesion which cotrols the degree of similarity among objects in clusters. This input has a value in [0,1] and is meaningful for the user to decide and set, and is intuitively more appealing than the inputs required in other clustering methods.

Our various experiments and analyses of the results show that CLUC generates natural clusters compared to existing methods on real datasets. We also observed that CLUC satisfies the following properties :

- It is scalable to large dataset.
- It has ability to cluster high dimensional datasets.
- It can detect noise.

- The quality of generated clusters are not affected by the order of objects in the dataset.
- It takes only one input parameter which is also intuitive to users and easy to select.
- The generated results are interpretable and meaningful.
- It is a one-tuple-at-a-time process and does not need the entire dataset in the main memory to process.
- It requires a small memory to keep cluster description during the run-time.

CLUC tries to assign objects to clusters to which they have the highest cohesion. From this point of view, CLUC is similar to K-mode as it uses a local optimization. However, unlike K-mode, CLUC does not require the number of clusters in advance and there is no seed for the initial clustering. In contrast to K-mode, CLUC does not consider any special object as a cluster representative. Each time, the cohesion between an object and the entire cluster is measured, and not just similarity between the object and the cluster's representative. On the other hand, CLUC is similar to CLOPE in that they both find the number of clusters automatically. Considering the quality, CLUC generates high quality clustering that is similar to LIMBO, a recently proposed technique which is hierarchical, unlike CLUC.

Further work is required to study, other issues including:

- Improving efficiency of our algorithm. We should find a way to limit scanning of the dataset to two or three times for all datasets. Since convergence of CLUC in some cases showed to require many iterations, finding some information through a pre-processing phase to guide the run-time is an important research direction.
- Applying CLUC to the dataset with mixed types of data, categorical and numerical.

REFERENCES

[1] P. Andritsos, P. Tsaparas, R. Miller and K. C. Sevcik, LIMBO: Scalable Clustering of Categorical Data, The 9th International Conference on Extending DataBase Technology (EDBT), Mar 2004.

[2] M. Ankerst, M. Breunig H. , P. Kriegel, and J. Sander, OPTICS: Ordering Points to Identify the Clustering Structure, Proceedings of ACM SIGMOD'99, Philadelphia, PA, ACM Press, New York, pages 49-60, 1999.

[3] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, Automatic subspace clustering of high dimensional data for data mining applications, Proceedings of ACM-SIGMOD International Conference on Management of Data, Seattle, Washington, June 1998.

[4] D. Barbara, Y. Li, and J. Couto, COOLCAT: An Entropy-based Algorithm for Categorical clustering, ACM Press, pages 582-589, 2002.

[5] M. Chen, J. Han, and P.S. Yu, Data mining: An overview from a database perspective, IEEE Transactions on Knowledge and Data Engineering, pages 866-883, December 1996.

[6] S. Connell and A.K. Jain: Learning Prototypes for on-line handwritten digits, Proceedings of the 14th International Conference on Pattern Recognition. Brisbane,Autralia, pages 182-184, 1998.

[7] A. P. Dempster, N. M. Laird, and D. B. Rubin, Maximum likelihood from incomplete data via the EM algorithm, Journal of the Royal Statistical Society, Series B, 34:1--38, 1977.

[8] M. Ester, H. Kriegel, J. Sander, and X. Xu, A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise, Proceedings of 2nd International Conference on KDD, 1996.

[9] Y. Gan, Clustering Algorithms for Data and Knowledge Exploration, PHD Thesis, Department of Industrial Engineering, Graduate College of the University of Iowa, Iowa city , Iowa, August, 2003.

[10] A. Gionis, H. Mannila, and P. Tsaparas, Clustering Aggregation, The 21st International Conference on Data Engineering (ICDE), pages 341-352, 2005.

[11] S. Guha, R. Rastogi, and K. Shim, ROCK: A robust clustering algorithm for categorical attributes, International Conference on Data Engineering, 1999.

[12] S. Guha, R. Rastogi and K. Shim, CURE: An Efficient Clustering Algorithm for Large Databases, Proceedings of ACM-SIGMOD International Conference Management of Data, Seattle, WA, 1998.

[13] J. Han, and M. Kamber, Data Mining Concepts and Techniques, Morgan Kaufmann, 2000.

[14] J. He, A. H. Tan, C. L. Tan, and S. Y. Sung, On Quantitative Evaluation of Clustering Systems, Information Retrieval and Clustering, Kluwer Academic Publishers, 2002.

[15] L. Hermes, T. Zöller, J. M. Buhmann, Parametric Distributional Clustering for Image Segmentation, European Conference on Computer Vision, 2002.

[16] Z. Huang, A fast clustering algorithm to cluster very large categorical data sets in data mining, In SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, Tucson, AZ, 1997.

[17] A. Jain, R. Dubes, Algorithm for Clustering Data. Prentice Hall, New Jersey, 1998.

[18] A. Leuski, Evaluating Document Clustering for Interactive Information Retrieval, International Conference on Information and Knowledge Management (CIKM), pages 33-40, 2001.

[19] D. P. Mercer, Clustering Large Datasets, Technical Report, Department of Statistics, Linacre Graduate College of Oxford University, London, October, 2003.

[20] C. Michel, Cardinal nominal or similarity measures in comparative evaluation of information retrieval process, The 2nd International Conference on Language Resources & Evaluation (LREC), 2000.

[21] B. Mobasher, R. Cooley, and J. Srivastava. Creating adaptive web sites through usage-based clustering of urls. In IEEE Knowledge and Data Engineering Workshop (KDEX'99), November 1999.

[22] A. Nemalhabib and N. Shiri. "CLUC: A Natural Clustering Algorithm for Categorical Datasets based on Cohesion", Proceedings of the 21st ACM Symposium on Applied Computing (SAC), Dijon, France, April 23-27, 2006.

[23] L. Neto, J. and A. D. Santos and C. Antonio, A. Kaestner and A. A. Freitas, Document Clustering and Text Summarization, Proceedings of the 4th International Conference on Practical Applications of Knowledge Discovery and Data Mining, pages 41-55, 2000.

[24] M. Peters, and M. J. Zaki, CLICK: Clustering Categorical Data using K-partite Maximal Clique, the 21st International Conference on Data Engineering (ICDE), pages 355-356, 2005.

[25] N. Tishby, F.C. Pereira, and W. Bialek, The information Bottleneck Method, In 37th Annual Allerton Conference on Communication, Control and Computing, Urban-Champaign, IL, 1999.

[26] K. Wang, C. Xu,, and B. Liu. Clustering transactions using large items, Proceedings of the Conference on Information and Knowledge Management (CIKM), Kansas, Missouri, 1999.

[27] W. Wang, J. Yang, and R. Muntz, STING: A statistical information grid approach to spatial data mining, Proceedings of the 23rd VLDB Conference, pages 186-195, Athens, Greece, 1997.

[28] Rui Xu, Wunsch, D., II , Survey of Clustering Algorithms, IEEE Transactions on Neural Networks,Volume 16, No.3, page 645 – 678, May, 2005.

[20] Y. Yiling, G. Xudong, Y. Jinyuan, CLOPE: A Fast and Effective Clustering Algorithm for Transactional Data, Proceeding of KDD '02, pp 682--687, 2002.

[30] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 103-114, Montreal, Canada, 1996.

[31] UCI Machine Learning Repository http://www.ics.uci.edu/~mlear/MLRepository .html

[32] Datagen http://www2.dcc.ufmg.br/~meira/ch/tp2/datgen