Data Clustering Techniques

Qualifying Oral Examination Paper

Periklis Andritsos
University of Toronto
Department of Computer Science
periklis@cs.toronto.edu

March 11, 2002

1 Introduction

During a cholera outbreak in London in 1854, John Snow used a special map to plot the cases of the disease that were reported [Gil58]. A key observation, after the creation of the map, was the close association between the density of disease cases and a single well located at a central street. After this, the well pump was removed putting an end to the epidemic. Associations between phenomena are usually harder to detect, but the above is a very simple, and for many researchers, the first known application of cluster analysis.

Since then, cluster analysis has been widely used in several disciplines, such as statistics, software engineering, biology, psychology and other social sciences, in order to identify natural groups in large amounts of data. These data sets are constantly becoming larger, and their dimensionality prevents easy analysis and validation of the results. Clustering has also been widely adopted by researchers within computer science and especially the database community, as indicated by the increase in the number of publications involving this subject, in major conferences.

In this paper, we present the state of the art in clustering techniques, mainly from the data mining point of view. We discuss the procedures clustering involves and try to investigate advantages and disadvantages of proposed solutions. Finally, we shall present our suggestions for future research in the field.

The structure of this work is as follows: Section 2 outlines the stages commonly followed when performing clustering techniques. Section 3 discusses the different kinds of data we might have in hand, and metrics that define their similarities or dissimilarities. Section 4 introduces the main established clustering techniques and several key publications that have appeared in the data mining community. Section 5 distinguishes previous work done on numerical data and discusses the main algorithms in the field of categorical clustering. Section 6 suggests challenging issues in categorical data clustering and presents a list of open research topics. Finally, Section 7 concludes our work.

2 Problem Definition and Stages

There are several definitions for clustering. We will borrow the one given by Jain and Dubes [JD88]:

> "Cluster analysis organizes data by abstracting underlying structure either as a grouping of individuals or as a hierarchy of groups. The representation can then be investigated to see if the data group according to preconceived ideas or to suggest new experiments".

In brief, cluster analysis groups data objects into clusters such that objects belonging to the same cluster are similar, while those belonging to different ones are dissimilar. The notions of similarity and dissimilarity will become clear in later section.

The above definition indicates that clustering cannot be a one-step process. In one of the seminal texts on Cluster Analysis, Jain and Dubes divide the clustering process in the following stages [JD88]:

Data Collection : Includes the careful extraction of relevant data objects from the underlying data sources. In our context, data objects are distinguished by their individual values for a set of *attributes* (or *measures*).

Initial Screening : Refers to the massaging of data after its extraction from the source, or sources. This stage is closely connected to a process widely used in *Data Warehousing*, called *Data Cleaning* [JLVV99].

Representation : Includes the proper preparation of the data in order to become suitable for the clustering algorithm. Here, the similarity measure is chosen, the characteristics and dimensionality of the data is examined.

Clustering Tendency : Checks whether the data in hand has a natural tendency to cluster or not. This stage is often ignored, especially in the presence of large data sets.

Clustering Strategy : Involves the careful choice of clustering algorithm and initial parameters.

Validation : This is one of the last and, in our opinion, most under-studied stages. Validation is often based on manual examination and visual techniques. However, as the amount of data and their dimensionality grow, we have no means to compare the results with preconceived ideas or other clusterings.

Interpretation : This stage includes the combination of clustering results with other studies, e.g., classification, in order to draw conclusions and suggest further analysis.

The following sections present solutions proposed for the above stages. We start with the collection of data and examination of their types and measures, which are defined in the next section.

3 Data Types and their Measures

In clustering, the objects of analysis could be persons, salaries, opinions, software entities and many others. These objects must be carefully presented in terms of their characteristics. These characteristics are the main variables of the problem and their choice greatly influences the results of a clustering algorithm.

A comprehensive categorization of the different types of variables met in most data sets provides a helpful means for identifying the differences among data elements. We present a classification based on two schemes: the *Domain Size* and the *Measurement Scale* [And73].

3.1 Classification Based on the Domain Size

This classification distinguishes data objects based on the size of their domain, that is, the number of distinct values the data objects may assume. In the following discussion we assume a database $\mathcal{D}$, of n *objects*, or *tuples*. If $\hat{x}$, $\hat{y}$ and $\hat{z}$ are three data objects belonging to $\mathcal{D}$, each one of them has the form: $\hat{x} = (x_1, x_2, \ldots, x_k)$, $\hat{y} = (y_1, y_2, \ldots, y_k)$ and $\hat{z} = (z_1, z_2, \ldots, z_k)$, where k is the *dimensionality*, while each x_i, y_i and z_i, $1 \leq i \leq k$, is a *feature*, or an *attribute* of the corresponding object. Henceforth, the term "data types" will refer to "attribute data types". We have the following classes, [And73]:

1. An attribute is *continuous* if its domain is *uncountably infinite*, *i.e.*, its elements cannot be put into a one-to-one correspondence with the set of positive integers. This means that between any two values of the attribute, there exists an infinite number of values. Examples of such attributes could be the temperature and the colour or sound intensity.

2. An attribute is *discrete* if its domain is a *finite set*, *i.e.*, a set whose elements can be put into a one-to-one correspondence with a finite subset of the positive integers. Examples could be the number of children in a family or the serial numbers of books.

The class of *binary* attributes consists of attributes whose domain includes exactly two discrete values. They comprise a special case of discrete attributes, and we present as examples the `Yes/No` responses to a poll or the `Male/Female` gender entries of an employees' database.

3.2 Classification Based on the Measurement Scale

This classification distinguishes attributes according to their measurement scales. Suppose we have an attribute i and two tuples $\hat{x}$ and $\hat{y}$, with values x_i and y_i for this attribute, respectively. Then we have the following classes [And73]:

1. A *nominal scale* distinguishes between categories. This means that we can only say if $x_i = y_i$ or $x_i \neq y_i$. Nominal-scaled attribute values cannot be totally ordered. They are just a generalization of binary attributes, with a domain of more than two discrete values. Examples include the place of birth and the set of movies currently playing in Toronto.

2. An *ordinal scale* involves nominal-scaled attributes with the additional feature that their values can be totally ordered, but differences among the scale points cannot be quantified. Hence, on top of $x_i = y_i$ and $x_i \neq y_i$, we can assert if $x_i < y_i$ or $x_i > y_i$. Examples include the medals won by athletes.

3. An *interval scale* measures values in a (roughly) linear scale [HK01]. With interval scaling we can tell not only if one value comes before or after another, but also how far before or after. If $x_i > y_i$, since values are put on a linear scale, we may also say that $\hat{x}$ is $x_i - y_i$ units different from $\hat{y}$ with respect to attribute i. Examples include the book serial numbers or TV channel numbers.

4. A *ratio scale* is an interval scale with a meaningful zero point. Examples include weight, height and the number of children in a family.

More examples as well as a cross-classification of attribute data types will be given at the end of this section. Nominal- and ordinal-scaled attributes are called *qualitative* or *categorical* attributes, while interval- and ratio-scaled are called *quantitative* [And73].

3.3 Variable Standardization

It is common that quantitative attributes are mainly measured using specific units, such as kilograms and centimeters. Measuring units affect the cluster analysis results, since, for example, changing measurement units from kilograms to pounds for weight, may lead to a different result. The remedy to this problem is called *standardization*, which leads to unit-less variables. Given an attribute $i, 1 \leq i \leq k$, and n tuples $\hat{x}^{(1)}, \hat{x}^{(2)}, \ldots, \hat{x}^{(n)}$ there is a two step procedure to standardize it [HK01]:

1. Find the *mean absolute deviation*, s_i, of i, for all n tuples in the database $\mathcal{D}$:

$$s_i = \frac{1}{n}\Big(|x_i^{(1)} - m_i| + |x_i^{(2)} - m_i| + \cdots + |x_i^{(n)} - m_i|\Big)$$

   where, $(x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(n)})$ are the n values of attribute i and m_i is the mean value of i.

2. Find the *standardized measurement*, or *z-score*:

$$z_i^{(j)} = \frac{x_i^{(j)} - m_i}{s_i}$$

   for all $1 \leq j \leq n$.

Notice that in the expression of $z_i^{(j)}$ we do not use the *standard deviation* σ_i. This is done because s_i is considered more robust than σ_i in the presence of outliers [HK01]: the differences $|x_i^{(j)} - m_i|$ are not squared in s_i and thus the effect of outliers is reduced.

Standardization is optional and its usage depends on the application and the user. It is merely used to remove the measurement units and give each variable a common numerical property, or in other words equal weights.

3.4 Proximity Indexes

Once the characteristics of the data have been determined, we are faced with the problem of finding proper ways to decide how far, or how close the data objects are from each other. Proximity indexes are measures of alikeness or association between pairs of data objects. A proximity index can measure either *similarity* or *dissimilarity* [JD88]: the more two objects resemble each other the larger their similarity is and the smaller their dissimilarity. Speaking about dissimilarity it can be measured in many ways and one of them is *distance*. Moreover, distance can be measured in many ways, and this is by using distance measures. All measures depend on the type of attributes we are analyzing. For example, having categorical attributes we cannot use distance measures that require a geometrical orientation of the data; such data has no such orientation inherent in it.

From all measures, special interest has been given to those called *metrics* (we usually encounter *distance metrics*). Given three data points $\hat{x}$, $\hat{y}$ and $\hat{z}$, all in $\mathcal{D}$ as described at the beginning of Section 3, a distance metric d should satisfy [HK01]:

1. $d(\hat{x}, \hat{y}) \geq 0$: non-negativity;
2. $d(\hat{x}, \hat{y}) = 0$ if and only if $\hat{x} = \hat{y}$: distance of an object to itself is 0;

3. $d(\hat{x}, \hat{y}) = d(\hat{y}, \hat{x})$: symmetry;

4. $d(\hat{x}, \hat{z}) \leq d(\hat{x}, \hat{y}) + d(\hat{y}, \hat{z})$: triangle inequality;

Anderberg gives a thorough review of measures and metrics, also discussing their interrelationships [And73]. Note that any metric is also a measure, while a measure is not always a metric. To avoid any confusions, we shall be using the term measure, mentioning whether it computes similarity or dissimilarity. We shall revert to the word metric wherever this is appropriate. Here, we give a brief description of measures for each type of attributes [HK01]:

Interval-Scaled Attributes : After standardization, the dissimilarity between $\hat{x}$ and $\hat{y}$ is computed using the following distance metrics:

- *Minkowski Distance*, defined as:

$$d(\hat{x}, \hat{y}) = \Big(\sum_{i=1}^{n} |x_i - y_i|^q \Big)^{1/q}$$

  where q is a positive integer.

- *Euclidean Distance*, defined as:

$$d(\hat{x}, \hat{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

  Note that this is equal to the Minkowski distance for $q = 2$.

- *Manhattan Distance*, defined as:

$$d(\hat{x}, \hat{y}) = \sum_{i=1}^{n} |x_i - y_i|$$

  Note that this is equal to the Minkowski distance for $q = 1$.

- *Maximum Distance*, defined as:

$$d(\hat{x}, \hat{y}) = max_{i=1}^{n} |x_i - y_i|$$

  Note that this is equal to the Minkowski distance for $q \to \infty$.

Binary Attributes : Before introducing the proximity indexes for binary variables, we introduce the concept of *contingency tables* [HK01]. Such a table is given in Table 1.

	$\hat{y}:1$	$\hat{y}:0$	
$\hat{x}:1$	α	β	$\alpha+\beta$
$\hat{x}:0$	γ	δ	$\gamma+\delta$
	$\alpha+\gamma$	$\beta+\delta$	τ

Table 1: Contingency Table for two binary objects $\hat{x}$ and $\hat{y}$, where $\tau = \alpha + \beta + \gamma + \delta$

For objects $\hat{x}$ and $\hat{y}$ with only binary values, we denote one of the values by 1 and the second by 0. Thus, this table contains the following information:

- α is the number of 1s in both objects;

- β is the number of attribute values that are equal to 1 in $\hat{x}$ and 0 in $\hat{y}$;
- γ is the number of attribute values that are equal to 1 in $\hat{y}$ but 0 in $\hat{x}$;
- δ is the number of 0s in both objects;

After that, we have the following proximity indexes:

- *Simple Matching Coefficient*, defined as:

$$d(\hat{x}, \hat{y}) = \frac{\alpha + \delta}{\tau}$$

  if both of the values $\hat{x}$ and $\hat{y}$ can take are of equal weight, *i.e.*, $\hat{x}$ and $\hat{y}$ are *symmetric*.

- *Jaccard Coefficient*, defined as:

$$d(\hat{x}, \hat{y}) = \frac{\alpha}{\alpha + \beta + \gamma}$$

  Notice that this coefficient disregards the number of $0 - 0$ matches. Hence, it is mainly used for cases where one of the possible states (described as 1) has a higher weight than the other, *i.e.*, the binary attributes are *asymmetric*.

Special consideration has to be given in the meaning of *existence* when encountering binary variables. If it is clear that one of the values of the variable denotes presence and the other one absence, then it is useful to talk in terms of existence. For example, this is the case when such a variable has to do with the presence of children in the appropriate attribute value of an employee's record. On the other hand, if the binary variable defines a dichotomy, then we can just measure 0-0 and 1-1 matches or 0-1 and 1-0 mismatches. This is the case when we store `Male/Female` gender values in a database.

Nominal-Scaled Attributes : The dissimilarity between objects $\hat{x}$ and $\hat{y}$ is given by:

$$d(\hat{x}, \hat{y}) = \frac{p - m}{p}$$

where m is the number of matches and p is the total number of attributes.

Ordinal-Scaled Attributes : These are treated in a similar way as interval-scaled, in terms of measuring dissimilarities. Assume i is an ordinal-scaled attribute with M_i states (domain size). The steps we follow are [HK01]:

1. The M_i states are ordered, $[1 \dots M_i]$, so we can replace each value with its corresponding rank, $r_i \in \{1 \dots M_i\}$;
2. Each ordinal attribute could have a different domain size, so it is often necessary to convert each state onto a value of the $[0.0, 1.0]$ interval. This is achieved with the following:

$$z_i^{(j)} = \frac{r_i^{(j)} - 1}{M_i - 1}$$

3. Dissimilarity can be computed with one of the measures for interval-scaled attributes using the $z_i^{(j)}$'s.

Ratio-Scaled Attributes : There are several methods to compute dissimilarities between these variables. One solution is to use a logarithmic formula on each attribute x_i, *i.e.*, $q_i = log(x_i)$. The q_i's now can be treated as interval-scaled. This logarithmic conversion is helpful in situations where the measurements of the variable are on an exponential scale. Hence, the use of the logarithmic transformation depends on the definition of the variables and the application. It may be more beneficial to treat ratio-scaled variables without any transformation.

Mixed Attributes : Suppose we have k attributes of mixed type, two data objects $\hat{x}$ and $\hat{y}$ and an indicator $\delta^f_{\hat{x}\hat{y}}$ that signifies the degree of dissimilarity between $\hat{x}$ and $\hat{y}$ in attribute f. Dissimilarity $d(\hat{x}, \hat{y})$ is defined as:

$$d(\hat{x}, \hat{y}) = \frac{\sum_{f=1}^{k} \delta^f_{\hat{x}\hat{y}} d^f_{\hat{x}\hat{y}}}{\sum_{f=1}^{k} \delta^f_{\hat{x}\hat{y}}}$$

where the indicator $\delta^f_{\hat{x}\hat{y}} = 0$ if either (1): $x_i^{(f)}$ or $x_j^{(f)}$ is missing (i.e., there is no measurement of attribute f for object $\hat{x}$ or object $\hat{y}$), or (2) $x_i^{(f)} = x_j^{(f)} = 0$ and attribute f is asymmetric binary; otherwise, $\delta^f_{\hat{x}\hat{y}} = 1$. The contribution of variable f to the dissimilarity between $\hat{x}$ and $\hat{y}$, $d^f_{\hat{x}\hat{y}}$ is computed dependent on its type:

- If f is binary or nominal: $d^f_{\hat{x}\hat{y}} = 0$ if $x_i^{(f)} = x_j^{(f)}$; otherwise $d^f_{\hat{x}\hat{y}} = 1$
- If f is interval-based: $d^f_{\hat{x}\hat{y}} = \frac{|x_i^{(f)} - x_j^{(f)}|}{max_h x_h^{(f)} - min_h x_h^{(f)}}$, where h runs over all non-missing objects for attribute f.
- If f is ordinal or ratio-scaled: compute the ranks $r_i^{(f)}$ and $z_i^{(f)} = \frac{r_i^{(f)} - 1}{M_f - 1}$, and treat $z_i^{(f)}$ as interval-scaled.

Table 2 gives a cross-classification of attribute types, [And73], together with metrics appropriate for measuring their distance.

	Domain Size			
Scale	CONTINUOUS	DISCRETE	BINARY	**Metric**
RATIO	Temperature (^oK), weight, height	Number of children,houses	N/A	*Euclidean or Manhattan. Special transformation may be needed prior to this*
INTERVAL	Temperature (^oC)	Book Serial Numbers, TV channel numbers	N/A	*Euclidean or Manhattan*
ORDINAL	Sound intensity, Color intensity	athletes' medals, clothes' size	Tall/Short, Big/Small etc.	*After representing values with their ranks, treat them as interval-scaled*
NOMINAL	N/A: requires an uncountably infinite number of distinct values	Color, Favorite Actors	Yes/No, On/Off etc.	*Simple Matching Coefficient or Jaccard Coefficient*

Table 2: Cross–classification of Attributes and their Metrics

Note two things: First, there is a distinction between temperature measured in ^oK and ^oC. This is done because the zero point of the Kelvin scale is absolute zero, whereas the zero point of the Celsius scale is the water's freezing temperature. Hence, it is reasonable to say that 100^oK is twice as hot as 50^oK. However, such a comparison would be unreasonable in the Celsius scale. That's why the Kelvin scale is considered a ratio scale while Celsius as interval [And73]. Second, some of the attribute types could belong to more than one category. For example, colour is usually considered as nominal (same as categorical here). However,

we all know that for each colour there is a point on the spectrum line for it, making it an ordinal-scaled attribute.

One final note in regard to the types of data sets and their handling is that certain attributes, or all of them, may be assigned weights. Sometimes upon removal of the measurement units, *i.e.* after standardization, user's judgment or understanding of the problem can be further taken into consideration to assign weights so that each variable contributes to the mean, range or standard deviation of the composite in a manner consistent with her objectives in the analysis [And73]. For example, if she analyzes a data set of soccer players, she might want to give more weight to a certain set of attributes, such as the athlete's height and age, than others, such as the number of children or cars each of them has. Finally, weights can be used in the distance measure above. For example, given the Euclidean distance, if each attribute is assigned a weight w_i, $1 \leq i \leq k$, we then have the *weighted Euclidean Distance*, defined as:

$$d(\hat{x}, \hat{y}) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2}$$

Now that we have studied the different kinds of attributes, we are ready to move on to the basic clustering algorithms starting with a general classification in the next section.

4 Categorization of Clustering Techniques and Previous Work

Many diverse techniques have appeared in order to discover cohesive groups in large datasets. In the following two section we present the two classic techniques for clustering as well as more specific ones, respectively.

4.1 Basic Clustering Techniques

We distinguish two types of clustering techniques: *Partitional* and *Hierarchical*. Their definitions are as follows [HK01]:

Partitional : Given a database of n objects, a partitional clustering algorithm constructs k partitions of the data, where each cluster optimizes a clustering criterion, such as the minimization of the *sum of squared distance from the mean* within each cluster.

One of the issues with such algorithms is their high complexity, as some of them exhaustively enumerate all possible groupings and try to find the global optimum. Even for a small number of objects, the number of partitions is huge. That's why, common solutions start with an initial, usually random, partition and proceed with its refinement. A better practice would be to run the partitional algorithm for different sets of initial k points (considered as representatives) and investigate whether all solutions lead to the same final partition.

Partitional Clustering algorithms try to locally improve a certain criterion. First, they compute the values of the similarity or distance, they order the results, and pick the one that optimizes the criterion. Hence, the majority of them could be considered as greedy-like algorithms.

Hierarchical : Hierarchical algorithms create a hierarchical decomposition of the objects. They are either *agglomerative* (*bottom-up*) or *divisive* (*top-down*):

(a) *Agglomerative* algorithms start with each object being a separate cluster itself, and successively merge groups according to a distance measure. The clustering may stop when all objects are in a single group or at any other point the user wants.

These methods generally follow a greedy-like bottom-up merging.

(b) *Divisive* algorithms follow the opposite strategy. They start with one group of all objects and successively split groups into smaller ones, until each object falls in one cluster, or as desired.

Divisive approaches divide the data objects in disjoint groups at every step, and follow the same pattern until all objects fall into a separate cluster. This is similar to the approach followed by divide-and-conquer algorithms.

Most of the times, both approaches suffer from the fact that once a merge or a split is committed, it cannot be undone or refined.

Figure 1(a) gives an example of two divisive algorithms performed in the same data set, with different initial parameters. A ''+'' sign denotes the centre of clusters, which in this case is defined as the mean of the values of a particular cluster. At the same time, Figure 1(b) depicts the *dendrogram* produced by either a divisive or agglomerative clustering algorithm.

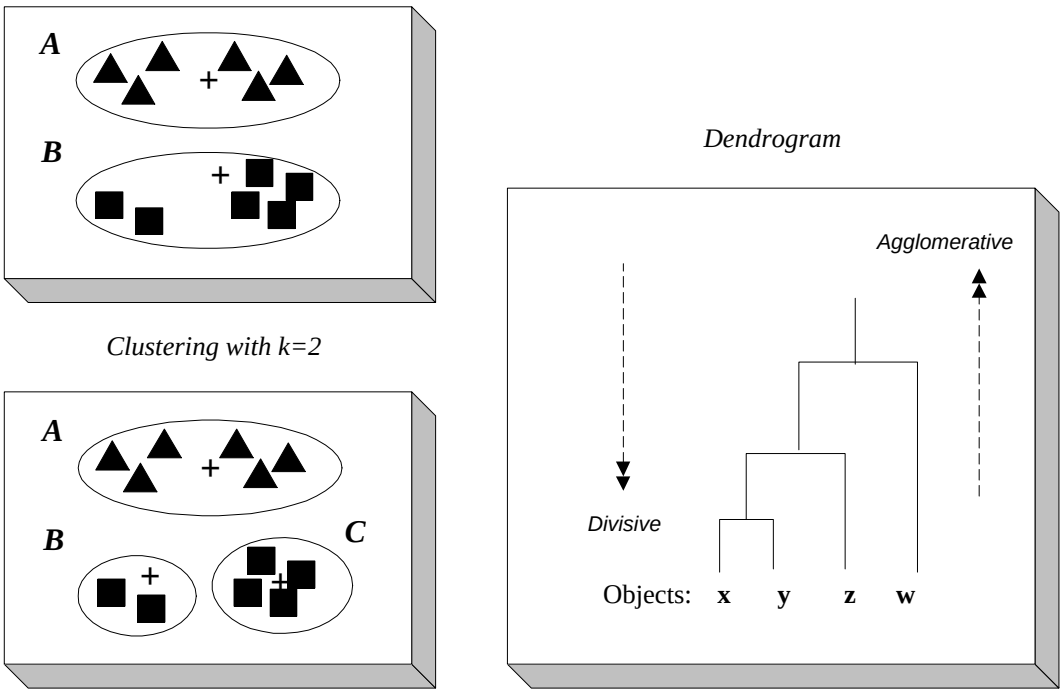


Figure 1: Examples of the classic clustering algorithms, where k is the number of clusters

Partitional and hierarchical methods can be integrated. This would mean that a result given by a hierarchical method can be improved via a partitional step, which refines the result via iterative relocation of points. Other classes of clustering algorithms are given in the next subsection.

4.2 Data Mining Clustering Techniques

Apart from the two main categories of partitional and hierarchical clustering algorithms, many other methods have emerged in cluster analysis, and are mainly focused on specific problems or specific data sets available. These methods include [HK01]:

Density-Based Clustering : These algorithms group objects according to specific density objective functions. Density is usually defined as the number of objects in a particular neighborhood of a data objects. In these approaches a given cluster continues growing as long as the number of objects in the neighborhood exceeds some parameter. This is considered to be different from the idea in partitional algorithms that use iterative relocation of points given a certain number of clusters.

Grid-Based Clustering : The main focus of these algorithms is spatial data, *i.e.*, data that model the geometric structure of objects in space, their relationships, properties and operations. The objective of these algorithms is to quantize the data set into a number of cells and then work with objects belonging to these cells. They do not relocate points but rather build several hierarchical levels of groups of objects. In this sense, they are closer to hierarchical algorithms but the merging of grids, and consequently clusters, does not depend on a distance measure but it is decided by a predefined parameter.

Model-Based Clustering : These algorithms find good approximations of model parameters that best fit the data. They can be either partitional or hierarchical, depending on the structure or model they hypothesize about the data set and the way they refine this model to identify partitionings. They are closer to density-based algorithms, in that they grow particular clusters so that the preconceived model is improved. However, they sometimes start with a fixed number of clusters and they do not use the same concept of density.

Categorical Data Clustering : These algorithms are specifically developed for data where Euclidean, or other numerical-oriented, distance measures cannot be applied. In the literature, we find approaches close to both partitional and hierarchical methods.

For each category, there exists a plethora of sub-categories, *e.g.*, density-based clustering oriented towards geographical data [SEKX98], and algorithms for finding clusters. An exception to this is the class of categorical data approaches. Visualization of such data is not straightforward and there is no inherent geometrical structure in them, hence the approaches that have appeared in the literature mainly use concepts carried by the data, such as co-occurrences in tuples. On the other hand, categorical data sets are in abundance. Moreover, there are data sets with mixture of attribute types, such as the United States Census data set (see `http://www.census.gov/`) and data sets used in schema discovery [**?**]. As will be discussed, current clustering algorithms focus on situations in which all attributes of an object are of a single type.We believe that cluster analysis of categorical and mixed type data sets is an intriguing problem in data mining.

But what makes a clustering algorithm efficient and effective ? The answer is not clear. A specific method can perform well on one data set, but very poorly on another, depending on the size and dimensionality of the data as well as the objective function and structures used. Regardless of the method, researchers agree that characteristics of a good clustering technique are [HK01]:

- *Scalability*: The ability of the algorithm to perform well with large number of data objects (tuples).
- *Analyze mixture of attribute types*: The ability to analyze single as well as mixtures of attribute types.

- *Find arbitrary-shaped clusters*: The shape usually corresponds to the *kinds* of clusters an algorithm can find and we should consider this as a very important thing when choosing a method, since we want to be as general as possible. different types of algorithms will be biased towards finding different types of cluster structures/shapes and it is not always an easy task to determine the shape or the corresponding bias. Especially when categorical attributes are present we may not be able to talk about cluster structures.

- *Minimum requirements for input parameters*: Many clustering algorithms require some user-defined parameters, such as the number of clusters, in order to analyze the data. However, with large data sets and higher dimensionalities, it is desirable that a method require only limited guidance from the user, in order to avoid bias over the result.

- *Handling of noise*: Clustering algorithms should be able to handle deviations, in order to improve cluster quality. Deviations are defined as data objects that depart from generally accepted norms of behavior and are also referred to as outliers. Deviation detection is considered as a separate problem.

- *Sensitivity to the order of input records*: The same data set, when presented to certain algorithms in different orders, may produce dramatically different results. The order of input mostly affects algorithms that require a single scan over the data set, leading to locally optimal solutions at every step. Thus, it is crucial that algorithms be insensitive to the order of input.

- *High dimensionality of data*: The number of attributes/dimensions in many data sets is large, and many clustering algorithms cannot handle more than a small number (eight to ten) of dimensions. It is a challenge to cluster high dimensional data sets, such as the U.S. census data set which contains 138 attributes.

  The appearance of large number of attributes is often termed as the *curse of dimensionality*. This has to do with the following [HAK00]:

  1. As the number of attributes becomes larger, the amount of resources required to store or represent them grows.
  2. The distance of a given point from the nearest and furthest neighbor is almost the same, for a wide variety of distributions and distance functions.

  Both of the above highly influence the efficiency of a clustering algorithm, since it would need more time to process the data, while at the same time the resulting clusters would be of very poor quality.

- *Interpretability and usability*: Most of the times, it is expected that clustering algorithms produce usable and interpretable results. But when it comes to comparing the results with preconceived ideas or constraints, some techniques fail to be satisfactory. Therefore, easy to understand results are highly desirable.

Having the above characteristics in mind, we present some of the most important algorithms that have influenced the clustering community. We will attempt to criticize them and report which of the requirements they meet or fail to meet. We shall treat clustering algorithms for categorical data in a separate section.

4.3 Partitional Algorithms

This family of clustering algorithms includes the first ones that appeared in the Data Mining Community. The most commonly used are *k-means*, [JD88, KR90], *PAM (Partitioning Around Medoids)*, [KR90], *CLARA (Clustering LARge Applications)*, [KR90] and *CLARANS (Clustering LARge ApplicatioNS)*, [NH94].

The goal in *k-means* is to produce k clusters from a set of n objects, so that the *squared-error* objective function:

$$E = \sum_{i=1}^{k} \Sigma_{p \in C_i} |p - m_i|^2$$

is minimized. In the above expression, C_i are the clusters, p is a point in a cluster C_i and m_i the mean of cluster C_i. The mean of a cluster is given by a vector, which contains, for each attribute, the mean values of the data objects in this cluster and. Input parameter is the number of clusters, k, and as an output the algorithm returns the centers, or means, of every cluster C_i, most of the times excluding the cluster identities of individual points. The distance measure usually employed is the Euclidean distance. Both for the optimization criterion and the proximity index, there are no restrictions, and they can be specified according to the application or the user's preference. The algorithm is as follows:

1. Select k objects as initial centers;
2. Assign each data object to the closest center;
3. Recalculate the centers of each cluster;
4. Repeat steps 2 and 3 until centers do not change;

The algorithm is relatively scalable, since its complexity is, $\mathcal{O}(Ikn)$ [1], where I denotes the number of iterations, and usually $k \ll n$.

PAM is an extension to *k-means*, intended to handle outliers efficiently. Instead of cluster centers, it chooses to represent each cluster by its *medoid*. A medoid is the most centrally located object inside a cluster. As a consequence, medoids are less influenced by extreme values; the mean of a number of objects would have to "follow" these values while a medoid would not. The algorithm chooses k medoids initially and tries to place other objects in clusters whose medoid is closer to them, while it swaps medoids with non-medoids as long as the quality of the result is improved. Quality is also measured using the squared-error between the objects in a cluster and its medoid. The computational complexity of *PAM* is, $\mathcal{O}(Ik(n-k)^2)$, , with I being the number of iterations, making it very costly for large n and k values.

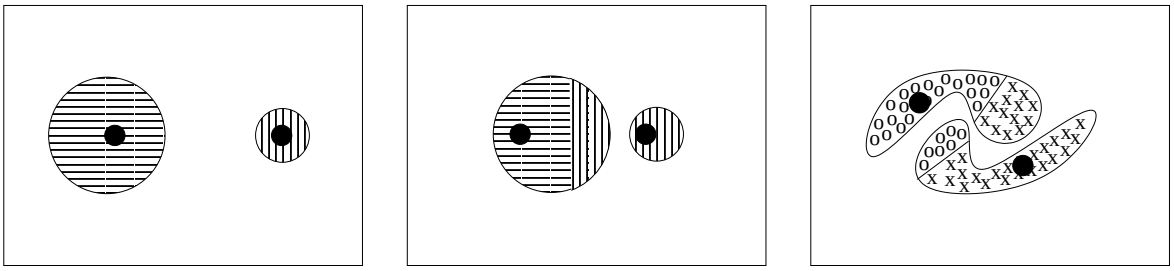
A solution to this is the *CLARA* algorithm, by Kaufman and Rousseeuw [KR90]. This approach works on several samples of size s, of the n tuples in the database, applying *PAM* on each one of them. The output depends on the s samples and is the "best" result given by the application of *PAM* on these samples. It has been shown that CLARA works well with 5 samples of $40 + k$ size [KR90], and its computational complexity becomes, $\mathcal{O}(k(40+k)^2 + k(n-k))$. Note that there is a quality issue when using sampling techniques in clustering: the result may not represent the initial data set, but rather a locally optimal solution. In *CLARA* for example, if "true" medoids of the initial data are not contained in the sample, then the result is guaranteed not to be the best.

[1] In this paper, we consider time complexities, unless otherwise specified

CLARANS, combines *PAM* with sampling, as well. Specifically, clustering is performed as a search in a graph: the nodes of the graph are potential solutions, *i.e.*, a set of k medoids. Two nodes are neighboring if they differ by one medoid. The *CLARANS* approach works as follows:

1. Randomly choose k medoids;
2. Randomly consider one of the medoids to be swapped with a non-medoid;
3. If the cost of the new configuration is lower, repeat step 2 with new solution;
4. If the cost is higher, repeat step 2 with different non-medoid object, unless a limit has been reached (the maximum value between 250 and $k(n-k)$;
5. Compare the solutions so far, and keep the best;
6. Return to step 1, unless a limit has been reached (set to the value of 2);

CLARANS compares an object with every other, in the worst case and for every of the k medoids. Thus, its computational complexity is, $\mathcal{O}(kn^2)$, which does not make it suitable for large data sets.



(a) Well separated Clusters

(b) Clusters of different sizes close to each other

(c) Aribtrary-Shaped Clusters

Figure 2: Three applications of the *k-means* algorithm

Figure 2 presents the application of *k-means* on three kinds of data sets. The algorithm performs well on appropriately distributed (separated) and spherical-shaped groups of data (Figure 2(a)). In case the two groups are close to each other, some of the objects on one might end up in different clusters, especially if one of the initial cluster representatives is close to the cluster boundaries (Figure 2(b)). Finally, *k-means* does not perform well on non-convex-shaped clusters (Figure 2(c)) due to the usage of Euclidean distance. As already mentioned, *PAM* appears to handle outliers better, since medoids are less influenced by extreme values than means, something that *k-means* fails to perform in an acceptable way.

CLARA and *CLARANS* are based on the clustering criterion of *PAM*, *i.e.*, distance from the medoid, working on samples of the data sets they are applied on, and making them more scalable. However, their efficiency and effectiveness highly depend on the sample size and its *bias*. A *bias* is present in a sample when the data objects in it have not been drawn with equal probabilities.

Finally, their application is restricted to numerical data of lower dimensionality, with inherent well separated clusters of high density.

4.4 Hierarchical Algorithms

As we already mentioned, standard hierarchical approaches suffer from high computational complexity, namely $\mathcal{O}(n^2)$. Some approaches have been proposed to improve this performance and one of the first ones is *BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)* [ZRL96]. It is based on the idea that we do not need to keep whole tuples or whole clusters in main memory, but instead, their sufficient statistics. For each cluster, *BIRCH* stores only the triple (n, LS, SS), where n is the number of data objects in the cluster, LS is the linear sum of the attribute values of the objects in the cluster and SS is the sum of squares of the attribute values of the objects in the cluster. These triples are called *Cluster Features (CF)* and kept in a tree called *CF-tree*. In the paper by Zhang et al. [ZRL96] it is proved how standard statistical quantities, such as distance measures, can be derived from the CF's.

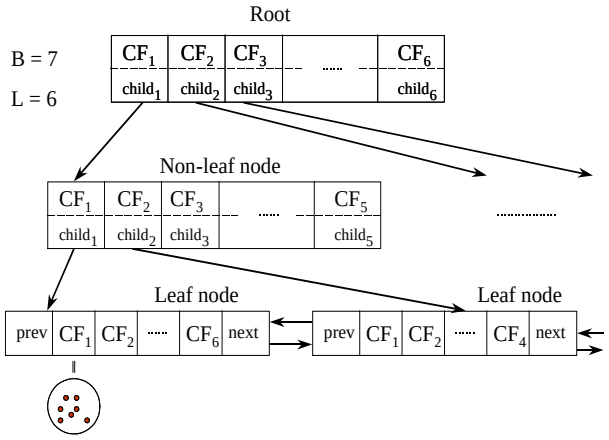


Figure 3: A CF-Tree used by the *BIRCH* algorithm, [HK01]

CF-trees are characterized by two parameters. These are the *Branching Factor, B* and the *Threshold, T*, the former being the maximum number of children for a non-leaf node and the latter the maximum distance between any pair of points, *i.e.* the *diameter*, in sub-clusters stored at leaf nodes. An example of a CF-tree is given in Figure 3. All nodes store CF's: non-leaf ones store the sums of the CF's of their children, while leaf nodes the CF's of the data objects themselves. *BIRCH* works as follows:

1. The data objects are loaded one by one and the initial CF-tree is constructed: an object is inserted into the closest leaf entry, *i.e.* sub-cluster. If the diameter of the this sub-cluster becomes larger than T, the leaf node, and possible others, are split. When the object is properly inserted in a leaf node, all nodes towards the root of the tree are updated with necessary information.

2. If the CF-tree of stage 1 does not fit into memory, build a smaller CF-tree: the size of a CF-tree is controlled by parameter T and thus choosing a larger value for it will merge some sub-clusters making the tree smaller. Zhang et al. show how this stage does not require to start reading the data from the beginning and guarantees the creation of a smaller tree.

3. Perform clustering: leaf nodes of the CF-tree hold sub-cluster statistics; in this stage *BIRCH* uses these statistics to apply some clustering technique, *e.g. k-means*, and produce an initial clustering

4. Redistribute the data objects using centroids of clusters discovered in step 3: this is an optional stage

which requires an additional scan of the data set and re-assigns the objects to their closest centroids. Optionally, this phase also includes the labeling of the initial data and discarding of outliers.

The algorithm obviously requires one scan over the data, *i.e.*, its computational complexity is $\mathcal{O}(n)$, with an optional additional scan to fix the problems of the first pass. The disadvantages, however, have to do with the quality of the discovered clusters. First of all, since it uses Euclidean distance, it works well only on well distributed numerical data. Second, and most important, is the issue of the order of input: parameter T affects the cluster sizes and thus their naturalness, forcing objects that should be in the same cluster to end up in different ones, while duplicate objects could be attracted by different clusters, if they are presented to the algorithm in different order. Finally, *BIRCH* has not been tested in higher dimensional data sets, so its performance in this respect is questionable.

Choosing one representative for each cluster might degrade the quality of the result. *CURE (Clustering Using REpresentatives)* [GRS98] chooses to use more than one point in each cluster as its representatives. As a consequence, *CURE* is able to capture clusters of varying shapes and varying sizes, in large databases, like the ones in Figure 4.

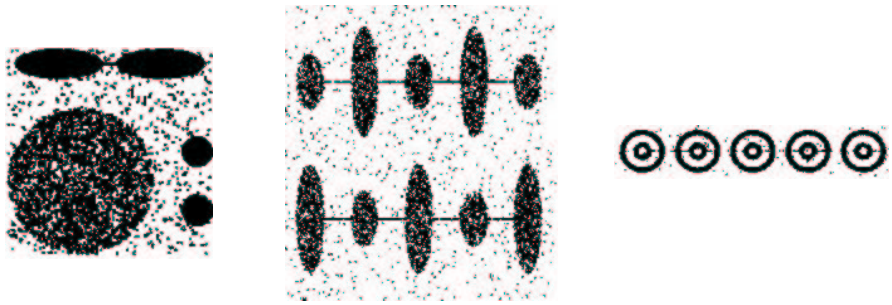


Figure 4: Varying shapes and sizes of data sets for *CURE*, [GRS98]

The algorithm is summarized below:

1. Draw a random sample from the data set;
2. Partition this sample into equal sized groups: the main idea here is to partition the sample into p partitions, each of size $\frac{n'}{p}$, where n' is the size of the sample;
3. Cluster the points of each group: we perform an initial clustering until each partition has $\frac{n'}{pq}$, $q > 1$ clusters;
4. Eliminate outliers: this is a two phase procedure. First, as clusters are being formed for some time until the number of clusters decreases below a certain fraction of the initial number of clusters. Second, in case outliers are sampled together during sampling phase, the algorithm eliminates small groups;
5. Cluster the partial cluster. The representative points are shrunken towards the center, *i.e.* replaced by other points closer to the center, by a *shrinking factor* α;
6. Mark the data with corresponding labels;

The computational complexity of the algorithm is $\mathcal{O}(n^2 \log n)$. As discussed, it is a reliable method for arbitrary shapes of clusters and has been shown to perform well on two-dimensional data sets. However, it appears to be sensitive to the parameters, such as the number of representative objects, the shrinking factor and the number of partitions.

In general, *BIRCH* outperforms *CURE* in time complexity, but is lacking cluster quality. Finally, they both handle outliers well.

4.5 Density-Based Algorithms

Clusters can be thought of as regions of high density, separated by regions of no or low density. Density here is considered as the number of data objects in the "neighborhood".

The most popular one is probably *DBSCAN (Density-Based Spatial Clustering of Applications with Noise*, [EKSX96]. The algorithm finds, for each object, the neighborhood that contains a minimum number of objects. Finding all points whose neighborhood falls into the above class, a cluster is defined as the set of all points transitively connected by their neighborhoods. *DBSCAN* finds arbitrary-shaped clusters while at the same time not being sensitive to the input order. Besides, it is incremental, since every newly inserted point only affects a certain neighborhood. On the other hand, it requires the user to specify the radius of the neighborhood and the minimum number of objects it should have; optimal parameters are difficult to determine. In this work, Ester et al. employ a spatial index to help finding neighbors of a data point. Thus, the complexity is improved to $\mathcal{O}(n \log n)$, as opposed to $\mathcal{O}(n^2)$ without the index. Finally, if Euclidean distance is used to measure proximity of objects, its performance degrades for high dimensional data.

OPTICS (Ordering Points To Identify the Clustering Structure) [ABKS96] is an extension to *DBSCAN* that relaxes the strict requirements of input parameters. *OPTICS* computes an augmented *cluster ordering*, such as in Figure 5, to automatically and interactively cluster the data. The ordering represents the density-based clustering structure of the data and contains information that is equivalent to density-based clustering obtained by a range of parameter settings [HK01]. *OPTICS* considers a minimum radius that makes a neighborhood legitimate for the algorithm, *i.e.*, having the minimum number of objects, and extends it to a maximum value. *DBSCAN* and *OPTICS* are similar in structure and have the same computational complexity, $\mathcal{O}(n \log n)$.

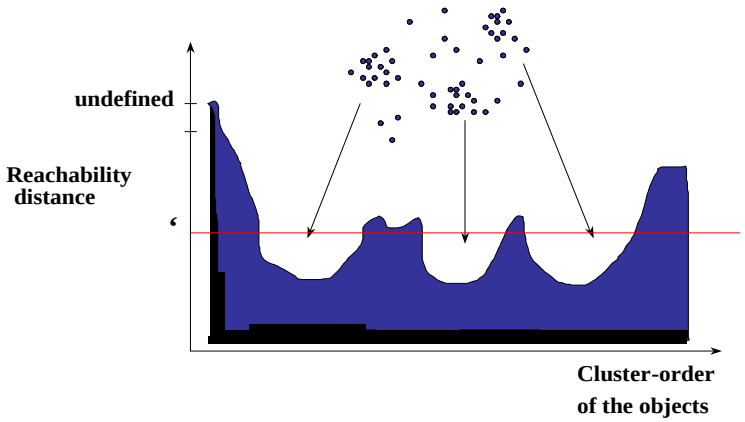


Figure 5: Augmented cluster ordering in *OPTICS* [HK01]

DENCLUE (DENsity-based CLUstEring) [HK98] is the third representative and uses a different approach to cluster data objects:

1. The influence of an object to its neighborhood is given by an *influence function*;
2. Overall density is modeled as the sum of the influence functions of all objects;
3. Clusters are determined by *density attractors*, where density attractors are local maxima of the overall density function.

The influence function can be an arbitrary one, as long as it is determined by the distance $d(\hat{x}, \hat{y})$ between the objects. Examples are the *square wave* influence function:

$$f_{Square}(\hat{x}, \hat{y}) = \begin{cases} 0 & \text{if } d(\hat{x}, \hat{y}) > \sigma \\ 1 & \text{otherwise} \end{cases}$$

where σ is a threshold, or a *Gaussian*:

$$f_{Gaussian}(\hat{x}, \hat{y}) = e^{-\frac{d(\hat{x}, \hat{y})^2}{2\sigma^2}}$$

Application of the second function is given in Figure 6. Density attractors are the peak values (local maxima) in the graph. After that, a *center-defined* cluster for an attractor x^* is a subset C, where the density function at x^* is no less than ξ.

(a) Data Set (b) Gaussian

Figure 6: *DENCLUE* using a Gaussian influence function [HK01]

We can see that *DENCLUE* highly depends on the threshold ξ (*noise threshold*) and parameter σ (*density parameter*). On the other hand, it involves the following advantages [HK01]:

1. Has a solid mathematical foundation;
2. Handles outliers;
3. Allows a compact mathematical description of arbitrary-shaped clusters even in high dimensional data sets.
4. Uses grid cells and keeps info about the ones that do actually contain objects.

DENCLUE's computational complexity is $\mathcal{O}(n \log n)$. Density-based algorithms do not perform any sampling on the data, which may increase the cost. This is done because there might be a difference in the density of the sample from the density of the whole data set [HVB00]. Another important issue here is the distinction between the previously discussed algorithms, *BIRCH* and *CURE*, and density based ones. The latter ones grow their clusters according to the density criterion, while the former try to accommodate data objects in specific clusters, with the use of extra passes over the data sometimes for optimization purposes. Moreover, density-based methods are generally insensitive to the order of input, while *BIRCH* depends on it.

4.6 Other Approaches

In this subsection, we present some algorithms that belong to the rest of the categories. First, we deal with grid-based clustering, which is mainly oriented towards spatial data sets. The main concept of these algorithms is the quantization of the data space into a number of cells. *STING (STatistical INformation Grid)*, [WYM97], *WaveCluster*, [SCZ98] and *CLIQUE (CLustering In QUEst*, [AGGR98] are three representatives of this family of algorithms.

STING breaks the spatial data space into a finite number of cells using a rectangular hierarchical structure, as the one in Figure 7. It then processes the data set and computes the mean, variance, minimum, maximum and type of distribution of the objects within each cell. As we go higher in the structure, statistics are being summarized from lower levels (similar to the summarization done with CF's in a CF-tree). New objects are easily inserted in the grid and spatial queries can be answered visiting appropriate cells at each level of the hierarchy. A spatial query is defined as one that retrieves information of spatial data and their interrelationships. *STING* is highly scalable, since it requires one pass over the data, but uses a multi-resolution method that highly depends on the granularity of the lowest level. Multi-resolution is the ability to decompose the data set into different levels of detail (an example is given in Figure 8). Finally, when merging grid cells to form clusters, children are not properly merged (because they only correspond to dedicated parents) and the shapes of clusters have vertical and horizontal boundaries, conforming to the boundaries of the cells.

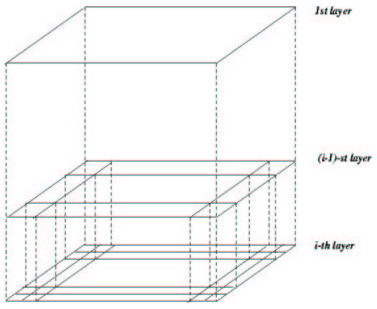


Figure 7: The multi-resolution grid used by *STING* [HK01]

On the other hand, *WaveCluster*, which employs a multi-resolution approach as well, follows a different strategy. It uses *Wavelets* to find arbitrary shaped clusters at different levels of resolution. A *wavelet transform* is a signal processing method that decomposes a signal into different frequency bands. Figure 8 shows this application of the algorithm on a data set. The leftmost image corresponds to high resolution, the middle one to medium and the rightmost to lower resolution. Hence, applying this transform into clustering helps in detecting clusters of data objects at different levels of detail. The algorithm handles outliers

well, and is highly scalable, $\mathcal{O}(n)$, but not suitable for high dimensional data sets. Compared to *BIRCH*, *CLARANS* and *DBSCAN*, *WaveCluster* was found to perform better [HK01].

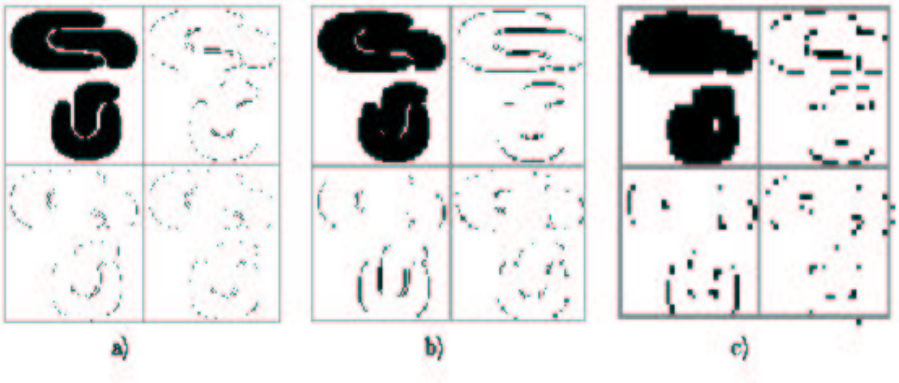


Figure 8: The application of *WaveCluster*, [HK01]

In higher dimensions, clusters might exist in a subset of dimensions, *i.e.* a *subspace*. *CLIQUE* is a subspace clustering algorithm. It partitions the data into hyper-rectangles, *i.e.* rectangles with more than two dimensions, and finds the dense ones, *i.e.* the ones with a certain number of objects in a given neighborhood; unions of such rectangles constitute the clusters. *CLIQUE* first finds 1-dimensional dense cells, then 2-dimensional dense rectangles and so on, until all dense hyper-rectangles of dimensionality k are found. As for its efficiency, it handles high dimensional data sets well, is insensitive to the input order and computes clusters in $\mathcal{O}(n)$ time.

Finally, we present the *Expectation-Maximization (EM)* algorithm [BFR98], which is generally considered as a *model-based* algorithm or just an extension to the *k-means* algorithm [HK01]. Indeed, *EM* assigns each object to a dedicated cluster according to the probability of membership for that object. The probability distribution function is the *multivariate Gaussian* and main goal is the iterative discovery of good values for its parameters, with objective function the logarithm of the likelihood of the data, given how well the probabilistic model fits it. The algorithm can handle various shapes of clusters, while at the same time it can be very expensive since hundreds of iterations may be required for the iterative refinement of parameters. Bradley et al. also propose a scalable solution to *EM*, based on the observation that data can be *compressed*, *maintained* in main memory or *discarded* [BFR99]. Objects are discarded if their cluster membership is ascertained, they are compressed if they cannot be discarded but belong to a tight sub-cluster, and retained in memory otherwise.

The algorithms discussed so far are given in Table 3, together with some of their characteristics. This table indicates the input parameters required by each algorithm (2nd column), the type of data sets it is optimized for (3rd column), the cluster structure (4th column), whether it handles noise or not (5th column) and its computational complexity (6th column). In general complexity can be given in terms of the number of operations in main memory or the I/O cost required. In our clustering techniques, we assume number of in-memory operations and thus a complexity of $\mathcal{O}(\backslash)$ does not necessarily mean that there is only one scan over the data. It means that there is a constant number of in-memory operations and maybe multiple scans of the data set.

Partitional Methods					
Algorithm	*Input Parameters*	*Optimized For*	*Cluster Structure*	*Outlier Handling*	*Computational Complexity*
$k-means$	Number of Clusters	Separated Clusters	Spherical	No	$\mathcal{O}(Ikn)$
PAM	Number of Clusters	Separated Clusters, Small Data Sets	Spherical	No	$\mathcal{O}(Ik(n-k)^2)$
$CLARA$	Number of Clusters	Relatively Large Data Sets	Spherical	No	$\mathcal{O}(ks^2+k(n-k))$
$CLARANS$	Number of Clusters, Maximum Number of Neighbors	Spatial Data Sets, Better Quality of Clusters than *PAM* and *CLARA*	Spherical	No	$\mathcal{O}(kn^2)$
Hierarchical Methods					
$BIRCH$	Branching Factor, Diameter Threshold	Large Data Sets	Spherical	Yes	$\mathcal{O}(n)$
$CURE$	Number of Clusters, Number of Cluster Representatives	Arbitrary Shapes of Clusters, Relatively Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n^2 \log n)$
Density-Based Methods					
$DBSCAN$	Radius of Clusters, Minimum Number of Points in Clusters	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n \log n)$
$DENCLUE$	Radius of Clusters, Minimum Number of objects	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n \log n)$
$OPTICS$	Radius of Clusters (min,max), Minimum Number of objects	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n \log n)$
Miscellaneous Methods					
$STING$	Number of cells in lowest level, Number of objects in cell	Large Spatial Data Sets	Vertical and Horizontal Boundaries	Yes	$\mathcal{O}(n)$
$WaveCluster$	Number of Cells for each Dimension, Wavelet, Number of application of Transform	Arbitrary Shapes of Clusters, Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n)$
$CLIQUE$	Size of the Grid , Minimum Number of Points within each Cell	High Dimensional Large Data Sets	Arbitrary	Yes	$\mathcal{O}(n)$
$ScalableEM$	Initial Gaussian Parameters, Convergence Limit	Large Data Sets with Approximately Uniform Distribution	Spherical	No (?)	$\mathcal{O}(n)$

n=number of objects, k=number of clusters, s=size of sample, I=number of iterations

Table 3: Properties of Various Clustering Algorithms

5 Clustering Databases with Categorical Data

In this section of the paper, we consider databases with attributes whose values are categorical. These values, as already mentioned, cannot be ordered in a single way and, therefore, clustering of such data is a challenge. We summarize the characteristics of such data in the following list:

- Categorical Data have no single ordering: there are several ways in which they can be ordered, but there is no single one which is more semantically sensible than others.
- Categorical Data can be visualized depending on a specific ordering.
- Categorical Data define no *a priori* structure to work with [GKR98];
- Categorical Data can be mapped onto unique numbers and, as a consequence, Euclidean distance could be used to prescribe their proximities, with questionable consequences though;

One sensitive point is the last one. Guha et. al. give an example why this entails several dangers [GRS99]: assume a database of objects 1 through 6 with the following tuples: (a) $\{1, 2, 3, 5\}$, (b) $\{2, 3, 4, 5\}$, (c) $\{1, 4\}$, and (d) $\{6\}$[2]. These objects could be viewed as vectors of 0's and 1's denoting the presence of these objects inside the corresponding tuples. The four tuples become:

(a) $\{1, 2, 3, 5\} \rightarrow \{1, 1, 1, 0, 1, 0\}$;

(b) $\{2, 3, 4, 5\} \rightarrow \{0, 1, 1, 1, 1, 0\}$;

(c) $\{1, 4\} \rightarrow \{1, 0, 0, 1, 0, 0\}$;

(d) $\{6\} \rightarrow \{0, 0, 0, 0, 0, 1\}$;

Now, using Euclidean distance between tuples (a) and (b), we get:

$$(1^2 + 0^2 + 0^2 + 1^2 + 0^2 + 0^2)^{1/2} = \sqrt{2}$$

and this is the smallest distance between pairs of tuples, forcing (a) and (b) to be merged using a centroid-based hierarchical algorithm. The centroid of the new cluster is $\{0.5, 1, 1, 0.5, 1, 0\}$. In the following steps, (c) and (d) have the smallest distance and, thus will be merged. However, this corresponds to a merge of tuple $\{1, 4\}$ with tuple $\{6\}$, which have no objects in common, assuming here that matching based on presence is more important than matching based on absence. After that, we reach the conclusion that using a binary mapping of categorical attributes and Euclidean distance, some tuples that should not be in the same cluster end up being together. In this particular case, Hamming distance would perform better.

It becomes apparent, then, that we need different methods, and especially different similarity measures, to discover "natural" groupings of categorical data. The following subsections introduce the most important Clustering Algorithms on databases with categorical attributes.

[2] tuples are considered sets of categorical values, whose identifiers we report

5.1 The *k-modes* Algorithm

The first algorithm in the database community, oriented towards categorical data sets is an extension to *k-means*, called *k-modes* [Hua98]. The idea is the same as in *k-means* and the structure of the algorithm does not change. The only difference is in the similarity measure used to compare the data objects.

More specifically the differences are:

1. a different dissimilarity measure is used;
2. the *means* are replaced by *modes*;
3. a frequency based method is used to update modes.

Given two categorical data objects $\hat{x}$ and $\hat{y}$, their dissimilarity is found using the following expression:

$$d(\hat{x}, \hat{y}) = \sum_{i=1}^{n} \delta(x_i, y_i)$$

where

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

Intuitively, the above expression counts the number of mis-matches the two data objects have on their corresponding attributes. Note that every attribute is given the same weight. If we consider the frequencies of the values in a data set, then the dissimilarity expression becomes:

$$d(\hat{x}, \hat{y}) = \sum_{i=1}^{n} \frac{n_{x_i} + n_{y_i}}{n_{x_i} n_{y_i}} \delta(x_i, y_i)$$

where n_{x_i} and n_{y_i} are the numbers of objects in the data set with attributes values x_i and y_i for attribute i, respectively. The mode of a set is the value that appears the most in this set. For a data set of dimensionality n, every cluster c, $1 \leq c \leq k$, has a mode defined by a vector $Q^c = (x_1^c, x_2^c, \ldots, x_n^c)$. The set of Q^c's that minimize the expression:

$$E = \sum_{c=1}^{k} \sum_{\hat{x} \in c} d(\hat{x}, Q^c)$$

is the desired output of the method.

The similarities, in structure and behavior, with *k-means* are obvious, with *k-modes* carrying, unfortunately, all the disadvantages of the former. An interesting extension to data sets of both numerical and categorical attributes is that of *k-prototypes* [Hua97]. It is an integration of *k-means* and *k-modes* employing:

- s^r: dissimilarity on numeric attributes;
- s^c: dissimilarity on categorical attributes;
- dissimilarity measure between two objects:

  $$s^r + \gamma s^c$$

  where γ is a weight to balance the two parts and avoid favoring either type of attribute.

γ is a parameter specified by the user.

5.2 The *ROCK* Algorithm

ROCK (RObust Clustering using linKs) [GRS99] is a hierarchical algorithm for categorical data. Guha et al. propose a novel approach based on a new concept called the *links* between data objects. This idea helps to overcome problems that arise from the use of Euclidean metrics over vectors, where each vector represents a tuple in the data base whose entries are identifiers of the categorical values. More precisely, *ROCK* defines the following:

- two data objects $\hat{x}$ and $\hat{y}$ are called *neighbors* if their similarity exceeds a certain threshold θ given by the user, *i.e.*, $sim(\hat{x}, \hat{y}) \geq \theta$.
- for two data objects, $\hat{x}$ and $\hat{y}$, we define: $link(\hat{x}, \hat{y})$ is the number of common neighbors between the two objects, *i.e.*, the number of objects $\hat{x}$ and $\hat{y}$ are both similar to.
- the *interconnectivity* between two clusters C_1 and C_2 is given by the number of *cross-links* between them, which is equal to $\sum_{\hat{x}_q \in C_1, \hat{x}_r \in C_2} link(\hat{x}_q, \hat{x}_r)$.
- the expected number of links in a cluster C_i is given by $n_i^{1+2f(\theta)}$. In all the experiments presented $f(\theta) = \frac{1-\theta}{1+\theta}$

In brief, *ROCK* measures the similarity of two clusters by comparing the *aggregate interconnectivity* of two clusters against a user-specified static *interconnectivity model*. After that, the maximization of the following expression comprises the objective of *ROCK*:

$$E = \sum_{i=1}^{k} n_i \cdot \sum_{\hat{x}_q, \hat{x}_r \in C_i} \frac{link(\hat{x}_q, \hat{x}_r)}{n_i^{1+f(\theta)}}$$

The overview of *ROCK* is given in Figure 9.

Figure 9: Overview of *ROCK* [GRS99]

As we see, a random sample is drawn and a clustering algorithm (hierarchical) is involved to merge clusters. Hence, we need a measure to identify clusters that should be merged at every step. This measure between two clusters C_i and C_j is called the *goodness measure* and is given by the following expression:

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

where $link[C_i, C_j]$ is now the number of cross-links between clusters:

$$link[C_i, C_j] = \sum_{\hat{x}_q \in C_i, \hat{x}_r \in C_j} link(\hat{x}_q, \hat{x}_r)$$

The pair of clusters for which the above goodness measure is maximum is the best pair of clusters to be merged.

The computational complexity of *ROCK* is $\mathcal{O}(n^2 + nm_m m_a + n^2 \log n)$, where: m_m is the maximum number of neighbors of a data object and m_a is the average number of neighbors for a data object.

5.3 The *STIRR* Algorithm

STIRR (Sieving Through Iterated Relational Reinforcement) [GKR98] is one of the most influential methods for clustering categorical data sets. It uses an iterative approach where data objects are considered to be similar if the items with which they appear together in the database have a large overlap, regardless of the fact that the objects themselves might never co-occur. For example, car types `Civic` and `Accord` are similar since tuples `[Honda, Civic,1998]` and `[Honda,Accord,1998]` have a large overlap, *i.e.*, the values `Honda` and `1998`.

KEY FEATURES OF THE APPROACH

1. There is *no a-priori quantization*. This means that clustering the categorical data sets is purely done through their patterns of co-occurrence, without trying to impose an artificial linear order or numerical structure on them.
2. Gibson et al., wish to define a notion of similarity among items of the database that will apply even to items that *never occur together* in a tuple; their similarity is based on the fact that the sets of items with which they do co-occur have large overlap.
3. Viewing each tuple in the database as a set of values, the authors treat the entire collection of tuples as an abstract *set system*, or *hyper-graph* (Figure 10).

Besides the above, spectral methods relate "good" partitions of an undirected graph to the eigenvalues and eigenvectors of certain matrices derived from the graph. *STIRR* employs spectral partitioning on *hyper-graph clustering* using *non-linear dynamical systems*, instead of eigenvectors and proposes a weight-propagation method which works roughly as follows:

- It first seeds a particular item of interest, *e.g.*, `Honda`, with a small amount of weight. This is not a required assignment, since all weights can be initialized to 1;
- This weight propagates to items with which `Honda` co-occurs frequently;
- These items, having acquired a weight, propagate it further (back to other automobile manufacturers, perhaps);
- The process iterates until it converges;

We are now ready to present some of the main technical details of the approach. Following are the descriptions of the concepts used throughout this technique.

Representation : each possible value in each possible attribute is represented by an abstract node; an example of a data set represented this way, is given in Figure 10.

Configuration : the assignment of a weight w_v to each node v; we will refer to the entire configuration as w;

Normalization function $N(w)$: re-scales weights of the nodes associated with each attribute, so that their squares add up to 1 and ensure orthonormality.

Combining Operator $\oplus$: this is defined by any of the following:

1. *product operator*, Π: $\bigoplus(w_1, \ldots, w_k) = w_1 w_2 \ldots w_k$.
2. *addition operator*: $\bigoplus(w_1, \ldots, w_k) = w_1 + w_2 + \ldots + w_k$.
3. a generalization of the addition operator that is called the S_p *combining rule*, where p is an odd natural number. $S_p(w_1, \ldots, w_k) = (w_1^p + \ldots + w_k^p)^{(1/p)}$. Addition is simply an S_1 rule.
4. a *limiting* version of the S_p rules, which is referred to as S_∞. $S_\infty(w_1, \ldots, w_k)$ is equal to w_i, where w_i has the largest absolute value among the weights in $\{w_1, \ldots, w_k\}$.

Dynamical System : repeated application of a function f on some set of values.

Fixed points : points such that $f(u) = u$, for all nodes u.

Function f : maps one configuration to another and is defined as follows:

To update w_v:
  for every tuple $\tau = \{v, u_1, \ldots, u_{k-1}\}$, containing v do
    $x_\tau \leftarrow \bigoplus(w_{u_1}, \ldots, w_{u_{k-1}})$
  $w_v \leftarrow \sum_\tau x_\tau$

Tuple	Attribute a	b	c
1.	A	W	1
2.	A	X	1
3.	B	W	2
4.	B	X	2
5.	C	Y	3
6.	C	Z	3

is represented as

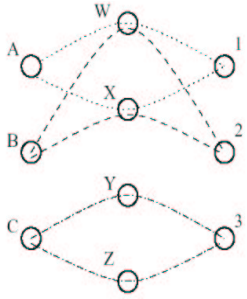


Figure 10: Representation of a database in *STIRR* [GKR98]

From the above, a choice of S_1 for $\bigoplus$ involves a linear term for each of the tuples, while Π and S_p, for $p > 1$, involve a non-linear one. The latter ones include the potential to represent co-occurence in a stronger way.

Finally, one of the key components of the approach is the choice of an initial configuration. Such a configuration could be chosen in two ways:

1. If we do not want to focus on the weight in a particular portion of the set of tuples, then we could adopt a *uniform initialization*, *i.e.*, all weights are set to 1 and are then being normalized, or we could adopt a *random initialization*, where all weights are taking values from the $[0, 1]$ interval with a normalization phase following again.
2. If we want to focus on a particular weight, we give this weight a higher value than the other.

The paper by Gibson et al. [GKR98] presents some useful theorems from spectral graph theory, to prove that *STIRR* converges and more over gives a result where some values have negative weights, while others have positive weights. Experimental results are given, when the algorithm is applied on a bibliographical database where database publications are successfully distinguished from theoretical ones. However, there are no experiments where more than two clusters were discovered. Finally, *STIRR* requires one pass over the data set.

5.4 The *CACTUS* Algorithm

An improvement to *STIRR* came from the *CACTUS (Clustering Categorical Data Using Summaries* algorithm [GGR99]. The main idea here is that summary information constructed from the data set is sufficient for discovering well-defined clusters. This way, the algorithm is able to find types of clusters that *STIRR* cannot discover, such as clusters with overlapping cluster-projections on any attribute and clusters where two or more clusters share the same cluster projection. An overview of *CACTUS* has as follows:

- **Summarization**: summaries of data are computed;
- **Clustering**: using the summaries, candidate clusters are computed;
- **Validation**: the set of candidate clusters are validated, after the clustering phase;

In *CACTUS* a set of categorical attributes $\{A_1, \ldots, A_n\}$ is assumed with domains $\{D_1, \ldots, D_n\}$, which are considered to be very small. Then, an *interval region* is defined by $S = S_1 \times \ldots \times S_n$ if for every i: $S_i \subseteq D_i$. If $a_i \in D_i$ and $a_j \in D_j$, $i \neq j$, the support $\sigma(a_i, a_j)$ is:

$$\sigma(a_i, a_j) = \Big|\{t \in D : t.A_i = a_i \text{ and } t.A_j = a_j\}\Big|$$

i.e., the number of tuples where a_i and a_j co-occur. Now the support of the region S, $\sigma(S)$ is the number of tuples in the data set that belong to S. If all attributes are independent and their values are equally likely:

$$\text{Expected Support of } S: E[\sigma(S)] = |D| \cdot \frac{|S_1| \times \ldots \times |S_n|}{|D_1| \times \ldots \times |D_n|}$$

and

$$\text{Expected Support of } (a_i, a_j): E[\sigma(a_i, a_j)] = |D| \cdot \frac{1}{|D_i| \times |D_j|}$$

Values a_i and a_j are now *strongly connected* if:

$$\sigma(a_i, a_j) > \alpha \cdot E[\sigma(a_i, a_j)]$$

and a cluster is defined by the following:

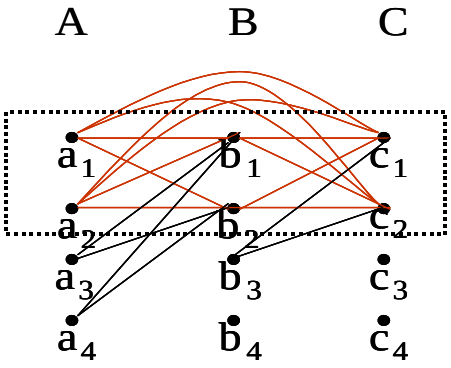
Cluster in *CACTUS* : $C = C_i \times \ldots \times C_n$ is a cluster if and only if:

1. for all i, j, C_i and C_j are *strongly connected*, *i.e.* all pairs of values in them are strongly connected;
2. C_i is maximal for all i
3. $support(C)$ is α times the expected;

The above definition implies that clusters could be regions, as shown in Figure 11, where region $\{a_1, a_2\} \times \{b_1, b_2\} \times \{c_1, c_2\}$, (dotted area), defines a cluster. After that, we can delve into the stages of *CACTUS*.

In the *summarization* phase, two types of summaries are computed:

- *inter-attribute* summaries: counts of all strongly connected attribute value pairs from different attributes;



(a)

Figure 11: Example using *CACTUS* [GGR99]

- *intra-attribute* summaries: computation of similarities between attribute values of the same attribute;

In the *clustering* phase, *CACTUS* analyzes each attribute, in order to compute all *cluster projections*. For an attribute A_i, its projection is a subset of values from this attribute, that is strongly connected to the attribute values of every other attribute A_j, $i \neq j$, denoted by S_i^j. For example, if we consider attribute A in Figure 11, then we can compute $S_A^B = \{a_1, a_2, a_3, a_4\}$ and $S_A^C = \{a_1, a_2\}$ and the projection of A is $S_A^B \bigcap S_A^C$. After that, the clustering phase synthesizes, in a level-wise manner, candidate clusters on sets of attributes from the cluster-projections on individual attributes. Intuitively, this step extends the previous one by increasing the dimensionality of clusters by one.

Finally, in the *validation* phase, the algorithm recognizes false candidates by checking if the support of each candidate cluster is greater than the required threshold.

CACTUS is a scalable algorithms since it requires only one pass of the data set. A second pass is needed for the validation phase, without any implications on scalability. The authors present sufficient experimental results that prove a better performance compared to *STIRR*, in terms of time and the number of attributes.

5.5 Discussion

Clustering Categorical data is a challenging issue as we already argued. The algorithms we just discussed introduce a variety of methods to tackle this problem and give different solutions in terms of their performance, with respect to the time it takes for the algorithms to run when the number of tuples and dimensions change. On the other hand, quality of produced clusters is measured by the user's expertise and examination of the results. Due to space considerations, it is impossible to present all technical details of the approaches. However, our intention is to compare these categorical clustering algorithms and stress their advantages and weaknesses.

Hence, we first introduced *k-modes* and *k-prototypes*, which first appeared in the database community, after the observation that *means*, in the *k-means* method, could be replaced by *modes* so as to compare categorical attributes. Both *k-modes* and *k-prototypes* are scalable but do not handle outliers well.

k-modes is a partitional method. The first, and probably the only, representative algorithm from the

hierarchical family is *ROCK*. Its novelty is based on the assumption that an attribute value, in addition to its occurrence, should be examined according to the number of other attribute values it exists with. *ROCK* works totally differently than *k-modes* not only because it is hierarchical but also due to the fact that it works on samples of the data. It is more scalable than other sampling techniques but less than *k-modes*. In our opinion, the main disadvantage of *ROCK* is that it employs sampling and the results highly depend on it.

A well presented and novel technique is *STIRR*, based on the iterative computation of new weights for the nodes of a graph. The idea seems similar to the work done by Jon Kleinberg on discovering authoritative sources on the web [Kle98], however *STIRR* highly depends on the choice of the *combining operator*, $\oplus$ and the notion of the iterative function, f, which defines a *dynamical system*. Gibson et. al. argue that there hasn't been much proved about the behavior of dynamical systems in the literature and they base their proof of convergence and the discovery of final clusters, on results that come from the spectral graph theory research area. An additional observation is that *STIRR* gives a result where each value has acquired either a positive or negative weight, and the reporting of final clusters might involve a heavy post-processing stage. Moreover, choosing different initial configurations, the authors discovered different partitions of their data set, which leads to the conclusion that initial weights have an impact on the final result. Note that the different clusterings could be meaningful, but still they are not the same and cannot be directly compared with each other. On the other hand, *STIRR* converges quickly and identifies clusters in the presence of irrelevant values, *i.e.*, values that co-occur with no other values [GKR98].

Finally, we presented *CACTUS*, an approach that entails the computation of summaries from the underlying data set (a concept similar to the one used in *BIRCH*), as well as techniques used to find frequent item-sets in databases. *CACTUS*'s summaries have to do with the support of pairs of attribute values from different attributes and the similarity of values that belong to the same attribute. The disadvantage of this algorithm is that, as the number of dimensions grow, when computing the summaries, each attribute value of a tuple is compared with every other value while the results can be kept in main memory only if the domain size of every attribute is very small. We believe that larger domains would have a destructive result in the current implementation, which requires intensive CPU computations.

Table 4 gives an overview of the features of each algorithm. In our opinion it becomes obvious that there is no "optimal solution" for the problem of categorical data and the choice of an algorithm highly depends on the knowledge of the data itself, the parameters of individual approaches and the resources that are available. From our presentation of clustering methods, it is also obvious that there are no good techniques for handling large data sets of mixed attributes. *k-prototypes* or *EM* can be employed but their performance degrades as the database grows in size and number of dimensions. At the same time, the majority of data sets nowadays contain attributes with mixed values, which require robust clustering techniques. Note that some of the values might be missing either due to reporting errors or unavailability of information, something that is not explicitly handled by the presented approaches.

In the next section we give our suggestions for future research.

6 Research Challenges

As we discussed earlier, the problem of cluster analysis becomes very interesting, and at the same time challenging, when the data in hand contain categorical attributes. The number of algorithms, however, for the discovery of groups in such data is limited, compared to the research devoted on data sets with numerical data. Further, few algorithms (perhaps only *EM*) deal with mixtures of values, *i.e.*, attributes of numerical and categorical values. Ideally, a clustering algorithm should:

Categorical Clustering Methods				
Algorithm	*Input Parameters*	*Optimized For*	*Outlier Handling*	*Computational Complexity*
$k-prototypes$	Number of Clusters	Mixed Data Sets	No	$\mathcal{O}(n)$
$ROCK$	Number of Clusters	Small Data Sets with Noise	Yes	$\mathcal{O}(n^2 + nm_m m_a + n^2 \log n)$
$STIRR$	Initial Configuration, Combining Operator, Stopping Criteria	Large Data Sets with Noise	Yes	$\mathcal{O}(n)$
$CACTUS$	Support Threshold α, Validation Threshold	Large Data Sets with Small Dimensionality and Small Attribute Domain Sizes	Yes	$\mathcal{O}(n)$

n=number of objects, k=number of clusters, m_m,m_a=maximum and average number of neighbors for an object, respectively.

Table 4: Properties of Categorical Clustering Algorithms

- scale well, *i.e.*, at most one scan of the data is required;
- handle deviations efficiently;
- discover arbitrary-shaped clusters (only for non-categorical attributes);
- be insensitive to the order of input;
- give reasonable execution times in the presence of high dimensional data sets;
- present a succinct model of the clusters.

In the algorithms we presented, some of the above desiderata are not met. For example, few of the algorithms work well with large number of attributes, due to the inherent sparsity, while there are considerable trade-offs in the others. In particular, the output of *BIRCH* depends not only on the parameters but also on the order of input of the data set.

After that, we believe that there are some key points worth mentioning in this challenging area, embodying promising opportunities for further research:

Numerical versus Categorical. So far, there is no clear comparison between algorithms on numerical and categorical data. The former ones have, in general, better mathematical properties, and the quality of some of them can be quantified [HVB00]. Besides, given a data set, proper visualization may sometimes give hints as to which technique to be used. This does not hold for categorical databases and we cannot tell which algorithm works better on them, *i.e.*, which algorithm discovers the most "natural" groups. Their performance is usually given in terms of their scalability.

In such a situation we can formulate a comparison of algorithms in terms of their performance on numerical data sets only. Numerical values can be considered as categorical and moreover a numerical algorithm can be chosen that works well on it. A real challenge would now be to perform clustering using a categorical technique and discover differences in the results. We should stress here that the main interest is not in the performance of algorithms in terms of time and space, but basically in terms of the clusters found.

Comparison of Clusterings. Categorical data presents difficult challenges for graphical display. On the other hand, numerical data can be graphically displayed, introducing hints for the number of clusters or shapes for them, most of the times. If one is working on categorical values, the expertise of domain knowledgeable users is needful, and considerable post-processing of the results is essential in order to assess the goodness of the results. In both numerical and categorical algorithms, even the same method gives different results for different runs (if the order of input or the algorithm parameters change), and the comparison between the partitionings becomes a difficult problem. In the community of Software Engineering, Tzerpos and Holt recently introduced a measure, called *MoJo*, to quantify how far apart two different partitionings of the same data set are from each other [TH99]. Their approach counts the number of *Moves*, of objects, and *Joins*, of partitions, required to convert one of them to the other. They deal only with software entities, and in this context they give a weight of one to each of the operations. A similar technique does not exist for cluster analysis algorithms that involve data sets with large amounts of data and high dimensionality. Furthermore, *MoJo*'s linear performance is not formally proved.

Such an approach would be beneficial in the comparison of our clustering algorithms. However, we need to propose a symmetric measure and weigh the operations we should perform to convert

one partition to the other. For example, speaking in information theoretical terms, the move of one object to a different partition might have a positive or negative change in the partitions entropy. In this respect, we will attempt to quantify the answer of the following question: *Given two clusterings $\mathcal{A}$ and $\mathcal{B}$ over the same data set, what is the amount of information (usually given by the number of bits) needed for the clusters $A_i \in \mathcal{A}, 1 \leq i \leq k$ and $B_j \in \mathcal{B}, 1 \leq j \leq m$, with k not necessarily equal to m, to communicate their values to each other*. Given this quantity, we may then be able to measure the quality of clustering $\mathcal{A}$ given $\mathcal{B}$ and vice versa.

Schema Discovery Through Clustering. There is the case where large database tables are the product of several joins among smaller tables, which eventually expire or become unavailable. In other cases the schema of a large table is the result of a schema mapping application, as in *Clio* [**?**], and the *Amalgam* project [**?**]. We believe that clustering, as a means of a succinct representation of similar groups, might lead to the break up of large relations into smaller ones, strongly coupled with smaller schemas. In this case, a cluster comparison seems necessary, while at the same time we need efficient handling of absent values in the data set.

Stability of Clustering Techniques. Data sets evolve and, as a consequence, their interdependencies and groupings change. One of the desired properties of the cluster analysis is to remain stable whenever the data changes by small amounts, that is new data objects are inserted and existing ones change or get deleted. But, how stable are *BIRCH*, *STIRR*, *CACTUS* and the collection of methods we described ? If the data set analyzed consists of daily transactions and a daily clustering technique is part of the data mining process, it is easy to understand that all changes in the data affect the results, where extreme deviations from previous ones are unwanted. The *stability* of categorical clustering algorithms is an under-studied issue and has not attracted much attention. It would be interesting to know how much the output of a clustering algorithm is affected when the input changes slightly. We intend to propose a measure of *stability* and the effects of changes in the data set, *e.g.* measure the difference in the resulting clusters.

Other Techniques in Clustering. When performing clustering on categorical data, it is obvious that the techniques used are based on co-occurrences of the data objects or the number of neighbors they have, and at the same time do not deal with mixed attribute types. *STIRR* adopts theory from the dynamical systems area and spectral graph theory to give a solution. *CACTUS* employs techniques similar to the ones used in frequent item-set discovery and summarizes information in a similar way as *BIRCH* does.

It is our belief that there exist methods not yet applied to categorical attributes which mainly lead to more succinct result (recall that *STIRR* needs a painful post-processing step to describe the results). For instance, there are techniques employed by the machine learning community which are used to cluster documents according to terms they contain [ST00]. It is our interest to examine the properties of these methods and investigate whether it can be effectively applied to categorical as well as mixed attribute types.

7 Conclusions

Clustering lies at the heart of data analysis and data mining applications. The ability to discover highly correlated regions of objects when their number becomes very large is highly desirable, as data sets grow and their properties and data interrelationships change. At the same time, it is notable that any clustering "is a division of the objects into groups based on a set of rules – it is neither true or false" [Eve93].

In this paper we described the process of clustering from the data mining point of view. We gave the properties of a "good" clustering technique and the methods used to find meaningful partitionings. At the same time, we concluded that research has emphasized numerical data sets, and the intricacies of working with large categorical databases is left to a small number of alternative techniques. We claimed that new research solutions are needed for the problem of categorical data clustering, and presented our ideas for future work.

References

[ABKS96] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proceedings of the International Conference on Management of Data, (SIGMOD)*, volume 28(2) of *SIGMOD Record*, pages 49–60, Philadelphia, PA, USA, 1–3 June 1996. ACM Press.

[AGGR98] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. Automatic Subspace Clustering of High Dimensional Data for Data Mining Applications. In *Proceedings of the International Conference on Management of Data, (SIGMOD)*, volume 27(2) of *SIGMOD Record*, pages 94–105, Seattle, WA, USA, 1–4 June 1998. ACM Press.

[And73] Michael R. Anderberg. *Cluster analysis for applications*. Academic Press, 1973.

[BFR98] Paul S. Bradley, Usama Fayyad, and Cory Reina. Scaling Clustering Algorithms to Large Databases. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, (KDD)*, pages 9–15, New York, NY, USA, 27–31 August 1998. AAAI Press.

[BFR99] Paul S. Bradley, Usama Fayyad, and Cory Reina. Scaling EM (Expectation-Maximization) Clustering to Large Databases. Technical Report MSR-TR-98-35, Microsoft Research, Redmond, WA, USA, October 1999.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jöerg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, (KDD)*, pages 226–231, Portland, OR, USA, 2–4 August 1996. AAAI Press.

[Eve93] Brian S. Everitt. *Cluster Analysis*. Edward Arnold, 1993.

[GGR99] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. CACTUS: Clustering Categorical Data Using Summaries. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining, (KDD)*, pages 73–83, San Diego, CA, USA, 15–18 August 1999. ACM Press.

[Gil58] E. W. Gilbert. Pioneer Maps of Health and Disease in England. *Geographical Journal*, 124: 172–183, 1958.

[GKR98] David Gibson, Jon M. Kleinberg, and Prabhakar Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. In *Proceedings of the 24th International Conference on Very Large Data Bases, (VLDB)*, pages 311–322, New York, NY, USA, 24–27 August 1998. Morgan Kaufmann.

[GRS98] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: An Efficient Clustering Algorithm for Large Databases. In *Proceedings of the International Conference on Management of Data, (SIGMOD)*, volume 27(2) of *SIGMOD Record*, pages 73–84, Seattle, WA, USA, 1–4 June 1998. ACM Press.

[GRS99] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A Robust Clustering Algorithm for Categorical Atributes. In *Proceedings of the 15th International Confererence on Data Engineering, (ICDE)*, pages 512–521, Sydney, Australia, 23–26 March 1999. IEEE Press.

[HAK00] Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the Nearest Neighbor in High Dimensional Spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases, (VLDB)*, pages 506–515, Cairo, Egypt, 10–14 September 2000. Morgan Kaufmann.

[HK98] Alexander Hinneburg and Daniel A. Keim. An Efficient Approach to Clustering in Large Multimedia Databases with Noise. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, (KDD)*, pages 58–65, New York, NY, USA, 27–31 August 1998. AAAI Press.

[HK01] Jiawei Han and Michelle Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[Hua97] Zhexue Huang. Clustering Large Data Sets with Mixed Numeric and Categorical Values. In *Proceedings of the 1st Pacific-Asia Conference on Knowledge Discovery and Data Mining, (PAKDD)*, pages 21–34, Singapore, 1997. Springer.

[Hua98] Zhexue Huang. Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values. *Workshop on Research Issues on Data Mining and Knowledge Discovery, (DMKD)*, 2(3): 283–304, 1998.

[HVB00] Maria Halkidi, Michalis Vazirgiannis, and Yannis Batistakis. Quality Scheme Assessment in the Clustering Process. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, volume 1910 of *Lecture Notes in Computer Science*, pages 265–276, Lyon, France, 13-16 September 2000. Springer.

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

[JLVV99] Matthias Jarke, Maurizio Lenzerini, Yannis Vassiliou, and Panos Vassiliadis. *Fundamentals of Data Warehouses*. Springer, 1999.

[Kle98] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, San Francisco, CA, USA, 25–27 January 1998. ACM Press.

[KR90] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[NH94] Raymond T. Ng and Jiawei Han. Efficient and Effective Clustering Methods for Spatial Data Mining. In *Proceedings of the 20th International Conference on Very Large Data Bases, (VLDB)*, pages 144–155, Santiago, Chile, 12–15 September 1994. Morgan Kaufmann.

[SCZ98] Gholamhosein Sheikholeslami, Surojit Chatterjee, and Aidong Zhang. WaveCluster: A Multi-Resolution Clustering Approach for Very Large Spatial Databases. In *Proceedings of the 24th International Conference on Very Large Data Bases, (VLDB)*, pages 428–439, New York, NY, USA, 24–27 August 1998. Morgan Kaufmann Publishers.

[SEKX98] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-Based Clustering in Spatial Databases: The Algortihm GDBSCAN and Its Applications. *Workshop on Research Issues on Data Mining and Knowledge Discovery, (DMKD)*, 2(2): 169–194, 1998.

[ST00] Noam Slonim and Naftali Tishby. Document Clustering Using Word Clusters via the Information Bottleneck Method. In *Proceedings of the 23d Annual International ACM Conference on Research and Development in Information Retrieval, (SIGIR)*, pages 208–215, Athens, Greece, 24-28 July 2000. ACM Press.

[TH99] Vassilios Tzerpos and Richard C. Holt. MoJo: A Distance Metric for Software Clustering. In *Proceeedings of the 6th Working Conference on Reverse Engineering, (WCRE)*, pages 187–195, Atlanta, GA, USA, 6–8 October 1999. IEEE Press.

[WYM97] Wei Wang, Jiong Yang, and Richard R. Muntz. STING: A Statistical Information Grid Approach to Spatial Data Mining. In *Proceedings of the 23rd International Conference on Very Large Data Bases, (VLDB)*, pages 186–195, Athens, Greece, 26–29 August 1997. Morgan Kaufmann Publishers.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient Data Clustering Method for Very Large Databases. In *Proceedings of the International Conference on Management of Data, (SIGMOD)*, volume 25(2) of *SIGMOD Record*, pages 103–114, Motreal, QB, Canada, 4–6 June 1996. ACM Press.