Scalable Clustering of Categorical Data and Applications

by

Periklis Andritsos

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Scalable Clustering of Categorical Data and Applications

Periklis Andritsos

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2004

Clustering is widely used to explore and understand large collections of data. In this thesis, we introduce LIMBO, a scalable hierarchical categorical clustering algorithm based on the *Information Bottleneck (IB)* framework for quantifying the relevant information preserved when clustering. As a hierarchical algorithm, LIMBO can produce clusterings of different sizes in a single execution. We also define a distance measure for categorical tuples and values of a specific attribute. Within this framework, we define a heuristic for discovering candidate values for the number of meaningful clusters.

Next, we consider the problem of database design, which has been characterized as a process of arriving at a design that minimizes redundancy. Redundancy is measured with respect to a prescribed model for the data (a set of constraints). We consider the problem of doing database redesign when the prescribed model is unknown or incomplete. Specifically, we consider the problem of finding structural clues in a data instance, which may contain errors, missing values, and duplicate records. We propose a set of tools based on LIMBO for finding structural summaries that are useful in characterizing the information content of the data. We study the use of these summaries in ranking functional dependencies based on their data redundancy.

We also consider a different application of LIMBO, that of clustering software artifacts. The majority of previous algorithms for this problem utilize structural information in order to decompose large software systems. Other approaches using non-structural in-

formation, such as file names or ownership information, have also demonstrated merit. We present an approach that combines structural and non-structural information in an integrated fashion. We apply LIMBO to two large software systems, and the results indicate that this approach produces valid and useful clusterings.

Finally, we present a set of weighting schemes that specify objective assignments of importance to the values of a data set. We use well established weighting schemes from information retrieval, web search and data clustering to assess the importance of whole attributes and individual values.

Contents

List of Tables

List of Figures

Chapter 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 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. The data sets of interest are becoming larger, and their dimensionality prevents easy analysis and validation of the results. Clustering has also been widely addressed 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.

Data Clustering refers to the specific problem of partitioning a set of objects into a fixed number of subsets such that the similarity of the objects in each subset is maximized and the similarity across subsets is minimized. Clustering is regarded as a specific branch of the *Data Mining* field. Some of the important requirements that data mining

algorithms should fulfill are *scalability* in terms of memory requirements and execution time, and consistent *quality* of the results as the size of the input grows. Especially the scalability requirement distinguishes data mining algorithms from the algorithms used in *Machine Learning*.

Most of the clustering algorithms present in the literature focus on data sets where objects are defined on a set of numerical values. In such cases, the dissimilarity (or similarity) of objects can be decided using well-studied measures that are derived from geometric analogies. Special attention should be paid when the data sets to be clustered contain multiple attributes that describe each object, but where the domains of the individual attributes are unordered. We term such data sets *categorical*.

In this thesis, we will first present LIMBO, a scalable algorithm that can be applied to categorical data. This algorithm is based upon a related machine learning algorithm. However, existing techniques are not effective when the input data is very large. Our algorithm scales well with the size of the data, incurring little loss in the quality of the results. We have applied this algorithm on data organized into database tables as well as data from the field of *Reverse Engineering*. Reverse Engineering is the process of analyzing a subject system in order to identify the system's components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction [CI90]. Our main objective is to find natural groupings of software artifacts, such as files or functions.

The growth of distributed databases has led to larger and more complex databases the structure and semantics of which are more difficult to understand. In heterogeneous applications, data may be exchanged or integrated. This integration may introduce anomalies such as duplicate records, missing values, or erroneous values. In addition, the lack of documentation or the unavailability of the original designers can make the task of understanding the structure and semantics of databases a very difficult one. Using LIMBO, we explore the benefits of using clustering to identify anomalies as well as the

existence of duplicate data values in large data sets and define a set of techniques that help in the redesign of large integrated sources.

Finally, we evaluate the quality of the LIMBO algorithm when the attributes and/or individual values have different degrees of influence in the clustering process. The co-occurrence of values in the data, as well as specific properties they hold, such as dependencies given through a graph structure, may lead to better clustering relative to the presumption of uniform importance of individual attributes and/or attribute values. We have explored a number of approaches for assigning importance to values in relational databases, as well as in data sets from software reverse engineering.

We believe that the field of clustering categorical data and its applications may benefit from our approach and the techniques we present in this thesis. However, the work presented here is not complete. Several challenges have appeared through the completion of this work, and we present our suggestions for future work at the end of the thesis.

More specifically the contributions we make in this thesis are given below.

- **Scalable Clustering for Categorical Data.**

  We introduce LIMBO, a scalable hierarchical categorical clustering algorithm that builds on the *Information Bottleneck (IB)* [TPB99] framework for quantifying the relevant information preserved when clustering. As a hierarchical algorithm, LIMBO has the advantage that it can produce clusterings of different sizes in a single execution. We use the IB framework to define a distance measure for categorical tuples and we also present a novel distance measure for categorical attribute values. We show how the LIMBO algorithm can be used to cluster both tuples and attribute values. LIMBO handles large data sets by producing a summary model of a user-specified size for the data.

- **Information-Theoretic Techniques for Structure Discovery in Large Data Sets.**

  We consider the problem of doing data redesign in an environment where the pre-

scribed model is unknown or incomplete. Specifically, we consider the problem of finding structural clues in an instance of data, an instance which may contain errors, missing values, and duplicate records. We propose a set of information-theoretic techniques for finding structural summaries that are useful in characterizing the information content of the data, and ultimately useful in database design. We provide algorithms for creating these summaries over large, categorical data sets. We study the use of these summaries in a database design task, that of ranking functional dependencies based on their data redundancy.

- **Clustering Using Structural and Non-Structural Software Information.** The majority of the algorithms in the software clustering literature utilize structural information in order to decompose large software systems, *i.e.*, information derived from the implementation of a system. Other approaches use non-structural data, such as the file names or ownership information, and have also demonstrated merit. However, there is no intuitive way to combine information obtained from these two different types of techniques. We propose an approach that combines structural and non-structural information in an integrated fashion using the LIMBO hierarchical clustering algorithm. Our results indicate that this approach produces valid and useful clusterings of large software systems, and that LIMBO can be applied to evaluate the usefulness of various types of non-structural information to the software clustering process.

- **Evaluation of Value Weighting Schemes in Clustering Categorical Data.** We present a set of weighting schemes that allow for an objective assignment of importance on the values of a data set. We use well established weighting schemes from information retrieval, web search and data clustering to assess the importance of whole attributes and individual values. To our knowledge, this is the first work that considers weights in the clustering of categorical data. We perform cluster-

ing in the presence of importance for the values within the LIMBO framework. Our experiments were performed in a variety of domains, including data sets used previously in clustering research and three data sets from large software systems.

Thesis Organization

This thesis contains seven chapters. Chapter 2 surveys background material on the problem of clustering, presenting several existing algorithms used to cluster categorical data. Chapter 3 presents the foundation of our approach, which is termed *Information Bottleneck*, as well as LIMBO, our scalable algorithm for clustering categorical data. Chapter 4 introduces the set of novel information-theoretic techniques for the discovery of structure in data sets and our approach for ranking functional dependencies. Chapter 5 presents our results for clustering structural and non-structural software data for the purpose of reverse-engineering. Chapter 6 discusses approaches to finding objective weights for the importance of attributes and/or attributes values in a data set to be clustered. Finally, Chapter 7 concludes the thesis and presents directions for future research.

Chapter 2

Clustering State of The Art

In this chapter, we present the state of the art in clustering techniques, mainly from the data mining point of view. We discuss the steps clustering involves and investigate advantages and disadvantages of proposed solutions. We focus more on the solutions that are closer to our research.

2.1 Problem Definition and Stages

There are several definitions for clustering.[1] Intuitively, 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 [JD88]. The notions of similarity and dissimilarity will become clear in a later section.

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*.

[1]The word "clustering" is used both as a noun and as a verb to mean the process of partitioning.

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 are 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, if any.

Validation : Validation is often based on manual examination and visual techniques. However, as the amount of data and its dimensionality grow, we may 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.

This list of stages is given for exposition purposes since we do not propose solutions for each one of them. We mainly address the problem of *Clustering Strategy* by proposing a new and scalable algorithm for categorical data, and the problem of *Clustering Tendency* by proposing a heuristic for identifying appropriate values for the number of clusters that exist in a data set. Although we do not introduce new *Validation* techniques, we use a large number of measures, already given in clustering research, in order to achieve better results.

2.2 Data Types and Their Measures

A database consists of a set of records where each one of them is defined over a set of attributes. Each attribute has a domain, the set from which it takes its values, called *attribute values* or simply *values*. When we have a fixed number of attributes (often with different semantics), the records are ordered lists and are called *tuples*. In other cases, records are just sets often defined over a different number of values. In clustering, the objects of analysis are usually the tuples in a data set such as persons, salaries, opinions, software entities and many others. In the following chapters we also use individual values or whole attributes as our objects to be clustered. These objects must be carefully presented in terms of their characteristics, which greatly influences the results of a clustering algorithm.

A comprehensive categorization of the different types of attributes found in most data sets provides a helpful means for identifying the differences among data elements. A detailed taxonomy of the types is presented by Anderberg [And73]. In the following subsections we present a brief classification based on the *Domain Size* and the *Measurement Scale*.

2.2.1 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*. A list of sybmols used in this thesis is given in Appendix A. If $\hat{x}$, $\hat{y}$ and $\hat{z}$ are three data objects (tuples in this case) belonging to $\mathcal{D}$, each one of them has the form: $\hat{x} = (x_1, x_2, \ldots, x_m)$, $\hat{y} = (y_1, y_2, \ldots, y_m)$ and $\hat{z} = (z_1, z_2, \ldots, z_m)$, where m is the *dimensionality*, while each x_i, y_i and z_i, $1 \leq i \leq m$, is a *feature*, or an *attribute* of the corresponding object. We have the following main classes [And73]:

- An attribute is *continuous* if, between any two values of the attribute, there exists an infinite number of values. Examples of such attributes could be the temperature, colour, or sound intensity.

- An attribute is *discrete* if the elements of its domain 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 with domains of exactly two discrete values. They comprise a special case of discrete attributes, and we present as examples the `Yes/No` responses to a poll and the `Male/Female` gender entries of a database of employees.

2.2.2 Classification Based on the Measurement Scale

This classification distinguishes attributes according to their measurement scales. We can think of a scale as a way of ordering the data, and, thus, a way to compare the values of a particular domain. Suppose we have an attribute i and two objects $\hat{x}$ 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 for the two values we can either have $x_i = y_i$ 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 of a person 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, for the two values, we can either have $x_i = y_i$ or $x_i < y_i$ or $x_i > y_i$. Examples include the medals won by athletes (*e.g.*, gold, silver and bronze).

3. An *interval scale* measures values in a 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 number of years of education for a person or TV channel numbers.

4. A *ratio scale* is an interval scale with a meaningful zero point. Due to the zero point, ratios of the values, x_i/y_i are meaningful. Examples include weight, height and the number of children in a family.

Nominal- and ordinal-scaled attributes are called *qualitative* or *categorical* attributes, while interval- and ratio-scaled are called *quantitative* [And73]. It is common that quantitative (or numerical) attributes are measured in different units, such as kilograms and centimeters. To avoid problems that may arise from the existence of these units, *standardization* of the attributes may be needed in order to perform clustering under a common measurement unit [HK01]. In this thesis we focus on clustering techniques applied to categorical data. A more comprehensive definition of categorical data will be given in Section 2.4.

2.2.3 The Concepts of Similarity and Dissimilarity

When the characteristics of the attributes 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 one another. The notions of *similarity* or *dissimilarity* are employed to help in the process of classification. These terms are used together with the notion of "measure", "index" or "coefficient" [And73, JD88, HK01] in order to quantify the extent to which pairs of data objects can be deemed as similar or dissimilar. Generally speaking, the more two objects resemble each other, the larger their similarity is and the smaller their dissimilarity. Dissimilarity can be measured in many ways and one of them is *distance*.

Moreover, distance can be measured using any one of a variety of *distance measures*. All measures depend on the type of attributes we are analyzing. For example, with 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 2.2, 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 $\hat{x} = \hat{y}$: identity;
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]. To avoid any confusion, we shall be using the term measure, mentioning whether it computes similarity or dissimilarity.

Some indicative metrics used to determine the distance of objects $\hat{x}$ and $\hat{y}$, when they are defined over numerical attributes (mainly Interval-Scaled) are special cases of the *Minkowski Distance* defined as:

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

where q is a positive integer. Thus, we have the following:

- for $q = 2$, the *Euclidean Distance* is defined as:

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

- for $q = 1$, the *Manhattan Distance* is defined as:

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

- for $q \to \infty$, the *Maximum Distance* is defined as:

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

Notice that the above metrics cannot be used when we have a combination of numerical and categorical values.

We now present measures for assessing the similarity of binary valued attributes. Before doing so, we introduce the concept of *contingency tables* [HK01]. Such a table is given in Table 2.1.

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

Table 2.1: 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 attributes, i, for which $x_i = y_i = 1$;
- β is the number of attributes, i, for which $x_i = 1$ and $y_i = 0$;
- γ is the number of attributes, i, for which $x_i = 0$ and $y_i = 1$;
- δ is the number of attributes, i, for which $x_i = y_i = 0$;

After that, we have the following metrics to measure similarity:

- *Simple Matching Coefficient*, defined as:

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

- *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.

One final note regarding 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 by assigning 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, $1 \leq i \leq m$, we then have the *weighted Euclidean Distance*, defined as:

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

Attribute value weights will be discussed more in Chapter 6.

2.3 Categorization of Clustering Techniques and Previous Work

Once we have identified the types of data our data set contains and the measures that can be applied to them, we need to chose a specific clustering algorithm to produce groups of similar data. We expect clustering algorithms to produce groups of objects that satisfy a certain criterion and obey a particular strategy in order to combine (or

merge) intermediate results. Hence we expect every clustering technique to include three main components:

1. The measure used to assess similarity or dissimilarity between pairs of objects.

2. The particular strategy followed in order to merge intermediate results. This strategy obviously affects the way the final clusters are produced, since we may merge intermediate clusters according to the distance of their closest or furthest points, or the distance of the average of their points.

3. An objective function that needs to be minimized or maximized as appropriate, in order to produce final results.

Many diverse techniques have appeared in order to discover cohesive groups in large data sets. In the following two sections, we present the two classical techniques for clustering.

2.3.1 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, so that an objective function is optimized.

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 is why common solutions start with an initial, usually random, partition and proceed with its refinement. A better practice is to run the partitional algorithm for several different sets of k initial points (considered as representatives), and keep the result with the best quality.

Partitional clustering algorithms try to locally improve a certain criterion. The majority of them could be considered as greedy algorithms, *i.e.*, algorithms that at each step choose the best solution and may not lead to optimal results in the end. The best solution at each step is the placement of a certain object in the cluster for which the representative point is nearest to the object.

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]. All of them are applicable to data sets with numerical attributes.

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 chooses.

These methods generally follow a greedy 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 until a desired number of clusters is reached. This is similar to the approach followed by divide-and-conquer algorithms, *i.e.*, algorithms that, given an instance of the problem to be solved, split this instance into several, smaller, sub-instances (of the same problem), independently solve each of the sub-instances and then combine the sub-instance solutions so as to yield a solution for the original instance.

Figure 2.1(a) gives an example of two partitional clusterings performed on the same data set, with different initial parameters. A ‘‘+’’ sign denotes the centre of a cluster, which in this case is defined as the mean of the values of a particular cluster. At the same time, Figure 2.1(b) depicts the *dendrogram* produced by either a divisive or agglomerative clustering algorithm. A dendrogram is a tree structure that depicts the sequence of merges during clustering together with the corresponding values of distance (or similarity).

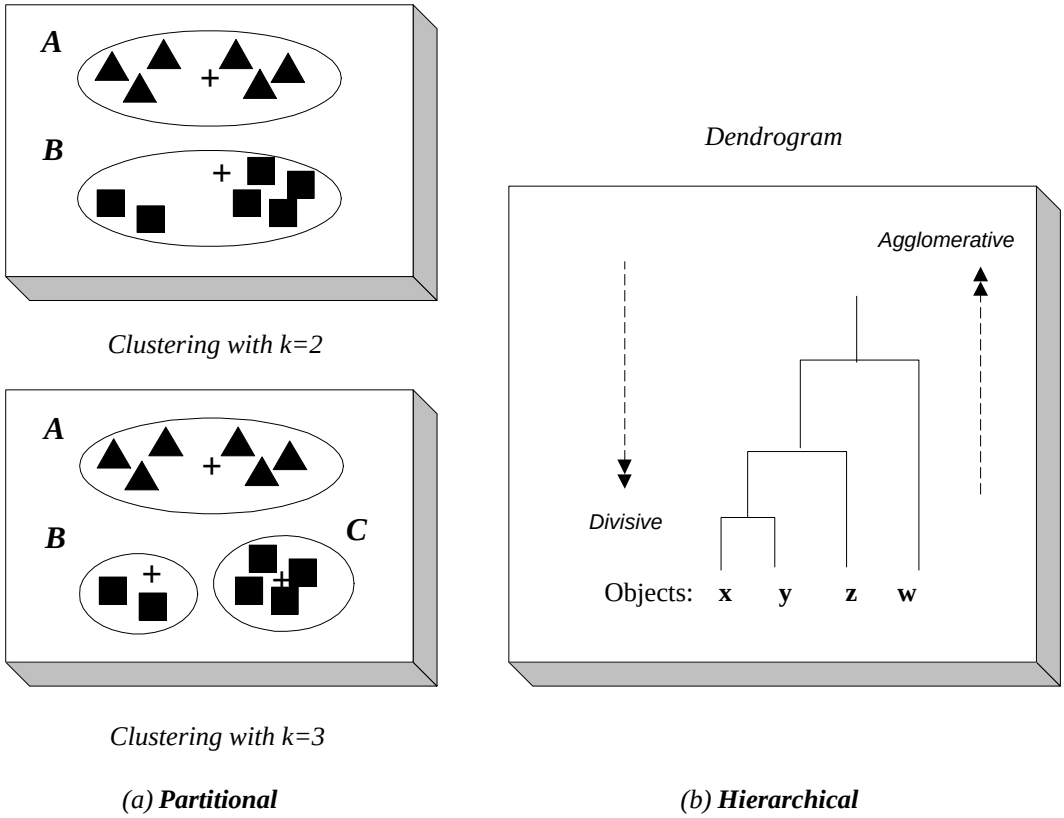


Figure 2.1: (a) Partitional clustering for $k = 2, 3$, (b) Hierarchical clustering dendrogram

Some of the representatives of this family of algorithms are *BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)* [ZRL96] and *CURE (Clustering Using REpresentatives)* [GRS98]. Both algorithms are applicable to data sets with numerical attributes.

Partitional and hierarchical methods can be integrated. For example, a result given by a hierarchical method can be improved via a partitional step, which refines the result

via iterative relocation of points.

2.3.2 Clustering Techniques in Data Mining

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. We briefly describe some of them below, and focus on the ones that are suitable for clustering categorical data [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 object. In these approaches, a given cluster continues growing as long as the number of objects in the neighborhood exceeds some parameter. Representatives are *DBSCAN (Density-Based Spatial Clustering of Applications with Noise* [EKSX96], *OPTICS (Ordering Points To Identify the Clustering Structure)* [ABKS96] and *DENCLUE (DENsity-based CLUstEring)* [HK98],

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, which is usually based on the number of objects that fall in a particular cell (or larger area) of the grid. Representatives of this family are *STING (STatistical INformation Grid)* [WYM97] *WaveCluster* [SCZ98] and *CLIQUE (CLustering In QUEst* [AGGR98].

Model-Based Clustering : These algorithms select a mathematical model for the data

and then find values 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 all use the same concept of density. A classical model-based clustering algorithm is the *Expectation-Maximization* algorithm [BFR99].

Categorical Data Clustering : These algorithms are specifically developed for data where Euclidean, or other numerically-oriented distance measures are not meaningful. In the literature, we find approaches close to partitional and hierarchical methods. In the next section, we analyze such algorithms in more detail.

For each category, there exists a plethora of sub-categories, *e.g.*, density-based clustering oriented toward geographical data [SEKX98]. An exception to this is the class of approaches to handling categorical data. 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, data sets that include some categorical attributes are abundant. Moreover, there are data sets with a mixture of attribute types, such as the United States Census data set (see `http://www.census.gov/`) and data sets used in data integration [MHH+01].

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 a good clustering technique should have the following characteristics [HK01]:

- *Scalability*: The ability of the algorithm to perform well with a large number of data objects (tuples).

- *Analyze mixture of attribute types*: The ability to analyze dataset with mixtures of attribute types, as well as homogeneous ones.

- *Find arbitrary-shaped clusters*: Different types of algorithms will be biased toward 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, it may not be relevant to talk about cluster structures.

- *Minimal 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 biasing 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.

- *Insensitivity to the order of input records*: The same data set, when presented to certain algorithms in different orders, may lead to dramatically different clusterings. The order of input mostly affects algorithms that perform only 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 can produce meaningful results only when the number of dimensions is small (*e.g.,* eight to ten).

The appearance of a 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 space required to store or represent the data set 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 factors strongly influence the efficiency of a clustering algorithm, since it requires more time to process the data, while at the same time the yielding clusters of very poor quality.

- *Interpretability and usability*: Most of the time, 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 analyze them and report which of the requirements they meet or fail to meet.

2.4 Clustering Databases with Categorical Data

In this section, we consider databases with attributes whose values are categorical. The problem of clustering becomes more challenging when the data is categorical, that is, when there is no inherent distance measure between data values. This is often the case in many domains where data is described by a set of descriptive attributes, some of which are neither numerical nor inherently ordered in any way. As a concrete example, consider a relation that stores information about movies. For the purpose of exposition, a movie is a tuple characterized by the attributes “director”, “actor/actress”, and “genre”. An

instance of this relation is shown in Table 2.2. In this setting, it is not immediately obvious what the distance, or similarity, is between the values "Coppola" and "Scorsese", or the tuples "Vertigo" and "Harvey". Categorical data contains values with no inherent

	director	actor	genre
t_1 (Godfather II)	Scorsese	De Niro	Crime
t_2 (Good Fellas)	Coppola	De Niro	Crime
t_3 (Vertigo)	Hitchcock	Stewart	Thriller
t_4 (N by NW)	Hitchcock	Grant	Thriller
t_5 (Bishop's Wife)	Koster	Grant	Comedy
t_6 (Harvey)	Koster	Stewart	Comedy

Table 2.2: An instance of the movie database

semantics. That is, the choice of a specific data value (perhaps "Coppola" or "F.F.C." or "francis ford coppola") has no inherent semantics. But the fact that "Coppola" is different from "Scorsese" is meaningful. Hence, any categorical clustering algorithm should be *generic* in that it "... treats data values as essentially uninterpreted objects..." [AHV95]. Genericity can be formalized by stating that the clustering algorithm should produce the same results with any permutation on the data values [AHV95, Hul84].

Without a clear measure of distance between data values, it is unclear how to define a measure of the quality of categorical clustering. Before introducing clustering algorithms for categorical data, 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 mapped onto unique numbers and, as a consequence, Euclidean distance could be used to measure their proximities, with questionable

consequences though.

Notice also that categorical data can be visualized by assuming a specific ordering. One sensitive point is the second one mentioned in the previous list. Guha et al. give an example why this entails several dangers [GRS99]: assume a database with objects defined over the values 1 through 6. The set of objects is: (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 the values inside the corresponding objects. The four objects 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 objects (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 objects. Thus, with a centroid-based hierarchical algorithm, (a) and (b) would be merged first. The centroid of the new cluster is $\{0.5, 1, 1, 0.5, 1, 0\}$. In the following step, (c) and (d) have the smallest distance, and thus will be merged. However, this corresponds to a merge of object $\{1, 4\}$ with object $\{6\}$, although the two objects have no values 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 objects that should not be in the same cluster end up being together.

It becomes apparent, then, that we need different methods, and especially different similarity measures, to discover "natural" groupings of categorical data. The follow-

[2] We consider sets of categorical values, whose identifiers we report.

ing subsections introduce the most important clustering algorithms on databases with categorical attributes. For each algorithm, we elaborate on the following issues:

- description of the algorithm;
- measure used to assess similarity or dissimilarity;
- objective function minimized or maximized as appropriate;
- merging strategy followed to produce intermediate clusters;
- computational complexity of the algorithm;
- set of input parameters.

2.4.1 The *k-modes* Algorithm

The first algorithm oriented toward categorical data sets is an extension to *k-means*, called *k-modes* [Hua98].

Description: The idea is the same as in *k-means* and the structure of the algorithm does not change. *k-modes* partitions a data set into a given number of clusters such that an objective function is minimized. The differences from the *k-means* algorithm can be summarized in the following list:

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

Distance measure: Given two categorical data objects $\hat{x}$ and $\hat{y}$, their distance is defined by 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.

Objective function: The mode of an attribute is the value that appears the most in this attribute. For example, if the values of a data set are $\{1, 2, 18, 3, 18\}$ its mode is the value 18. For a data set of dimensionality n, every cluster c, $1 \leq c \leq k$, has a mode defined by a vector $Q^c = (x_1^c, x_2^c, \ldots, x_n^c)$ whose entries are the modes of each attribute. 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.

Merging strategy: The algorithm starts with a usually random set of k tuples from the data set, each one of them being the representative of a cluster. Then, each remaining tuple is compared to the k representatives and placed in the cluster of the closest representative.

Computational complexity: The algorithm requires a linear number of in-memory operations and thus can be used for large inputs.

Input parameters: *k-modes* accepts as input the desired number of clusters, k.

The similarities, in structure and behavior, to *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.

The parameter γ is specified by the user.

2.4.2 The *ROCK* Algorithm

ROCK (RObust Clustering using linKs) [GRS99] is a hierarchical algorithm for categorical data.

Description: Guha et al. propose a novel approach based on the concept of *links* between data objects [GRS99]. 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 whose entries are identifiers of the categorical values. More precisely, if n_i is the number of objects in cluster C_i, *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 the similarity, we may use any measure that can be applied on such data.

- 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 to which $\hat{x}$ and $\hat{y}$ are both similar, *i.e.*, their similarity exceeds parameter θ.

- 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 $f(\theta) = \frac{1-\theta}{1+\theta}$

Similarity measure: In brief, *ROCK* measures the similarity of two clusters by comparing the *aggregate interconnectivity* of two clusters against a user-specified static *interconnectivity model*.

Objective function: The maximization of the following expression comprises the objective of *ROCK*:

$$E = \sum_{i=1}^{k} n_i \cdot \sum_{\hat{x}_q, \hat{x}_r \in C_i} \frac{link(\hat{x}_q, \hat{x}_r)}{n_i^{1+2f(\theta)}}$$

Merging strategy: The overview of *ROCK* is given in Figure 2.2.

Figure 2.2: Overview of *ROCK* [GRS99]

As we see, a random sample is drawn to be the cluster representatives and a clustering algorithm (hierarchical) is used 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.

Computational complexity: The number of in-memory operations 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.

Input parameters: *ROCK* requires as parameters the number of clusters k and the value of the similarity threshold θ.

2.4.3 The *STIRR* Algorithm

STIRR (Sieving Through Iterated Relational Reinforcement) [GKR98] is one of the most innovative methods for clustering categorical data sets and differs in spirit from the previous approaches.

Description: It uses an iterative approach where the values, rather than the tuples, are the data objects to be clustered. The key features of this approach are summarized below.

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. 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 2.3).

Similarity measure: There is no mathematical expression to assess the similarity of values in *STIRR*. They are considered to be similar if the values 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`. Generally speaking, the similarity of values is based on the fact that the sets of items with which they do co-occur have large overlap.

Objective function/strategy: In *STIRR*, we cannot clearly separate the objective of the algorithm from the main strategy followed. Spectral methods are used to perform the clustering. These 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*, 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, or alternatively 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 2.3.

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 orthonormality is ensured.

Combining Operator $\bigoplus$: 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

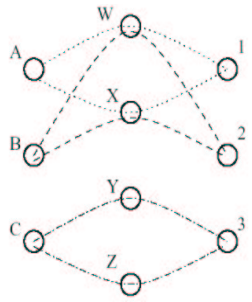


Figure 2.3: Representation of a database in *STIRR* [GKR98]

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-occurrence in a stronger way.

The paper by Gibson et al. [GKR98] presents some useful theorems from spectral graph theory, to prove that *STIRR* converges and moreover gives a result where some values have negative weights, while others have positive weights. Experimental results are given: the algorithm is applied to a bibliographical database and database publications are successfully distinguished from theoretical ones. However, there is no clustering defined for more than two clusters of values.

Computational complexity: *STIRR* requires one pass over the data set and a linear number of in-memory operations, making it suitable for large data sets.

Input parameters: The initial configuration (weights) is required as well as the combining operator and the stopping criteria.

2.4.4 The *COOLCAT* Algorithm

The *COOLCAT* algorithm [BCL02a, BCL02b], by Barbará, Couto and Li is an information-theoretic algorithm most similar to the algorithm we propose in this thesis.

Description: *COOLCAT* is very similar in spirit to the k-means algorithm. However, it includes an initial sampling phase, where the goal is the selection of k appropriate cluster representatives.

Similarity measure The algorithm uses *entropy* as the measure to assess similarity of objects.[3]

Objective function: The objective function is the minimization of entropy across clusters. Intuitively, entropy is a measure of the uncertainty we have in predicting the values of a specific random variable. Given the number of clusters to be produced, the objective of the algorithm is to partition a data set such that the entropy of the resulting clustering is minimized, or equivalently, the values within clusters can be predicted with maximum certainty.

Merging strategy: *COOLCAT* is an algorithm that relies on sampling, and it is non-hierarchical. Assuming that each attribute exists independently of the others, *COOLCAT* defines the entropy of a cluster, and the algorithm starts with a sample of points and, using a heuristic method, identifies a set of k initial clusters such that their pairwise entropies are large. It then places all remaining tuples of the data set in clusters such that the entropy of the whole system is minimized.

Computational complexity: This algorithm requires a single pass over the data and performs a linear number of in-memory computations.

Input parameters: *COOLCAT* requires the number of clusters k and the size of the

[3]More precise information-theoretic definitions are given in the next chapter.

initial sample as parameters. As it will be seen in the following chapter, through our experimental evaluation of *COOLCAT*, the quality of the clustering depends on the choice of initial sample.

2.4.5 Discussion

Clustering categorical data is a challenging problem, 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 clusters produced is measured by the user's expertise and examination of the results. 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.

The *k-modes* algorithm 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 with which it exists. *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*.

A well presented and novel technique is *STIRR*, based on the iterative computation of new weights for the nodes of a graph. The idea is related to the work done by Kleinberg on discovering authoritative sources on the web [Kle98], however *STIRR* strongly depends on the choice of the *combining operator*, $\bigoplus$ and the notion of the iteration function,

f, which defines a *dynamical system*. Gibson et al. argue that there has not been much proved about the behavior of dynamical systems in the literature. They base their proof of convergence and the discovery of final clusters, on results that come from the spectral graph theory research area. *STIRR* gives a result where each value has acquired either a positive or negative weight, producing just two clusters. The reporting of more clusters would require a costly 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 *COOLCAT*, an algorithm based on an information-theoretic quality measure, namely the entropy of a clustering. Algorithmically, it is mostly similar to the *k-modes* and *k-prototypes* algorithms, except for an initial sampling phase to choose (potential) "good" cluster representatives. However, the algorithm seems to be sensitive to this sampling phase, since it is possible that outliers get picked as such representatives. *COOLCAT* demonstrates linear computational complexity (number of in-memory operations) in the number of input records and, thus, can be used with large data sets.

In our opinion, the aforementioned clustering algorithms for categorical data each have some disadvantages that are difficult to overcome. For example, *STIRR* is not defined for clustering tuples and is not able to produce more than two clusters of attribute values. *ROCK* is not suitable for large data sets and *COOLCAT* is sensitive to the initial sampling process. During our study of these algorithms, we observed no common quality measure, so comparison of the results each algorithm gives on the same data set is difficult. In our work, we try to address these issues and provide an intuitive and

scalable solution to the problem of clustering both tuples and attribute values. Also, we assess the quality of the results of the other techniques using a variety of measures.

Table 2.3 gives an overview of the features of each algorithm. In our opinion it is obvious that there is no "optimal solution" for the problem of clustering categorical data, and the choice of an appropriate algorithm depends on many factors.

2.5 Conclusions

Clustering lies at the heart of data analysis and data mining applications. The ability to discover highly correlated or co-occurring sets 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 correct or incorrect [Eve93].

In this chapter, 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. Previous research has emphasized numerical data sets, and the intricacies of working with large categorical databases has been left to a small number of alternative techniques.

Our claim is that new research solutions are needed for the problem of clustering categorical data. In the next chapter, we introduce a new algorithm to cluster categorical data, and demonstrate its qualitative advantages as well as its efficiency. We also present the application of this algorithm to the discovery of structure in large categorical data sets that contain heterogeneous and/or dirty data.

Categorical Clustering Methods				
Algorithm	*Input Parameters*	*Optimized For*	*Outlier Handling*	*Computational Complexity (number of in-memory operations)*
$k-modes$	Number of Clusters	Data Sets with Well-separated Clusters	No	$\mathcal{O}(n)$
$k-prototypes$	Number of Clusters	Mixed Data Sets	No	$\mathcal{O}(n)$
$ROCK$	Number of Clusters, Similarity Threshold	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	$\mathcal{O}(n)$
$COOLCAT$	Size of Initial Sample, Number of Clusters	Large Data Sets with Well-separated Clusters	No	$\mathcal{O}(n)$

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

Table 2.3: Properties of categorical clustering algorithms

Chapter 3

LIMBO Clustering

In this chapter, we introduce LIMBO, a scalable hierarchical categorical clustering algorithm that builds on the *Information Bottleneck (IB)* framework for quantifying the relevant information preserved when clustering. As a hierarchical algorithm, LIMBO has the advantage that it can produce clusterings of different sizes in a single execution. We use the IB framework to define a distance measure for categorical tuples and we also present a novel information-theoretic distance measure for categorical attribute values. We show how the LIMBO algorithm can be used when the objects to be clustered are either the tuples of a data set or the values of one of its attributes. Finally, we present a heuristic for discovering candidate values for the number of meaningful clusters that exist in a categorical data set.

3.1 Introduction

The definition of clustering assumes that there is some well-defined notion of *similarity*, or *distance*, between data objects. As we saw in the previous chapter, when the objects are defined by a set of numerical attributes, there are natural definitions of distance based on geometric analogies. These definitions rely on the semantics of the data values themselves (for example, the values \$100,000 and \$110,000 are more similar than \$100,000 and \$1).

Consider the example from Table 2.2, expanded in Table 3.1 to show two possible

tuple clusterings, **C** and **D**. We already stated that in this setting it is not immediately obvious what the distance, or similarity, is between the values "Coppola" and "Scorsese", or the tuples "Vertigo" and "Harvey". Without a clear measure of distance between data values, it is unclear how to define a quality measure for categorical clustering. To do this, we employ *mutual information*, a measure from information theory. A good clustering is one where the clusters are *informative* about the data objects they contain. Since data objects are expressed in terms of attribute values, we require that the clusters convey information about the attribute values of the objects in the cluster. That is, given a cluster, we wish to narrow as much as possible the variety of attribute values associated with objects of the cluster. The quality measure of the clustering is then the mutual information of the clusters and the attribute values that occur in the respective clusters. Since a clustering is a summary of the data, some information is generally lost. Our objective will be to minimize this loss, or equivalently to minimize the increase in uncertainty as the objects are grouped into fewer and larger clusters.

	director	actor	genre	**C**	**D**
t_1 (Godfather II)	Scorsese	De Niro	Crime	c_1	d_1
t_2 (Good Fellas)	Coppola	De Niro	Crime	c_1	d_1
t_3 (Vertigo)	Hitchcock	Stewart	Thriller	c_2	d_1
t_4 (N by NW)	Hitchcock	Grant	Thriller	c_2	d_1
t_5 (Bishop's Wife)	Koster	Grant	Comedy	c_2	d_2
t_6 (Harvey)	Koster	Stewart	Comedy	c_2	d_2

Table 3.1: An instance of the movie database

Consider partitioning the tuples in Table 3.1 into two clusters. Clustering **C** groups the first two movies together into one cluster, c_1, and the remaining four into another, c_2. Note that cluster c_1 preserves all information about the actor and the genre of the movies it holds. For objects in c_1, we know with certainty that the genre is "Crime", the actor is "De Niro" and there are only two possible values for the director. Cluster c_2 involves only two different values for each attribute. Any other clustering into two clusters, will result in greater information loss. For example, in clustering **D**, d_2 is equally

as informative as c_1, but d_1 includes three different actors and three different directors. So, while in c_2 there are two equally likely values for each attribute, in d_1 the director is any of "Scorsese", "Coppola", or "Hitchcock" (with respective probabilities 0.25, 0.25, and 0.50). Similarly, for actor, the value in d is one of "De Niro", "Stewart" or "Grant" (with respective probabilities (0.50, 0.25, 0.25).

This intuitive idea was formalized by Tishby, Pereira and Bialek [TPB99]. They recast clustering as the compression of one random variable into a compact representation that preserves as much information as possible about another random variable. Their approach was named the *Information Bottleneck (IB)* method, and it has been applied to a variety of different areas. In this chapter, we consider the application of the IB method to the problem of clustering data sets of categorical data.

We formulate the problem of clustering relations with categorical attributes within the Information Bottleneck framework, and define dissimilarity between categorical data objects based on the IB method. Our contributions in this chapter are given in the following list (and published elsewhere [ATMS04]).

- We propose LIMBO, the first scalable hierarchical algorithm for clustering categorical data. As a result of its hierarchical approach, LIMBO allows us in a single execution to consider clusterings of various sizes. The size of the model LIMBO builds to summarize the data can be controlled to match the space available for use.

- We take advantage of the hierarchical nature of LIMBO to examine heuristics for determining the numbers of clusters into which a given data set can be most naturally partitioned.

- We use LIMBO to cluster both data objects in relational and market-basket data sets and to cluster attribute values. We define a novel distance measure between attribute values that allows us to quantify the degree of interchangeability of attribute

values within a single attribute.

- LIMBO builds and efficiently manages summary structures in a single pass over the data, making it the first scalable clustering algorithm amenable for use on categorical data streams.

- We empirically evaluate the quality of clusterings produced by LIMBO relative to other categorical clustering algorithms including the tuple clustering algorithms IB, ROCK [GRS99], and COOLCAT [BCL02b]; as well as the attribute value clustering algorithm STIRR [GKR98]. We compare the clusterings based on a comprehensive set of quality metrics including all metrics used in the evaluation of these related approaches. We also show that LIMBO is robust to different input orders of data.

3.2 The Information Bottleneck Method

In this section, we review some of the concepts from information theory that will be used in the rest of the thesis. We also provide an outline of the Information Bottleneck method, and its application to the problem of clustering.

3.2.1 Information Theory Basics

The following definitions can be found in any information theory textbook, *e.g.*, [CT91]. Let X denote a discrete random variable that takes values over the set $\mathbf{X}$, and let $p(x)$ denote the probability mass function of X. The *entropy* $H(X)$ of variable X is defined by

$$H(X) = -\sum_{x \in \mathbf{X}} p(x) \log p(x) \ .$$

The entropy $H(X)$ can be thought of as a lower bound on the number of bits on average required to describe the random variable X. Intuitively, entropy captures the "uncertainty" of variable X; the higher the entropy, the lower the certainty with which we can predict the value of the variable X.

Now, let X and Y be two random variables that range over sets $\mathbf{X}$ and $\mathbf{Y}$ respectively, and let $p(x, y)$ denote their joint distribution and $p(y|x)$ be the conditional distribution of Y given X. If, for example, X is a random variable representing the movies in Table 3.1, and Y a random variable representing the director and actor values, then $p(x, y)$ is the joint distribution of attribute value appearance in tuples. Then *conditional entropy* $H(Y|X)$ is defined as

$$\begin{aligned} H(Y|X) &= \sum_{x \in \mathbf{X}} p(x) H(Y|X = x) \\ &= -\sum_{x \in \mathbf{X}} p(x) \sum_{y \in \mathbf{Y}} p(y|x) \log p(y|x) \end{aligned}$$

Given X and Y, an important question that arises is: "to what extent can the value of one variable be predicted from knowledge of the value of the other variable?". When the two variables are independent, no information about one variable can be obtained through knowledge of the other one. This question has a quantitative answer through the notion of *mutual information*, $I(X;Y)$. Mutual information quantifies the amount of information that the variables hold about each other and is the amount of uncertainty (entropy) in one variable that is removed by knowledge of the value of the other one. Specifically, we have

$$\begin{aligned} I(X;Y) &= \sum_{x \in \mathbf{X}} \sum_{y \in \mathbf{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \\ &= \sum_{x \in \mathbf{X}} p(x) \sum_{y \in \mathbf{Y}} p(y|x) \log \frac{p(y|x)}{p(y)} \\ &= H(X) - H(X|Y) = H(Y) - H(Y|X) \end{aligned}$$

Mutual information is symmetric, non-negative and equals zero if and only if X and Y are independent.

Relative Entropy, or the *Kullback-Leibler (KL)* divergence, is a standard information-theoretic measure of the difference between two probability distributions. Given two

distributions p and q over a set $\mathbf{X}$, the relative entropy is

$$D_{KL}[p\|q] = \sum_{x\in\mathbf{X}} p(x)\log\frac{p(x)}{q(x)} \ .$$

Intuitively, the relative entropy $D_{KL}[p\|q]$ is a measure of the redundancy in an encoding that assumes the distribution q, when the true distribution is p. (Note that relative entropy is not symmetric with respect to p and q.)

In the remainder of the thesis and for each application of the aforementioned information-theoretic measure, we will appropriately define variables X and Y.

3.2.2 The Information Bottleneck Method

Clustering problems can be formulated to involve a collection of objects (e.g. customers, documents) represented as vectors in a feature space.

Let $\mathbf{X}$ denote a set of objects that we want to cluster (e.g., customers, tuples in a relation, web documents), and assume that the elements in $\mathbf{X}$ are expressed as vectors in a feature space $\mathbf{Y}$ (e.g., items purchased, words, attribute values, links). That is, each element of $\mathbf{X}$ is associated with a sequence of values from $\mathbf{Y}$. In Section 3.3, we describe in detail the types of datasets that we consider. Let $n = |\mathbf{X}|$ and $d = |\mathbf{Y}|$. Our data can then be conceptualized as an $n \times d$ matrix M, where each row holds the feature vector of an object in $\mathbf{X}$. Now let X, Y be random variables that range over the sets $\mathbf{X}$ and $\mathbf{Y}$ respectively. We normalize matrix M so that the entries of each row sum up to one. For some object $x \in \mathbf{X}$, the corresponding row of the normalized matrix holds the conditional probability $p(Y|X = x)$. An example of our representation will be given in Section 3.2.3. The information that one variable contains about the other can be quantified using the mutual information $I(X;Y)$ measure. Furthermore, if we fix a value $x \in \mathbf{X}$, the conditional entropy $H(Y|X = x)$ gives the uncertainty of a value for variable Y selected among those associated with the object x.

A *k-clustering* $\mathbf{C}_k$ of the elements of $\mathbf{X}$ partitions them into k clusters $\mathbf{C}_k = \{c_1, c_2, c_3, ..., c_k\}$, where each cluster $c_i \in \mathbf{C}$ is a non-empty subset of $\mathbf{X}$ such that $c_i \cap c_j = \emptyset$ for all i, j,

$i \neq j$, and $\cup_{i=1}^{k} c_i = \mathbf{X}$. Let C_k denote a random variable that ranges over the clusters in $\mathbf{C}_k$. We define k to be the *size* of the clustering. When k is fixed or when it is immaterial to the discussion, we will use $\mathbf{C}$ and C to denote the clustering and the corresponding random variable.

Now, let $\mathbf{C}$ be a specific clustering. Giving equal weight to each element $x \in \mathbf{X}$, we define $p(x) = \frac{1}{n}$. Then, for $c \in \mathbf{C}$, the elements of $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{C}$ are related as follows:

$$p(c|x) = \begin{cases} 1 & \text{if } x \in c \\ 0 & \text{otherwise} \end{cases}$$

$$p(c) = \sum_{x \in c} p(x)$$

$$p(y|c) = \frac{1}{p(c)} \sum_{x \in c} p(x)p(y|x) \ .$$

We seek clusterings of the elements of $\mathbf{X}$ such that, for $x \in c_i$, knowledge of the cluster identity, c_i, provides essentially the same prediction of, or information about, the values in $\mathbf{Y}$ as does the specific knowledge of x. Just as $I(X;Y)$ measures the information about the values in $\mathbf{Y}$ provided by the identity of a specific element of $\mathbf{X}$, $I(C;Y)$ measures the information about the values in $\mathbf{Y}$ provided by the identity of a cluster in $\mathbf{C}$. The higher $I(C;Y)$, the more informative the cluster identity is about the values in $\mathbf{Y}$ contained in the cluster. The relationship of all information-theoretic quantities involved in the clustering are given in Figure 3.1.

In the formalization of Tishby, Pereira and Bialek [TPB99], the problem of clustering is recast as the problem of compressing variable X while preserving information about variable Y. Formally, they define clustering as an optimization problem, where, for a given number k of clusters, we wish to identify the k-clustering that maximizes $I(C_k;Y)$. Intuitively, in this procedure, the information contained in X about Y is "squeezed" through a compact "bottleneck" clustering $\mathbf{C}_k$, which is forced to represent the "relevant" part in X with respect to Y. Tishby et al. [TPB99] prove that, for a fixed number k of clusters, the optimal clustering $\mathbf{C}_k$ partitions the objects in $\mathbf{X}$ so that the average

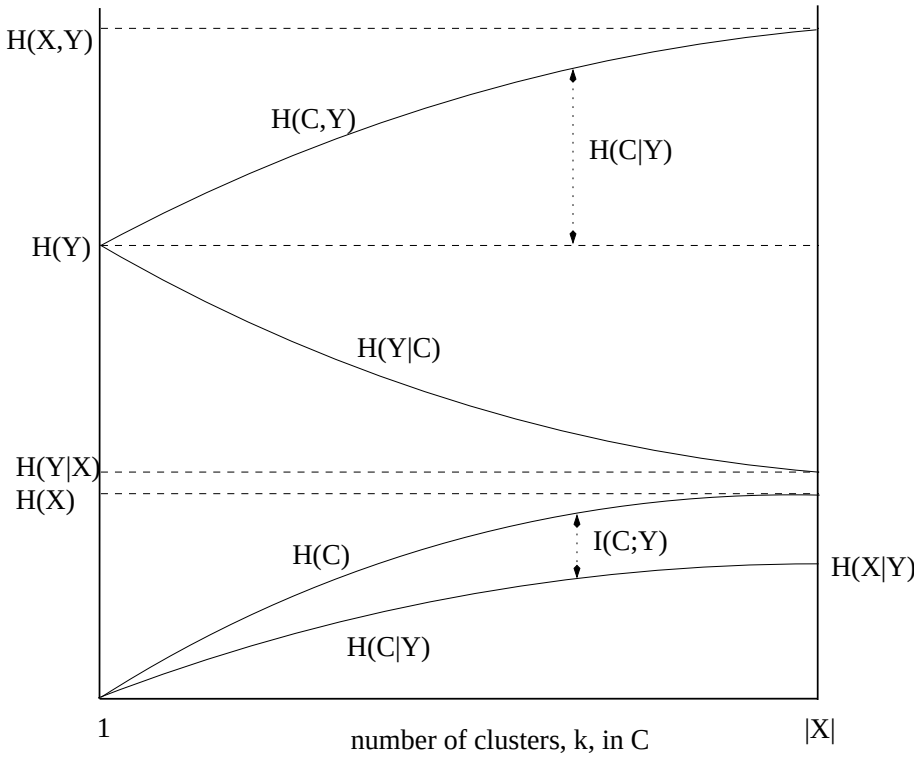


Figure 3.1: Information-theoretic quantities as number of clusters, k, changes.

relative entropy $\sum_{c \in \mathbf{C}_k, x \in \mathbf{X}} p(x,c) D_{KL}[p(y|x) \| p(y|c)]$ is minimized.

3.2.3 The Agglomerative Information Bottleneck Algorithm

Finding the optimal clustering is an NP-complete problem [GJ79]. Slonim and Tishby [ST99] propose a greedy agglomerative approach, the *Agglomerative Information Bottleneck (AIB)* algorithm, for finding an informative clustering.

The algorithm starts with the clustering $\mathbf{C}_n$, in which each object $x \in \mathbf{X}$ is assigned to its own cluster. Due to the one-to-one mapping between $\mathbf{C}_n$ and $\mathbf{X}$, $I(C_n; Y) = I(X; Y)$; that is, the clusters in $\mathbf{C}_n$ contain the same information about the values in the set $\mathbf{Y}$ as the tuples in $\mathbf{X}$. The algorithm then proceeds iteratively, for $n - k$ steps, reducing the number of clusters in the current clustering by one in each iteration. At step $n-\ell+1$ of the AIB algorithm, two clusters c_i, c_j in ℓ-clustering $\mathbf{C}_\ell$ are merged into a single component c^* to produce a new $(\ell-1)$-clustering $\mathbf{C}_{\ell-1}$. As the algorithm forms clusterings of smaller

size, the information that the clustering contains about the values in $\mathbf{Y}$ decreases; that is, $I(C_{\ell-1};Y) \leq I(C_\ell,Y)$. The clusters c_i and c_j to be merged are chosen such that the information loss in moving from clustering $\mathbf{C}_\ell$ to clustering $\mathbf{C}_{\ell-1}$ is minimized. This information loss is given by

$$\delta I(c_i,c_j) = I(C_\ell;Y) - I(C_{\ell-1};Y)$$

We can also view the information loss as the increase in the uncertainty. Recall that $I(C;Y) = H(Y) - H(Y|C)$. Since the value $H(Y)$ is independent of the clustering $\mathbf{C}$, we have that

$$\delta I(c_i,c_j) = I(C_\ell;Y) - I(C_{\ell-1};Y) = H(Y|C_{\ell-1}) - H(Y|C_\ell)$$

That is, the information loss measures the decrease in our ability to predict the values of Y given a cluster in C. Therefore, maximizing the mutual information $I(C;Y)$ is the same as minimizing the entropy of the clustering $H(Y|C)$.

After merging clusters c_i and c_j, the new component $c^* = c_i \cup c_j$ has

$$p(c^*|x) = \begin{cases} 1 & \text{if } x \in c_i \text{ or } x \in c_j \\ 0 & otherwise \end{cases} \tag{3.1}$$

$$p(c^*) = p(c_i) + p(c_j) \tag{3.2}$$

$$p(Y|c^*) = \frac{p(c_i)}{p(c^*)}p(Y|c_i) + \frac{p(c_j)}{p(c^*)}p(Y|c_j)\ . \tag{3.3}$$

Tishby et al. [TPB99] show that

$$\delta I(c_i,c_j) = [p(c_i) + p(c_j)] \cdot D_{JS}[p(Y|c_i), p(Y|c_j)] \tag{3.4}$$

where D_{JS} is the *Jensen-Shannon (JS)* divergence, defined as follows. Let $p_i = p(y|c_i)$ and $p_j = p(y|c_j)$, and let

$$\bar{p} = \frac{p(c_i)}{p(c^*)}p_i + \frac{p(c_j)}{p(c^*)}p_j$$

Input : Probability matrix $p(x, y)$, $x \in \mathbf{X}$ and $y \in \mathbf{Y}$, and number of clusters k.
Output : A $k-$clustering of X.

Initialization :
- For $i = 1 \dots n$:
  * $c_i = \{x_i\}$
  * $p(c_i) = p(x_i)$
  * $p(y|c_i) = p(y|x_i)$
- $C_n = \{c_1, c_2, \dots, c_n\}$
- For every $q, r = 1 \dots n$, with $q < r$
  calculate $d_{q,r} = \delta I(x_q, x_r)$

Main Loop:
- For $m = 1 \dots (n - k + 1)$:
  * find $\{i, j\} = argmin_{q,r}\{d_{q,r}\}$
  * merge $\{c_i, c_j\}$ into c^*:
    ** $p(c^*|x) = 1$ if $x \in c_i$ or $x \in c_j$
    ** $p(c^*) = p(c_i) + p(c_j)$
    ** $p(Y|c^*) = \frac{p(c_i)}{p(c^*)} p(Y|c_i) + \frac{p(c_j)}{p(c^*)} p(Y|c_j)$
  * update $C = \{C - \{c_i, c_j\}\} \cup \{c^*\}$
  * update $d_{q,r}$ costs according to c^*

Figure 3.2: The *AIB* algorithm

denote the weighted average distribution of distributions p_i and p_j. Then, the D_{JS} distance is:

$$D_{JS}[p_i, p_j] = \frac{p(c_i)}{p(c^*)} D_{KL}[p_i||\bar{p}] + \frac{p(c_j)}{p(c^*)} D_{KL}[p_j||\bar{p}] \ .$$

The D_{JS} distance is the average D_{KL} distance of p_i and p_j from $\bar{p}$. It is non-negative and equals zero if and only if $p_i \equiv p_j$. It is also bounded above by one, and it is symmetric in i and j but does not satisfy the triangle inequality. Thus, it is not a metric. We note that the information loss for merging clusters c_i and c_j, depends only on the clusters c_i and c_j, and not on other parts of the clusterings $\mathbf{C}_\ell$ and $\mathbf{C}_{\ell-1}$. The pseudo-code of the AIB algorithm is given in Figure 3.2 [ST99].

3.3 Clustering Categorical Data using the *IB* Method

In this section, we formulate the problem of clustering categorical data in the context of the Information Bottleneck method, and we consider some novel applications of the method. We consider two types of data: *relational* data and *market-basket* data.

3.3.1 Relational Data

In this case, the input to our problem is a set $\mathbf{T}$ of n tuples on m attributes $A_1, A_2, \ldots, A_m$. The domain of attribute A_i is the set $\mathbf{V}_i = \{A_i.v_1, A_i.v_2, \ldots, A_i.v_{d_i}\}$ so that identical values from different attributes are treated as distinct values. A tuple $t \in \mathbf{T}$ consists of m attribute values, where the value for the i^{th} attribute is taken from the set $\mathbf{V}_i$. Let $\mathbf{V} = \mathbf{V}_1 \cup \cdots \cup \mathbf{V}_m$ denote the set of all possible attribute values. Let $d = d_1 + d_2 + \cdots + d_m$ denote the size of $\mathbf{V}$. We represent our data as an $n \times d$ binary matrix M, where each $t \in \mathbf{T}$ is a d-dimensional row vector in M. If tuple t contains attribute value v, then $M[t, v] = 1$, otherwise $M[t, v] = 0$. Since every tuple contains one value for each attribute, each tuple vector contains exactly m 1's.

Now, let T and V be random variables that range over sets $\mathbf{T}$ and $\mathbf{V}$ respectively. Following the formalism of Section 3.2, we can substitute $\mathbf{X}$ with $\mathbf{T}$, $\mathbf{Y}$ with $\mathbf{V}$, X with T and Y with V in the corresponding measures and proceed by defining $p(t) = 1/n$, and normalizing matrix M so that the t^{th} row holds the conditional probability distribution $p(V|t)$. Since each tuple contains exactly m attribute values, for some $v \in \mathbf{V}$, $p(v|t) = 1/m$ if v appears in tuple t, and zero otherwise. Table 3.2 shows the normalized matrix M for the movie database example. We use abbreviations for the attribute values. For example d.H stands for director.Hitchcock. Given the normalized matrix, we can proceed with the application of the IB method to cluster the tuples in $\mathbf{T}$. Note that matrix M can be stored as a sparse matrix, so we do not need to materialize all $n \times d$ entries.

Our approach merges all attribute values into one variable, without taking into account the fact that the values come from different attributes. Alternatively, we could

	d.S	d.C	d.H	d.K	a.DN	a.S	a.G	g.Cr	g. T	g.C	p(t)
t_1	1/3	0	0	0	1/3	0	0	1/3	0	0	1/6
t_2	0	1/3	0	0	1/3	0	0	1/3	0	0	1/6
t_3	0	0	1/3	0	0	1/3	0	0	1/3	0	1/6
t_4	0	0	1/3	0	0	0	1/3	0	1/3	0	1/6
t_5	0	0	0	1/3	0	0	1/3	0	0	1/3	1/6
t_6	0	0	0	1/3	0	1/3	0	0	0	1/3	1/6

Table 3.2: The normalized movie table

define a random variable for every attribute A_i. We will now show the following:

Theorem 1 *Applying the Information Bottleneck method to the case of relational data, considering all attributes together is equivalent to considering each attribute independently.*

Proof 1 *Let V_i be a random variable that ranges over the set $\mathbf{V}_i$. For some $v \in \mathbf{V}_i$, and some $t \in \mathbf{T}$ we use $p(v|t)$ to denote the conditional probability $p(V = v|t)$, and $p_i(v|t)$ to denote the conditional probability $p(V_i = v|t)$. Also let $p(v)$ denote $p(V = v)$, and $p_i(v)$ denote $p(V_i = v)$. Since each tuple takes exactly one value in each attribute, $p_i(v|t) = 1$, if v appears in t, and zero otherwise. We have that $p(v|t) = \frac{1}{m}p_i(v|t)$, for all $1 \leq i \leq m$, $v \in \mathbf{V}_i$, and $t \in \mathbf{T}$. It follows that $p(v) = \frac{1}{m}p_i(v)$. Furthermore, let c denote a cluster, and let $|c|$ denote the number of tuples in c. Since $p(v|c) = \frac{1}{|c|}\sum_{t\in c} p(v|t)$, and $p_i(v|c) = \frac{1}{|c|}\sum_{t\in c} p_i(v|t)$, we have that $p(v|c) = \frac{1}{m}p_i(v|c)$. Now let $\mathbf{C}_k$ be a k-clustering, for $1 \leq k \leq n$, and let C_k be the corresponding random variable. We have that*

$$\begin{aligned} H(V) &= \frac{1}{m}\sum_{i=1}^{m} H(V_i) + \log m \\ H(V|C_k) &= \frac{1}{m}\sum_{i=1}^{m} H(V_i|C_k) + \log m \\ I(V;C_k) &= \frac{1}{m}\sum_{i=1}^{m} I(V_i;C_k)\ . \end{aligned}$$

Given that T and C_n convey the same amount of information about both V and V_i, the information loss for some clustering C can be expressed as

$$I(V;T) - I(V;C) = \frac{1}{m}\sum_{i=1}^{m}(I(V_i;T) - I(V_i;C))$$

Therefore, minimizing the information loss for variable V is the same as the minimizing the sum of the information losses for all individual variables V_i.

3.3.2 Market-Basket Data

Market-basket data describes a database of transactions for a store, where every tuple consists of the items purchased by a single customer. It is also used as a term that collectively describes a data set where the data objects are sets of values of a single attribute, and each object may contain a different number of values. In the case of market-basket data, the input to our problem is a set $\mathbf{T}$ of n tuples on a single attribute V, with domain $\mathbf{V}$. Tuple t_i contains d_i values. If d is the size of the domain $\mathbf{V}$, we can represent our data as an $n \times d$ matrix M, where each $t \in \mathbf{T}$ is a d-dimensional row vector in M. If tuple t contains attribute value v $c(v)$ times, then $M[t, v] = c(v)$, otherwise $M[t, v] = 0$.

Now, let T and V be random variables that range over sets $\mathbf{T}$ and $\mathbf{V}$ respectively. For tuple $t_i \in \mathbf{T}$, $1 \leq i \leq n$ we define

$$p(t_i) = 1/n$$

$$p(v|t_i) = \begin{cases} c(v)/d_i & \text{if } v \text{ appears in } t \ c(v) \text{ times} \\ 0 & \text{otherwise} \end{cases} .$$

We can now define the mutual information $I(T;V)$ and proceed with the Information Bottleneck method to cluster the tuples in $\mathbf{T}$.

3.4 *LIMBO* Clustering

The Agglomerative Information Bottleneck algorithm, described in Section 3.2.3, requires a number of operations with high computational complexity, namely $\mathcal{O}(n^2 d^2 \log n)$, which is prohibitive for large data sets. We now introduce the *scaLable InforMation BOttleneck, LIMBO,* algorithm, which uses distributional summaries in order to deal with large data sets. LIMBO is based on the idea that we do not need to keep whole tuples, or whole clusters in main memory, but instead, just sufficient statistics to describe them. LIMBO produces a compact summary model of the data, and then performs clustering on the

summarized data. In our algorithm, we bound the size of our summary model that contains the sufficient statistics. We use a B-tree-like indexing structure to efficiently manage the summary statistics and to reduce the in-memory computation. This data structure is similar to one used in BIRCH [ZRL96], a clustering algorithm for numerical data. However, an IB-inspired notion of distance and a novel definition of summaries to produce the solution, make our approach different. Moreover, in BIRCH, a heuristic is used to control the accuracy of the summary created. (When a space bound for the summary is reached, the accuracy threshold is heuristically changed in hopes of reducing the size of the summary.)

3.4.1 Distributional Cluster Features

We summarize a cluster of tuples in a *Distributional Cluster Feature (DCF)*. We will use the information in the relevant *DCF*s to compute the distance between two clusters or between a cluster and a tuple.

For this section, we shall use $\mathbf{T}$ to denote a set of tuples over a set $\mathbf{V}$ of attribute values, and T and V to denote the corresponding random variables. Also let $\mathbf{C}$ denote a clustering of the tuples in $\mathbf{T}$ and let C be the corresponding random variable. For some cluster $c \in \mathbf{C}$, the *Distributional Cluster Feature* (DCF) of cluster c is defined by the pair

$$DCF(c) = \Big(p(c), p(V|c)\Big)$$

where $p(c) = n(c)/n$ is the probability of cluster c, with $n(c)$ being the number of tuples in c, and $p(V|c)$ is the conditional probability distribution of the attribute values given the cluster c. We will use $DCF(c)$ and c interchangeably.

If c consists of a single tuple $t \in \mathbf{T}$, $p(t) = 1/n$, and $p(V|t)$ is computed as described in Section 3.2. For example, in the movie database, for tuple t_i, $DCF(t_i)$ corresponds to the i^{th} row of the normalized matrix M in Table 3.2. For larger clusters, the DCF is computed recursively as follows. Let c^* denote the cluster we obtain by merging two

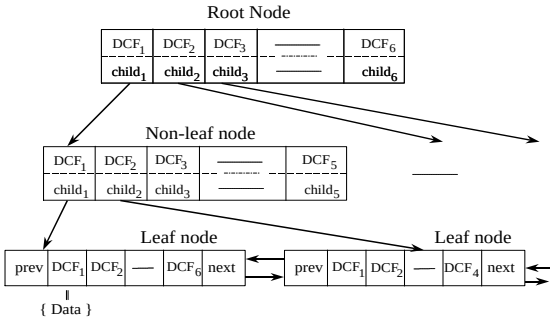


Figure 3.3: A *DCF* tree with branching factor 6

clusters c_1 and c_2. The *DCF* of the cluster c^* is equal to

$$DCF(c^*) = \Big(p(c^*), p(V|c^*)\Big)$$

where $p(c^*)$ and $p(V|c^*)$ are computed using Equations 3.2, and 3.3 respectively, where $Y = V$.

We define the distance, $d(c_1, c_2)$, between $DCF(c_1)$ and $DCF(c_2)$ as the information loss $\delta I(c_1, c_2)$ incurred by merging the corresponding clusters c_1 and c_2. The distance $\delta I(c_1, c_2)$ is computed using Equation 3.4, where $Y = V$. The information loss depends only on the clusters c_1 and c_2, and not on the clustering $\mathbf{C}$ of which they are part.

The *DCF*s can be stored and updated incrementally. The probability vectors are stored as sparse vectors. Each *DCF* provides a summary of the corresponding cluster that is sufficient for computing the distance between two clusters.

3.4.2 The *DCF* Tree

The *DCF* tree is a height-balanced tree as depicted in Figure 3.3. Each node in the tree contains at most B entries, where B is the *branching factor* of the tree. Each node entry stores one *DCF*. At any point in the construction of the tree, the *DCF*s at the leaves define a clustering of the tuples seen so far. Each non-leaf node stores *DCF*s that are produced by summarizing the *DCF*s of its children. The *DCF* tree is built in a B-tree-like dynamic fashion. The insertion algorithm is described in detail below. After

all tuples are inserted into the tree, the *DCF* tree embodies a compact representation in which a particular clustering of the data set is summarized by the information in the *DCF*s of the leaves.

3.4.3 The LIMBO Clustering Algorithm

The LIMBO algorithm proceeds in three phases. In the first phase, the *DCF* tree is constructed to summarize the data. In the second phase, the *DCF*s of the *DCF* tree leaves are merged into new *DCF*s to produce a chosen number of clusters. In the third phase, we associate each tuple with the *DCF* to which the tuple is closest.

Phase 1: Insertion into the *DCF* tree. Tuples are read and inserted one by one. Tuple t is converted into $DCF(t)$, as described in Section 3.4.1. Then, starting from the root, we trace a path downward in the *DCF* tree. When at a non-leaf node, we compute the distance between $DCF(t)$ and each *DCF* entry of the node, finding the closest *DCF* entry to $DCF(t)$. We follow the child pointer of this entry to the next level of the tree. When at a leaf node, let $DCF(c)$ denote the *DCF* entry in the leaf node that is closest to $DCF(t)$. $DCF(c)$ is the summary of some cluster c. At this point we need to decide whether t will be absorbed into the cluster c or not.

In our space-bounded algorithm, an input parameter S controls the maximum memory size for the *DCF* tree. Let E be the maximum size of a *DCF* entry (note that sparse *DCF*s may be smaller than E). We compute the maximum number of nodes ($N = S/(EB)$) and keep a counter of the number of used nodes as we build the tree. If there is an empty entry in the leaf node that contains $DCF(c)$, then $DCF(t)$ is placed in that entry. If there is no empty leaf entry and there is sufficient free space, then the leaf node is split into two leaves. We find the two *DCF*s in the leaf node that are farthest apart and we use them as seeds for the new leaves. The remaining *DCF*s, and $DCF(t)$ are placed in the leaf that contains the seed *DCF* to which they are closest. Finally, if the space bound has been reached, then we compare $\delta I(c, t)$ with the minimum distance of any two

DCF entries in the leaf. If $\delta I(c,t)$ is smaller than this minimum, we merge $DCF(t)$ with $DCF(c)$; otherwise, the two closest entries are merged and $DCF(t)$ occupies the freed entry. The space-bounded version of our algorithm is denoted by LIMBO$_S$.

Alternatively, if we wish to control the information that is lost at each merge, rather than the model size, we use a threshold on the distance $\delta I(c,t)$ (that is, the information loss incurred by merging tuple t into cluster c). We merge $DCF(t)$ with $DCF(c)$ only if $\delta I(c,t)$ is less than a specified threshold. In this way, we control the information lost in merging tuple t with cluster c. The selection of an appropriate threshold value will necessarily be data dependent, and there is a need for an intuitive way of allowing a user to set this threshold. If the threshold is chosen to be too small, then either the *DCF* tree can become huge or else the physical space available will be exhausted. A candidate guideline for setting the threshold is to choose a value that is a fraction of the average mutual information between the tuples $\mathbf{T}$ and the attribute values $\mathbf{V}$. Within a data set, every tuple contributes, on "average", $I(V;T)/n$ to the mutual information $I(V;T)$. We define the clustering threshold to be a multiple ϕ of this average and we denote the threshold by $\tau(\phi)$. That is,

$$\tau(\phi) = \phi \frac{I(V;T)}{n}$$

We can make a pass over the data, or use a sample of the data, to estimate $I(V;T)$. Given a value for ϕ ($0 \leq \phi \ll n$), if a merge incurs information loss more than ϕ times the "average" mutual information, then the new tuple is placed in a cluster by itself. Note that value $\tau(\phi)$ plays a role that is analogous to the *radius* of a cluster for numerical data. In the extreme case $\phi = 0.0$, we prohibit any information loss in our summary and our algorithm is the same as AIB. Empirically, we have shown that for values of ϕ close to 1.0 we obtain a concise and informative summarization. We denote this version of our algorithm by LIMBO$_\phi$ and discuss the effect of ϕ in Section 3.6.4.

When a leaf node is split, resulting in the creation of a new leaf node, the leaf's parent is updated, and a new entry is created at the parent node that describes the

newly created leaf. If there is space in the non-leaf node, we add a new *DCF* entry, otherwise the non-leaf node must also be split. This process continues upward in the tree until the root is either updated or split itself. In the latter case, the height of the tree increases by one.

Phase 2: Clustering. After the construction of the *DCF* tree, the leaf nodes hold the *DCF*s of a clustering $\tilde{\mathbf{C}}$ of the tuples in $\mathbf{T}$. Each $DCF(c)$ corresponds to a cluster $c \in \tilde{\mathbf{C}}$, and contains sufficient statistics for computing $p(V|c)$, and probability $p(c)$. We employ the *Agglomerative Information Bottleneck (AIB)* algorithm to cluster the *DCF*s in the leaves and produce clusterings of the *DCF*s. The final result is a reduced *DCF* tree in which the leaves correspond to clusters of a k-clustering for a chosen value of k.

The time for this phase depends upon the number of clusters in clustering $\tilde{\mathbf{C}}$. We note that any clustering algorithm is applicable at this phase of the algorithm. We assure that the number of the *DCF*s in the leaves is sufficiently small that the computation time of AIB does not dominate the running time of the full algorithm.

Phase 3: Associating tuples with clusters. In the final phase, we perform a scan over the data set and assign each tuple to the cluster whose cluster descriptor (a corresponding *DCF* summary) is closest to the tuple.

3.4.4 Analysis of LIMBO

We now present an analysis of the I/O and CPU costs for each phase of the LIMBO algorithm. In what follows, n is the number of tuples in the data set, d is the total number of attribute values, B is the branching factor of the *DCF* tree, S is the size of the available memory for the *DCF* tree in LIMBO$_S$ and k is the chosen number of clusters.

Phase 1: The I/O cost of this stage is a scan that involves reading the data set from the disk. For the CPU cost, when a new tuple is inserted, the algorithm considers a path

of nodes in the tree, and for each node in the path, it performs at most B operations (distance computations, or updates), each taking time $\mathcal{O}(d)$. Thus, if h is the height of the *DCF* tree produced in Phase 1, locating the correct leaf node for a tuple takes time $\mathcal{O}(hdB)$. The time required for splitting a node is $\mathcal{O}(dB^2)$. If U is the number of non-leaf nodes, then all splits are performed in time $\mathcal{O}(dUB^2)$ in total. Hence, the CPU cost of creating the *DCF* tree is $\mathcal{O}(nhdB + dUB^2)$. We observed experimentally that LIMBO produces compact trees of small height.

Phase 2: For values of S that produce clusterings of high quality the *DCF* tree is compact enough to fit in main memory. Hence, there is no I/O cost involved in this phase, since it involves only the clustering of the leaf node entries of the *DCF* tree. If L is the number of *DCF* entries at the leaves of the tree, then the AIB algorithm takes time $\mathcal{O}(L^2 d^2 \log L)$. In our experiments, $L \ll n$, so the CPU cost is low.

Phase 3: The I/O cost of this phase is the reading of the data set from the disk again. The CPU complexity is $\mathcal{O}(kdn)$, since each tuple is compared against the k *DCF*s that represent the clusters.

In the experimental section (Section 6.4), we show how actual execution times are distributed among the different phases of our algorithm.

3.5 Intra-Attribute Value Distance

In this section, we propose a novel approach that can be used within LIMBO to quantify the distance between attribute values of the same attribute. Categorical data is characterized by the fact that there is no inherent distance between attribute values. For example, in the movie database instance, given the values "Scorsese" and "Coppola", it is not apparent how to assess their similarity. Comparing the set of tuples in which they appear is not useful since every movie has a single director. In order to compare attribute values, we need to place them within a *context*. Then, two attribute values are

similar if the contexts in which they appear are similar. We define the context as the distribution these attribute values induce on the remaining attributes. For example, for the attribute "director", two directors are considered similar if they induce a "similar" distribution over the attributes "actor" and "genre".

Formally, let A' be the attribute of interest, and let $\mathbf{A}'$ denote the set of values of attribute A'. Also let $\tilde{\mathbf{A}} = \mathbf{A} \setminus \mathbf{A}'$ denote the set of attribute values for the remaining attributes. For the example of the movie database, if A' is the director attribute, with $\mathbf{A}' = \{d.S, d.C, d.H, d.K\}$, then $\tilde{\mathbf{A}} = \{ac.DN, ac.S, ac.G, g.Cr,\ g.T, g.C\}$. Let A' and $\tilde{A}$ be random variables that range over $\mathbf{A}'$ and $\tilde{\mathbf{A}}$ respectively, and let $p(\tilde{A}|v)$ denote the distribution that value $v \in \mathbf{A}'$ induces on the values in $\tilde{\mathbf{A}}$. For some $a \in \tilde{\mathbf{A}}$, $p(a|v)$ is the fraction of the tuples in $\mathbf{T}$ containing v, that also contain value a. Also, for some $v \in \mathbf{A}'$, $p(v)$ is the fraction of tuples in $\mathbf{T}$ that contain the value v. Table 3.3 shows an example of a table when A' is the director attribute.

director	ac.DN	ac.S	ac.G	g.Cr	g.T	g.C	p(d)
Scorsese	1/2	0	0	1/2	0	0	1/6
Coppola	1/2	0	0	1/2	0	0	1/6
Hitchcock	0	1/3	1/3	0	2/3	0	2/6
Koster	0	1/3	1/3	0	0	2/3	2/6

Table 3.3: The "director" attribute

For two values $v_1, v_2 \in \mathbf{A}'$, we define the distance between v_1 and v_2 to be the information loss $\delta I(v_1, v_2)$, incurred about the variable $\tilde{A}$ if we merge values v_1 and v_2. This is equal to the increase in the uncertainty of predicting the values of variable $\tilde{A}$, when we replace values v_1 and v_2 with $v_1 \vee v_2$. In the movie example, Scorsese and Coppola are the most similar directors.[1]

The definition of a distance measure for categorical attribute values is a contribution in itself, since it imposes some structure on an inherently unstructured problem. We can now define a distance measure between tuples as the sum of the distances of the individual

[1] A conclusion that agrees with a well-informed cinematic opinion.

attribute values. Another possible application is to cluster intra-attribute values. For example, in a movie database, we may be interested in discovering clusters of directors or actors.

Given the joint distribution of random variables A' and $\tilde{A}$, we can apply the LIMBO algorithm for clustering the values of attribute A'. Merging two values $v_1, v_2 \in A'$, produces a new cluster of values $v_1 \vee v_2$, where $p(v_1 \vee v_2) = p(v_1) + p(v_2)$, since v_1 and v_2 never appear together. Also,

$$p(a|v_1 \vee v_2) = \frac{p(v_1)}{p(v_1 \vee v_2)} p(a|v_1) + \frac{p(v_2)}{p(v_1 \vee v_2)} p(a|v_2) \ .$$

The problem of defining a context sensitive distance measure between attribute values is also considered by Das and Mannila [DM00]. They define an iterative algorithm for computing the *interchangeability* of two values. We believe that our approach gives a natural quantification of the concept of interchangeability. Furthermore, our approach has the advantage that it allows for the definition of distance between clusters of values, which can be used to perform intra-attribute value clustering. Gibson et al. [GKR98] proposed STIRR, an algorithm that clusters attribute values. STIRR does not define a distance measure between attribute values and, furthermore, is restricted to producing only two clusters of values (see Section 2.4). Palmer and Faloutsos [PF03] define a similarity function between categorical values by looking at how they co-occur with other values. They exploit the duality between random walks on graphs and electrical circuits to develop their similarity measure. Our work is also complementary to the work on keyword searching [BHN+02, HP02] where, given a keyword, a set of answers is retrieved. In our work, we produce sets of similar values, *i.e.* keywords, of a specific domain, and hence, given a value from this domain, we may retrieve other similar values from its cluster. However, we do not deal with keywords from different domains. Further experimentation is needed to better assess similarities and differences of our approach relative to the work on keyword searching. Finally, we intend to compare our algorithm

with the co-clustering approach of Dhillon, Malella and Modha [DMM03], where, given a data set, its tuples and values are clustered interchangeably so that the information loss is minimum.

3.6 Experimental Evaluation

In this section, we perform a comparative experimental evaluation of the LIMBO algorithms on both real and synthetic data sets. We compare both versions of LIMBO with other categorical clustering algorithms, including COOLCAT [BCL02a, BCL02b], which is the only other scalable information-theoretic clustering algorithm of which we know.

3.6.1 Algorithms

We compare the clustering quality of LIMBO with the following algorithms already presented in Chapter 1.

ROCK Algorithm [GRS99]. We use the *Jaccard Coefficient* for the similarity measure as suggested in the original paper. We remind the reader that ROCK uses θ as a parameter to assess initial similarity among all pairs of objects. For data sets that appear in the original ROCK paper, we set the threshold θ to the value suggested there, otherwise we set θ to the value that gave us the best results in terms of quality. For our experimentation, we use the implementation of Guha et al. [GRS99].

COOLCAT Algorithm [BCL02b]. COOLCAT differs from our approach in that it employs sampling, and it is non-hierarchical. COOLCAT starts with a sample of points and identifies a set of k initial tuples such that the minimum pairwise distance among them is maximized. These serve as representatives of the k clusters. All remaining tuples of the data set are then placed in one of the clusters such that, at each step, the increase in the entropy of the resulting clustering is minimized. For the experiments, we implement COOLCAT based on the CIKM paper by Barbarà et al. [BCL02b].

STIRR Algorithm [GKR98]. We compare this algorithm with our intra-attribute value clustering algorithm. In our experiments, we use our own implementation and report results for ten iterations, as suggested in the original paper.

LIMBO Algorithm. As already stated in Section 3.4, we have two versions of the LIMBO algorithm, one that is space-bounded, LIMBO$_S$ as well as a version, in which we control the loss of information at each merge (LIMBO$_\phi$). The choice of which version to use depends on the application we consider. For example, for an application where we need to control the information content of each cluster produced, we use LIMBO$_\phi$. When we apply the algorithm to a data set under limited memory budget, LIMBO$_S$ is more appropriate. We remind the reader that in the extreme case of $\phi = 0.0$, we prohibit any information loss in our summary. This is equivalent to setting $S = \infty$ in LIMBO$_S$. We discuss the effect of ϕ and S in Section 3.6.4. Algorithmically, only the merging decision in Phase 1 differs in the two versions, while all other phases remain the same for both LIMBO$_S$ and LIMBO$_\phi$.

In our implementation, we store DCFs as sparse vectors using a sparse vector library (`http://www.genesys-e.org/ublas/`).

3.6.2 Data Sets

We experimented with the data sets described below. The first three have been previously used for the evaluation of the aforementioned algorithms [BCL02b, GKR98, GRS99]. The synthetic data sets are used both for quality comparison, and for our scalability evaluation.

Congressional Votes: This relational data set was taken from the *UCI Machine Learning Repository*.[2] It contains 435 tuples of votes from the U.S. Congressional Voting Record of 1984. Each tuple is a congress-person's vote on 16 issues and each vote is boolean,

[2] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

either YES or NO. Each congress-person is classified as either Republican or Democrat. There are a total of 168 Republicans and 267 Democrats. There are 288 missing values that we treat as the same value when they belong to the domain of the same attribute and as different when they belong to the domain of different attributes.

Mushroom: The Mushroom relational data set also comes from the UCI Repository. It contains 8,124 tuples, each representing a mushroom characterized by 22 attributes, such as color, shape, odor, etc. The total number of distinct attribute values is 117. Each mushroom is classified as either poisonous or edible. There are 4,208 edible and 3,916 poisonous mushrooms in total. There are 2,480 missing values that we treat in the same way as in Votes.

Database and Theory Bibliography. This relational data set contains 8,000 tuples that represent research papers. About 3,000 of the tuples represent papers from database research and 5,000 tuples represent papers from theoretical computer science. Each tuple contains four attributes with values for the first Author, second Author, Conference/Journal and the Year of publication.[3] We use this data to test our value clustering algorithm by attempting to cluster conferences and journals according to the database or theoretical computer science areas.

Synthetic Data Sets. We produce synthetic data sets using a data generator available on the Web.[4] This generator offers a wide variety of options, in terms of the number of tuples, attributes, and attribute domain sizes. We specify the number of classes in the data set by the use of conjunctive rules of the form $(Attr_1 = a_1 \wedge Attr_2 = a_2 \wedge \ldots) \Rightarrow Class = c1$. The rules may involve an arbitrary number of attributes and attribute values. We name these synthetic data sets by the prefix DS followed by the number of classes in the data set, *e*.g., DS5 or DS10. The data sets contain 5,000 tuples, and 10

[3]Following the approach of Gibson et al. [GKR98], if the second author does not exist, then the name of the first author is copied instead. We also filter the data so that each conference/journal appears at least 5 times.

[4]`http://www.datgen.com/`

attributes, with domain sizes between 20 and 40 for each attribute. Three attributes participate in the rules the data generator uses to produce the class labels. Additional larger synthetic data sets are described in Section 3.6.6.

Web Data: This is a market-basket data set that consists of a collection of web pages. The pages were collected as described by Kleinberg [Kle98]. A query is made to a search engine, and an initial set of web pages is retrieved. This set is augmented by including pages that point to, or are pointed to by pages in the set. Then, the links between the pages are discovered, and the underlying graph is constructed. Following the terminology of Kleinberg [Kle98], we define a *hub* to be a page with non-zero out-degree, and an *authority* to be a page with non-zero in-degree.

Our goal is to cluster the authorities in the graph. The set of objects **T** is the set of authorities in the graph, while the set of attribute values **A** is the set of hubs. Each authority is expressed as a vector over the hubs that point to this authority. For our experiments, we use the data set used by Borodin et al. [BRRT01] for the "abortion" query. We applied a filtering step to assure that each hub points to more than 10 authorities and each authority is pointed to by more than 10 hubs. The data set contains 93 authorities related to 102 hubs.

All data sets are summarized in Table 3.4.

Data Set	Records	Attributes	Attr. Values	Missing
Votes	435	16	48	288
Mushroom	8,124	22	117	2,480
Bibliographic	8,000	4	9,587	0
Web Data	93	102	102	0
DS5	5,000	10	314	0
DS10	5,000	10	305	0

Table 3.4: Summary of the data sets used

3.6.3 Quality Measures for Clustering

Clustering quality lies in the eye of the beholder; determining the best clustering usually depends on subjective criteria. Consequently, we will use several plausible quantitative measures of clustering performance that are likely to match well with the subjective clustering quality in various situations.

Many data sets commonly used in testing clustering algorithms include a variable that is hidden from the algorithm, and specifies the class with which each tuple is associated. All data sets we consider include such a variable. This variable is *not* used by the clustering algorithms. While there is no guarantee that any given classification corresponds to an optimal clustering, it is nonetheless enlightening to compare clusterings with pre-specified classifications of tuples. To do this, we use the Min Classification Error, and Precision and Recall measures described below.

Information Loss, (IL): We use the information loss, $I(V;T) - I(V;C)$ to compare clusterings. The lower the information loss, the better the clustering. For a clustering with low information loss, given a cluster, we can predict the attribute values of the tuples in the cluster with relatively high accuracy.

Category Utility, (CU): Category utility [GC85] is defined as the difference between the expected number of attribute values that can be correctly guessed given a clustering, and the expected number of correct guesses with no such knowledge. Let $\mathbf{C}$ be a clustering. If A_i is an attribute with values v_{ij}, then CU is given by the following expression:

$$CU = \sum_{c \in \mathbf{C}} \frac{|c|}{n} \sum_{i}^{m} \sum_{j}^{|\mathbf{V}_i|} [P(A_i = v_{ij}|c)^2 - P(A_i = v_{ij})^2]$$

Min Classification Error, (E_{min}): Assume that the tuples in $\mathbf{T}$ are already classified into k classes $\mathbf{G} = \{g_1, \ldots, g_k\}$, and let $\mathbf{C}$ denote a clustering of the tuples in $\mathbf{T}$ into k clusters $\{c_1, \ldots, c_k\}$ produced by a clustering algorithm. Consider a one-to-one mapping,

f, from classes to clusters, such that each class g_i is mapped to the cluster $f(g_i)$. The *classification error* of the mapping is defined as

$$E = \sum_{i=1}^{k} \frac{\left| g_i \cap \overline{f(g_i)} \right|}{|T|}$$

where $\left| g_i \cap \overline{f(g_i)} \right|$ measures the number of tuples in class g_i that received the wrong label. The *optimal* mapping between clusters and classes, is the one that minimizes the classification error. We use E_{min} to denote the fraction of tuples mislabelled under the optimal mapping.

Precision, (P), **Recall, (R)**: Without loss of generality assume that the optimal mapping assigns class g_i to cluster c_i. We define precision, P_i, and recall, R_i, for a cluster c_i, $1 \leq i \leq k$ as follows.

$$P_i = \frac{|c_i \cap g_i|}{|c_i|} \qquad and \qquad R_i = \frac{|c_i \cap g_i|}{|g_i|} \ .$$

P_i and R_i take values between 0 and 1. Intuitively, P_i measures the accuracy with which cluster c_i reproduces class g_i, while R_i measures the completeness with which c_i reproduces class g_i. We define the precision and recall of the clustering as the weighted averages of the precision and recall over all classes. More precisely

$$P = \sum_{i=1}^{k} \frac{|g_i|}{|T|} P_i \qquad and \qquad R = \sum_{i=1}^{k} \frac{|g_i|}{|T|} R_i \ .$$

We think of precision, recall, and classification error as indicative values of the ability of the algorithm to reconstruct the indicated classes in the data set.

In our experiments, we report values for all of the above measures. For LIMBO and COOLCAT, the numbers are averages over 100 runs with different (random) orderings of the tuples. Our main results are averages over the 100 runs, but we also present statistics on the variability among the 100 runs.

3.6.4 Quality-Efficiency Trade-offs for LIMBO

In LIMBO, we can control the information loss during merges (using ϕ) or the size of the model (using S). Both ϕ and S permit a trade-off between the compactness of the model (number of leaf entries in the tree), and the expressiveness (information preservation) of the summarization it produces. For small values of ϕ and large values of S, we obtain a fine grain representation of the data set at the end of Phase 1. However, this results in a tree with a large number of leaf entries, which leads to a higher computational cost for both Phase 1 and Phase 2 of the algorithm. For large values of ϕ and small values of S, we obtain a compact representation of the data set (small number of leaf entries), which results in a smaller execution time, at the expense of increased information loss.

We now investigate this trade-off for a range of values for ϕ and S. Using a range for the values of B from $B = 2$ up to $B = 50$, we observed experimentally that the branching factor B does not significantly affect the quality of the clustering. We set $B = 4$, which results in an acceptable execution time for Phase 1. Figure 3.4 presents the execution times for LIMBO$_S$ and LIMBO$_\phi$ on the DS5 data set, as a function of ϕ and S. For $\phi = 0.25$ the Phase 2 time is 210 seconds (beyond the edge of the graph). The figures also include the size of the tree in KBytes, which was measured by a scan of the tree upon completion of Phase 1. In this figure, we observe that for small ϕ and large S the computational bottleneck of the algorithm is Phase 2. As ϕ increases and S decreases the time for Phase 2 decreases in a quadratic fashion. This agrees with the plot in Figure 3.6(a), where we observe that the number of leaves decreases also in a quadratic fashion. Due to the decrease in the size (and height) of the tree, time for Phase 1 also decreases, however, at a much slower rate. Phase 3, as expected, remains unaffected, and it is only a few seconds for all values of ϕ and S. For $\phi \geq 1.0$, and $S \leq 256$KB the number of leaf entries becomes sufficiently small that the computational bottleneck of the algorithm becomes Phase 1. For these values the execution time is dominated by the linear scan and insertion time of the data in Phase 1.

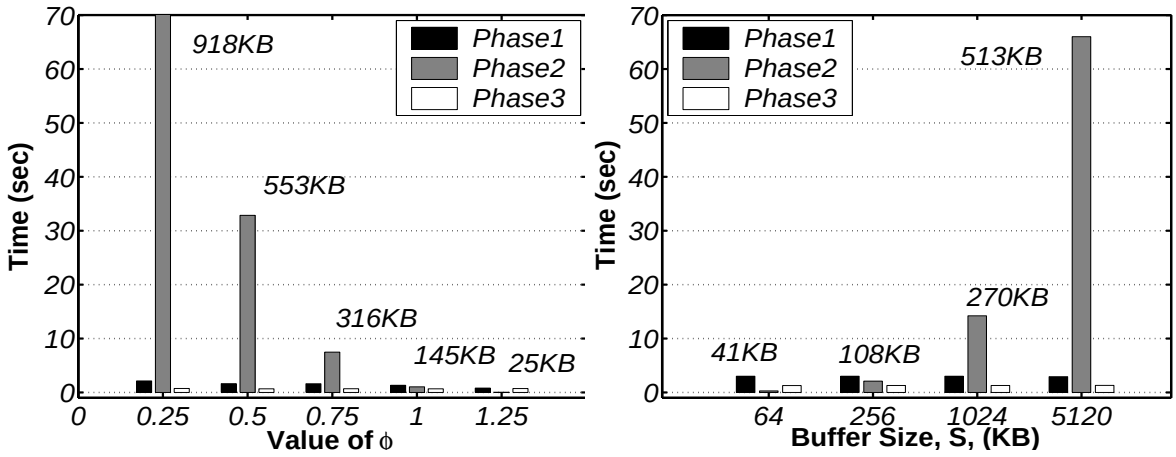


Figure 3.4: LIMBO$_\phi$ and LIMBO$_S$ execution times (DS5)

In addition to Figure 3.4, we produced the graph of Figure 3.5. This graph shows the size of the tree produced by LIMBO in Phase 2 and the corresponding time for clustering the leaves of this tree. The graph demonstrates that, for a given buffer size, LIMBO$_S$ requires up to 20% more time than LIMBO$_\phi$. This can be attributed to the difference in the merging strategies used when new tuples are converted into *DCF*s and need to be placed or merged in a leaf entry. LIMBO$_S$ performs more comparisons among leaf entries than LIMBO$_\phi$.

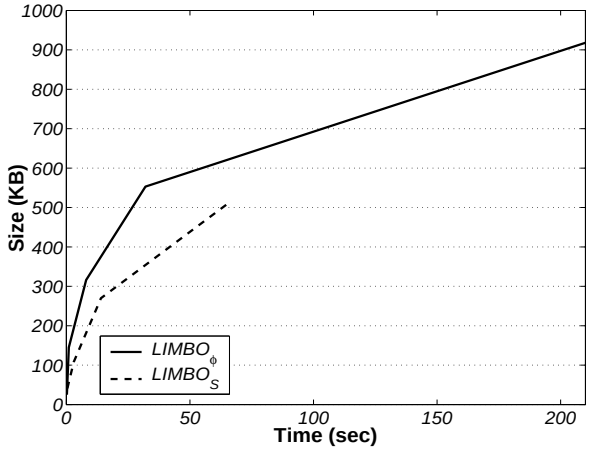


Figure 3.5: LIMBO$_\phi$ and LIMBO$_S$ model sizes over time (DS5)

We now study the change in the quality measures for the same range of values for ϕ and S. In the extreme cases of $\phi = 0.0$ and $S = \infty$, we only merge identical tuples,

	Votes	**Mushroom**	**DS5**	**DS10**
LIMBO$_\phi$	94.01%	99.77%	98.68%	98.82%
LIMBO$_S$	85.94%	99.34%	95.36%	95.28%

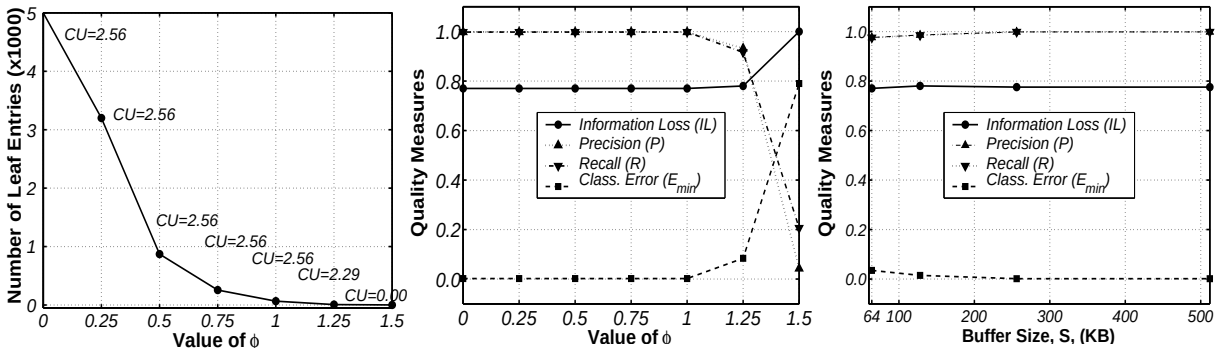
Table 3.5: Reduction in *DCF* leaf entries

and no information is lost in Phase 1. LIMBO then reduces to the AIB algorithm, and we obtain the same quality as AIB. Figures 3.6(b) and 3.6(c) show the quality measures for the different values of ϕ and S. The CU value (not plotted) is equal to 2.51 for $S \leq 256$KB, and 2.56 for $S > 256$KB. We observe that for $\phi \leq 1.0$ and $S > 256$KB, we obtain clusterings of *exactly* the same quality as for $\phi = 0.0$ and $S = \infty$, that is, the same clusterings as produced by the AIB algorithm. At the same time, for $\phi = 1.0$ and $S = 256$KB the execution time of the algorithm is only a small fraction of that of the AIB algorithm, which was a few minutes.

Similar observations hold for all other data sets. There is a range of values for ϕ and S where the execution time of LIMBO is dominated by Phase 1, while at the same time, we observe essentially no change (up to the third decimal digit) in the quality of the clustering. Table 3.5 shows the reduction in the number of leaf entries for each data set for LIMBO$_\phi$ and LIMBO$_S$. The numbers are computed using the expression:

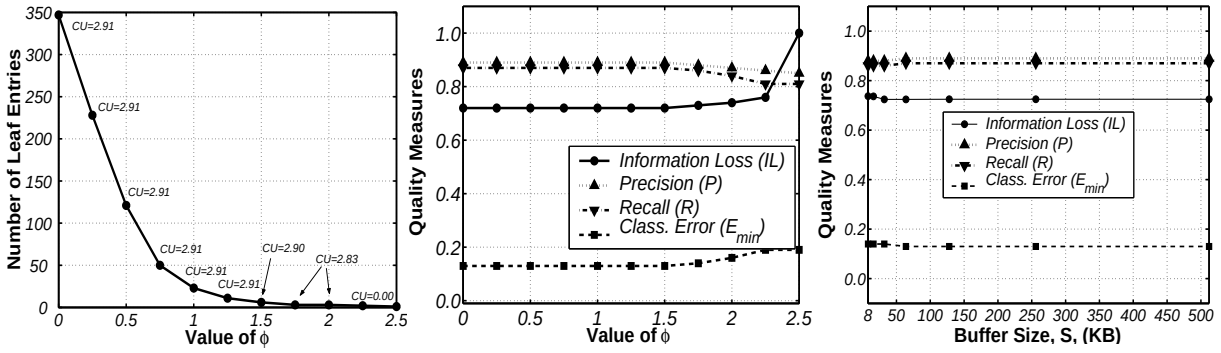
$$\frac{n-e}{n} \cdot 100\%$$

where n is the number of tuples and e the number of *DCF* leaf entries. The parameters ϕ and S are set so that the cluster quality is almost identical to that of AIB (as demonstrated in the tables in Section 3.6.5). These experiments demonstrate that in Phase 1 we can obtain significant compression of the data sets at no expense in the final quality of clustering. The consistency of LIMBO can be attributed in part to the effect of Phase 3, which assigns the tuples to cluster representatives, and recovers some of the information loss accepted in the previous phases. Thus, it is sufficient for Phase 2 to discover k well separated representatives. As a result, even for large values of ϕ and small values of S, LIMBO obtains essentially the same clustering quality as AIB. Similar



(a) (b) (c)

Figure 3.6: DS5: (a) Leaf entries, (b) LIMBO$_\phi$ quality, (c) LIMBO$_S$ quality



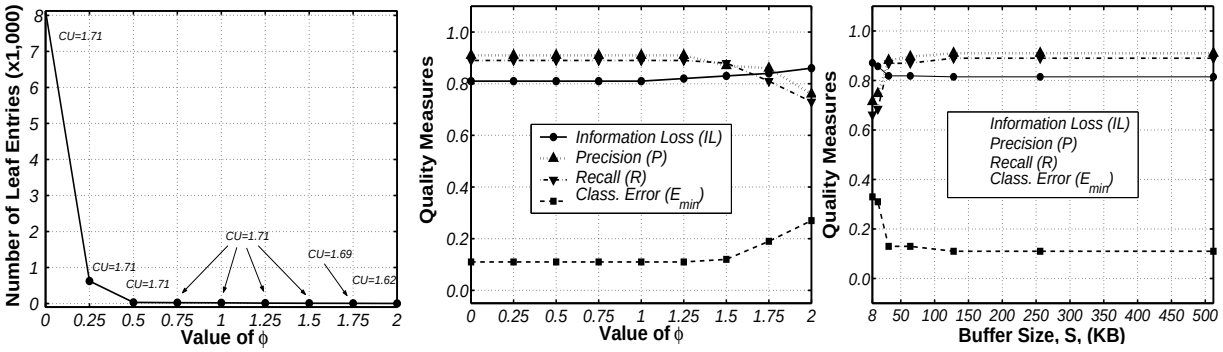
(a) (b) (c)

Figure 3.7: Votes: (a) Leaf entries, (b) LIMBO$_\phi$ quality, (c) LIMBO$_S$ quality

figures are presented for all other data sets; Figures 3.7(a), 3.7(b) and 3.7(c) for Votes, Figures 3.8(a), 3.8(b) and 3.8(c) for Mushroom and Figures 3.9(a), 3.9(b) and 3.9(c) for DS10. As a general observation, these graphs show that for LIMBO$_\phi$, it appears that $\phi \leq 1.0$ is always a good choice.. For data sets with small number of clusters $\phi = 1.25$ or $\phi = 1.5$ are also acceptable choices. On the other hand, for LIMBO$_S$ it can be seen that $S = 250KB$ is the safest choice with a minor loss in the quality.

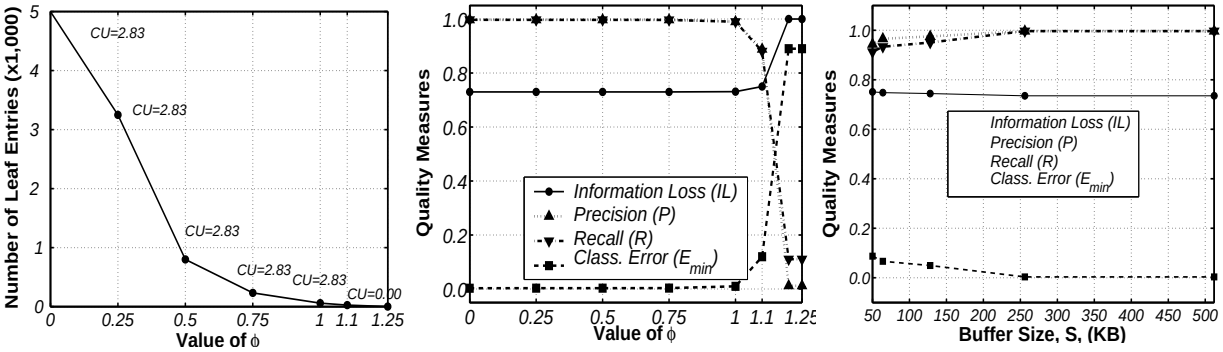
Finally, the graphs of the information-theoretic quantities involved in the clustering of Votes with LIMBO$_\phi$ ($\phi = 0.0$) are given in Figure 3.10.[5] Similar graphs are observed for all other data sets.

[5]We give the graphs in two separate figures to increase their readability.



(a) (b) (c)

Figure 3.8: Mushroom: (a) Leaf entries, (b) LIMBO$_\phi$ quality, (c) LIMBO$_S$ quality



(a) (b) (c)

Figure 3.9: DS10: (a) Leaf entries, (b) LIMBO$_\phi$ quality, (c) LIMBO$_S$ quality

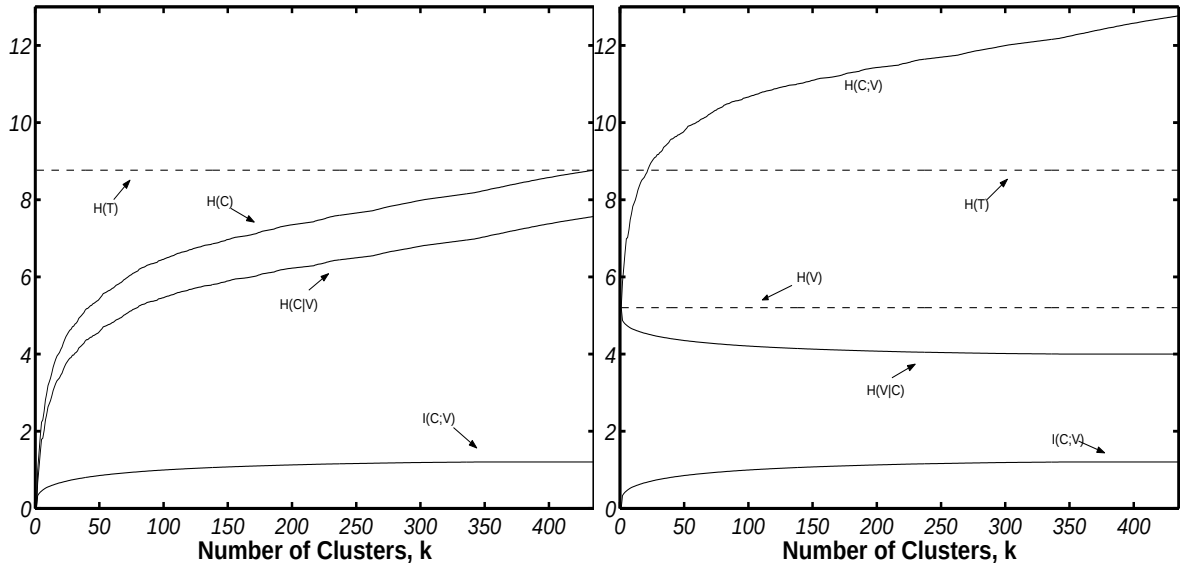


Figure 3.10: Information-theoretic quantities for LIMBO$_\phi$ as k changes (Votes)

3.6.5 Evaluation of LIMBO

Comparison to Optimal Clustering

To get a better feel for how LIMBO performs on categorical data, we implemented a brute force algorithm (BF), that generates all possible clusterings and selects the one with the lowest information loss. Note that BF is computationally feasible only if the number of tuples is quite small. We experimented with data sets of sizes $n = 7$ and $n = 8$ and for all clustering sizes k, $2 \leq k < n$ [6]. The data sets were produced using the synthetic data generator and they each contain five attributes, each one of them with a domain of size three. We compared the information loss of the Brute Force algorithm against that of LIMBO$_\phi$ for $\phi = 0.0$ (i.e., starting with each tuple in its own cluster). Results are given in Table 3.6.

The combinations of n, k did not contain clusterings with the same information loss. For all these combinations in Table 3.6, LIMBO produces a clustering with information loss equal to the minimum information loss (the numbers agreed even in more significant

[6]The total number of clusterings for given values of n and k is equal to the Stirling number of second order. Even for $n = 20$ and $k = 5$, this number is equal to approximately $7.5 \cdot 10^{11}$.

n, k	**Clusterings**	**BF**	**LIMBO**
7,2	63	74%	74%
7,3	301	46%	46%
7,4	350	21%	21%
7,5	140	16%	16%
7,6	21	7%	7%
8,2	127	65%	65%
8,3	966	45%	45%
8,4	1701	39%	39%
8,5	1050	21%	21%
8,6	266	16%	16%
8,7	28	9%	9%

Table 3.6: Information loss of brute force and LIMBO ($\phi = 0.0$)

digits than the ones presented in Table 3.6).These results are indicative of the ability of the LIMBO algorithm to find an optimal clustering when the number of tuples is small.

Tuple Clustering

Table 3.7 shows the results for all algorithms on all quality measures for the Votes and Mushroom data sets. For LIMBO$_\phi$, we present results for $\phi = 1.0$ while for LIMBO$_S$, we present results for $S = 128K$, since that is enough space for LIMBO$_S$ to match the quality of AIB. We can see that both versions of LIMBO have results almost identical to the quality measures for $\phi = 0.0$ and $S = \infty$, *i.e.*, the AIB algorithm. The *size* entry in the table holds the number of *DCF* entries in all leaf nodes for LIMBO, and the sample size for COOLCAT. For the Votes data set, we use the whole data set as a sample, while for Mushroom, we use 1,000 tuples. As Table 3.7 indicates, LIMBO's quality is superior to ROCK and COOLCAT for both data sets. In terms of information loss, LIMBO created clusters which retained most of the initial information about the attribute values. With respect to the other measures, LIMBO outperforms all other algorithms, exhibiting the highest CU, P and R in all data sets tested, as well as the lowest E_{min}.

We also evaluate LIMBO's performance on two synthetic data sets, namely DS5 and DS10. These data sets allow us to evaluate our algorithm on data sets with more than

Votes (2 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 0.0$,$S = \infty$)[**AIB**]	384	**72.52**	**0.89**	**0.87**	**0.13**	**2.89**
LIMBO ($\phi = 1.0$)	23	**72.55**	**0.89**	**0.87**	**0.13**	**2.89**
LIMBO ($S = 128KB$)	54	**72.54**	**0.89**	**0.87**	**0.13**	**2.89**
COOLCAT ($s = 435$)	435	73.55	0.87	0.85	0.15	2.78
ROCK ($\theta = 0.7$)	-	74.00	0.87	0.86	0.16	2.63
Mushroom (2 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 0.0$,$S = \infty$)[**AIB**]	8124	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
LIMBO ($\phi = 1.0$)	18	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
LIMBO ($S = 128KB$)	54	**81.46**	**0.91**	**0.89**	**0.11**	**1.71**
COOLCAT ($s = 1,000$)	1,000	84.57	0.76	0.73	0.27	1.46
ROCK ($\theta = 0.8$)	-	86.00	0.77	0.57	0.43	0.59

Table 3.7: Results for real data sets (bold fonts indicate results for LIMBO)

two classes. The results are shown in Table 3.8. We observe again that LIMBO has the lowest information loss and produces nearly optimal results with respect to precision and recall.

For the ROCK algorithm, we observed that it is very sensitive to the threshold value, θ, and in many cases, the algorithm produces one giant cluster that includes tuples from most classes. This results in poor precision and recall.

Comparison with COOLCAT

COOLCAT exhibits average clustering quality, as indicated by the P and R measures, that is close to that of LIMBO. It is interesting to examine how COOLCAT behaves when we consider other statistics. In Table 3.9, we present statistics for 100 runs of COOLCAT and LIMBO on different orderings of the Votes and Mushroom data sets. We present LIMBO's results for $\phi = 1.0$, which are very similar to those for $\phi = 0.0$. For the Votes data set, COOLCAT exhibits information loss as high as 95.31% with a variance of 12.25%. For all runs, we use the whole data set as the sample for COOLCAT.

For the Mushroom data set, the situation is better, but still the variance in information loss is as high as 3.5%. The sample size was 1,000 for all runs. Table 3.9 indicates

DS5 (n=5000, 10 attributes, 5 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 0.0$,$S = \infty$)[**AIB**]	5000	**77.56**	**0.998**	**0.998**	**0.002**	**2.56**
LIMBO ($\phi = 1.0$)	66	**77.56**	**0.998**	**0.998**	**0.002**	**2.56**
LIMBO ($S = 1024KB$)	232	**77.57**	**0.998**	**0.998**	**0.002**	**2.56**
COOLCAT ($s = 125$)	125	78.02	0.995	0.995	0.05	2.54
ROCK ($\theta = 0.0$)	-	85.00	0.839	0.724	0.28	0.44
DS10 (n=5000, 10 attributes, 10 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 0.0$,$S = \infty$)[**AIB**]	5000	**73.50**	**0.997**	**0.997**	**0.003**	**2.82**
LIMBO ($\phi = 1.0$)	59	**73.51**	**0.994**	**0.996**	**0.004**	**2.82**
LIMBO ($S = 1024KB$)	236	**73.52**	**0.996**	**0.996**	**0.004**	**2.82**
COOLCAT ($s = 125$)	125	74.32	0.979	0.973	0.026	2.74
ROCK ($\theta = 0.0$)	-	78.00	0.830	0.818	0.182	2.13

Table 3.8: Results for synthetic data sets (bold fonts indicate results for LIMBO)

that LIMBO behaves in a more stable fashion than COOLCAT over runs with different input orders. Notably, for the Mushroom data set, LIMBO's performance is exactly the same in all runs, and for Votes the variability in performance is very small. This indicates that LIMBO is insensitive to the input order of data.

The performance of COOLCAT appears to be sensitive to the following factors: the choice of representatives, the sample size, and the ordering of the tuples. After detailed examination, we found that the runs with maximum information loss for the Votes data set correspond to cases where COOLCAT selected an outlier as an initial representative. The Votes data set contains three such tuples, which are far from all other tuples, and they are naturally picked as representatives.[7] Reducing the sample size decreases the probability of selecting outliers as representatives, however it increases the probability of missing one of the clusters. In this case, high information loss may occur if COOLCAT picks as representatives two tuples that are not maximally far apart. Finally, there are cases where the same representatives may produce different results under different input orderings. As tuples are inserted to the clusters, the representatives "move" closer to the

[7] A newer version of COOLCAT includes a step designed to avoid such selections [BCL02a]. We plan to implement this step for future experiments.

VOTES		Min	Max	Avg	Var
LIMBO ($\phi = 1.0$)	IL	71.98	73.29	72.55	0.083
	CU	2.83	2.94	2.89	0.0006
LIMBO ($S = 128KB$)	IL	71.98	73.68	72.54	0.08
	CU	2.80	2.93	2.89	0.0007
COOLCAT ($s = 435$)	IL	71.99	95.31	73.55	12.25
	CU	0.19	2.94	2.78	0.15
MUSHROOM		Min	Max	Avg	Var
LIMBO ($\phi = 1.0$)	IL	81.45	81.45	81.45	0.00
	CU	1.71	1.71	1.71	0.00
LIMBO ($S = 1024KB$)	IL	81.46	81.46	81.46	0.00
	CU	1.71	1.71	1.71	0.00
COOLCAT ($s = 1000$)	IL	81.60	87.07	84.57	3.50
	CU	0.80	1.73	1.46	0.05

Table 3.9: Statistics for IL(%) and CU over 100 trials

inserted tuples, thus making the algorithm sensitive to the ordering of the data set.

In terms of in-memory operations, both LIMBO and COOLCAT include a stage that requires a quadratic number of them. For LIMBO this is Phase 2. For COOLCAT this is the step where all pairwise entropies between the tuples in the sample are computed. We experimented with both algorithms having the same input size for this phase, *i.e.*, we made the sample size of COOLCAT equal to the number of *DCF* leaf entries. leaves for LIMBO. Results for the Votes and Mushroom data sets are shown in Tables 3.10 and 3.11. LIMBO outperforms COOLCAT in all runs, for all quality measures. The two algorithms are closest in quality for the Votes data set with input size 27, and farthest apart for the Mushroom data set with input size 275. COOLCAT appears to perform relatively better with the smaller sample size, while LIMBO's performance is essentially unaffected by the choice of input size.

Web Data

Since this data set has no predetermined cluster labels, we use a different evaluation approach. We applied LIMBO with $\phi = 0.0$ and clustered the authorities into three clusters. (The choice of k is discussed in detail in Section 3.7.) The total information

Sample Size = Leaf Entries = 384					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO	**72.52**	**0.89**	**0.87**	**0.13**	**2.89**
COOLCAT	74.15	0.86	0.84	0.15	2.63
Sample Size = Leaf Entries = 27					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO	**72.55**	**0.89**	**0.87**	**0.13**	**2.89**
COOLCAT	73.50	0.88	0.86	0.13	2.87

Table 3.10: LIMBO vs COOLCAT on Votes with same number of objects as input to their corresponding expensive stages

Sample Size = Leaf Entries = 275					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
COOLCAT	83.50	0.76	0.73	0.27	1.46
Sample Size = Leaf Entries = 18					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
COOLCAT	82.10	0.82	0.81	0.19	1.60

Table 3.11: LIMBO vs COOLCAT on Mushroom with same number of objects as input to their corresponding expensive stages

loss was 61%. Figure 3.11 shows the authority to hub table, after permuting the rows so that we group together authorities in the same cluster, and the columns so that each hub is assigned to the cluster to which it has the most links. LIMBO manages to characterize

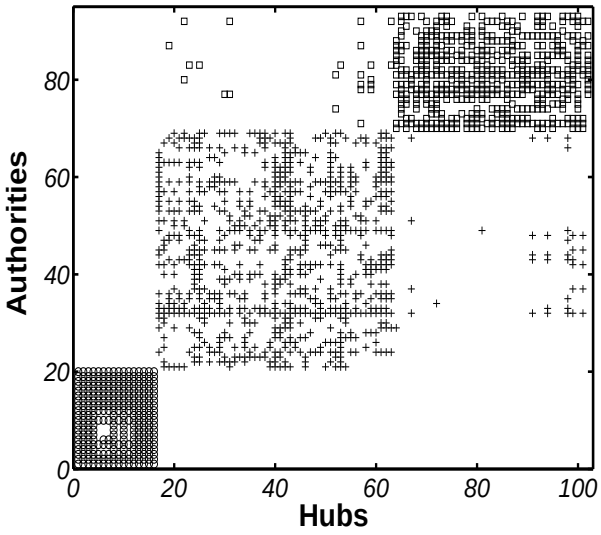


Figure 3.11: Clustering of web data

the structure of the web graph. Authorities are clustered in three distinct clusters, such that the ones in the same cluster share many hubs, while the ones in different clusters have very few hubs in common. The three different clusters correspond to different web communities, and different viewpoints on the issue of abortion. The first cluster (authorities 70 to 93) consists of "pro-choice" pages. The second cluster (authorities 21 to 69) consists of "pro-life" pages. The third cluster (authorities 1 to 20) contains a set of pages from `cincinnati.com` that were included in the data set by the algorithm that collects the web pages, despite having no apparent relation to the abortion query. We supplement the visual evidence of Figure 3.11 by some quantitative evidence: for the authorities in a cluster, we calculate the percentage of their hub linkages that are to hubs with the majority of the linkages to authorities in the same cluster. Hence, for the "pro-choice" cluster this percentage is 95.59%, for the "pro-life" cluster it is 95.55% and for the `cincinatti.com` cluster it is 100%. A complete list of the results can be found in Tables 3.12, 3.13 and 3.14. In almost all cases the URLs and Titles of the web pages

(Pro-Life Cluster) 49 web pages	
URL	**Title**
http://www.afterabortion.org	After Abortion: Information on the aftereffects of abortion
http://www.lifecall.org	Abortion Absolutely Not! A Several Sources Prolife Website
http://www.lifeinstitute.org	International abortion reports and prolife news
http://www.rtl.org	Information at Right to Life of Michigan's homepage
http://www.silentscream.org	Silent Scream Home Page
http://www.heritagehouse76.com	Empty title field
http://www.peopleforlife.org	Abortion, Life and Choice
http://www.stmichael.org/OCL/OCL.html	Orthodox Christians For Life Resource Page
http://www.worldvillage.com/wv/square/chapel/safehaven	Empty title field
http://www.prolife.com	Pro-Life America
http://www.roevwade.org	RoevWade.org
http://www.nrlc.org	National Right to Life Organization
http://www.hli.org	Human Life International (HLI)
http://www.pregnancycenters.org	Pregnancy Centers Online
http://www.prolife.org/ultimate	Empty title field
http://www.mich.com/ buffalo	Catholics United for Life
http://www.prolife.org	Empty title field
http://www.prolife.org/mssl	Empty title field
http://www.pfli.org	Pharmacists for Life International
http://www.rockforlife.org	Rock For Life
http://members.aol.com/nfofl	Empty title field
http://www.serve.com/fem4life	Feminists For Life of America
http://www.cc.org	Welcome to Christian Coalition of America Web site
http://www.cwfa.org	Concerned Women for America (CWA)
http://www.prolifeaction.org	Pro-Life Action League
http://www.ru486.org	The RU-486 Files
http://www.operationrescue.org	End Abortion in America
http://www.orn.org	Operation Rescue National
http://www.priestsforlife.org	Priests for Life Index
http://www.abortionfacts.com	Abortion facts and information, statistics, hotlines and helplines
http://www.prolifeinfo.org	The Ultimate Pro-Life Resource List
http://www.feministsforlife.org	Feminists For Life of America
http://www.marchforlife.org	The March For Life Fund Home Page
http://www.bfl.org	BFL Home Page
http://www.ppl.org	Presbyterians Pro-Life Home
http://www.wels.net/wlfl	WELS Lutherans for Life
http://www.lifeissues.org	Life Issues Institute, Inc.
http://netnow.micron.net/ rtli	Right To Life of Idaho, Inc. Home Page
http://www.ohiolife.org	Ohio Right To Life
http://www.wrtl.org	Index
http://www.powerweb.net/dcwrl	Dodge County Right to Life Home Page
http://www.nccn.net/ voice	newvoice
http://www.bethany.org	Bethany Christian Services
http://www.prolife.org/LifeAction	Empty title field
http://www.ovnet.com/ voltz/prolife.htm	Pirate Pete's Pro-Life page
http://www.prolife.org/cpcs-online	Empty title field
http://www.care-net.org	Empty title field
http://www.frc.org	FAMILY RESEARCH COUNCIL
http://www.ldi.org	Life Dynamics

Table 3.12: Pro-life cluster of the web data set

are indicative of their content.

The information loss of ROCK and COOLCAT was 72% and 67%, respectively.

Intra-Attribute Value Clustering

We now present results for the application of LIMBO to the problem of intra-attribute value clustering. For this experiment, we use the Bibliographic data set. We are interested in clustering the conferences and journals, as well as the first authors of the papers. We compare LIMBO with STIRR, an algorithm for clustering attribute values.

Following the description of Section 3.5, for the first experiment we set the random variable A' to range over the conferences/journals, while variable $\tilde{A}$ ranges over first and second authors, and the year of publication. There are 1,211 distinct publication venues

(Pro-Choice Cluster) 24 web pages	
URL	**Title**
http://www.gynpages.com	Abortion Clinics OnLine
http://www.prochoice.org	NAF - The Voice of Abortion Providers
http://www.cais.com/agm/main	The Abortion Rights Activist Home Page
http://hamp.hampshire.edu/ clpp/nnaf	National Network of Abortion Funds
http://www.ncap.com	National Coalition of Abortion Providers
http://www.wcla.org	Welcome to the Westchester Coalition for Legal Abortion
http://www.repro-activist.org	Abortion Access Project
http://www.ms4c.org	Medical Students for Choice
http://www.feministcampus.org	Feminist Campus Activism Online: Welcome Center
http://www.naral.org	NARAL: Abortion and Reproductive Rights: Choice For Women
http://www.vote-smart.org	Project Vote Smart
http://www.plannedparenthood.org	Planned Parenthood Federation of America
http://www.rcrc.org	The Religious Coalition for Reproductive Choice
http://www.naralny.org	NARAL/NY
http://www.bodypolitic.org	Body Politic Net News Home
http://www.crlp.org	CRLP - The Center for Reproductive Law and Policy
http://www.prochoiceresource.org	index
http://www.caral.org	CARAL
http://www.protectchoice.org	Pro-Choice Public Education Project
http://www.agi-usa.org	The Alan Guttmacher Institute: Home Page
http://www.ippf.org	International Planned Parenthood Federation (IPPF)
http://www.aclu.org/issues/reproduct/hmrr.html	Empty title field
http://www.nationalcenter.org	The National Center for Public Policy Research
http://wlo.org	Women Leaders Online and Women Organizing for Change

Table 3.13: Pro-choice cluster of the web data set

('Cincinnati' Cluster) 20 web pages	
URL	**Title**
http://cincinnati.com/traffic	Traffic Reports: Cincinnati.Com
http://careerfinder.cincinnati.com	CareerFinder: Cincinnati.Com
http://autofinder.cincinnati.com	Cincinnati Post and Enquirer
http://classifinder.cincinnati.com	Classifieds: Cincinnati.Com
http://homefinder.cincinnati.com	HomeFinder: Cincinnati.Com
http://cincinnati.com/freetime	Cincinnati Entertainment: Cincinnati.Com
http://cincinnati.com/freetime/movies	Movies: Cincinnati.Com
http://cincinnati.com/freetime/dining	Dining: Cincinnati.Com
http://cincinnati.com/freetime/calendars	Calendars: Cincinnati.Com
http://cincinnati.com	Cincinnati.Com
http://cincinnati.com/helpdesk	HelpDesk: Cincinnati.Com
http://cincinnati.com/helpdesk/feedback	HelpDesk: Cincinnati.Com
http://cincinnati.com/helpdesk/circulation/circulation.html	HelpDesk: Cincinnati.Com
http://cincinnati.com/helpdesk/circulation/subscribe.html	HelpDesk: Cincinnati.Com
http://cincinnati.com/search	Search our site: Cincinnati.Com
http://mall.cincinnati.com	Cincinnati.Com Advertiser Index
http://cincinnati.com/advertise	The Daily Fix: Cincinnati.Com
http://cincinnati.com/helpdesk/classifieds	HelpDesk: Cincinnati.Com
http://cincinnati.com/copyright	Cincinnati.Com - Your Key to the City
http://www.gannett.com	Gannett home page

Table 3.14: 'Cincinnati' cluster of the web data set

in the data set; 815 are database venues, and 396 are theory venues.[8] Results for $\phi = 1.0$ and $S = 5MB$ are shown in Table 3.15. LIMBO's results are superior to those of STIRR with respect to all quality measures. The difference is especially pronounced in the P and R measures.

Algorithm	*Leaves*	$IL(\%)$	P	R	E_{min}
LIMBO ($\phi = 1.0$)	47	**94.01**	**0.90**	**0.90**	**0.11**
LIMBO ($S = 5MB$)	16	**94.02**	**0.90**	**0.89**	**0.12**
STIRR	-	98.01	0.56	0.55	0.45

Table 3.15: Bibliography clustering using LIMBO and STIRR

We now turn to the problem of clustering the first authors. Variable A' ranges over the set of 1,416 distinct first authors in the data set, and variable $\tilde{A}$ ranges over the rest of the attributes. We produce two clusters, and we evaluate the results of LIMBO and STIRR based on the distribution of the papers that were written by first authors in each cluster. Figures 3.12 and 3.13 illustrate the clusters produced by LIMBO and STIRR, respectively. The x-axis in both figures represents publishing venues while the y-axis represents first authors. If an author has published a paper in a particular venue, this is represented by a point in each figure. The thick horizontal line separates the clusters of authors, and the thick vertical line distinguishes between theory and database venues. Database venues lie on the left of the line, while theory venues lie on the right of the line.

From these figures, it is apparent that LIMBO yields a better partition of the authors than STIRR. The upper half corresponds to a set of theory researchers with almost no publications in database venues. The bottom half, corresponds to a set of database researchers with very few publications in theory venues. Our clustering is slightly smudged by the authors between index 400 and 450 that appear to have a number of publications in theory. These are drawn into the database cluster due to their co-authors. STIRR,

[8]The data set is pre-classified, so class labels are known.

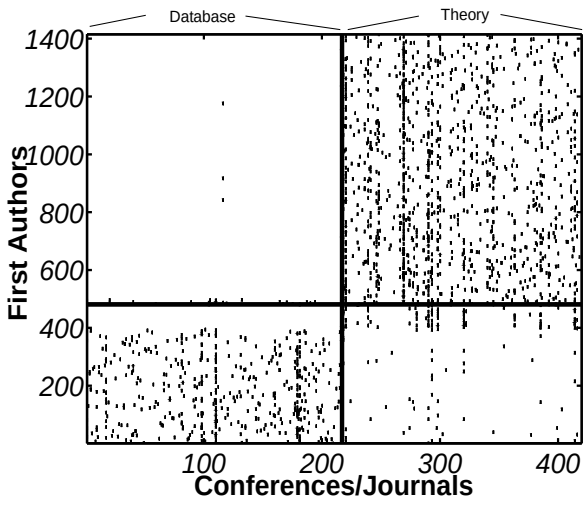


Figure 3.12: LIMBO clusters of first authors

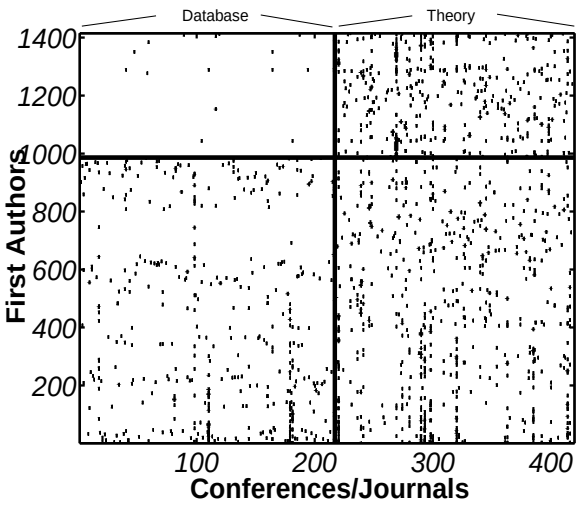


Figure 3.13: STIRR clusters of first authors

on the other hand, creates a well separated theory cluster (upper half), but the second cluster contains authors with publications almost equally distributed between theory and database venues. We supplement the visual evidence of Figures 3.12 and 3.13 by some quantitative evidence: for the first author in a cluster, we calculate the percentage of the venues that are to authors in the same cluster. Hence, for LIMBO this percentage is 77.13% in the database cluster and 96.34% in the theory cluster while for STIRR it is 1.66% in the database cluster and 91.95% in the theory.

3.6.6 Scalability Evaluation

In this section, we study the scalability of LIMBO algorithm, and we investigate how the parameters of LIMBO affect the execution time. First we consider a sample of the DBLP data set. The DBLP data set was created from the XML file found at `http://dblp.uni-trier.de/xml/`. It contains 50,000 tuples, 13 attributes and 57,187 attribute values. More details on the creation of this sample are given in Chapter 4. To study the scalability using DBLP, we created samples of 10,000, 20,000, 30,000 and 40,000 tuples from the initial data set and ran LIMBO$_\phi$ with $\phi = 1.0$. Figure 3.14 shows that the execution time increases linearly as the size of the data set increases.

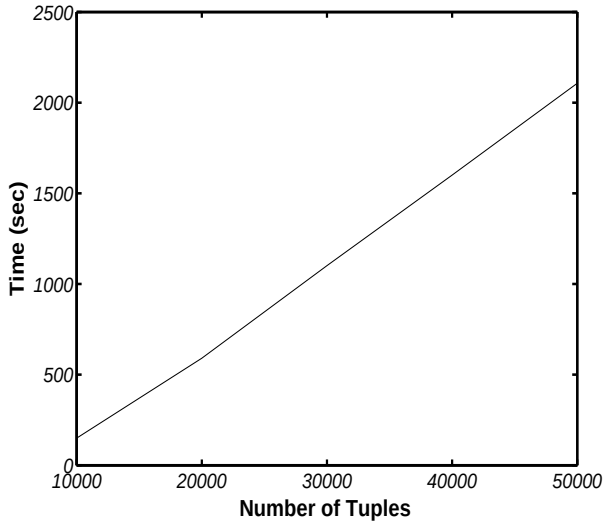


Figure 3.14: DBLP execution times

Next, we study the execution time of both LIMBO$_\phi$ and LIMBO$_S$. We consider four data sets of size $500K$, $1M$, $5M$, and $10M$, each containing 10 clusters and 10 attributes with 20 to 40 values each (there were no missing values). All data sets were created with the synthetic data set generator. The first three data sets are samples of the $10M$ data set.

For LIMBO$_S$, the size and the number of leaf entries of the *DCF* tree at the end of Phase 1 is controlled by the parameter S. For LIMBO$_\phi$, we study Phase 1 in detail. As we vary ϕ, Figure 3.15 demonstrates that the execution time for Phase 1 decreases at a steady rate for values of ϕ up to 1.0. For $1.0 < \phi < 1.5$, execution time drops significantly. This decrease is due to the reduced number of splits and the decrease in the *DCF* tree size. In the same plot, we show some indicative sizes of the tree demonstrating that the vectors that we maintain remain relatively sparse. The average density of the *DCF* tree vectors, *i.e.*, the average fraction of non-zero entries remains between 41% and 87%. Figure 3.16 plots the number of leaf entries as a function of ϕ.[9] We observe that for $\phi > 1.0$ LIMBO produces a manageable *DCF* tree, with fewer than 11,000 leaf entries, leading to fast execution time in Phase 2. Furthermore, in all our experiments

[9]The y-axis of Figure 3.16 has a logarithmic scale.

the height of the tree was never more than 11, and the occupancy of the tree, *i.e.*, the number of occupied entries over the total possible number of entries, was always above 85.7%, indicating that the memory space was well used.

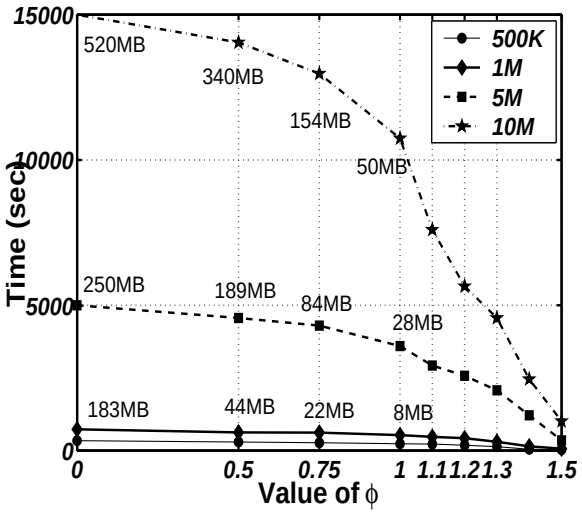


Figure 3.15: Phase 1 execution times

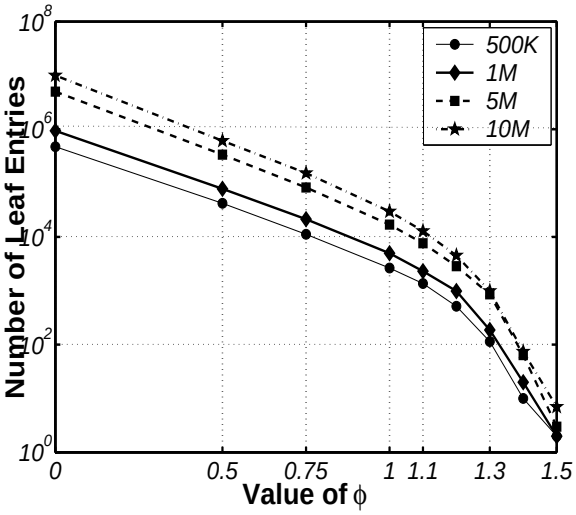


Figure 3.16: Phase 1 leaf entries

Thus, for $1.0 < \phi < 1.5$, we have a *DCF* tree with manageable size, and fast execution time for Phases 1 and 2. For our experiments, we set $\phi = 1.2$ and $\phi = 1.3$. For LIMBO$_S$ we use buffer sizes of $S = 1MB$ and $S = 5MB$. We now study the total execution time of the algorithm for these parameter values. The graph in Figure 3.17 shows the execution time for LIMBO$_\phi$ and LIMBO$_S$ on the data sets we consider. In this figure, we observe that execution time scales in a linear fashion with respect to the size of the data set for both versions of LIMBO. We also observed that the clustering quality remained unaffected for all values of ϕ and S, and it was the same *across* the data sets (except for information loss in the $1M$ data set, which differed by 0.01%). Precision (P) and Recall (R) were 0.999, and the classification error (E_{min}) was 0.0013, indicating that LIMBO can produce clusterings of high quality, even for large data sets.

In our next experiment, we varied the number of attributes, m, in the $5M$ and $10M$ data sets and ran both LIMBO$_\phi$, with $\phi = 1.2$, and LIMBO$_S$, with a buffer size of $5MB$. Figure 3.18 shows the execution time as a function of the number of attributes, for

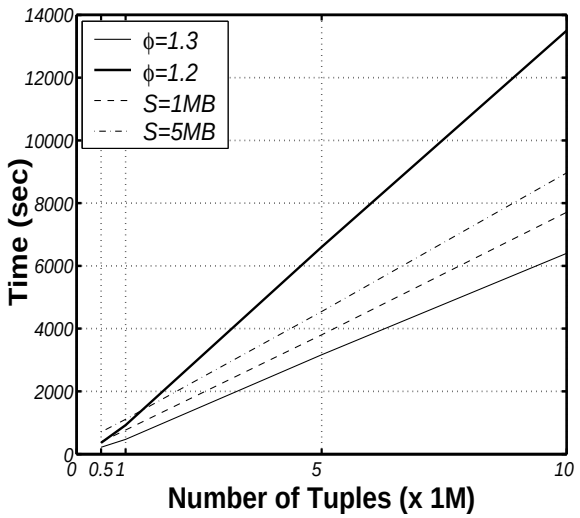


Figure 3.17: Execution time (m=10)

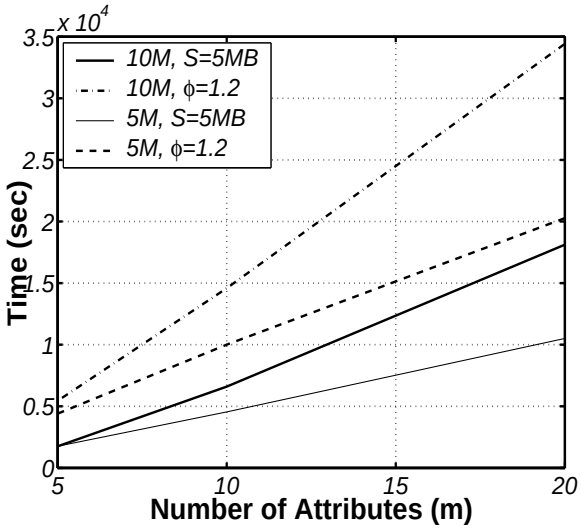


Figure 3.18: Execution time

LIMBO$_{\phi,S}$	*IL(%)*	*P*	*R*	E_{min}	*CU*
$m = 5$	49.12	0.991	0.991	0.0013	2.52
$m = 10$	60.79	0.999	0.999	0.0013	3.87
$m = 20$	52.01	0.997	0.994	0.0015	4.56

Table 3.16: LIMBO$_\phi$ and LIMBO$_S$ quality

different data set sizes. In all cases, execution time increased linearly with the number of attributes. Table 3.16 also presents the quality results for all values of m for both LIMBO algorithms. The quality measures (except for CU) are essentially the same for different sizes of the data set. (The CU measure is not normalized and consequently tends to be larger for data sets with more attributes.)

Finally, we varied the number of clusters from $k = 10$ up to $k = 50$ in the $10M$ data set, for $\phi = 1.2$ and $S = 5MB$. As expected from the analysis of LIMBO in Section 3.4.4, the number of clusters affected the execution time only in Phase 3. Recall from Figure 3.4 in Section 3.6.4 that Phase 3 is a small fraction of the total execution time. Indeed, as we increase k from 10 to 50, we observed just 1.1% increase in the execution time for LIMBO$_\phi$, and just 2.5% for LIMBO$_S$ in the total execution time of all phases of the algorithm.

3.6.7 Information Loss in Higher Dimensions

As a final experiment, we investigate the effect of increasing dimensionality for the LIMBO clustering algorithm. Our experiment is similar in spirit to the one presented by Hinneburg, Aggarwal and Keim [HAK00] for numerical data. We constructed a synthetic data set of 1000 tuples and dimensionality ranging from 1 up to 200 attributes, each one having up to 15 values in its domain.

We calculated all pairwise distances using the expression δI of Equation 3.4. The plot of Figure 3.19 depicts the quantity $\delta I_{max} - \delta I_{min}$ for each dimensionality m. Figure 3.19

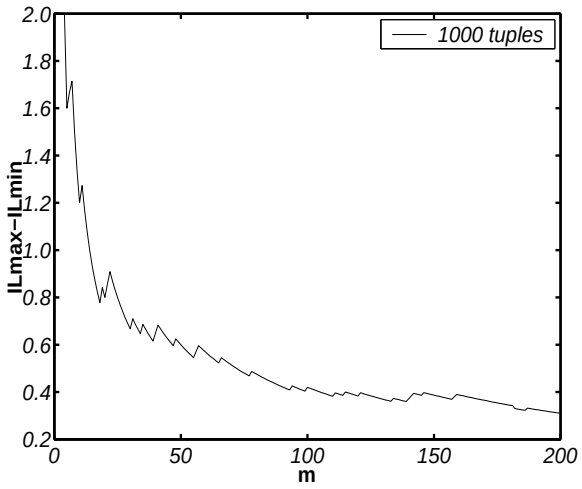


Figure 3.19: Quantity $\delta I_{max} - \delta I_{min}$ for different values of m

shows that the difference between the maximum and minimum values of δI becomes smaller and smaller as dimensionality grows in a nearly monotonic fashion. This result agrees with the one given by Hinneburg et al. for the L_p metrics when $p \geq 3$ [HAK00], indicating that information loss entails the same problems incurred by other metrics used for numerical data when dimensionality grows.

3.7 Estimating k

Automatically identifying an appropriate number of clusters in a data set is an important aspect of the clustering problem. In most cases, there is no single correct answer. In this

section, we discuss some information-theoretic measures that can be used in hierarchical algorithms to identify the most "natural" clustering sizes for a given data set.

The first measure we consider is the rate of change of mutual information, $\delta I(V;C_k) = I(V;C_{k+1}) - I(V;C_k)$. Thinking in reverse, $\delta I(V;C_k)$ captures the amount of information we gain if we break a cluster in two, to move from a clustering of size k to a clustering of size $k+1$. For small values of k, breaking up the clusters results in large information gains. As k increases, the information gain decreases. An appropriate value for k is when the gain $\delta I(V;C_k)$ becomes sufficiently small as k increases. This has also been discussed by Slonim and Tishby [ST99].

The mutual information $I(V;C)$ captures the coherence of the clusters, that is, how similar the tuples within each cluster are. For a good clustering, we require that the elements within the clusters be similar, but also that the elements *across* clusters be dissimilar. We capture the dissimilarity across clusters in the conditional entropy $H(C|V)$. Intuitively, $H(C|V)$ captures the *purity* of the clustering. For a clustering $\mathbf{C}$ with very low $H(C|V)$, for each cluster c in $\mathbf{C}$, there exists a set of attribute values that appear almost exclusively in the tuples of cluster c. The lower the $H(C|V)$, the purer the clusters. The value of $H(C_k|V)$ is minimized for $k = 1$, where $H(C_k|V) = 0$. An appropriate value for k is when $H(C_k|V)$ is sufficiently low, and $k > 1$. Furthermore, the value $\delta H(C_k|V) = H(C_k|V) - H(C_{k-1}|V)$ gives the increase in purity when merging two clusters to move from a clustering of size k to one of size $k-1$. High values of $\delta H(C_k|V)$ mean that the two merged clusters are similar. Low values imply that the two merged clusters are dissimilar. The latter case suggests k as a candidate value for the number of clusters. We do not have an automatic way of determining how small the value of $\delta H(C_k|V)$ should be in order to choose a k value and, hence, we use the minimum value of $\delta H(C_k|V)$ and other values that are close to it.

We propose a combination of these measures, as a way of identifying the appropriate number of clusters. When the number of clusters in not known in advance, we run Phase

2 of the LIMBO algorithm up to $k = 1$, keeping a record of the *DCF* leaf entries for each value of k in the range of interest. We also keep track of the mutual information $I(V; C_k)$, and the conditional entropy $H(C_k|V)$. Observing the behavior of these two measures, provides us with candidate values for k, for which we run Phase 3 of LIMBO, using the corresponding set of leaf entries. We illustrate this procedure using the Web data set. Figure 3.20 presents the plots for $H(C_k|V)$, $I(C_k|V)$ (left), and $\delta H(C_k|V)$, $\delta I(V; C_k)$ (right) for the Web data. From the plots, we can conclude that for $k = 3$, both $\delta I(V; C_k)$ and $H(C_k|V)$ are sufficiently close to zero to mean that the clustering is both pure and informative. This becomes obvious when looking at the clustering of Web data in Figure 3.11. Note that $\delta H(C_2|V)$ is very low, which means that producing a 2-clustering will result in merging two dissimilar clusters.

Similar curves for the Votes data set are given in Figure 3.21 and for the Mushroom data set in Figure 3.22. For the Votes data set, the $\delta I(V; C_k)$ curve clearly suggests $k = 2$ as the right number of clusters. The $H(C_k|V)$ measure is not equally informative since it increases steadily for increasing k. For $k = 7$, $\delta H(C_k|V)$ takes a value close to zero. When the AIB algorithm moves from 7 to 6 clusters, it merges two highly dissimilar clusters. This suggests $k = 7$ as a candidate value for the number of clusters. This value was also examined in the evaluation of the COOLCAT algorithm [BCL02a].

For the mushroom data set, the $\delta I(V; C_k)$ curve does not give a clear indication of what the right number of clusters is. The $H(C_k|V)$ and $\delta H(C_k|V)$ curves demonstrate that we have very low improvement in purity when we move from 3 to 2 clusters. In fact, after careful examination of the corresponding curves for the individual attributes, we observed that for some attributes, $\delta H(C_k|V_i)$ is close, or equal to zero for $k = 3$. Therefore, the two clusters that are merged when moving from a 3-clustering to a 2-clustering are completely separated with respect to these attributes. This suggest $k = 3$ as candidate for the number of clusters. Looking at the clusterings we observed that generating 3 clusters results in breaking up the cluster that contained mostly poisonous mushrooms

in the 2-clustering. Interestingly, the class of poisonous mushrooms is the concatenation of two classes: "poisonous" and "not recommended" mushrooms. Therefore, $k = 3$ is a valid candidate for the clustering size. Another possible number of clusters suggested by the curves is $k = 8$.

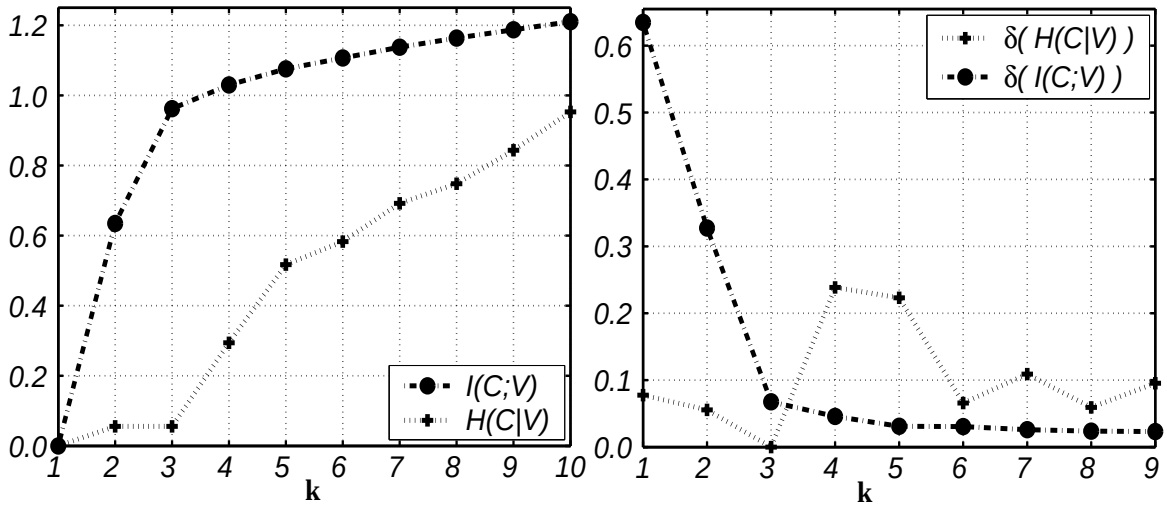


Figure 3.20: $I(C;V)$, $H(C|V)$, and $\delta I(C;V)$, $\delta H(C|V)$ for web data as functions of the number of clusters

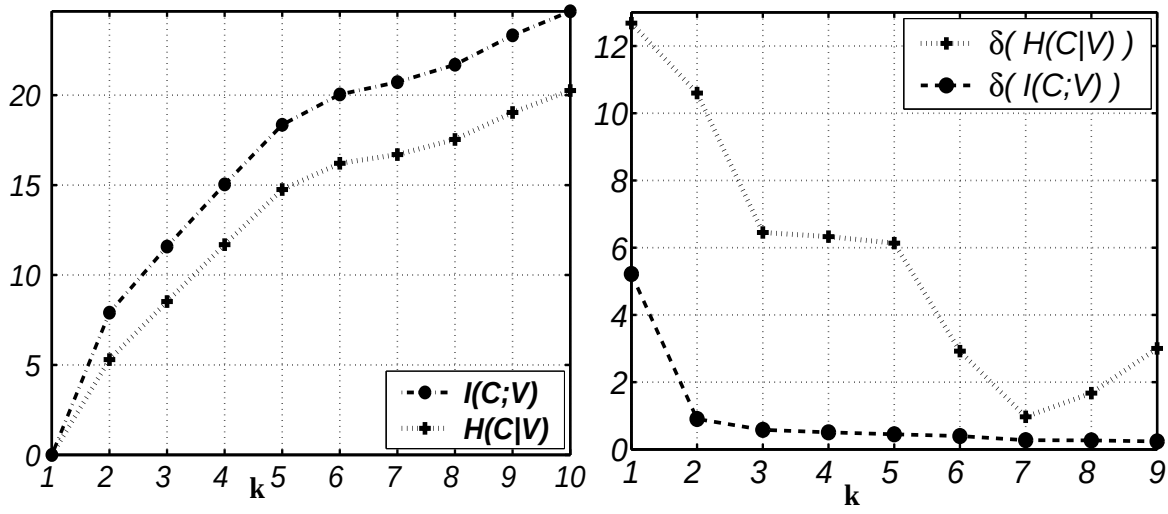


Figure 3.21: $I(C;V)$, $H(C|V)$, and $\delta I(C;V)$, $\delta H(C|V)$ for Votes as functions of the number of clusters

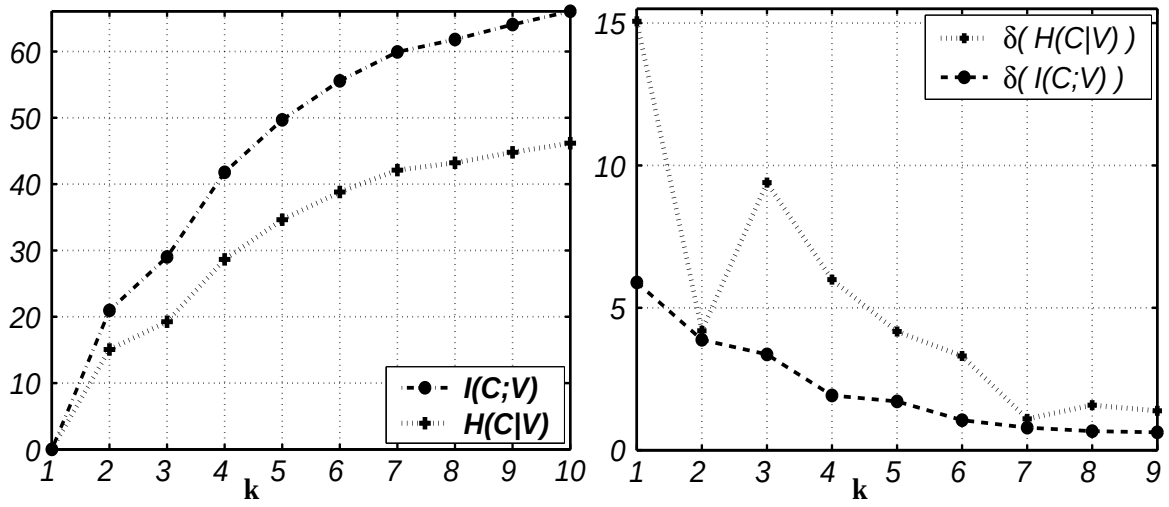


Figure 3.22: $I(C;V)$, $H(C|V)$, and $\delta I(C;V)$, $\delta H(C|V)$ for Mushroom as functions of the number of clusters

3.8 Conclusions

In this chapter, we presented LIMBO, a scalable hierarchical clustering algorithm and evaluated its effectiveness in trading off either quality for time or quality for space to achieve compact, yet accurate, models for small and large categorical data sets. We have shown LIMBO to have advantages over other information theoretic based clustering algorithms, including AIB (in terms of scalability) and COOLCAT (in terms of clustering quality and parameter stability). We have also shown advantages in quality over other scalable and non-scalable algorithms designed to cluster either categorical data objects or values. LIMBO builds a model in one pass over the data in a fixed amount of memory, while keeping information loss in the model close to the minimum possible. In addition, to the best of our knowledge, LIMBO is the only scalable categorical clustering algorithm that is hierarchical. Using its compact summary model, LIMBO efficiently builds clusterings for a large range (typically hundreds) of values of k. Furthermore, we are also able to produce statistics that let us directly compare clusterings and select appropriate values for the number of clusters, k.

Chapter 4

LIMBO-Based Techniques for Structure Discovery

Database design has been characterized as a process of arriving at a design that maximizes the information content of each piece of data (or equivalently, one that minimizes redundancy). Information content (or redundancy) is measured with respect to a prescribed model for the data, a model that is often expressed as a set of constraints. In this chapter, we consider the problem of doing database redesign in an environment where the prescribed model is unknown or incomplete. Specifically, we consider the problem of finding structural clues in an instance of data, an instance which may contain errors, missing values, and duplicate records. We propose a set of information-theoretic techniques for finding structural summaries that are useful in characterizing the information content of the data, and ultimately useful in database design. We provide algorithms for creating these summaries over large, categorical data sets us. We study the use of these summaries in one specific physical design task, that of ranking functional dependencies based on their data redundancy. We show how our ranking can be used by a database designer tool to acquire hints about the structure of the data and potentially derive meaningful vertical decompositions of a relation. We present an evaluation of the approach on real data sets. The results of this chapter have been published in a series of papers [AFF+02, MA03, AM03, AMT04].

4.1 Introduction

The growth of distributed databases has led to larger and more complex databases, the structure and semantics of which are increasingly difficult to understand. In heterogeneous applications, data may be exchanged or integrated. This integration may introduce anomalies such as duplicate records, missing values, or erroneous values. In addition, the lack of documentation or the unavailability of the original designers can make the task of understanding the structure and semantics of databases a very difficult one.

No matter how carefully a database was designed initially, there is no guarantee that the data semantics are preserved as it evolves over time. It is usually assumed that the schema and constraints are trustworthy, which means that they provide an accurate model of the time-invariant properties of the data. However, in both legacy databases and integrated data this may not be a valid assumption. Hence, we may need to redesign a database to find a model (a schema and constraints) that better fit the current data.

In this chapter, we consider the problem of mining a database instance for structural clues that may help a designer in identifying a better database design. Our work is in the spirit of several recent approaches that propose tools to help an analyst in the process of understanding and cleaning a database [DJMS02, RH01]. However, while these approaches focus on providing data summaries that help in the process of integration [RH01] or querying [DJMS02], we focus on data summaries that reveal information about the database design and its information content.

To approach this problem, it is important to understand what makes a database design good. Database design has been characterized as a process of arriving at a design that maximizes the information content of each piece of data (or equivalently, one that minimizes redundancy) [AL03]. In other cases, a good database design is the one that assures better performance given a specific workload [PA04]. In our work, information content (or redundancy) is measured with respect to a prescribed model for the data, a model that is often expressed as a set of constraints or dependencies. In their recent work,

Arenas and Libkin presented information-theoretic measures for comparing database designs [AL03]. Given a schema and a set of constraints, the information content of a design is precisely characterized. Their approach has the benefit that it permits two designs for the same database to be compared directly.

However, to characterize the information content, it was necessary to have a prescribed model, that is, a fixed set of constraints. Consider the following example.

	Director	*Actor*	*Genre*
t_1	Coppola	Grant	Thriller
t_2	Coppola	DeNiro	Thriller
t_3	Scorsese	Grant	Thriller

Figure 4.1: Examples of duplication and redundancy

Clearly, there is considerable duplication of values in this instance. However, what we consider to be redundant will depend on the constraints expressed on the schema. If the functional dependency $Actor \rightarrow Genre$ holds, then the value `Thriller` in tuple t_3 is redundant. That is, if we remove this value from the third tuple, it could be inferred from the information in the first tuple. However, the value `Thriller` in the second tuple is not redundant. If we lose this value, we will not know the Genre of this tuple. So while the value `Thriller` is duplicated in t_2, it is not redundant. But if we change the constraints and, instead of $Actor \rightarrow Genre$, we have the dependency $Director \rightarrow Genre$, then the situation is reversed. The value `Thriller` is redundant in t_2, but the one in t_3 is not.

Understanding redundancy is at the heart of database design. In this work, we consider how to summarize the potential redundancy in an instance. At their core, our techniques find duplicate or similar values. However, unlike techniques based on counting (for example, frequent item-set mining [AIS93]), we use information-theoretic clustering techniques to identify and summarize the information content of a database in the presence of duplicate or similar values.

Our contributions are the following.

• We propose a set of information-theoretic techniques that use clustering to discover duplicate records, sets of correlated attribute values, and groups of attributes that share such values.

• We provide a set of efficient algorithms that can be used to identify duplication in large, categorical data sets. Our algorithms are based on LIMBO and generate compact summaries of the data that can be used by an analyst to identify errors or to understand the information content of a data set.

• We present several applications of our summaries to the data quality problems of duplicate elimination and the identification of anomalous values. We also present applications to the database design problems of horizontally partitioning an integrated or overloaded relation and to ranking functional dependencies based on their information content. For the latter application, we show how our techniques can be used in combination with a dependency miner to help understand and use discovered dependencies.

Horizontal and Vertical Decompositions

The problem of the decomposition of database relations has been studied extensively in the past. It has been argued that the decomposition of a relation [De 87]:

1. reduces the redundancy of the relation (same information does not get repeated);
2. speeds-up query answering (the stored relations are of smaller size);
3. improves the understanding of the data.

Decomposition, however, is mostly done in a *vertical* fashion, which means that given a set of functional dependencies the decomposed relations satisfy some semantic constraints. Vertical decomposition highly depends on the presence of functional dependencies to separate independent pieces of information.

De Bra and Paredaens [DP83a, DP83b] introduced a normal form based on the notion of *horizontal decomposition*. This decomposition serves as exception handlers, in that it

separates the tuples of a relation so that a number of them satisfy a set of functional dependencies and the rest of the tuples do not. They argue that it is more of a technical need to use horizontal decompositions since information (tuples) that do not satisfy certain constraints can be stored separately or hidden from users. This decomposition produces smaller relations that are potentially easier to understand. However, a horizontal decomposition cannot be performed without the existence of functional dependencies.

Horizontal decompositions have been used in different work. For example, Chan et al. [CDF+82] use horizontal decompositions within a system that supports a more semantically rich database language called ADAPLEX. In order to achieve better inter and intra entity type information grouping, they store homogeneous sets of tuples from database relations into separate tables. Their main objective is to facilitate the querying of these tables.

In real life, it may be the case that functional dependencies do not hold in a database over time as new tuples are inserted or relations are integrated. In such situations, vertical and horizontal decompositions may not be the suitable approach in order to separate the different types of information present in a relation. In this thesis, we use a clustering approach in order to identify and separate duplicate information in a data set and potentially re-design (decompose) it. We complement the work of decomposing a relation based on functional dependencies in that our technique allows for a ranking of these dependencies.

4.2 Related Work

Our work is motivated by two independent lines of work. The first on data quality browsers, such as Potter's Wheel [RH01] and Bellman [DJMS02]. The second on the information-theoretic foundation of data design [AL03, DR00].

Data browsers aim to help an analyst understand, query, or transform data. In addition to sophisticated visualization techniques (and adaptive query or transformation

processing techniques [RH01]), they employ a host of statistical summaries to permit real-time browsing. In our work, we consider the generation of summaries that could be used in a data browser for database (re)design. These summaries complement the summaries used in Bellman, where the focus is on identifying co-occurrence of values across different relations (to identify join paths and correspondences between attributes of different relations). Instead, we present a set of techniques for identifying and summarizing various forms of duplication within a relation.

Arenas and Libkin provide information-theoretic measures for comparing database designs [AL03]. Given a schema and a set of constraints, the information content of a design is precisely characterized. The measures used are computationally infeasible and they rely on having a clean instance that conforms to a specified set of constraints. Our techniques are based on an efficient information-theoretic clustering approach. Because we are using unsupervised learning, we are able to create informative summaries even without an accurate model (set of constraints) for the data.

Constraint or dependency mining is a related field of study where the goal is to find all dependencies that hold on a given instance of the data (that is, all dependencies that are not invalidated by the instance). Such approaches include the discovery of functional [SF93, HKPT99, WGR01] and multi-valued [SF00] dependencies. Our work complements this work by providing a means of characterizing the redundancy captured by a dependency. We have found that constraint miners reveal hundreds or thousands of potential (approximate) dependencies when they are run on large, real data sets. Our work helps a data analyst understand and quickly identify interesting dependencies within these large sets. For our work, interesting are the dependencies which, if used in a potential decomposition remove a great deal of redundancy from a relation.

The importance of automated data design and redesign tools has been recognized in reports on the state of database research. Yet, the advances that have been made in this area are largely limited to physical design tools that help tailor a design to best

meet the performance characteristics of a workload [RD02, ACN01]. Cluster Analysis has been used for vertical partitioning [HS75, NCWD84, NR89], however these techniques partition the attributes of a relation based only on their usage in workload queries and not the data. On the other hand, fractured mirrors [RD02] store different versions of the same database, which are combined during query optimization. This technique is also based on the usage of the attributes in queries.

Finally, our work complements work on duplicate elimination [HS95, SE00, SB02]. We propose a technique to identify duplicates based on the information content of tuples. Our approach does not consider how to identify or use distance functions for measuring the difference between values (which is the main focus of related work in this area). Value differences are due to the representation of the same entity in different ways. For example, the values `F.F. Coppola` and `F. Coppola` correspond to the same director. Differences also appear due to data entry errors. An example would be the values `Scorsese` and `Scorcese`.

4.3 Clustering and Duplication

Schemas, like structured query languages that use them, treat data values largely as uninterpreted objects. This property has been called *genericity* [AHV95] and is closely tied to data independence, the concept that schemas should provide an abstraction of a data set that is independent of the internal representation of the data. That is, the choice of a specific data value (perhaps "Pat" or "Patricia") has no inherent semantics and no influence on the schema used to structure director values. The semantics captured by a schema are independent of such choices. For query languages, genericity is usually formalized by saying that a query must commute with all possible permutations of data values (where the permutations may be restricted to preserve a distinguished set of constants) [AHV95].

This property becomes important when one considers clustering algorithms. Cluster-

ing assumes that there is some well-defined notion of similarity between data objects. Many clustering methodologies employ similarity functions that depend on the semantics of the data values. For example, if the values are numbers, a Euclidean distance function may be used to define similarity. However, we do not want to impose any application-specific semantics on the data values within a database.

Again, we assume a model where a set $\mathbf{T}$ of n tuples is defined on m attributes $(A_1, A_2, \ldots, A_m)$. The domain of attribute A_i is the set $\mathbf{V}_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,d_i}\}$. Any tuple $t \in \mathbf{T}$ takes exactly one value from the set $\mathbf{V}_i$ for the i^{th} attribute. Moreover, a functional dependency between attribute sets $W \subseteq \mathbf{A}$ and $Z \subseteq \mathbf{A}$, denoted by $W \rightarrow Z$, holds if, whenever the tuples in $\mathbf{T}$ agree on the W values, they also agree on their corresponding Z values.

To apply information-theoretic techniques, we will treat relations as distributions. For each tuple $t \in \mathbf{T}$ containing a value $v \in \mathbf{V}$, we will set the probability that v appears in tuple t to $1/m$ as described by the formalism of Section 3.3. If we go back to the example of Figure 4.1, we consider a representation of its tuples as given in Figure 4.2. Each row corresponds to a tuple and has a non-zero value for each one of the values it stores in the original relation and sums up to one.

	Coppola	*Scorsese*	*Grant*	*DeNiro*	*Thriller*
t_1	1/3	0	1/3	0	1/3
t_2	1/3	0	0	1/3	1/3
t_3	0	1/3	1/3	0	1/3

Figure 4.2: Example of tuple representation for the relation of Figure 4.1

Similarly, the representation we consider for the values of the same data set is given in Figure 4.3. Each row in the table of Figure 4.3 characterizes the occurrence of values in tuples, and for each value there is a non-zero entry for the tuples in which it appears. As in the tuple representation, each row sums up to one.

Using such representations, we consider a number of clustering approaches for identifying duplication in tuples, values, and attributes. All of our techniques are based on

	t_1	t_2	t_3
Coppola	1/2	1/2	0
Scorsese	0	0	1
Grant	1/2	0	1/2
DeNiro	0	1	0
Thriller	1/3	1/3	1/3

Figure 4.3: Example of value representation for the relation of Figure 4.1

LIMBO.

4.4 Duplication Summaries

In this section, we present a suite of structure discovery tasks that can be performed using LIMBO, the information-theoretic clustering algorithm introduced in the previous chapter. We will see how, from information about tuples, we can build summaries about the attribute values and, from this, summaries about the attributes of a relation.

The input to our problem here is the set of tuples $\mathbf{T}$ and the set $\mathbf{V} = \mathbf{V}_1 \cup \ldots \cup \mathbf{V}_m$, which denotes the set of all possible values. Let $d = d_1 + d_2 + \ldots + d_m$ denote the size of set $\mathbf{V}$. We shall denote by V and T the random variables that range over sets $\mathbf{V}$ and $\mathbf{T}$, respectively. In our approach, we want to control the information loss during merges of objects that correspond to tuples and attribute values, and, therefore, we shall use the LIMBO$_\phi$ version of the LIMBO algorithm. LIMBO$_S$ can also be used, without direct control of information loss in the merges.

4.4.1 Tuple Clustering

Again, we seek to find clusters of tuples that preserve the information about the values they contain as much as possible. We use the formalism introduced in Section 3.3.1 for the clustering of the values represented by random variable T in order to preserve as much information as possible about the values represented by V. When clustering tuples, the parameter ϕ is denoted by ϕ_T, which controls the loss of information when newly inserted tuples are merged with existing sub-clusters during Phase 1 of LIMBO. We shall use ϕ

values for attribute value clustering, denoted by ϕ_V and for attribute grouping, denoted by ϕ_A. We tried different values for these parameters and present results with the ones that produced the best results.

Duplicate Tuples

Duplicate tuples can be introduced through data integration. Different sources may store information about the same entity. The values stored may differ slightly so when integration is performed, two entries may be created for the same entity. As an example, we can imagine a situation where employee information is integrated from different sources and employee numbers are represented differently in the sources. After integration, it is natural to expect tuples referring to the same employee that differ only in their employee number (or perhaps some other attributes if one database is more up-to-date than another). To identify duplicate or nearly duplicate tuples we proceed as follows.

1. Apply Phase 1 of LIMBO to construct tuples summaries using a chosen value of ϕ_T.

2. Eliminate any *DCF* leaf entries that represent only a single tuple (*i.e.,* with $p(c^*) = 1/n$).

3. Apply Phase 3 of LIMBO to associate each tuple of the initial data set with the remaining summary to which it is closest, where proximity is measured by the information loss in merging the two.

Step 1 of the above procedure determines the accuracy of the representation of groups of tuples in the summaries at the leaf level of the *DCF*-tree and applies Phase 1 of LIMBO. If $\phi_T = 0.0$, we merge only identical tuples and the representation is exact. As we increase ϕ_T, the summaries permit larger differences in the duplicate values in a group. Step 2 eliminates summaries of single tuples, while Step 3 associates tuples with summaries that represent *groups of tuples* (more than one tuple). It is then natural to explore the sets of tuples associated with each single summary to find candidate duplicates or near duplicates.

Horizontal Partitioning

Horizontal partitioning is an application of tuple clustering approach given in the previous chapter. Horizontal partitioning can be useful on relations that have been overloaded with different types of data [DJ03]. For example, Table 4.1, originally designed to store movies that have a release date (released movies), may have been reused to store future releases. Released movies have a value on their *Release Data* attribute while future releases do not and, thus, when integrated the latter ones get filled with NULL values. Performing horizontal partitioning on this table, our goal is to separate the information in the first four tuples (released movies) from the last three tuples (future releases).

Movie	*Director*	*Actor*	*Genre*	*Release Date*
Godfather	Scorsese	DeNiro	Crime	1974
Good Fellas	Coppola	DeNiro	Crime	1998
Vertigo	Hitchcock	Stewart	Thriller	1958
N by NW	Hitchcock	Grant	NULL	1959
Alexander	Luhrman	NULL	NULL	NULL
Water	Mehta	NULL	NULL	NULL
Life Without Me	Coizet	NULL	NULL	NULL

Table 4.1: A relation with two heterogeneous clusters

In horizontal partitioning, we are seeking to separate different types of tuples based on the similarities in their attribute values. Specifically, we try to identify whether there is a natural clustering that separates out tuples having different characteristics.

4.4.2 Attribute Value Clustering

As in tuple clustering, we can build clusters of attribute values that maintain as much information about the tuples in which they appear as possible. The parameter ϕ in this case will be denoted by ϕ_V, and small values of it allow for the identification of almost perfectly co-occurring groups of attribute values. Such groups of values appear within the same attributes of subsets of the tuples. Our intention is to cluster values that mainly come from different attributes and cluster them together in order to characterize their

duplication. For example, the values "DeNiro" and "Crime" in Table 4.1 will be clustered together, since they appear within the same attributes of two different tuples.

A useful connection between tuple and attribute value clustering is drawn when the number of tuples is large. We can use a $\phi_T > 0.0$ value to cluster the tuples, and then attribute values can be expressed over the (much smaller) set of tuple clusters instead of individual tuples. Attribute value clustering can then be performed as described above. This technique is referred to as *Double Clustering* [EYS01].

Contrary to tuple clustering, our goal here is to cluster the values represented in random variable V so that they retain information about the tuples in T in which they appear.

Presuming that the representation of our data set is the same as a market basket data set (Section 3.3.2), we represent our data as a $d \times n$ matrix N, where $N[v,t] = 1$ if attribute value $v \in \mathbf{V}$ appears in tuple $t \in \mathbf{T}$ (for each value $c(v) = 1$), and zero otherwise. Note that the vector of a value v contains $d_v \leq d_i$ 1's, $1 \leq d_i \leq n$. For a value $v \in \mathbf{V}$, we define:

$$p(v) = 1/d \tag{4.1}$$

$$p(t|v) = \begin{cases} 1/d_v & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

Intuitively, we consider each value v to be equi-probable and normalize matrix N so that the v^{th} row holds the conditional probability distribution $p(T|v)$. Consider the example relation depicted in Figure 4.4. Figure 4.5 (left) shows the normalized matrix N for the relation in Figure 4.4. Together with N, we define a second matrix, O, which keeps track of the frequency of the attribute values in their corresponding attributes. The matrix O is defined as a $d \times m$ matrix were $O[v,A] = d_v$ if value v appears d_v times in attribute V. Intuitively, each entry of matrix $O[v,A]$ stores the support of a value v in attribute A of the relation. For our example, matrix O is given on the right-hand-side of Figure 4.5. Note that for a value v: $\sum_j O[v,A_j] = d_v$ and for an attribute A:

A	B	C
a	1	p
a	1	r
w	2	x
z	2	x
y	2	x

Figure 4.4: Duplication in attribute pairs (A, B) and (B, C)

N	t_1	t_2	t_3	t_4	t_5	$p(a)$
$\{a\}$	1/2	1/2	0	0	0	1/9
$\{w\}$	0	0	1	0	0	1/9
$\{z\}$	0	0	0	1	0	1/9
$\{y\}$	0	0	0	0	1	1/9
$\{1\}$	1/2	1/2	0	0	0	1/9
$\{2\}$	0	0	1/3	1/3	1/3	1/9
$\{p\}$	1	0	0	0	0	1/9
$\{r\}$	0	1	0	0	0	1/9
$\{x\}$	0	0	1/3	1/3	1/3	1/9

O	A	B	C
$\{a\}$	2	0	0
$\{w\}$	1	0	0
$\{z\}$	1	0	0
$\{y\}$	1	0	0
$\{1\}$	0	2	0
$\{2\}$	0	3	0
$\{p\}$	0	0	1
$\{r\}$	0	0	1
$\{x\}$	0	0	3

Figure 4.5: Matrix N (left) and O (right) for the table in Figure 4.4

$\sum_l O[v_l, A] = n$.

Given matrix N, we can use mutual information $I(V;T)$ to cluster the attribute values in $\mathbf{V}$ into clusters $\mathbf{C}_V$ such that the loss of information $I(C_V;T) - I(V;T)$ is minimized at each step. Intuitively, we seek sets of attribute values in $\mathbf{C}_V$ that retain information about the tuples in which they appear. Such sets of values may contain values that appear in the same attributes of more than one tuple. We will show how to characterize the sets of attribute values in the clusters of $\mathbf{C}_V$ in the next subsection.

Set $\mathbf{V}$ may include a large number of values, making the AIB algorithm computationally infeasible. Thus, we perform the clustering using LIMBO, where the *DCF*s are extended in order to include information from matrix O. If $c*$ is the summary of a particular cluster of values, we define the *Attribute Distributional Cluster Features (ADCF)* as a triplet:

$$ADCF(c^*) = \Big(p(c^*), p(T|c^*), O(c*)\Big)$$

where $p(c^*)$ and $p(T|c^*)$ are defined as in Section 3.4 and $O(c^*) = \sum_{c \in c^*} O(c)$, *i.e.*, $O(c^*)$

is the sum of the rows of matrix O that correspond to the sub-clusters c^* represents. Essentially, $O(c^*)$ stores the support of the values in cluster c^* in each of the attributes.

As in tuple clustering, we use LIMBO to identify duplicate or near duplicate values in the data set.

1. Apply Phase 1 of LIMBO to construct tuple summaries using a chosen value of ϕ_V.

2. Eliminate any $ADCF$ leaf entries that represent a single value ($i.e$, with $p(c^*) = 1/d$).

3. Apply Phase 3 of LIMBO to associate each attribute values of the initial data set with the remaining summary to which it is closest, where proximity is measured by the information loss in merging the two.

By augmenting DCFs in this way, we are able to perform value clustering on the value matrix N together with O at the same time. Hence, we are able to find sets of attribute values (of arbitrary size) together with their counts (that is, the number of tuples in which they appear) using one execution of our clustering algorithm. Specifically, we require only three passes over the data set. One pass to construct the matrices N and O, one pass to perform Phase 1 of LIMBO and a final pass to perform Phase 3.

In our example, in executing LIMBO while allowing no loss of information during merges ($\phi_V = 0.0$), attribute values `a` and `1` are clustered, as are values `x` and `2`. These values have perfect co-occurrence in the tuples of the original relation. The clustering of values with $\phi_V = 0.0$ is depicted on the left-hand-side of Figure 4.6. The resulting matrix

N	t_1	t_2	t_3	t_4	t_5	$p(a)$
$\{a, 1\}$	1/2	1/2	0	0	0	2/9
$\{w\}$	0	0	1	0	0	1/9
$\{z\}$	0	0	0	1	0	1/9
$\{y\}$	0	0	0	0	1	1/9
$\{2, x\}$	0	0	1/3	1/3	1/3	2/9
$\{p\}$	1	0	0	0	0	1/9
$\{r\}$	0	1	0	0	0	1/9

O	A	B	C
$\{a, 1\}$	2	2	0
$\{w\}$	1	0	0
$\{z\}$	1	0	0
$\{y\}$	1	0	0
$\{2, x\}$	0	3	3
$\{p\}$	0	0	1
$\{r\}$	0	0	1

Figure 4.6: Clustered matrix N (left) and O (right)

O of our example is depicted on the right-hand-side of Figure 4.6. For the cluster $\{a, 1\}$ of values the corresponding row of O becomes $(2, 2, 0)$, which means that the values of this cluster appear two times in attribute A and two times in attribute B. In general, O stores the cumulative counts of the occurrences of clusters of values inside the attributes of a relation. Both N and O contain important information, and the next sub-section describes their use in finding duplicate and non-duplicate groups of values.

It is critical to emphasize the role of parameter ϕ_V. As already explained, ϕ_V is used to control the loss of information during the merges of newly inserted values with existing sub-clusters in the leaf entries of the tree. Besides this, it plays a significant role in identifying "almost" perfect co-occurrences of values. To illustrate this consider the relation in Figure 4.7. This relation is the same as the one in Figure 4.4 except for value `x` in the second tuple.

A	B	C
a	1	p
a	1	x
w	2	x
z	2	x
y	2	x

Figure 4.7: No perfect correlation of attribute B and C due to value `x` in the second tuple

Constructing matrices N and O can be done as explained before. However, when trying to cluster with $\phi_V = 0.0$, our method does not place values `x` and `2` together since they do not exhibit perfect correlation any more. This may be a result of an erroneous placement of `x` in the second tuple, or a difference in the representation among data sources that were integrated in this table. Moreover, the functional dependency $C \rightarrow B$ that holds in the relation of Figure 4.4 now becomes *approximate* in that it does not hold in all the tuples. To capture such anomalies, we perform clustering with $\phi_V > 0.0$, which allows for some small loss of information when merging $ADCF$ leaves in the $ADCF$-tree. Matrices N and O for $\phi_V = 0.1$ are depicted in Figure 4.8. An interesting distinction

N	t_1	t_2	t_3	t_4	t_5	$p(a)$
$\{a,1\}$	1/2	1/2	0	0	0	2/8
$\{w\}$	0	0	1	0	0	1/8
$\{z\}$	0	0	0	1	0	1/8
$\{y\}$	0	0	0	0	1	1/8
$\{2,x\}$	0	1/8	7/24	7/24	7/24	2/8
$\{p\}$	1	0	0	0	0	1/8

O	A	B	C
$\{a,1\}$	2	2	0
$\{w\}$	1	0	0
$\{z\}$	1	0	0
$\{y\}$	1	0	0
$\{2,x\}$	0	3	4
$\{p\}$	0	0	1

Figure 4.8: Matrix N (left) and O (right), ($\phi_V = 0.1$)

between the notions of approximation in our method and in the methods that have appeared in the literature should be made. These methods [SF93, HKPT99] characterize approximate duplication based upon the removal of whole tuples from the relation. This, however, might lead in the loss of useful information. If, for example, a particular set of values are not perfectly duplicated due to a single tuple, the removal of this tuple could result in the elimination of values with high importance in the attributes that do not contain duplicate values. In contrast, our method is valued-based. This means that no tuple elimination is required, and the control of the loss of information in the model through parameter ϕ_V is enough to determine the approximate nature of duplication. Hence, using as input the example of Figure 4.8, our method with $\phi_V = 0.1$ determines that value `x` in the second tuple affects the perfect duplication of pairs $\{2, x\}$ less than any other value, and thus the value with which it co-occurs (value `1`) is merged with the cluster $\{2, x\}$.

As already mentioned, tuple clustering and attribute value clustering can be combined when the size of the input is large. In a situation where the number of tuples is very large, we can define the mutual information $I(T;V)$, and cluster the tuples in $\mathbf{T}$ into the clusters represented by C_T. Usually $|C_T| \ll |T|$ and we can use it to define $I(V;C_T)$ and speed up the clustering of attribute values.

4.4.3 Grouping Attributes

Knowing the similarities among the values in the data set, we are able to express attributes over the duplicate groups of values they contain and derive information about duplication-based attribute proximity. These similarities can be used in the ranking of functional dependencies found to hold in a particular instance. Again, the information loss in merging attributes can be controlled through a ϕ value denoted by ϕ_A. Typically, the number of attributes m is much smaller than the number of tuples n, so we use small values of ϕ_A.

The rows of the compressed matrix N represent groups of values as conditional probability distributions on the tuples in which they appear either exactly for $\phi_V = 0.0$, or approximately for $\phi_V > 0.0$. From these rows and the corresponding rows of the compressed matrix O, we can infer which groups of attribute values appear as duplicates in the set of attributes. We are looking for clusters of values that make their appearance in more than one tuple and more than one attribute. More formally, we define the following.

• C_V^D denotes the set of duplicate groups of attribute values. A set of values c^D belongs to C_V^D if and only if there are at least two tuples t_i, t_j for which both $p(t_i|c^D) \neq 0$ and $p(t_j|c^D) \neq 0$, and at the same time there are at least two attributes A_x and A_y such that both $O[c^D, A_x] \neq 0$ and $O[c^D, A_y] \neq 0$. Intuitively, C_V^D contains these sets of values that appear within more than two attributes and more than two tuples in the data set, and, thus naturally co-occur in it.

• C_V^{ND} denotes the set of non-duplicate groups of attribute values. This set is comprised of all values in $C_V - C_V^D$. These are sets that appear just once in the tuples of the data set.

In our example, it is easy to see from Figure 4.6 that $C_V^D = \Big\{\{a, 1\}, \{2, x\}\Big\}$ and $C_V^{ND} = \Big\{\{w\}, \{z\}, \{y\}, \{p\}, \{r\}\Big\}$. Now, from these groups, C_V^D contains "interesting" information in that it may lead to a grouping of the attributes such that attributes in

the same group contain more duplicate values than attributes in different groups.

Formally, if $\mathbf{A}$ is the set of attributes and A the random variable that takes its values from this set, we only express the members of $\mathbf{A}$ over C_V^D (since we are only interested in duplicated values) through the information kept in matrix O. We denote these members of $\mathbf{A}$ by $\mathbf{A}^D$ and the random variable that takes values from this set by A^D. Then, we cluster the attributes in $\mathbf{A}^D$ into a clustering C_A^D, such that the information $I(C_A^D; C_V^D)$ remains maximum. Intuitively, we can cluster the attributes such that the information about the duplicate groups of attribute values that exist in them, remains as high as possible. Using C_V^D instead of the whole set C_V, we focus on the set of attributes that will potentially offer higher duplication, while at the same time we reduce the size of the input for this task.

Since set $\mathbf{A}$ usually includes a tractable number of attributes, we can use LIMBO with $\phi_A = 0.0$ and produce a full clustering of the attributes (*i.e.*, produce all clusterings up to $k = 1$). By performing an agglomerative clustering (in Phase 2) over the attributes, at each step we cluster together a pair that creates a group with the maximum possible duplication. For our example, Figure 4.9 depicts the table of attributes expressed over the set C_V^D as explained above, and using the information in matrix O (the rows that correspond to the members of C_V^D). Note that we have the same matrix both for $\phi_V = 0$ and $\phi_V = 0.1$, and that in this example $\mathbf{A} = \mathbf{A}^D$. We name this matrix F. Normalizing rows of F so that they sum up to one, we can proceed with our algorithm and cluster the attributes. All the merges performed are depicted in the *dendrogram* given in Figure 4.10. The horizontal axis of the dendrogram shows the information loss incurred at each merging point. Initially, all attributes form singleton clusters. The first merge with the least amount of information (0.13) loss occurs between attributes B and C and after that, attribute A is merged with the previous cluster (with an information loss of 0.52).

Looking back at our example of Figure 4.4, we can see that attributes B and C contain more tuples with the duplicate group of values $\{2, x\}$ than A and B do with respect to

F	$\{a, 1\}$	$\{2, x\}$
$\{A\}$	2	0
$\{B\}$	2	3
$\{C\}$	0	4

Figure 4.9: Matrix F before normalization

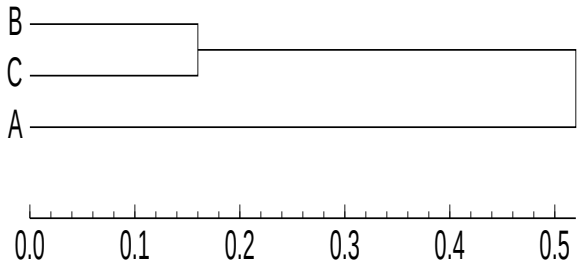


Figure 4.10: Attribute cluster dendrogram

the group of values $\{a, 1\}$.

4.5 Ranking Dependencies

In this section, we show how to use our attribute clustering to rank a set of functional dependencies holding on an instance.

A desirable goal of structure discovery is to derive clues with respect to a potential decomposition of an integrated data set. As we pointed out, duplication is not the same as redundancy. To understand the relationship, we turn to work on mining for constraints (dependencies). There have been several approaches to the discovery of functional [SF93, HKPT99, WGR01] and multivalued [SF00] dependencies. However, none of the approaches presents a characterization of the resulting dependencies. In this section, we present a procedure that performs a ranking of the functional dependencies found to hold on an instance, based on the redundancy they represent in the initial relation.

A good indication of the amount of duplication of the values in C_V^D in a cluster of attributes C_A is the entropy $H(C_V^D|C_A)$. The entropy captures how skewed the distribution of C_V^D in C_A is. Skewed distributions are expected to have higher duplication. The lower the entropy, the more skewed the distribution. The following proposition shows that the clusters of attributes formed early in the clustering process have smaller entropy than those formed later in the clustering process.

Proposition 1 *Given sets of attributes C_{A1}, C_{A2} and C_{A3}, if the information loss of merging C_{A1} and C_{A2} into C_1 is smaller than the information loss of merging C_{A1} and C_{A3} into C_2, then the duplication in C_1 is larger than the duplication in C_2.*

Proof 2 *If the clustering before the merge is C, we have that $\delta I(C_{A1}, C_{A2}) < \delta I(C_{A1}, C_{A3})$ and*

$$\begin{aligned} I(C; C_V^D) - I(C_1; C_V^D) &< I(C; C_V^D) - I(C_2; C_V^D) \\ I(C_1; C_V^D) &> I(C_2; C_V^D) \\ H(C_V^D) - H(C_V^D|C_1) &> H(C_V^D) - H(C_V^D|C_2) \\ H(C_V^D|C_1) &< H(C_V^D|C_2) \end{aligned}$$

The last inequality states that given C_1, the duplicate groups of values appear more times than in C_2, which implies that duplication is higher in C_1 than in C_2.

The above result justifies the observation that if we scan the dendrogram of a full clustering of the attributes of $\mathbf{A}^D$, the sub-clusters that get merged first are the ones with the highest duplication. Upon the creation of the dendrogram, if we have a set of functional dependencies FD, we can rank them according to how much of the duplication in the initial relation is removed after their use in the decomposition. Given a functional dependency that contains attributes with high duplication, we may then say that the duplicate values in these attributes are redundant. The more redundancy the set of attributes of a functional dependency remove from the initial relation, the more interesting it is for our purposes. Knowing all values of information loss across all merges (in a sequence Q) of attribute sub-clusters, we can proceed with algorithm FD-RANK given in Figure 4.11 to rank the functional dependencies in FD.

Intuitively, if we have the sequence of all merges Q of the attributes in matrix F (the set C_A^D) with their corresponding information losses, we first initialize the rank of each dependency to be the maximum information loss realized during the full clustering procedure (Step 1.a). For the set of values that participate in a functional dependency (Step 1.b), we update its rank with the highest information loss of a merge where all attributes are merged and this information loss is below a percentage, specified by ψ, of the maximum information loss (Step 1.c). At this point we can break ties among the functional dependencies that acquire the same ranking based on the number of participating attributes; we rank the ones with more attributes higher than others. Step 2 collapses two functional dependencies with the same left-hand-side and ranks, into a

FD-RANK
Input : Set FD, merge sequence Q, threshold $0 \leq \psi \leq 1$
Output : Set FD_{ranked}

1. For each $fd \in FD : X \rightarrow A$ (A single attribute):
   (1.a) $rank(fd) = max(Q)$ (max inf. loss in Q);
   (1.b) $S = X \cup A$;
   (1.c) $rank(fd) = IL(G)$, the inf. loss at merge G
   where all attributes in S participate and
   $IL(G) <= \psi \cdot max(Q)$;
2. If $fd_1 : X \rightarrow A_1$ and $fd_2 : X \rightarrow A_2$
   with $rank(fd_1) = rank(fd_2)$, set $fd_{12} : X \rightarrow A_1A_2$
3. Order the set FD in ascending order of
   $rank$ to produce FD_{ranked}

Figure 4.11: The FD-RANK algorithm

single functional dependency. Finally, Step 3 orders set FD in ascending order of their corresponding ranks. We currently have no guideline for choosing ψ and in the future, we are planning to investigate this issue as well as the influence of ψ in our results. This parameter is used here as a cut-off point.

In the example of Section 4.4.2, the maximum information loss realized in the attribute clustering is approximately 0.52. This is the initial rank the dependencies $A \rightarrow B$ and $C \rightarrow B$ acquire. With a $\psi = 0.5$, we only update the rank of functional dependency $C \rightarrow B$ with an information loss of the merge of attributes B and C, since this is the only merge lower than 0.26 ($\psi \cdot 0.52$). At this point, $C \rightarrow B$ is the highest ranked functional dependency since it contains attributes with the highest redundancy in it. Indeed, looking back at the initial relation, if we use the dependency $C \rightarrow B$ to decompose the relation into relations S1=(B,C) and S2=(A,C), the reduction of tuples, and thus the redundancy reduction, is higher than using $A \rightarrow B$ to decompose into relations S1'=(A,B) and S2'=(A,C).

Finally, if f is the number of functional dependencies in FD, finding the greatest common merge which is smaller than ψ times the maximum information loss realized,

can be done in $\mathcal{O}(f \cdot m \cdot (m-1))$ time, since we can have at most m attributes participating in a dependency and should traverse at most $(m-1)$ merges to find the desired common merge of all of them. The final step of ordering the dependencies according to their ranks has a worst-case complexity of $\mathcal{O}(f \cdot \log f)$. Thus, the total complexity is $\mathcal{O}(f \cdot m \cdot (m-1) + f \cdot \log f)$. If $f \gg m^2$, which is often the case in practice, the previous complexity is dominated by the number of dependencies (first term).

4.6 Experiments

We ran a set of experiments to determine the effectiveness of the techniques discussed in this chapter in the structure discovery process. We report on the results found in each data set we used and provide evidence of the usefulness of our approach.

Data Sets. In our experiments we used the following data sets.

• *DB2 Sample Database:* This is a data set we constructed out of the small database that is pre-installed with IBM DB2.[1] We built a single relation after joining three of the tables in this database, namely tables `EMPLOYEE`, `DEPARTMENT` and `PROJECT`. The schema of the tables together with their key (the attributes separated by a line at the top of each box) and foreign key (arrows) constraints are depicted in Figure 4.12. The relational algebra expression we used to produce the single relation was (we use the initials of each relation):

$$R = \Big((E \bowtie_{WorkDepNo=DepNo} D) \bowtie_{DepNo=DepNo} P \Big)$$

Relation R contains 90 tuples with 19 attributes and 255 attribute values. We used this instance to illustrate the types of "errors" we are able to discover using our information-theoretic methods

[1] `http://www-3.ibm.com/software/data/db2/udb/`

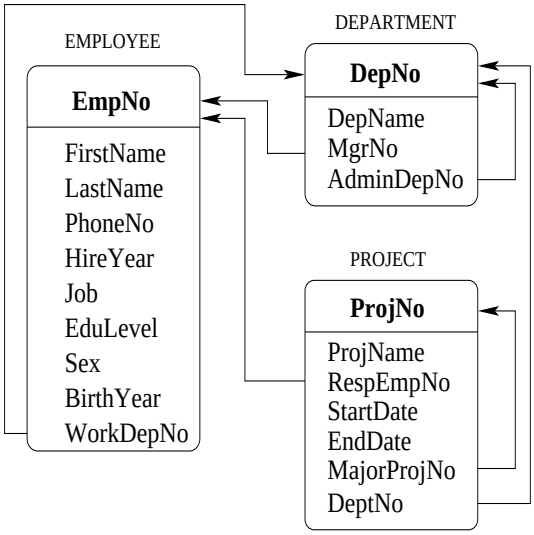


Figure 4.12: DB2 Sample

• *DBLP Database:* This data set was created from the XML document found at `http://dblp.uni-trier.de/xml/`. This document stores information about different types of computer science publications. In order to integrate the information into a single relation, we chose to use the Clio schema mapping tool that permits the creation of queries to transform the information stored in XML format into relations [PVM+02]. We specified a target schema (the schema over which the tuples in the relation are defined) containing the 13 attributes depicted in Figure 4.13. We specified correspondences between the source XML schema and the attributes in Figure 4.13. The queries given by the mapping tool were used to create a relation that contained $50,000$ tuples and $57,187$ attribute values. Each tuple contains information about a single author and, therefore, if a particular publication involved more than one author, the mapping created one additional tuple for each one of them. Moreover, the highly heterogeneous information in the source XML document (information regarding conference, journal publications, etc.) introduced a large number of `NULL` values in the tuples of the relation. We used this highly heterogeneous relation to demonstrate the strength of our approaches in suggesting a better structure than the target relation we initially specified.

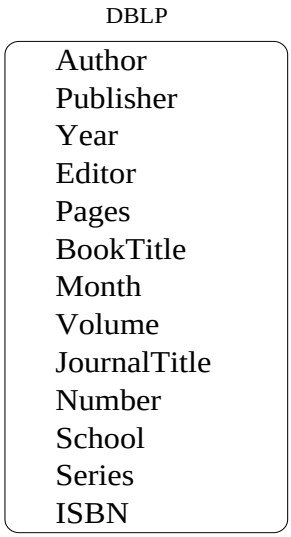


Figure 4.13: DBLP

Functional Dependency Discovery. Our goal is not to rediscover functional dependencies, but rather provide a ranking of those that hold on a database instance in order to help a database designer acquire hints about the data and potentially re-organize it. For the purposes of our study, we used FDEP [SF93] as the method to discover functional dependencies. Other methods could also be used.

A maximal invalid dependency is a dependency that does not hold in the data set and the addition of any attribute to the dependency makes it valid. On the other hand, a minimal valid dependency is a dependency that holds in the data set and any addition of an attribute makes it invalid. FDEP first computes all maximal invalid dependencies by pairwise comparison of all tuples and then it computes the minimal valid dependencies from the maximal set of invalid dependencies. The algorithm proposed by Savnik and Flach [SF93] performs the second step using a depth-first search approach. During this approach the set of maximal invalid dependencies are used to test whether a functional dependency holds. It also prunes the search space, which consists of the set of all candidate functional dependencies.

After computing the functional dependencies using FDEP, we computed the minimum cover using Maier's algorithm [Mai80].

Duplication Measures. In order to evaluate the amount of redundancy removed from the initial data set, and thus the ranking of functional dependencies, we used two measures. These measures are the *Relative Attribute Duplication ($\mathcal{RAD}$)* and *Relative Tuple Reduction ($\mathcal{RTR}$)* defined below.

• *Relative Attribute Duplication:* Given a relation, R, of n tuples, a set $C_A = \{A_1, A_2, \ldots, A_j\}$ with $j \geq 1$ of attributes, and the projection $t_{C_A} = \pi_{C_A}(R)$ of tuples assuming bag semantics on the attributes of C_A, we define

$$\mathcal{RAD}(C_A) = \left(1 - \frac{H(t_{C_A}|C_A)}{\log_2(n)}\right)$$

Intuitively, $\mathcal{RAD}$ captures the fraction of bits we save in the representation of C_A due to repetition of values. However, the above definition does not clearly distinguish between the duplication of differently sized relations. For example, consider two relations on a single attribute with the first one having the same value in its three tuples and the second one the same value in its two tuples. The above definition will suggest that both relations have $\mathcal{RAD}$ equal to one, missing the fact that the first relation contains more duplication than the second (since it contains more tuples). To overcome this we introduce the next measure.

• *Relative Tuple Reduction:* Given a relation, R, of n tuples, a set $C_A = \{A_1, A_2, \ldots, A_j\}$ with $j \geq 1$ of attributes, and $t_{C_A} = \pi_{C_A}(R)$ the set of n' tuples (assuming bag semantics) projected on the set C_A, we define

$$\mathcal{RTR}(C_A) = \left(1 - \frac{n'}{n}\right)$$

Intuitively, $\mathcal{RTR}$ quantifies the fractional reduction in the number of tuples that we get if we project the tuples of a relation over C_A.

Overall $\mathcal{RAD}$ and $\mathcal{RTR}$ offer two different measures of the extent to which values are repeated in the relation. A closer look at $\mathcal{RAD}$ reveals that this measure is more *width-sensitive*. From the definition of conditional entropy, the numerator of the fraction

in $\mathcal{RAD}$ can be considered as the weighted entropy of the tuples in a particular set of attributes, where the weights are taken as the probability of this set of attributes. On the other hand, $\mathcal{RTR}$ is more *size-sensitive* in that it can quantify the duplication within different set of tuples taken over the same set of attributes.

4.6.1 Small Scale Experiments

In this phase of our experiments, we performed a collection of tests in the DB2 sample data set to see how effective our techniques are in finding exact or near duplicate tuples and values in the data. This data set was used since it is a "clean" one and errors can be introduced to illustrate the potential of our methods.

Application of Tuple Clustering

Exact Tuple Duplicates. Our method can identify exact duplicates introduced in the data set in any order. These duplicates are found by setting $\phi_T = 0.0$.

Typographic, Notational and Schema Discrepancies. Such errors may be introduced when the same information is recorded differently in several data sources and then integrated into a single database. For example, this might be the case where the employee numbers are stored following different schemes (typographical or notational errors). On the other hand, this might also be the case where unknown values during integration are filled with `NULL` values in order to satisfy the common integrated schema (schema discrepancies). An example of the latter case was given in Table 4.1.

To identify this type of error, we introduced additional tuples into the data set where some of the values in their attributes differ from the values in the corresponding attributes of their matching tuples in the data set. First, we set the value of ϕ_T to 0.1 and performed a study with various numbers of erroneous tuples and attribute values. Then, we fixed the number of erroneous tuples that we inserted to five and performed a study where the ϕ_T and the number of erroneous attribute values varied. We changed the same number of attribute values in each of the inserted tuples every time. The results of both experiments

are given in Table 4.2. From this table, the strength of our method in determining groups

#Err. Tuples=5		#Err. Tuples=20	
Errors	**Found**	**Errors**	**Found**
1	5	1	20
2	5	2	20
4	5	4	19
6	4	6	17
10	4	10	15

$\phi_T = 0.2$		$\phi_T = 0.3$	
Errors	**Found**	**Errors**	**Found**
1	5	1	4
2	5	2	3
4	4	4	3
6	3	6	2
10	3	10	2

Table 4.2: DB2 Sample results of erroneous tuples, for $\phi_T = 0.1$ (left) and #err. tuples=5 (right)

of tuples that do not differ a lot is evident. The table on the left indicated that, for a small number of "dirty" tuples inserted, our method fails to discover some approximate duplicates only when the number of attribute values on which they differ is more than half the number of attributes in the schema. The same table, shows that, as the number of these duplicates increases, the performance of the method deteriorates gracefully. The table on the right, where the number of inserted tuples is five, shows that, as the accuracy of the chosen model in the summaries decreases (larger ϕ_T values), the identification of approximate duplicates becomes more difficult, since in these cases more tuples are associated with the clusters of the constructed summaries.

Duplicates found using tuple clustering are presented to the user, and an inspection of the tuple clusters reveals whether these are interesting ones, *i.e.*, duplicates corresponding to the same physical entities represented by the tuples. Phase 3 was effective in that it never failed to identify the correct correspondences of tuples with their summaries in the leaf entries of the tree.

Application of Attribute Value Clustering

In this section, we present experiments on attribute value clustering.

Value correlations. Using $\phi_T = 0.0$ (no clustering of tuples is performed), and $\phi_V = 0.0$, we first looked for perfect correlations among the values, that is, groups of attributes

values that naturally appear exclusively together in the tuples. Our clustering method successfully discovered such groups of values that make up the set C_V^D.

Although with $\phi_V = 0.0$ we do not get anything more than the perfectly correlated sets of values, we believe that this information is critical in that it aligns our method with that of Frequent Itemset counting [AIS93]. However, with higher values of ϕ_V, we are able to discover potential entry errors.

Value Errors. In this experiment, we introduced errors similar to the ones in tuple clustering, however our goal here is to locate the values that are "responsible" for them. For better results, we may combine the results of tuple and attribute value clustering. We performed experiments for the same set of tuples that were artificially inserted when we performed tuple clustering, where we counted the number of correct placements of "dirty" values in the clusters of attribute values that appear almost exclusively together in the tuples. That is, we wanted to see if a dirty value was correctly clustered with the value it replaced. Results of these experiments are given in Table 4.3. As in tuple

#Err. Tuples=5		#Err. Tuples=20	
Errors	**Found**	**Errors**	**Found**
1	1	1	1
2	2	2	2
4	4	4	4
6	5	6	5
10	9	10	7

$\phi = 0.2$		$\phi = 0.3$	
Errors	**Found**	**Errors**	**Found**
1	1	1	1
2	2	2	1
4	2	4	2
6	4	6	2
10	7	10	6

Table 4.3: DB2 Sample results of erroneous values, for $\phi_T = 0.1$ (left) and #Err. Tuples=10 (right)

clustering, our method performs well even if the number of inserted tuples is quite large relative to the size of the initial data set.

Attribute Grouping

Having information about duplicate values in C_V^D, we built matrix F. The dendrogram that was produced for $\phi_V = 0.0$ and $\phi_A = 0.0$ is depicted in Figure 4.14. Again, the horizontal axis represents information loss. In this data set, the maximum information

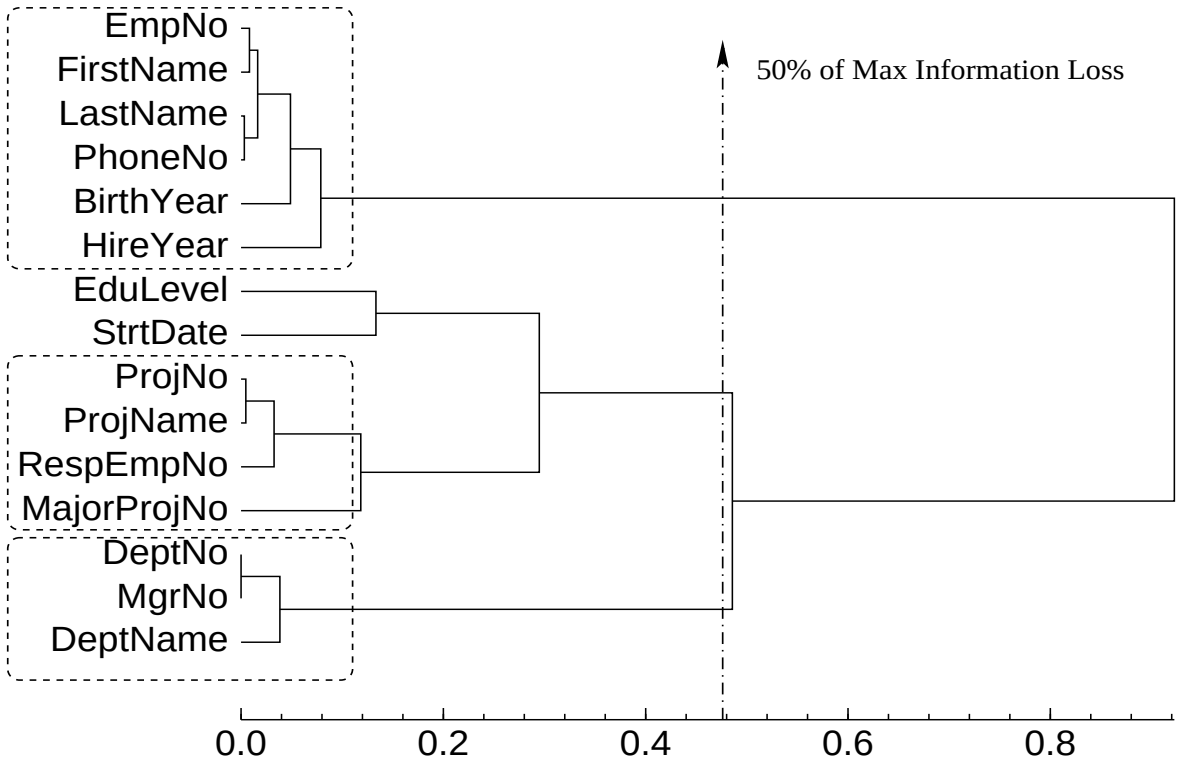


Figure 4.14: DB2 Sample attribute clusters

loss realized was 0.922. As indicated by the boxes, our attribute grouping has separated the attributes of the initial sub-schemas to a large extent, with the only exception being attributes `EduLevel` and `StartDate`. From the dendrogram, we could also identify that pairs (`EmpNo, FirstName`), (`LastName, PhoneNo`), (`ProjNo, ProjName`) and (`DeptNo, MgrNo`) exhibit the highest redundancy in the data set, a result that agrees with the data instance as well as our intuition.

In addition to the previous experiment, we increased the value of ϕ_V to 0.1 and 0.2 respectively. The set of attributes in C_A^D remained the same for $\phi_V = 0.1$, while attribute `ProjEndDate` was included when $\phi_V = 0.2$. However, there was large information loss when this attribute was merged with other attributes. In both experiments, the sequence of the merges remained the same, indicating that our attribute grouping is stable in the presence of errors (a constant number of errors are initially present, but more of them are discovered with higher ϕ_V values).

Ranking of Functional Dependencies

Having the sequence of merged attributes, we used Fd-rank to identify which functional dependencies, if used in a decomposition, would help in the removal of high amounts of redundancy from the initial data set. FDEP initially discovered 106 functional dependencies, and the minimum cover consisted of nine dependencies given below in order of their increasing rank using Fd-rank with $\psi = 0.5$.

1. [DeptNo]→[DeptName,DeptMgrNo]
2. [DeptName]→[DeptMgrNo]
3. [EmpNo]→[EmpBirthYear,EmpFName,EmpLName,EmpPhoneNo,EmpHireYear]
4. [ProjNo]→[ProjName,ProjRespEmpNo,ProjStDate,ProjMajorProjNo]
5. [ProjRespEmpNo]→[ProjStDate,ProjMajorProjNo]
6. [ProjRespEmpNo,EmpEdLevel,EmpBirthYear]→[EmpNo,EmpFName,EmpLName, EmpPhoneNo,DeptName,EmpHireYear,DeptMgrNo,Dep tNo]
7. [EmpNo]→[DeptNo,DeptName,DeptMgrNo,EmpEdLevel]
8. [ProjNo]→[DeptNo,DeptName,DeptMgrNo]
9. [ProjRespEmpNo]→[DeptNo,DeptName,DeptMgrNo]

Finally, Table 4.4 shows the $\mathcal{RAD}$ and $\mathcal{RTR}$ values for the nine functional dependencies of the minimum cover, if their attributes are used to project the tuples in the initial relation. Table 4.4 shows that our ranking identifies dependencies with high redundancy (high $\mathcal{RAD}$ and $\mathcal{RTR}$ values). Decompositions of the initial relation according to the ordered list of dependencies would lead to the removal of considerable amounts of redundancy. This is attributed to the fact that correlations of the corresponding attributes are high. However, the attribute value clusters in C_V^D have lower support in the initial

FD	$\mathcal{RAD}$	$\mathcal{RTR}$
1.	0.947	0.922
2.	0.965	0.922
3.	0.924	0.878
4.	0.872	0.800
5.	0.871	0.800
6.	0.577	0.523
7.	0.456	0.345
8.	0.435	0.311
9.	0.234	0.300

Table 4.4: $\mathcal{RAD}$ and $\mathcal{RTR}$ values for DB2 Sample

data set. This fact is also visible in the dendrogram, where the attributes of `Department` have a lower information loss than those of `Employee` and `Project`, and according to Proposition 1, they can remove more redundancy.

4.6.2 Large Scale Experiments

For these experiments, we used the larger DBLP data set. We performed a different series of experiments which, in large integrated relations, could be part of a structure discovery task.

The DBLP data set contains integrated information. The relation contains tuples describing computer science publications that appeared as part of conference proceedings, journals, theses, etc. As we already argued, integrated information may have anomalies due to the discrepancies between the source and the target schemas. More specifically, most conference publications have their `Journal` attributes filled with `NULL` values. Some conference publications, though, appear as part of a `Series` publication, (like SIGMOD publications in the SIGMOD Record journal).

Before performing horizontal partitioning, we performed attribute grouping in order to identify which attributes would be most useful in such a partitioning. We used $\phi_T = 0.5$, which reduced the number of tuples to 1361 and then performed the attribute grouping with $\phi_A = 0.0$. The result of this grouping is depicted in Figure 4.15. From the dendrogram, we observe that a number of attributes demonstrate an almost perfect correlation in that the information loss in merging them is very small. These are the

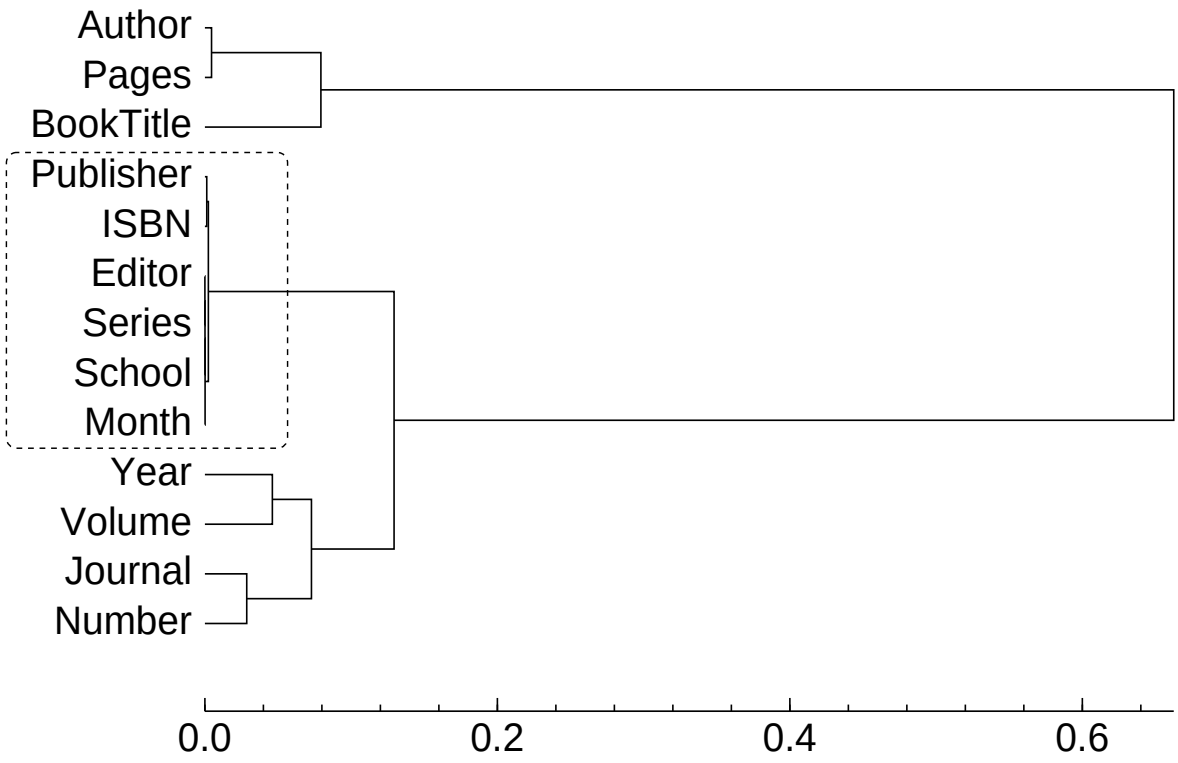


Figure 4.15: DBLP attribute clusters

attributes (dashed box) with zero or almost zero information loss, indicating an almost one-to-one correspondence among their values. This is true since the value that prevails in this set of attributes is the `NULL` value. A manual inspection of the data set revealed that the set of attributes {`Publisher`, `ISBN`, `Editor`, `Series`, `School`, `Month`} contains over 98% of `NULL` values, an anomaly introduced during the transformation of XML data into the integrated schema.

Having a set of attributes with limited non-missing information, the horizontal partitioning produced unexpected results. We performed all three Phases of our algorithm to cluster the tuples into three groups. The result contained a huge cluster of $49,998$ tuples and two clusters of one tuple in each. However, this result was very informative. All the tuples in the relation are almost duplicates on many attributes and `NULL` values forced them into the same summary. Hence, our first observation here is that the six attributes with `NULL` values can be set aside in the analysis without considerable loss of information

about the tuples. At the same time, if our goal is the definition of a possible schema for the relation, the existence of a huge percentage of `NULL` values suggests that these attributes contain very large amounts of duplication and should be stored separately, before any horizontal partitioning.

After the previous observation, we projected the initial relation onto the attribute set {`Author`, `Pages`, `BookTitle`, `Year`, `Volume`, `Journal`, `Number`}. Then we performed a horizontal partitioning of the tuples. Using our heuristic for choosing k as described in Section 3.7, we determined that $k = 3$ was a natural grouping for this data. The loss of initial information after Phase 3 was only 9.45%, indicating that the clusters are highly informative. The characteristics of the three clusters are given in Table 4.5. We now consider each cluster separately, and we report results of our attribute grouping and functional dependency ranking.

Cluster	*Tuples*	*AttributeValues*
c_1	35892	43478
c_2	13979	21167
c_3	129	326

Table 4.5: Horizontal partitions

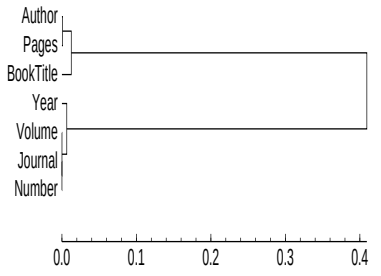


Figure 4.16: Cluster 1

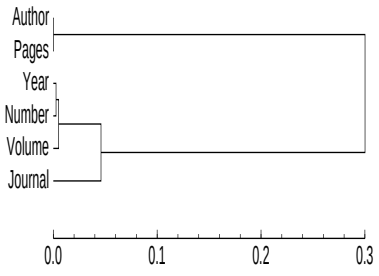


Figure 4.17: Cluster 2

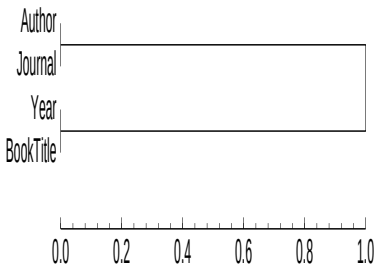


Figure 4.18: Cluster 3

Cluster 1: This horizontal partition contains all Conference publications where the `BookTitle` attribute was a non-`NULL` value in every tuple. Using $\phi_T = 0.5$ and $\phi_V = 1.0$ (we use a large ϕ_V value due to the large number of attribute values), we performed the grouping of attributes and the result is given in Figure 4.16. This dendrogram of

the attributes in C_A^D reveals that there is zero distance among the `Volume`, `Journal` and `Number` attributes. Indeed, these are attributes that exclusively contained `NULL` values in this cluster. In addition, we found almost zero distance between attributes `Author` and `Pages`, which happens due to an almost one-to-one mapping between their values in this sample of the DBLP data set (author tuples had unique `Pages` values in this particular cluster). Finally, `BookTitle` is closer to the `Author` and `Pages` attributes as conference titles are correlated with the authors. Having the sequence of attribute merges, we used FDEP to find functional dependencies that hold in c_1 and Fd-rank with $\psi = 0.5$ to rank them. There were 12 dependencies and the minimum cover contained 11. All dependencies together with their $\mathcal{RAD}$ and $\mathcal{RTR}$ values are given in Table 4.6. Looking at the top-two dependencies along with the $\mathcal{RAD}$ and $\mathcal{RTR}$ values of their sets of attributes we see that maximum redundancy reduction can be achieved if we use them in a decomposition. Although these two dependencies that were ranked highest did not contain conference attributes, they are highly informative in that the `NULL` values in the attributes they cover indicate removal of more redundancy. A database designer may not choose to decompose the relation according to these functional dependencies but rather exclude them from this cluster. Hence, the ranking of dependencies may give hints regarding the structure of the data.

FD	$\mathcal{RAD}$	$\mathcal{RTR}$	$Rank$
[Volume]→[Journal]	1.0	1.0	1
[Number]→[Journal]	1.0	1.0	2
[BookTitle]→[Journal]	0.3661	0.9349	3
[Year,BookTitle,Volume]→[Journal,Number]	0.2201	0.8419	4
[Pages,Number]→[Volume,Journal]	0.1406	0.6237	5
[Pages]→[Journal]	0.1407	0.6238	6
[Author,Number]→[Volume,Journal]	0.0430	0.2313	7
[Pages,BookTitle]→[Volume,Journal]	0.0191	0.0986	8
[Author,BookTitle,Volume]→[Journal,Number]	0.0052	0.0374	9
[Author,Year]→[Journal,Number]	0.0089	0.0472	10
[Author,Pages]→[Journal,Number]	0.0082	0.0145	11

Table 4.6: Ranked dependencies for c_1.

Cluster 2: The second horizontal partition contains journal publications where the `Journal`, `Volume` and `Number` attributes had non-`NULL` values. Again, using $\phi_T = 0.5$ and $\phi_V = 1.0$ (given the number of the attribute values) the dendrogram produced is depicted in Figure 4.17. The first observation is that all attributes in C_A^D are generally characteristics of journal publications. We see that correlations appear among `Journal`, `Volume`, `Number` and `Year`, which is something natural to assume in such publications. For example, the SIGMOD Record journal appears once every quarter and the values of the `Number` attribute are 1 through 4. Finally, using the sequence of merges of the attributes in C_A^D, we ranked the functional dependencies holding in this partition. FDEP discovered a set of functional dependencies whose minimum cover contained 3 dependencies. Using FD-RANK with $\psi = 0.5$, we ranked the dependencies, which are given in Table 4.7 together with the $\mathcal{RAD}$ and $\mathcal{RTR}$ values of the sets of attributes they involve. Note that the first two dependencies had the same rank. However, the first dependency has more attributes and is ranked at the top. Using the first dependency in a decomposition, it separates `Year` from the other attributes, something that is counter-intuitive. Decomposing over the first dependency will remove redundancy but a designer must decide if the dependency is time-invariant and semantically meaningful. Notice, also, the big difference in the $\mathcal{RAD}$ and $\mathcal{RTR}$ values of the third dependency compared to the first two ones. This can be attributed to the almost distinct values that appear in the attributes of the Author and Pages.

FD	$\mathcal{RAD}$	$\mathcal{RTR}$	$Rank$
[Author,Volume,Journal,Number]→[Year]	0.754	0.881	1
[Author,Year,Volume]→[Journal]	0.858	0.982	1
[Author,Year,Pages]→[Volume]	0.0001	0.001	2

Table 4.7: Ranked dependencies for c_2.

Cluster 3: The last horizontal partition was much smaller than the previous two, and contained miscellaneous publications, such as technical reports, theses, etc. It also contained a very small number of conference and journal publications that were written by

a single author. The dendrogram produced based on the C_A^D set is given in Figure 4.18. Given the nature and the size of the cluster, the attribute associations are rather random, and we did not find any functional dependencies in the partition, a fact suggesting that this relation does not have internal structure.

The initial horizontal partitioning we used was beneficial. While the initial relation defined on all 13 attributes contained hundreds of functional dependencies, mainly due to the attributes containing `NULL` values, the clusters we produced had a small number of dependencies (or none) defined on their attributes. This makes the understanding of their structure an easier task. As to the decomposition of the individual partitions using the ranked dependencies, we should be careful in that higher ranked dependencies cannot always be used in a decomposition. The mining algorithms used to derive these dependencies often produce a large number of them, some of which are accidental (attributes either contain unique values or a large number of NULL values that co-occur with others). Thus, our ranking provides useful hints to help a designer interactively examine the data.

4.7 Conclusions

We have presented an approach to discover duplication in large data sets. We presented a set of information-theoretic techniques based on clustering that discover duplicate, or almost duplicate, tuples and attribute values in a relation instance. From the information collected about the values, we then presented an approach that groups attributes so that duplication in each group is as high as possible. The groups of attributes with large duplication provide important clues for the re-design of the schema of a relation. Using these clues, we introduced a novel approach to rank the set of functional dependencies that are valid in an instance.

Chapter 5

Software Clustering Based on Information Loss

In this chapter, we consider a different application of the LIMBO algorithm, that of clustering software artifacts. The majority of the algorithms in the software clustering literature utilize structural information in order to decompose large software systems. Other approaches, such as using file names or ownership information, have also demonstrated merit. However, no intuitive way to combine information obtained from these two different types of techniques has been proposed previously.

In this chapter, we present an approach that combines structural and non-structural information in an integrated fashion. LIMBO is used as the clustering algorithm. We apply LIMBO to two large software systems in a number of experiments. The results indicate that this approach produces valid and useful clusterings of the components of large software systems. LIMBO can also be used to evaluate the usefulness of various types of non-structural information in the software clustering process.

Portions of this chapter have been published by Andritsos and Tzerpos [AT03].

5.1 Introduction

It is widely believed that an effective decomposition of a large software system into smaller, more manageable subsystems can be of significant help to the process of understanding, redocumenting, or reverse engineering the system in question. As a result, the

software clustering problem has attracted the attention of many researchers in the last two decades.

The majority of the software clustering approaches presented in the literature attempt to discover clusters by analyzing the dependencies between software artifacts, such as functions or source files [MMCG99, Kos00, Lut02, SP89, Sch91, CS90, MOTU93, HB85, TH00]. Software engineering principles such as information hiding or *high-cohesion*, *low-coupling* are commonly employed to help determine the boundaries between clusters. Well-designed software systems are organized into cohesive subsystems that are loosely interconnected. A cohesive subsystem is characterized by its knowledge of a design decision that it hides from other subsystems. Typically, the elements of a cohesive subsystem exhibit a large degree of interdependency [Par72].

We distinguish two types of information about software artifacts. *Structural information* is extracted from the system implementation by program analysis tools on the behaviour of the system. It includes function calls, variable references, inter-process communications, header file inclusions and system build dependencies. On the other hand, *non-structural information* is based on the organization of the development group. It includes the names of developers of particular parts of the code, as well as simple statistics derived from the system, such as the number of lines of code. More general concepts can also be used, such as design principles, the goals for which particular modules were built and the functionality of sub-systems.

Using naming information, such as file names or words extracted from comments in the source code [AL97, MMM93] may be the best way to cluster a given system. The ownership architecture of a software system, *i.e.*, the mapping that shows which developer is responsible for what part of the system, can also provide valuable hints [BH98]. Some researchers have also attempted to combine structural and non-structural information [AL99]. Others have proposed ways of bringing clustering into a more general data management framework [AM01].

Even though existing approaches have shown that they can be quite effective when applied to large software systems, there are still several issues that can be identified:

1. There is no guarantee that the developers of a legacy software system have followed software engineering principles, such as high-cohesion, low-coupling. As a result, the validity of the clusters discovered following such principles, as well as the overall contribution of the decomposition obtained to the reverse engineering process, can be challenged.

2. Software clustering approaches based on high-cohesion, low-coupling fail to discover utility subsystems, *i.e.*, collections of utilities that do not necessarily depend on each other, but are used in many parts of the software system (they may or may not be omnipresent nodes [MOTU93]). Such subsystems do not exhibit high-cohesion, low-coupling, but they are frequently found in manually-created decompositions of large software systems.

3. It is not clear what types of non-structural information are appropriate for inclusion in a software clustering approach. Some choices are listed in Table 5.1. Clustering

Developer names
Lines of code
Directory structure
Date of creation
Date/Time of last maintenance
Revision control logs

Table 5.1: Candidate non-structural features

based on the lines of code of each source file is probably inappropriate, but what about using timestamps? Ownership information has been shown to be valuable [BH99], but its usefulness in an automatic approach has not been evaluated.

In this chapter, we present an approach that addresses these issues. Our approach is based on minimizing information loss during the software clustering process. The

objective of software clustering is to reduce the complexity of a large software system, especially when this system is visualized, by replacing a set of objects with a cluster. Thus, the decomposition obtained is easier to understand. However, this process also reduces the amount of information conveyed by the clustered representation of the software system. Using LIMBO, we create decompositions that convey as much information as possible by choosing clusters that represent their contents as accurately as possible. In other words, one can predict with high probability the features of a given object just by knowing the cluster to which it belongs.

Our approach clearly addresses the first issue raised above. It makes no assumptions about the software engineering principles followed by the developers of the software system. It also creates decompositions that convey as much information about the software system as possible, a feature that should be helpful to the reverse engineer. As will be shown in Section 5.2, our approach can discover utility subsystems as well as ones based on high-cohesion, low-coupling. Finally, any type of non-structural information may be included in our approach. As a result, our approach can be used in order to evaluate the usefulness of various types of information such as timestamps or ownership. In fact, we present such a study in Section 5.3.2.

5.2 Clustering Using *LIMBO*

Throughout this section we will use as an example the *dependency graph* of an imaginary software system given in Figure 5.1. For our purposes, we assume that the edges of the graph correspond to only one type of dependency among the nodes. This graph contains three program files f_1, f_2 and f_3 and two utility files u_1 and u_2. This software system is clearly too trivial to require clustering. However, it will serve as an example of how our approach discovers various types of subsystems.

Our approach starts by translating the dependencies shown in Figure 5.1 into the matrix shown in Table 5.2. The rows of this matrix represent the artifacts to be clustered,

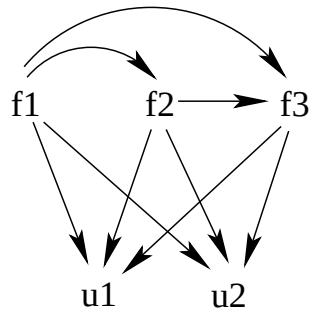


Figure 5.1: Example dependency graph

while the columns represent the features that describe these artifacts. Since our example contains only structural information (non-structural information will be added in Section 5.2.2), the features of a software artifact are other artifacts. To avoid confusion, we will represent the software artifacts to be clustered with italic letters, *e.g.*, f_1, u_1, and the corresponding features with bold letters, *e.g.*, $\mathbf{f_1}, \mathbf{u_1}$. Note that the directed arcs in Figure 5.1 are treated as undirected arcs in creating the matrix of Table 5.2.

	$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_3}$	$\mathbf{u_1}$	$\mathbf{u_2}$
f_1	0	1	1	1	1
f_2	1	0	1	1	1
f_3	1	1	0	1	1
u_1	1	1	1	0	0
u_2	1	1	1	0	0

Table 5.2: Example matrix from dependencies in Figure 5.1

Let X denote a discrete random variable taking its values from a set $\mathbf{X}$. In our example, $\mathbf{X}$ is the set $\{f_1, f_2, f_3, u_1, u_2\}$. Let Y be a second random variable taking values from the set $\mathbf{Y}$ of all the features in the software system. In our example, $\mathbf{Y}$ is the set $\{\mathbf{f_1}, \mathbf{f_2}, \mathbf{f_3}, \mathbf{u_1}, \mathbf{u_2}\}$.

The normalized matrix of Table 5.2 is depicted in Table 5.3.

Notice that this is a representation equivalent to the market-basket representation of Section 3.3.2, and is suitable for clustering by LIMBO.

$X \backslash Y$	$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_3}$	$\mathbf{u_1}$	$\mathbf{u_2}$
f_1	0	1/4	1/4	1/4	1/4
f_2	1/4	0	1/4	1/4	1/4
f_3	1/4	1/4	0	1/4	1/4
u_1	1/3	1/3	1/3	0	0
u_2	1/3	1/3	1/3	0	0

Table 5.3: Normalized matrix of system features

5.2.1 Structural Example

By using the representation of Section 3.3.2, we can compute all pairwise values of information loss (δI). These values are given in Table 5.4. The value in position (i, j) indicates the information loss we would incur, if we chose to group the i-th and the j-th artifact together.

	f_1	f_2	f_3	u_1	u_2
f_1	-	0.10	0.10	0.17	0.17
f_2	0.10	-	0.10	0.17	0.17
f_3	0.10	0.10	-	0.17	0.17
u_1	0.17	0.17	0.17	-	**0.00**
u_2	0.17	0.17	0.17	**0.00**	-

Table 5.4: Pairwise δI values for vectors of Table 5.3

Clearly, if utility files u_1 and u_2 are merged into the same cluster, c_u, we lose no information about the system. This agrees with our intuition just by observation of Figure 5.1, which suggests that u_1 and u_2 have exactly the same structural features. On the other hand, we lose some information if f_1 and f_2 are merged into the same cluster c_f. The same loss of information is incurred if any pair among the program files forms a cluster. Table 5.5 depicts the new matrix after forming clusters c_f and c_u. Intuitively, c_u represents the dependencies of its constituents *exactly* as well as u_1 and u_2 before the merge, while c_f is *almost* as good. We compute the probabilities of the two new clusters using Equation 3.2 from Section 3.2.3 as $p(c_f) = 2/5$ and $p(c_u) = 2/5$, while the new distributions $p(Y|c_f)$ and $p(Y|c_u)$ are calculated using Equation 3.3 of the same section. The values obtained are shown in Table 5.5.

$X \backslash Y$	$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_3}$	$\mathbf{u_1}$	$\mathbf{u_2}$	$p(c^*)$
c_f	1/8	1/8	1/4	1/4	1/4	2/5
f_3	1/4	1/4	0	1/4	1/4	1/5
c_u	1/3	1/3	1/3	0	0	2/5

Table 5.5: Normalized matrix after forming c_f and c_u

The new matrix of pairwise distances is given in Table 5.6. It suggests that c_f should next be merged with f_3 as their δI value is the minimum. Thus, our approach is able to discover both utility subsystems (such as c_u) as well as cohesive ones (such as the cluster containing f_1, f_2, and f_3).

	c_f	f_3	c_u
c_f	-	**0.04**	0.26
f_3	**0.04**	-	0.24
c_u	0.26	0.24	-

Table 5.6: Pairwise δI after forming c_f and c_u

5.2.2 Example Using Non-Structural Information

One of the strengths of our approach is its ability to consider various types of information about the software system. Our example so far has employed only structural data. We now expand it to involve non-structural data as well, such as the name of the developer, or the location of an artifact.

All we need to do is extend the universe $\mathbf{Y}$ to include the values of non-structural features. This way our algorithm is able to cluster the software system components in the presence of meta-information about software artifacts. The files of Figure 5.1 together with their *developer* and *location* are given in Table 5.7.

The normalized matrix when $\mathbf{Y}$ is extended to $\{\mathbf{f_1}, \mathbf{f_2}, \mathbf{f_3}, \mathbf{u_1}, \mathbf{u_2}, \mathbf{Alice}, \mathbf{Bob}, \mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}\}$ and shown in Table 5.8.

After that, $I(X;Y)$ is defined and clustering is performed as before, but without necessarily giving the same results. This will be illustrated in the experimental evaluation section of this chapter.

	Developer	Location
f_1	**Alice**	$\mathbf{p_1}$
f_2	**Bob**	$\mathbf{p_2}$
f_3	**Bob**	$\mathbf{p_2}$
u_1	**Alice**	$\mathbf{p_3}$
u_2	**Alice**	$\mathbf{p_3}$

Table 5.7: Non-structural features for the files in Figure 5.1

	$\mathbf{f_1}$	$\mathbf{f_2}$	$\mathbf{f_3}$	$\mathbf{u_1}$	$\mathbf{u_2}$	**Alice**	**Bob**	$\mathbf{p_1}$	$\mathbf{p_2}$	$\mathbf{p_3}$
f_1	0	1/6	1/6	1/6	1/6	1/6	0	1/6	0	0
f_2	1/6	0	1/6	1/6	1/6	0	1/6	0	1/6	0
f_3	1/6	1/6	0	1/6	1/6	0	1/6	0	1/6	0
u_1	1/5	1/5	1/5	0	0	1/5	0	0	0	1/5
u_2	1/5	1/5	1/5	0	0	1/5	0	0	0	1/5

Table 5.8: Normalized matrix of system dependencies with structural and non-structural features

5.3 Experimental Evaluation

5.3.1 Experiments with Structural Information Only

In order to evaluate the applicability of LIMBO to the software clustering problem, we applied it to two large software systems with known widely accepted decompositions, and compared its outputs to those of other well-established software clustering algorithms.

The two large software systems we used for our experiments were of comparable size, but of different development philosophy:

1. **TOBEY**. This is a proprietary industrial system that is under continuous development. It serves as the optimizing back end for a number of IBM compiler products. The version we worked with was comprised of 939 source files and approximately 250,000 lines of code. An authoritative decomposition of TOBEY was obtained through a series of interviews with its developers.

2. **Linux**. We experimented with version 2.0.27a of this free operating system that is probably the most significant existing open-source system. This version had 955 source files and approximately 750,000 lines of code. An authoritative decomposi-

tion of Linux was presented by Bowman, Holt and Brewester [BHB99].

The software clustering approaches used for comparison were the following:

1. **ACDC**. This is a pattern-based software clustering algorithm that attempts to recover subsystems commonly found in manually-created decompositions of large software systems [TH00].

2. **Bunch**. This is a suite of algorithms that attempt to find a decomposition that optimizes a quality measure based on high-cohesion, low-coupling. We experimented with two versions of a hill-climbing algorithm, we will refer to as NAHC and SAHC (for nearest- and shortest-ascent hill-climbing) [MMCG99].

3. **Cluster Analysis Algorithms**. We also compared LIMBO to several hierarchical agglomerative cluster analysis algorithms. We used the Jaccard co-efficient as similarity measure because it has been shown to work well in a software clustering context [AL99]. We experimented with four different algorithms: single linkage (SL), complete linkage (CL), weighted average linkage (WA), and unweighted average linkage (UA).

Many data sets commonly used in testing clustering algorithms include a variable that is hidden from the algorithm, and specifies the class with which each tuple is associated. As mentioned above, the market-basket data sets we experimented with include such a variable, since an authoritative decomposition is available for all of them. This variable is *not* used by the clustering algorithms.

To evaluate the results of our clustering, we used the MoJo distance measure[1] [TH99, WT03]. MoJo distance between two different partitions A and B of the same data set is defined as the minimum number of *Move* or *Join* operations one needs to perform

[1] A Java implementation of MoJo is available for download at:
`http://www.cs.yorku.ca/~bil/downloads`

in order to transform either A to B or vice versa. *Move* refers to assigning a tuple to a different cluster, while *Join* refers to merging two clusters into one. Intuitively, the smaller the MoJo distance between an automatically created clustering A and the accepted partitioning B, the more effective the algorithm that created A can be considered to be.

In order to choose an appropriate number of clusters for the non-structural data, we start by creating decompositions for all values of k between two and a large value. For the experiments performed here, the chosen value was 100. We felt that clusterings of higher cardinality would not be useful from a reverse engineering point of view. Moreover, this value was always sufficient to allow us to choose an appropriate k.

Let C_k be a clustering of k clusters and C_{k+1} a clustering of $k+1$ clusters. Both clusterings are given after Phase 3 of Limbo is performed. Hence, this phase may produce different results when we move from $k+1$ to k clusters. If the cluster representatives created in Phase 2 are well created, then these neighboring clusterings must differ in only one cluster after Phase 3. Starting from a large k value, we compare consecutive clustering results and stop if one of the clusterings, C_{k+1}, can be produced after a single merge of two clusters of the second clustering, C_k. Using MoJo, we can detect these clusterings by computing the value of the index from C_{k+1} to C_k, *i.e.*, the value of $MoJo(C_{k+1}, C_k)$. If this value is equal to one, this means that the difference in the two clusterings is translated to a simple merge of two clusters of C_{k+1}, to produce the k clusters of C_k. In our experiments we report the first value of k, where $MoJo(C_{k+1}, C_k) = 1$ after all three phases of LIMBO have been performed.

For the experiments presented in this section, all algorithms were provided with the same input, the dependencies between the software artifacts to be clustered. The traditional cluster analysis algorithms were run with a variety of cut-point heights, which is a user-specified threshold on the distance between sub-clusters. The smallest MoJo distance obtained is reported below.

	TOBEY		Linux	
	k	MoJo	k	$MoJo$
LIMBO	80	311	56	237
ACDC	94	320	66	342
NAHC	33	382	35	249
SAHC	15	482	15	353
SL	67	688	9	402
CL	153	361	154	304
WA	139	351	70	309
UA	131	354	38	316

Table 5.9: (k,MoJo) pairs between decompositions proposed by eight different algorithms and the authoritative decompositions for TOBEY and Linux

Table 5.9 presents the results of our experiments. As can be seen, LIMBO created a decomposition that is closer to the authoritative one for both TOBEY and Linux, although the nearest-ascent hill-climbing (NAHC) algorithm of Bunch comes very close in the case of Linux, as does ACDC in the case of TOBEY. The cluster analysis algorithms perform respectably, but as expected, are not as effective as the specialized software clustering algorithms. Notice that although the MoJo values that correspond to the LIMBO algorithm are quite large, they are smaller than the values for other algorithms. Moreover, all MoJo values correspond to different numbers of clusters.

We believe that the fact that LIMBO performed better than other algorithms can be attributed mostly to its ability to discover utility subsystems. An inspection of the authoritative decompositions for TOBEY and Linux revealed that they both contain such collections of utilities. Since in our experience that is a common occurrence, we are optimistic that similar results can be obtained for other software systems as well.

The results of these experiments indicate that the idea of using information loss minimization as a basis for software clustering has definite merit. Even though further experimentation is required in order to assess the usefulness of LIMBO in the reverse engineering process, it is clear that it can automatically create decompositions that are close to the ones prepared manually by humans.

We also tested LIMBO's efficiency with both systems. The time required to cluster the components of a software system depends on the number of clusters. For a given number of clusters k, LIMBO was able to produce a C_k clustering within 31 seconds. Figure 5.2 presents LIMBO's execution time for both example systems, and all values of k from 2 to 100.

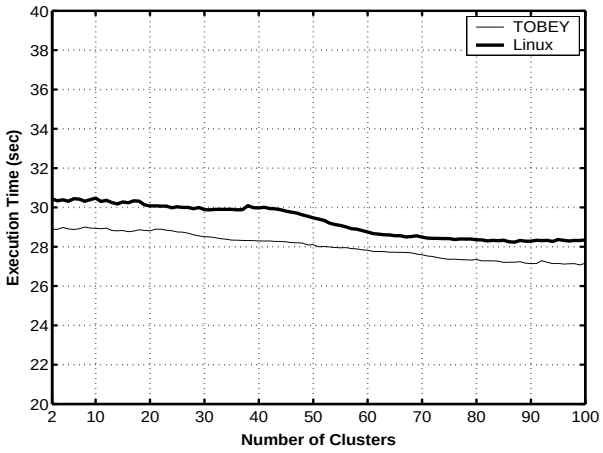


Figure 5.2: LIMBO execution time

As can be seen in Figure 5.2, execution time varies only slightly as k increases. As a result, obtaining an appropriate clustering for either example system was a matter of minutes. The similarity in the execution times of LIMBO for the two systems does not come as a surprise, since the number of source files to be clustered was similar (939 in TOBEY and 955 in Linux). Finally, we provide the execution times for each phase of the LIMBO algorithm. Figure 5.3(a) shows the execution time for Phase1, Figure 5.3(b) the execution time for Phase 2 and Figure 5.3(c) the execution time of Phase 3. From the graphs in these figures, we observe the decrease in the execution time of Phase 2 while the execution time of Phase 3 increases linearly with increasing k.

5.3.2 Experiments with Non-Structural Information Added

In this section, we utilize LIMBO's ability to combine structural and non-structural information seamlessly in order to evaluate the usefulness of certain types of information

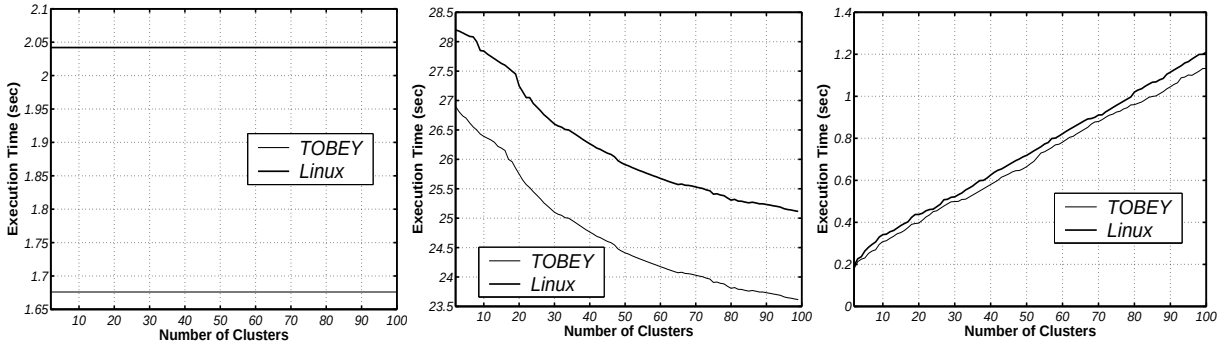


Figure 5.3: (a) Phase 1, (b) Phase 2, (c) Phase 3 execution times of LIMBO

to the reverse engineering process. We present results for the application of LIMBO to Linux when non-structural features are present. We will test the quality of clustering when the following features are added to the structural information:

- *Developers (dev)*: This feature gives the ownership information, *i.e.*, the names of the developers involved in the implementation of the file. Where the developer was unknown, we used a unique dummy value for each file.

- *Directory Path (dir)*: In this feature, we include the full directory path for each file. In order to increase the similarity of the files residing in similar directory paths, we include the set of all sub-paths for each path. For example, the directory information for file `drivers/char/ftape/ftape-io.c` is the set {`drivers`, `drivers/char`, `drivers/char/ftape`} of directory paths.

- *Lines of Code (loc)*: This feature is the number of lines of code in the files. We discretized the values by dividing the full range of *loc* values into the intervals $(0, 100]$, $(100, 200]$, $(200, 300]$, etc. Each file is given a feature such as `RANGE1`, `RANGE2`, `RANGE3`, etc.

- *Time of Last Update (time)*: This feature is derived from the time-stamp of each file on the disk. We include only the month and year of latest modification to the file.

In order to investigate the results LIMBO produced with these non-structural features, we consider all possible combinations of them added to the structural information. These combinations are depicted in the lattice of Figure 5.4. At the bottom of this lattice, we have only the structural dependencies, and as we follow a path upwards, different non-structural features are added. Thus, in the first level of the lattice, we only add individual non-structural features. Each addition is represented by a different type of arrow at each level of the lattice. For example, the addition of *dir* is given by a solid arrow. As the

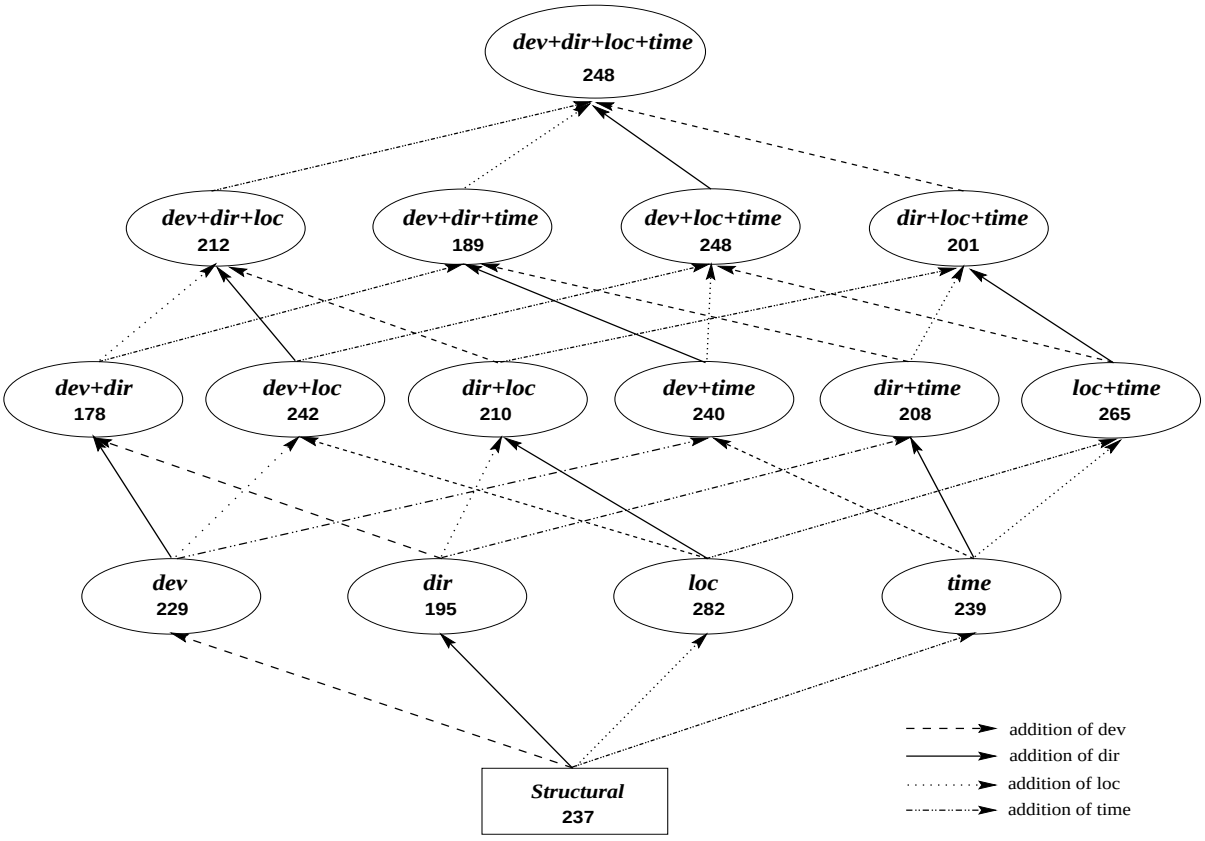


Figure 5.4: Lattice of combinations of non-structural features for the Linux system

lattice of Figure 5.4 suggests, there are fifteen possible combinations of non-structural features that can be added to the structural information.

Each combination of non-structural features in Figure 5.4 is annotated with the MoJo distance between the decomposition created by LIMBO and the authoritative one. The results are also given, in ascending order of the MoJo distance value, in Table 5.10. The

table also includes the number of clusters that the proposed decomposition had in each case.

	Clusters	MoJo
dev+dir	69	178
dev+dir+time	37	189
dir	25	195
dir+loc+time	78	201
dir+time	18	208
dir+loc	74	210
dev+dir+loc	49	212
dev	71	229
structural	56	237
time	66	239
dev+time	73	240
dev+loc	73	242
dev+loc+time	45	248
dev+dir+loc+time	48	248
loc+time	34	265
loc	85	282

Table 5.10: Number of clusters and MoJo distance between the proposed and the authoritative decomposition.

Certain combinations of non-structural data produce clusterings with a smaller MoJo distance to the authoritative decomposition than the clustering produced when using structural information alone. This indicates that the inclusion of non-structural information has the potential to increase the quality of the decomposition obtained. However, in some of the cases the MoJo distance to the authoritative decomposition has increased significantly.

A closer look reveals some interesting observations:

- Following a solid arrow in the lattice always leads to a smaller MoJo value (with the exception of the topmost one where the value is actually the same). This indicates that the inclusion of directory structure information produces better decompositions, an intuitive result.

- Following a dashed arrow leads to a smaller MoJo value as well, although the

difference is not as dramatic as before (the topmost dashed arrow is again an exception). Still, this indicates that ownership information has a positive effect on the clustering obtained, a result that confirms the findings of Holt and Bowman [BH99].

- Following a dotted arrow consistently decreases the quality of the decomposition obtained. (A marginal exception exists between *dir+time* and *dir+loc+time*.) This confirms our expectation that using the lines of code as a basis for software clustering is not a good idea.

- Finally, following the arrows that indicate addition of *time*, leads mostly to worse clusterings but only marginally. This indicates that time could have merit as a clustering factor for some software systems. Better results might be achieved by examining the revision control logs of a system in order to obtain information about which files are being developed around the same time.

The few exceptions to the above trends that we encountered occurred in the top part of the lattice. This is due to the fact that, when a number of factors have already been added to the algorithm's input, the effect of a new factor will not be as significant, and in fact it might be eclipsed by the effect of other factors. As a result, we believe that the lower part of the lattice provides more revealing results than the top part.

When the structural information was removed from LIMBO's input, the results were not as good. In fact, *loc* and *time* produced rather random decompositions. The situation was better for *dir* and *dev* (MoJo distances to the authoritative decomposition of 407 and 317 respectively), but still quite far from the results obtained from the combination of structural and non-structural information. This result indicates that an effective clustering algorithm needs to consider both structural and non-structural information in order to produce decompositions that are close to the conceptual architecture of a software system.

Finally, the execution times observed for the experiments that involved non-structural information were almost identical to those that involved both structural and non-structural information.

In summary, the results of our experiments show that directory structure and ownership information are important factors for the software clustering process, while lines of code is not. Further research is, of course, required in order to determine whether these results hold true for a variety of software systems, or are particular to the Linux kernel.

5.4 Conclusions

This chapter demonstrated that information loss minimization is a valid basis for a software clustering approach. A strength of our approach is that it can incorporate in the software clustering process any type of information relevant to the software system. We experimented and assessed the usefulness of adding four different types of non-structural information to supplement the structural information.

Chapter 6

Evaluating Value Weighting Schemes in LIMBO

All algorithms presented so far deem all attributes and data values present in a data set as equally important. In this chapter, we present a set of weighting schemes that allow for objective assignments of importance to the values of a data set. We use well established weighting schemes from information retrieval, web search and data clustering to assess the importance of whole attributes and individual values. To the best of our knowledge, this is the first work that considers weights in the clustering of categorical data.

We perform clustering in the presence of importance for the values within the LIMBO framework. Our experiments were performed on dataset from a variety of domains, including data sets used before in clustering research and three data sets from large software systems. We report results as to which weighting schemes show merit in the decomposition of data sets.

The results of this chapter have been submitted for publication [AT04].

6.1 Introduction

Current algorithms treat all attributes in a relation and all individual values in a data set equally. However, a domain expert clustering a particular data set would invariably assign different importance to particular attributes based on her intuition. Similarly, she

might consider certain data values as more important than others for the determination of the clusters.

The premise for the work presented in this chapter is that by assigning different importance to attributes and/or individual values, we can direct the clustering process toward a more meaningful result. We do this by implementing a number of weighting schemes that are based on existing techniques from information retrieval and clustering of categorical values. Experiments conducted using the LIMBO algorithm demonstrate the merit of the various weighting schemes and suggest possible improvements.

In particular, we investigate weighting schemes that apply to two different types of data sets:

1. *Relational Data Sets.* These are data sets where tuples are defined over a set of different attributes. We employ two techniques from information retrieval and one from spectral graph theory, in order to produce weights for the attributes and/or individual values in such data sets:

   - *Term Frequency-Inverse Document Frequency (TF.IDF).*
   - *Mutual Information* (conveyed by a particular value about the rest of the values).
   - *Linear Dynamical Systems.* These systems iterate a function over a graph in order to update the weights of its nodes. They are used in the STIRR algorithm described in Section 2.4.

2. *Graph-based data sets.* These are data sets where the objects to be clustered are in the form of a graph that represents interdependencies between them. Such structures appear in numerous domains, such as in hyperlinked documents, where the objective is to group web pages with similar content, or the reverse engineering of software systems, where the objective is to decompose software systems into meaningful components in order to better understand and maintain them.

In addition to the weighting schemes applied to relational data sets, we also employ the following two weighting schemes for graph-based data sets:

- We use the well known *PageRank* algorithm [BP98] to assign importance to specific values.
- We utilize usage data, such as weblogs or information obtained by profiling software systems.

Our work is different in spirit from the work presented in the literature on *Feature Selection* [LM98]. In feature selection for clustering [Tal99], the main focus is on the elimination of whole attributes to improve the performance of the underlying algorithm. An initial evaluation of a weighting scheme without attribute elimination is presented by Modha and Spangler for numerical data and the k-means algorithm [MS03]. On the other hand *Term Weighting Schemes* have appeared in Information Retrieval to ensure better search results [BR99, DS03, SB88]. These techniques, though, assume a class label assigned to every tuple and evaluate attributes according to how well they predict these labels. Finally, Gravano et al., use the *TF.IDF* weighting schemes for approximate text joins within a database system [GIKS03].

6.2 Incorporating Weights

We consider both relational and market-basket data with the representations given in Section 3.3.1 and 3.3.2, respectively. We denote by $\mathbf{T}$ the set of tuples to be clustered and T the random variable that takes its values from set $\mathbf{T}$. Similarly, $\mathbf{V}$ is the set of attribute values and V the random variable taking values from $\mathbf{V}$. When dealing with graph-based data sets, we first transform them into market-basket data sets, and then use the corresponding representation. Since the objective with a graph-based data set is to cluster the nodes of the graph, the transformation into market-basket data proceeds as follows: Each node n_i of the graph corresponds to a tuple, while the values in the

tuple is the set of nodes that are adjacent to n_i in the graph.

6.2.1 Incorporating Weighting Schemes

In both relational and market-basket data, we normalized each row of matrix M to sum to one in order to make it a probability distribution. This way we consider the appearance of a value in a tuple as probable as any of the other value in the same tuple. If we represent importance with numerical weights, the aforementioned conceptualizations of our data sets involve values with equal weights.

Our goal is to study how particular weighting schemes over the data we cluster influence the resulting clusters. Before introducing these schemes, we describe how to apply a weighting scheme through an example. Consider the tuples of the market-basket data set given in Table 6.1. According to the equations from Section 3.3.1, we set $p(t_i) = 1/4$, $1 \leq t_i \leq 4$, and the matrix M that is used to represent this data set is given in Table 6.2.

t_1	a	b	c	d	e
t_2	b	c	e		
t_3	d	e			
t_4	a	b	d		

Table 6.1: Market-basket data

	a	b	c	d	e	$p(t)$
t_1	1/5	1/5	1/5	1/5	1/5	1/4
t_2	0	1/3	1/3	0	1/3	1/4
t_3	0	0	0	1/2	1/2	1/4
t_4	1/3	1/3	0	1/3	0	1/4

Table 6.2: Market-basket data representation

By applying Equation 3.4 to the example data set, we can compute all pairwise values of information loss (δI) that would result from merging tuples. These values are given in Table 6.3. The value in position (i, j) indicates the information loss we would incur, if we chose to group the i-th and the j-th tuple together.

	t_1	t_2	t_3	t_4
t_1	-	0.1182	0.1979	0.1182
t_2	0.1182	-	0.2977	0.3333
t_3	0.1979	0.2977	-	0.2977
t_4	0.1182	0.3333	0.2977	-

Table 6.3: Pairwise δI values for vectors of Table 6.2

From the information losses of Table 6.3, we conclude that the algorithm should merge either pair (t_1, t_2) or (t_1, t_4), which have the lowest value of 0.1182. We also notice that pairs (t_2, t_3) and (t_3, t_4) are equidistant.

Let us now assume that a particular weighting scheme has assigned weights to the five values in the example (larger weights correspond to more important values). Denoting the vector of weights with w, we may have $w = (0.01, 0.01, 0.01, 0.96, 0.01)$. This rather extreme weight distribution considers value d to be the most important one. In order to have the importance of each value reflected in matrix M, we replace each appearance of a value in a tuple with its weight, and normalize the rows of matrix M so that they sum up to one. Using the example vector w given above, the new matrix M is given in Table 6.4.

	a	b	c	d	e	$p(t)$
t_1	0.01	0.01	0.01	0.96	0.01	1/4
t_2	0	0.3333	0.3333	0	0.3333	1/4
t_3	0	0	0	0.9897	0.0103	1/4
t_4	0.0102	0.0102	0	0.9796	0	1/4

Table 6.4: Data representation with weights

The new pairwise distances between tuples are given in Table 6.5.

	t_1	t_2	t_3	t_4
t_1	-	0.4511	0.0076	0.0050
t_2	0.4511	-	0.4833	0.4834
t_3	0.0076	0.4833	-	0.0077
t_4	0.0050	0.4834	0.0077	-

Table 6.5: Pairwise δI values for vectors of Table 6.4

In the presence of importance for the values, there are no ties in the information losses among the tuples in this particular example. Moreover, the closest pair is now (t_1, t_4), which are almost identical since they both share the value with highest importance. The clustering algorithm, as an initial step, will merge tuples t_1 and t_4 into cluster t_{14} and the new probability distribution $p(V|t_{14})$ is given in Table 6.6.

	a	b	c	d	e	$p(t)$
t_{14}	0.0101	0.0101	0.0050	0.9698	0.0050	1/2
t_2	0	0.3333	0.3333	0	0.3333	1/4
t_3	0	0	0	0.9897	0.0103	1/4

Table 6.6: New data representation with weights

The new pairwise distances are given in Table 6.7.

	t_{14}	t_2	t_3
t_{14}	-	0.3820	0.3298
t_2	0.3820	-	0.4833
t_3	0.3298	0.4833	-

Table 6.7: New pairwise δI values for vectors of Table 6.6

This table dictates that tuple t_3 and cluster t_{14} should be merged next.

After this illustrative example, we are now ready to formally define the data set representation in the presence of weights for the attribute values. If $\mathbf{V}$ is the universe of all d values that appear in the data set and w a vector of their importances, where $|w| = d$, we represent our data as an $n \times d$ matrix M, where $M[t, v] = w(v)$ if attribute value $v \in \mathbf{V}$ appears in tuple $t \in \mathbf{T}$, and zero otherwise. For a tuple $t \in \mathbf{T}$, we define:

$$p(t) = 1/n \tag{6.1}$$

$$p(v|t) = \begin{cases} w(v)/\sum_{v' \in \mathbf{V}}(w(v')) & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

The only difference from the representation of market-basket data in Section 3.3.2 is in the definition of distribution $p(V|t)$, where we first replace each entry equal to 1 with the weight of the corresponding value v and normalize each vector so that it sums

up to one. Note that our definition is sufficiently general to cover both relational and market-basket data sets. In the former case, if we only have weights for each attribute rather than for each value, we may proceed as above after giving each value the weight of its corresponding attribute.

In the following section, we present weighting schemes for both attributes and values.

6.3 Data Weighting Schemes

In this section, we present in detail the weighting schemes we consider for our data sets.

6.3.1 Mutual Information

The first weighting scheme that we propose is based on mutual information. Given a set of attributes $A_1, A_2, \ldots, A_m$, we can define a probability distribution of the values of each one of them. The dependence score for attribute A_i and A_j, is computed as the mutual information $I(A_i; A_j)$ given by the following equation

$$I(A_i; A_j) = H(A_i) - H(A_i|A_j) = H(A_j) - H(A_j|A_i)$$

Note that mutual information is symmetric and, the lower its value, the weaker the dependence between A_i and A_j . We suggest computing the weight $MI(A_i)$ for each attribute A_i as the average mutual information between A_i and each other attribute:

$$MI(A_i) = \frac{1}{m-1} \sum_{j=1, j\neq i}^{m} I(A_i; A_j)$$

The higher the value of $MI(A_i)$, the more important A_i is.

Given a relational data set, we compute the weight of each one of the attributes and label the values of the data sets with the weights of their corresponding attributes. More formally, if v_{ij} is the j-th value that belongs to the set of values $\mathbf{V}_i$ of attribute A_i, then $w(v_{ij}) = MI(A_i)$.

The previous definition of MI holds for relational data sets. In the case of market-basket data sets, the tuples are expressed over a single attribute. Hence, we need to

define the probability distribution in a different manner. For each value $v_i \in \mathbf{V}_i$, we define the probability

$$P_{present}(v_i) = \frac{\text{number of times } v_i \text{ appears}}{n} \tag{6.3}$$

Equation 6.3 is the probability of finding value v_i in a randomly selected tuple in the data set. Therefore, using $P_{present}$ and $P_{absent} = 1 - P_{present}$ we can compute the entropy $H(v_i)$ of value v_i. Similarly we can define the joint distribution of pairs of values $v_i \in \mathbf{V}_i$ and $v_j \in \mathbf{V}_j$ and compute the joint entropy $H(v_i, v_j)$. Given the mutual information of values $v_i \in \mathbf{V}_i$ and $v_j \in \mathbf{V}_j$, the MI value of v_i can be computed by

$$MI(v_i) = \frac{1}{d-1} \sum_{j=1, j \neq i}^{d} I(v_i; v_j)$$

6.3.2 Linear Dynamical Systems

In this section, we restate the definition of Dynamical Systems from Chapter 2. Dynamical Systems have been previously used in the clustering of attribute values in a relational data set [GKR98]. In this case, the data set is represented as a hypergraph whose nodes are the values in the data set, and there is an undirected edge between two values that appear in a tuple together. An example of a relational data set together with its hypergraph is given in Figure 6.1.

	A	B	C
t_1	a	w	1
t_2	a	x	1
t_3	b	y	2

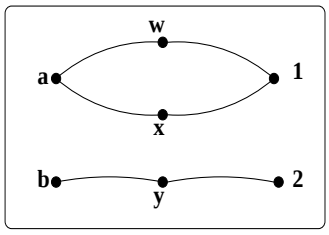


Figure 6.1: Relational data set with its hypergraph

Given a set of d values, the initial set of weights, which is called the *initial configuration*, is a d-dimensional vector w of real numbers. The dynamical system repeatedly applies a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. The configuration in which the values in the

d-dimensional vector do not change, that is $f(w_i) = w_{i-1}$, where i indexes the successive weight configurations, is called a *fixed point* of the dynamical system.

The dynamical system that f describes is given in Figure 6.2 [GKR98]. Following the steps, we update the weight w_v of each value v.

Dynamical System

To update weight w_v:
For each tuple $\tau = \{v, u_1, \ldots, u_m\}$
containing v do:
$\chi_\tau = \bigoplus(u_1, \ldots, u_m)$
$w_v \leftarrow \sum_\tau \chi_\tau$

Figure 6.2: Updating weights in a dynamical system

In Figure 6.2, the symbol $\bigoplus$ denotes the combination operator. Several choices for the combination operator have appeared in the literature [GKR98]. We shall use the summation operator, hence the term *Linear Dynamical Systems (LDS)*. Intuitively, for each value, we sum the weights of the values with which it co-occurs in the data set. To update all the values in the data set, a full pass over the data is required. In each iteration, we normalize the weight vector w so that the weights sum to one and check if $f(w_i) = w_{i-1}$. If this is the case, the dynamical system has converged and the final set of weights is stored in w_i. If not, more iterations are performed until we reach a fixed point. In our experiments, we performed ten iterations of the dynamical system, since this has been shown to perform well [GKR98]. Alternatively, a threshold ϵ can be used in the comparison $f(w_i) = w_{i-1}$. If $f(w_i) - w_{i-1} \leq \epsilon$ then the iterations cease, otherwise a new one starts. Instead of ϵ, we can also bound the number of iterations. Generally, little is known with respect to the theoretical justification as to why Dynamical Systems converge [GKR98].

In our work, we use Linear Dynamical Systems to derive weights for the values in both kinds of data sets. The more a value co-occurs with other values in the data set the higher its weight.

6.3.3 TF.IDF

In this section, we introduce the use of the well-established *Term Frequency-Inverse Document Frequency (TF.IDF)* weighting scheme from information retrieval [BR99]. Given a collection of d values $\mathbf{V}$ and n tuples $\mathbf{T}$, the *TF.IDF* weight of a value $v \in \mathbf{V}$ is defined as

$$TF.IDF(v) = tf(v) \cdot \log\Big(idf(v)\Big)$$

where $tf(v)$ (term frequency) is the frequency of value v in a tuple $t \in \mathbf{T}$ and $idf(v)$ (inverse document frequency) is the fraction n/n_v, with n_v being the number of tuples containing the value v. For relational data sets all values have $tf(v) = 1$ for obvious reasons. Drawing the analogy with information retrieval, we consider our tuples as a set of documents and our values as the set of terms over which these documents are expressed.

Intuitively, the *TF.IDF* weight of a value is high if this value appears many times within a tuple and at the same time a smaller number of times in the collection of the tuples. The latter means that this value conveys high discriminatory power. For example, in a data warehouse of software artifacts, file `stdio.h`, which is used by a large number of software files will have a lower *TF.IDF* compared to file `my_vector.h`, which is connected to a smaller fraction of files.

Once vector w of the weights of all values in $\mathbf{V}$ is defined, we normalize it so that it sums up to one. Hence, the resulting weights correspond to the impact of the values in the data set. Note that the *TF.IDF* scheme can be applied to both relational and market-basket data.

6.3.4 PageRank

PageRank is a weighting scheme proposed and widely used in search engines [BP98] to compute a web page's importance (or relevance). *PageRank* can be used when the

relationships among different web pages are given by a graph. We shall draw an analogy with a data set whose values are related to each other, and this relationship is realized through a directed graph. (Note that, in the case of *Dynamical Systems*, there is no direction associated with the edges of the hypergraph.) The main idea behind *PageRank* is that a value v is deemed important if it is being pointed to by *good* values.

More precisely, let us denote by G the graph that relates the values in $\mathbf{V}$. *PageRank* performs a random walk over the nodes of G. The walk starts at a random node according to some distribution, usually uniform. Intuitively, the weight of a node n_0 is the number or frequency of visits to this node. The *PageRank* of a node n_0 with $C(n_0)$ outgoing links is computed as [BP98]:

$$PR(n_0) = (1-\alpha) + \alpha\Big(\frac{PR(n_1)}{C(n_1)} + \ldots + \frac{PR(n_s)}{C(n_s)}\Big) \tag{6.4}$$

where $n_1, \ldots, n_s$ are the nodes that point to n_0. The parameter α is a *damping factor*, which can be set between 0 and 1. A common value for α is 0.85 [BP98], the value we used in our experiments.

The *PageRank* of each page depends on the *PageRank* of the pages that point to it. To reach a final weight vector w of *PageRank* weights, $PR(v)$ of each value can be calculated using a simple iterative algorithm. Vector w corresponds to the principal eigenvector of the normalized adjacency matrix of G. As in the case of *Dynamical Systems*, the iterations stop when the vector w of PR values, as given by Equation 6.4, converges.

6.3.5 Usage Data

Edges in a graph-based data set indicate only potential relationships between the objects they connect. For example, a link on a webpage indicates a potential path that a user might follow. A procedure call in the call graph of a software system may or may not be executed when the system is run. Furthermore, it is quite common that particular edges are heavily used, while others are used only rarely.

These observations indicate that the static picture of a graph-based data set might belie what actually happens when the system it represents is in use. It is intuitive to conjecture that the amount of usage of a particular object is related to its importance.

For this reason, the fifth weighting scheme we implemented for this work is based on usage data acquired from a dynamic trace. Assuming that each edge in the graph-based data set is associated with a weight that represents its usage, each value in the corresponding market-basket data set was assigned a weight equal to the weight of the edge that connects the node represented by the value to the node represented by the tuple. (The unweighted transformation of a graph-based data set to a market-basket data set was described in Section 6.2.) The weights of the values within each tuple were then normalized prior to the execution of LIMBO.

In contrast to the *PageRank* weighting scheme, here the same value might be given a different non-normalized weight when it appears in different tuples since, for example, a particular procedure will not be called with the same frequency by all its callers.

6.3.6 Weight Transformations

An interesting observation that was confirmed by early experiments is that the weights assigned by the weighting schemes presented so far may not always be beneficial to the clustering process. For example, nodes deemed highly relevant by *PageRank* may not be as important for clustering purposes. In software clustering, there is the well-established notion of "omnipresent" nodes [MOTU93], i.e., nodes with large in- or out-degree. It is often beneficial to minimize the effect such nodes have in the clustering process.

For this reason, we investigated several variations to the five weighting schemes by a process of weight smoothing. More precisely, for each weighting scheme, we also applied the following variations:

- The values with the largest weights were identified, and their weight was modified to the minimum weight in the data set. We performed experiments where the values

affected were in the top 5, 10, or 20 percentile.

- All values were sorted according to their weight. Intuitively, we reverse the order in which the values are weighted. If v_1 is the value with the smallest weight, and v_d is the value with the largest weight, this variation assigns a new weight to v_i equal to the old weight of v_{d-i+1}.

6.4 Experimental Evaluation

We perform a comparative evaluation using the LIMBO clustering algorithm on both relational and market-basket data sets. Our intention here is not to test the scalability or the performance of LIMBO under different parameter settings. The effects of these parameters have been discussed in Chapter 3.

We experimented with the following six data sets.

6.4.1 Relational Data Sets

The first two relational data sets we used are the Votes and Mushroom data sets described in Chapter 3. The third data set is the DBLP data set described in Chapter 4.

The first two data sets have been previously used for the evaluation of clustering algorithms [BCL02b, GKR98, GRS99, ATMS04] as well as in our experiments with LIMBO in Chapters 3 and 4.

Congressional Votes. We ran LIMBO with $\phi = 0.0$ and use this data set to test the performance of the weighting schemes on data sets with small attribute domains.

Mushroom. We used $\phi = 0.5$ for this data set to reduce the number of leaf entries to approximately 350 with no loss in quality.

DBLP Bibliography. We used this highly heterogeneous relation containing a large number of values (including missing ones) to demonstrate the strength of our approach

in suggesting a clustering. The ϕ value used was 1.2 due to the larger size of this data set.

6.4.2 Market-basket Data Sets

In addition to the TOBEY and LINUX data sets as described and used in Chapter 5, we performed experiments on a third data set, that of the Mozilla source code.

TOBEY. We used $\phi = 0.0$ for this data set.

LINUX. Due to the relatively small size of this data set, we used $\phi = 0.0$ again.

Mozilla. The third market-basket data set we used for our experiments was derived from Mozilla, an open-source web browser. We experimented with version 1.3, which was released in March 2003. It contains approximately 4 million lines of C and C++ source code.

We built Mozilla under Linux and extracted its static dependency graph using CPPX, and a dynamic dependency graph using *jprof*. A decomposition of the Mozilla source files for version M9 was presented by Godfrey and Lee [GL00]. For the evaluation portion of our work, we used an updated decomposition for version 1.3 [Xia04].

Mozilla was the only software system that was used to evaluate the usage data weighting scheme. The main reason for this was the fact that, in order to extract meaningful usage data from a software system, one needs a comprehensive test suite that ensures good coverage of as many execution paths as possible. Such a test suite was not available for TOBEY or LINUX. However, we were able to use the Mozilla "smoketests" [GL00] for this purpose. The dynamic dependency graph we obtained contained information about 1202 of the 2432 source files that are compiled under Linux. The results presented in this section are based on the classification of these 1202 files. A number of ϕ values between 0.0 and 1.2 gave the same results. We report experiments with $\phi = 1.2$.

6.4.3 Quality Measures for Clustering

Clustering quality lies in the eye of the beholder; determining the best clustering usually depends on subjective criteria. For this reason, we will use a variety of evaluation measures in order to assess the merit of the obtained clusterings.

Category Utility (CU): The definition of Category Utility [GC85] was presented in Chapter 3. In order to compare clusterings with different number of clusters, Fisher's COBWEB clustering system [Fis87] introduced the average CU value per cluster. This value is defined as $CU_{avg} = \frac{CU}{k}$, where k is the number of clusters. We use this measure in order to evaluate the clusterings obtained from the relational data sets.

MoJo: We use MoJo here, similar to the way in which it was used for non-structural data in Chapter 5 and in Section 5.3.1, to derive an appropriate value for the number of clusters and assess the quality of the results.

Information Loss, (IL): We also use the information loss $IL = I(A;T) - I(A;C_k)$ to compare clusterings. The lower the information loss, the better the clustering. For a clustering with low information loss, given a cluster, we can predict the attribute values of the tuples in the cluster with relatively high accuracy. We present IL as a percentage of the initial mutual information, $\frac{IL}{I(A;T)} \cdot 100\%$, lost after producing the desired number of clusters using each algorithm. However, special attention must be paid since clusterings with smaller k values tend to incur larger values of information loss. We report the value of IL as an indication of the information content of the resulting clusters.

6.4.4 Relational Data: Results and Observations

For the relational data sets, we ran LIMBO without any weights as well as in the presence of weights given by the *MI*, *LDS*, and *TF.IDF* weighting schemes. Figures 6.3, 6.4 and 6.5 depict the weight distributions of the three weighting schemes on the Votes, Mushroom

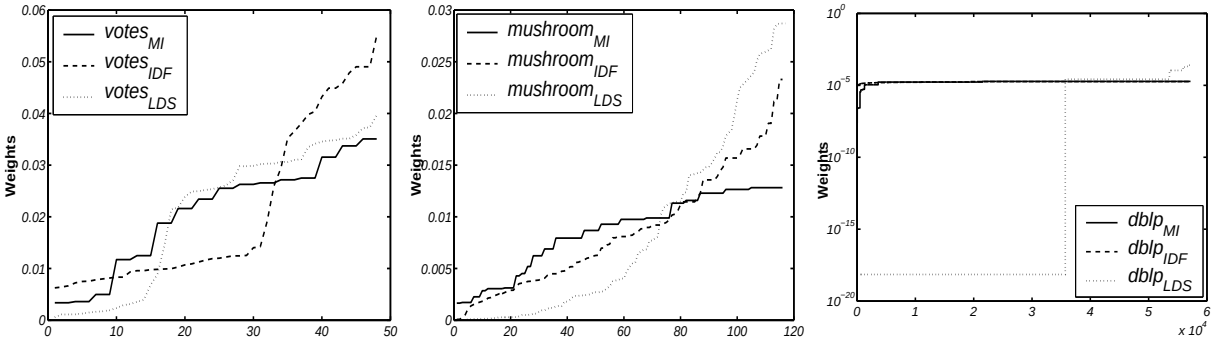


Figure 6.3: Votes weights Figure 6.4: Mushroom weights Figure 6.5: DBLP weights

and DBLP data sets, respectively. (The weight values were sorted in ascending order to facilitate visualization.)

For the Votes data set, Figure 6.3 indicates that *MI* and *LDS* assign weights in a similar fashion, although *LDS* does assign significantly smaller weights to 20% of the values. On the other hand, *TF.IDF* assigns smaller weights to about half of the values, while the weights increase for the rest of them. The latter ones correspond to YES or NO values that do not appear many times in the corresponding attributes of the data set.

The Mushroom data set contains values that are almost equally distributed in the attributes of the data set. Hence, as Figure 6.4 depicts, the *TF.IDF* scheme does not assign the highest weights as in Votes. A similar situation with respect to the weights produced by LDS is observed here as well. The weights for approximately 20% of the values are significantly lower than the rest. Finally, *MI* demonstrates behaviour similar to that in the Votes data set, which can be characterized as more conservative than the other weighting schemes.

The main lesson learned from the distribution of the weights in the DBLP data set (the y-axis is in logarithmic scale) is that *MI* and *TF.IDF* produce similar distributions of weights. On the other hand, the large number of missing values in different attributes and the large number of values in the tuples related to the same publication forced *LDS* to elicit two significantly different categories of weights.

The results of clustering the three relational data sets (without any weight transformation) are given in Table 6.8. In order to choose an appropriate number of clusters, we start by creating decompositions for all values of k between 2 and a large value. For the experiments performed for these data sets, the chosen value was 50. For these clusterings, we compute the value of CU_{avg} and choose the clustering that had the maximum CU_{avg} value, *i.e.* a clustering where values can be predicted with the highest accuracy in their corresponding clusters. The weighting schemes that performed best with respect to CU_{avg} are shown in bold.

Votes ($\phi = 0.0$)			
Scheme	k	CU_{avg}	$IL(\%)$
None	2	1.4017	73.25
MI	**2**	**1.4349**	**63.38**
LDS	**2**	**1.4397**	**71.67**
IDF	2	1.3850	77.47

Mushroom ($\phi = 0.5$)			
Scheme	k	CU_{avg}	$IL(\%)$
None	**3**	**1.0670**	**79.24**
MI	**3**	**1.0670**	**72.71**
LDS	4	1.0399	59.44
IDF	4	1.0399	58.11

DBLP ($\phi = 1.2$)			
Scheme	k	CU_{avg}	$IL(\%)$
None	3	0.4089	90.72
MI	2	0.6002	92.75
LDS	**3**	**0.6172**	**90.00**
IDF	**3**	**0.6174**	**89.04**

Table 6.8: Results for relational data sets

From these results, we observe that in the Votes data set there is hardly any difference among the three schemes. In all cases, the number of clusters with the lowest CU_{avg} is the same. *MI* and *LDS* produce slightly better quality results both with respect to the CU_{avg} and IL. A possible explanation for the similarity between the results could be the fact that the domain of all attributes is the same (YES, NO, UNKNOWN).

In the Mushroom data set the *MI* weighting scheme gave the best results. The value of CU_{avg} is the same as in the case where no weights were introduced. However, for the same number of clusters (three) in each case, *MI* resulted in a clustering with smaller IL value.

Finally, in the DBLP data set, all weighting schemes showed merit. This result is intuitive since weighting schemes balanced abnormalities, such as the high number of `NULL` values. For example, the *TF.IDF* scheme assigned a very small weight to the `NULL` values that appear almost exclusively in some attributes, driving the result of the clustering to more meaningful and informative clusters.

Votes ($\phi = 0.0$)			
Scheme	k	CU_{avg}	$IL(\%)$
None	2	1.4017	73.25
MI	2	1.4349	63.38
MI-5%	**2**	**1.4031**	**69.91**
MI-10%	2	1.1889	73.34
MI-20%	3	0.9266	65.52
LDS	2	1.4397	71.67
LDS-5%	2	1.4321	68.97
LDS-10%	**2**	**1.4393**	**68.89**
LDS-20%	2	1.4206	71.26
IDF	2	1.3850	77.47
IDF-5%	2	1.4386	76.73
IDF-10%	2	1.4426	75.82
IDF-20%	**2**	**1.4433**	**73.29**

Mushroom ($\phi = 0.5$)			
Scheme	k	CU_{avg}	$IL(\%)$
None	3	1.0670	79.24
MI	3	1.0670	72.71
MI-5%	4	1.0399	60.68
MI-10%	3	1.0666	69.09
MI-20%	**3**	**1.0670**	**69.24**
LDS	4	1.0399	59.44
LDS-5%	3	0.9719	68.71
LDS-10%	3	0.9718	65.68
LDS-20%	3	0.9717	65.50
IDF	4	1.0399	58.11
IDF-5%	4	1.0399	59.14
IDF-10%	4	1.0401	58.56
IDF-20%	**3**	**1.0666**	**69.17**

DBLP ($\phi = 1.2$)			
Scheme	k	CU_{avg}	$IL(\%)$
None	3	0.4089	90.72
MI	2	0.6002	92.75
MI-5%	3	0.6001	90.68
MI-10%	3	0.6001	92.09
MI-20%	**3**	**0.6201**	**89.24%**
LDS	3	0.6172	90.00
LDS-5%	3	0.6174	89.97
LDS-10%	3	0.6171	90.20
LDS-20%	3	0.6089	91.04
IDF	3	0.6174	89.04
IDF-5%	4	0.6171	89.45
IDF-10%	3	0.6072	90.56
IDF-20%	**3**	**0.6233**	**89.18%**

Table 6.9: Results for relational data sets with transformed weights (bold fonts show best results of transformed weights)

We also applied the same weighting schemes but with transformed weights, as explained in Section 6.3.6. The results obtained are shown in Table 6.9.

In the Votes data set, clustering results are worse when *MI* and *LDS* weight values are transformed. On the contrary, the results of *TF.IDF* are improved. *TF.IDF* is a scheme that gives higher weights to values that appear less often in the tuples. The transformed results prove that, in the case of this data set, such values are less important than the scheme presumes, and, by decreasing their value, the results are better.

The same observation holds for the case of DBLP, where single authors or conference names appear with high weights in the case of *TF.IDF*, we see that transforming the weights of these values to the weights that correspond to `NULL` values, the clusters are more informative and the publications better separated. In Mushroom, IDF-20% improves over IDF, but the result is still less good than for None and MI.

6.4.5 Market-Basket Data: Results and Observations

LIMBO was also applied to the three market-basket data sets using all weighting schemes and their variations. In the same way as in the relational data sets, in order to choose an appropriate number of clusters, we start by creating decompositions for all values of k

between 2 and a large value. For the experiments performed in market-basket data sets, the chosen value was 150, a value that turned out to be sufficient for our purposes.

Figures 6.6, 6.7, and 6.8 present the weight distribution for the three market-basket data sets and the applicable weighting schemes. In all figures, the y-axis is in logarithmic scale.

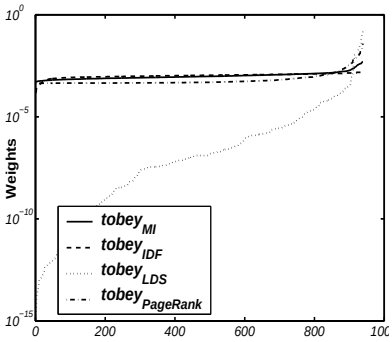


Figure 6.6: TOBEY weights

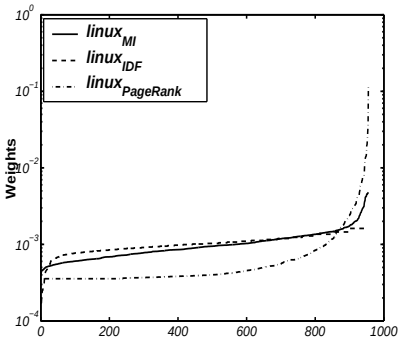


Figure 6.7: LINUX weights

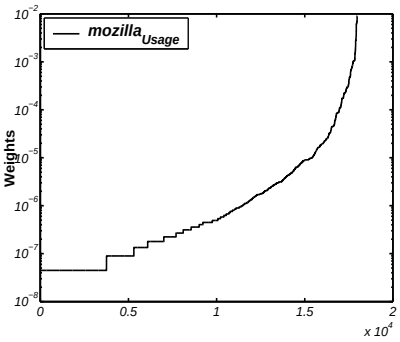


Figure 6.8: MOZILLA weights

In Figure 6.6 we present the weight distribution of all four schemes. We observe that *MI*, *TF.IDF* and *PageRank* produce weights in the same range. In the case of *LDS*, the weights produced are smaller and with a broader range. Larger weights correspond to nodes with large in- and out-degrees.

The weight distribution in the LINUX data set for all schemes follows the same pattern as in the TOBEY data set. To reveal the differences among the weights of *MI*, *TF.IDF* and *PageRank* schemes, we chose to omit the distribution of *LDS* in Figure 6.7. This figure shows that *MI* and *TF.IDF* elicit similar and more conservative weights compared to *PageRank*, which gives a high weight to a number of values. These values correspond to nodes that are pointed to by other important nodes in the graph of the Linux system. Finally, Figure 6.8 depicts only the weight distribution produced based on Mozilla usage data. The distributions for the other four schemes are similar to what they were in the previous two data sets described above. The main observation from the distribution of usage weights is that there is a wide range of weights. The smallest weights are five orders of magnitude smaller than the largest ones.

The clustering results we obtained are shown in Table 6.10. Weighting schemes performing best are shown in bold.

TOBEY ($\phi = 0.0$)			
Scheme	k	$MoJo$	$IL(\%)$
None	80	311	30.59
MI	33	341	42.94
LDS	59	476	20.61
IDF	**102**	**292**	**27.17**
PageRank	24	571	41.33

LINUX ($\phi = 0.0$)			
Scheme	k	$MoJo$	$IL(\%)$
None	56	237	36.03
MI	70	237	30.95
LDS	41	286	31.31
IDF	**81**	**225**	**20.09**
PageRank	24	340	39.66

MOZILLA ($\phi = 0.2$)			
Scheme	k	$MoJo$	$IL(\%)$
None	10	600	63.25
MI	125	428	23.64
LDS	32	528	36.31
IDF	**68**	**406**	**33.19**
PageRank	48	478	33.14
Usage	61	440	47.21

Table 6.10: Results for market-basket data sets

The *TF.IDF* weighting scheme outperforms all others, including the scheme that uses no weights. This can be attributed to the fact that the way *TF.IDF* assigns weights corresponds well to the way software architects would assign importance to artifacts of their system. Artifacts used by the majority of the system are probably library functions that are not very important (low *idf*), while artifacts rarely used are unlikely to be central to the system's structure (low *tf*).

The *LDS* weighting scheme performs quite poorly most likely because it assigns large importance to nodes of large in- and out-degree. This property is shared by the *PageRank* weighting scheme. Our results confirm that this is not a desirable property for the analysis of software data.

The usage data weighting scheme performs rather well with the Mozilla data set. Even though it is outperformed by *TF.IDF*, it still improves significantly on using the static dependency graph (represented by the None weighting scheme). Further experiments with more software systems are, of course, required to determine whether this is generally true.

Finally, the *MI* weighting scheme performs well consistently. With the exception of TOBEY, it is only slightly worse than *TF.IDF*. This might indicate that it is a weighting scheme that is not influenced by the type of data set used, a quite desirable property.

We also performed experiments with the transformed variations of the weighting

schemes. The results are shown in Table 6.11.

TOBEY ($\phi = 0.0$)

Scheme	k	$MoJo$	IL(%)
MI-5%	97	323	28.51
MI-10%	16	383	53.75
MI-20%	38	328	41.69
LDS-5%	42	486	20.61
LDS-10%	28	540	30.15
LDS-20%	37	447	34.27
IDF-5%	67	369	33.87
IDF-10%	27	333	46.41
IDF-20%	27	333	46.41
PageRank-5%	82	310	29.08
PageRank-10%	61	312	34.33
PageRank-20%	53	321	37.02
InvPageRank	**86**	**297**	**29.90**

LINUX ($\phi = 0.0$)

Scheme	k	$MoJo$	IL(%)
MI-5%	94	245	27.15
MI-10%	68	240	32.12
MI-20%	90	256	27.79
LDS-5%	52	281	28.29
LDS-10%	45	315	31.11
LDS-20%	31	305	37.02
IDF-5%	97	248	26.42
IDF-10%	47	257	37.60
IDF-20%	46	257	37.95
PageRank-5%	56	244	29.08
PageRank-10%	62	235	32.96
PageRank-20%	36	230	28.00
InvPageRank	**79**	**226**	**29.26**

MOZILLA ($\phi = 0.2$)

Scheme	k	$MoJo$	IL(%)
MI-5%	97	425	27.33
MI-10%	70	423	32.42
MI-20%	100	411	26.96
LDS-5%	100	423	26.90
LDS-10%	89	435	34.02
LDS-20%	69	452	32.84
IDF-5%	28	482	47.01
IDF-10%	132	419	23.19
IDF-20%	132	419	23.23
PageRank-5%	80	436	27.18
PageRank-10%	55	435	34.23
PageRank-20%	**98**	**407**	**27.12**
InvPageRank	91	416	28.34
Usage-5%	68	665	46.79
Usage-10%	73	673	46.80
Usage-20%	80	673	46.74
InvUsage	89	678	49.01

Table 6.11: Results for market-basket data sets with transformed weights

In agreement with our observations above, the performance of the *LDS* weighting scheme improves in certain cases. This phenomenon is even more dramatic with the *PageRank* weighting scheme. In many cases, the clustering obtained is only slightly worse than the one produced by *TF.IDF*.

Surprisingly, the *Inverse PageRank* weighting scheme produced results that were among the best. This indicates that importance for web search engines does not imply importance for clustering algorithms. In fact, quite the opposite seems to be the case.

As expected, *TF.IDF* yielded worse results when its weight structure was modified. A similar behaviour was observed for the usage data weighting scheme. This indicates that these weighting schemes in their pure form encapsulate well the properties of software decompositions.

Finally, *MI* is insensitive to the weighting scheme chosen, confirming our belief that it is a conservative, stable, and effective weighting scheme.

6.5 Conclusions

This chapter presented an evaluation of certain weighting schemes within a clustering algorithm for categorical data. We implemented and experimentally assessed the useful-

ness of such schemes on a variety of relational and market-basket data sets, the latter ones from the field of software reverse engineering.

Our approach employs the LIMBO clustering algorithm as described in Chapter 3. The only additional step required is the analysis of the data set and elicitation of value weights. From our experiments, we can reach the following general conclusions:

- When the number of clusters in a data set is small, the weighting schemes do not offer considerable merit. This is shown through our initial experiments on relational data sets.
- The *MI* weighting scheme performs consistently well in a variety of domains.
- When software graph-based data sets are clustered, the *TF.IDF* weighting scheme seems to perform best. Especially in software systems, this scheme decreases the effect of "omnipresent" nodes appropriately in order for the clusters to reflect natural groupings of the data.
- The *PageRank* weighting scheme seems to be inappropriate for clustering purposes. Interestingly, the *Inverse PageRank* weighting scheme performs well in the software clustering domain.
- The performance of the *LDS* weighting scheme is overall worse than the rest of the schemes. However, it was interesting to discover that it assigns weights in a more skewed fashion than other weighting schemes. Such a property might be desirable in domains other than the ones examined in this thesis.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we presented LIMBO, a scalable hierarchical clustering algorithm. We evaluated its effectiveness in trading off either quality for time or quality for space to achieve compact, yet accurate, models for small and large categorical data sets. We have shown that LIMBO has advantages over other information-theoretic based clustering algorithms including AIB (in terms of scalability) and COOLCAT (in terms of clustering quality and parameter stability). We have also shown advantages in quality over other scalable and non-scalable algorithms designed to cluster either categorical data objects or values. LIMBO builds a model in one pass over the data in a limited amount of memory while keeping information loss in the model close to minimal. In addition, to the best of our knowledge, LIMBO is the only scalable algorithm for clustering categorical data that is hierarchical. Using its compact summary model, LIMBO efficiently builds clusterings for a large range (typically hundreds) of values of k. Furthermore, we are also able to produce statistics that let us directly compare clusterings and select appropriate values for the number of clusters, k.

Next, we presented an approach to discovering structure. Our approach defines schema discovery as a problem where the schema of a relation is inconsistent with respect to the data, rather than the opposite. We presented a set of information-theoretic techniques based on LIMBO that discover duplicate, or almost duplicate, tuples and attribute

values in a relational instance. From the information collected about the values, we then presented an approach that groups attributes, so that differences within each group are as small as possible. The groups of attributes with large duplication provide important clues for the re-design of the schema of a relation. Using these clues, we introduced a novel approach to rank the set of functional dependencies that are valid in an instance.

We also presented the notion that information loss minimization is a valid basis for a software clustering approach. Our approach can exploit in the software clustering process any type of information relevant to the software system. We experimented with and assessed the usefulness of four different types of non-structural information.

Our final contribution was the evaluation of certain weighting schemes within the LIMBO clustering algorithm for categorical data. We implemented and experimentally assessed the usefulness of such schemes on a variety of relational and market-basket data sets, the latter ones from the field of software reverse engineering.

7.2 Future Work

Finally, our contributions in this thesis open several avenues for further research. We present the most important ones in the following sub-sections.

7.2.1 Clustering Numerical And Categorical Data

LIMBO is an approach that works well with categorical values, but it currently is not designed to deal with *mixed* data, that is, data that contains *both* categorical and numerical attributes. Simply ignoring the distinction and treating numerical attributes as categorical does not lead to effective solutions. For example in a database instance, an attribute "years" typically takes on numerical values. We could treat this attribute as a categorical attribute. However, in this case the values 1958 and 1959 are considered to be equally distant as are the values 1990 and 1947. Our distance function does not exploit the inherent ordering of numerical values.

LIMBO needs further extensions in order to handle data sets with numerical as well as categorical values well. It is our intention to propose a scheme for integrating numerical data with categorical data which will allow us to employ the same information-theoretic tools that were applied in LIMBO. Our approach should exploit the inherent geometric notion of distance between numerical attribute values. Our initial idea is to replace each numerical value by a *distribution* centered around the value. In this fashion, numerical values that are close in a geometric sense, will be close in the information-theoretic distance measure.

7.2.2 Clustering Categorical Data Streams

In many applications, data arrive in the form of continuous streams that need to be analyzed. Clustering of numerical *data streams* has recently received a great deal of attention [AHWY03, Bar02, OMM+02, GMM+03], but the problem is not well-studied for categorical data streams. In this section, we argue that LIMBO is the first scalable categorical clustering algorithm that can be used for streaming data. Requirements for the effective, and efficient clustering of streams include the following [AHWY03, Bar02]:

• *One Pass over the data*: The data points must be read in an incremental fashion and "discarded in favor of summaries whenever possible" [GMM+03].

• *Compact Representation of the clusters*: The size of data streams is usually large (or unbounded) and the model cannot grow in proportion. Typically, a bounded amount of memory is used to store useful summaries of previously seen data. The summaries must be as concise and representative of the data that has been seen as possible.

Data stream clustering algorithms usually include an *on-line* component, which accepts the data points from the stream and produces the appropriate summaries, and an *off-line* component, which maintains the summaries and produces higher-level clusters. LIMBO$_S$ satisfies both requirements for a stream clustering algorithm. It scans the data incrementally, associating each of the tuples with its closest *DCF* leaf entry and imposing

a strict bound on the size of the summary model. Phase 1 of LIMBO corresponds to the *on-line* component of the data stream clustering algorithm. Phase 2 corresponds to the *off-line* component of the algorithm. As our experiments in Chapter 3 showed, even with a small number of leaf entries (*i.e.*, small memory requirements), the quality of the clustering produced by LIMBO is very good. In a streaming environment, the original data is discarded and only the summary is kept. Hence, Phase 3 of LIMBO (which labels the original data with their cluster assignments) is not applicable.

It is instructive to compare LIMBO to other approaches for clustering numerical streams and to other approaches for clustering categorical data in order to understand how LIMBO complements and extends this work. A recent proposal for clustering streams [AHWY03] uses numerical cluster features (first proposed in BIRCH [ZRL96]) to summarize a *numerical* data stream. These cluster features cannot be used to summarize categorical data. To handle categorical streams, we can use the information-theoretic summaries of either LIMBO or COOLCAT. However, notice that the COOLCAT construction algorithm makes use of an initial sample of the whole data set. A set of k cluster representatives are chosen from this sample. Such a sample is not available in streaming environments. Furthermore, our experiments with COOLCAT demonstrated that the choice of cluster representatives has a significant effect on the quality of the algorithm. So, a sample of the initial portion of a data stream may not yield a high-quality clustering. Although LIMBO is using the same objective function as COOLCAT (the entropy of the clustering), LIMBO does not require an initial sample to initialize its summary data structure.

Finally, notice that COOLCAT is not hierarchical, rather it produces only a single clustering for a specific value of k. In a streaming environment, the goal is to produce a high-quality summary of the data. From this summary, we may, at different times, wish to understand the underlying clustering structure of the data. We can imagine that this structure may change as more data is processed. In particular, the number of clusters

in the data may change. LIMBO is designed to build and efficiently manage a summary structure significantly larger than that required for producing the best clustering for a static data set. This summary structure can be post-processed to understand the current clustering properties of the data stream. While COOLCAT could be used in a similar way (by setting k to be large), its data structures are not designed to efficiently manage a large, in-memory summary. Namely, the execution time for COOLCAT is proportional to the size of the summary it maintains.

7.2.3 Evaluating Other Structure Discovery Techniques

In Chapter 4, we described a particular aspect of the problem of reorganizing a large data set, that of ranking a set of functional dependencies. The ranking produced by Fd-rank is based on a clustering of the attributes of the data set. However, clustering may be too restrictive for this application. For instance, once we have grouped attributes A and B, we cannot then consider combination A and C, which may have high redundancy (although lower than that of A and B). If AB does not participate in an FD, then the high redundancy of AC might be missed.

We plan to investigate other techniques that decompose a relation with duplicate values. More precisely, we will focus on techniques that have been proposed as part of *Reconstructibility Analysis* [Kri86]. In brief, Reconstructibility Analysis proposes methods for decomposing a set of records defined on a number features into *simpler* sets of records, such that the information content of the decomposed sets is as close as possible to the original information content. To perform the decomposition, algorithms are proposed that navigate through a lattice of models that cover all the features of the initial data set and try to find the ones that produce the original set when merged, and at the same time are simpler, *i.e.*, require fewer degrees of freedom to be described.

7.2.4 Clustering and Histograms

Histograms are used to summarize the numerical values of data sets and, among other applications, speed up the process of query answering. Ioannidis [Ioa03] draws similarities between histograms and clustering. He speculates that the techniques that have been proposed for histograms and clustering can be brought together, and that the advantages and disadvantages of each one can be studied together. We are planning to study the unification of histograms and clustering techniques.

Besides the previous observations, we are also planning to investigate whether techniques for producing histograms of categorical data are feasible. The difficulty in building such histograms is the ordering of the data. A potential histogram of categorical data must be accompanied by a mapping of each of the values to the histogram buckets. Because that mapping is typically very large, the main advantage of histograms may be lost.

7.2.5 Other LIMBO Studies

Finally, we plan to study different properties of the LIMBO algorithm and extend its applicability. Namely we suggest the following two points of future research:

1. **Soft Clustering**: In the clustering framework we studied in the thesis, every object from the initial data sets is assigned to one and only one cluster. Several pieces of work present clustering algorithms that assign objects to more than one cluster with an associated probability. Since the initial proposal of the IB method includes all that is needed for such an approach, we plan to perform the so called *soft* clustering and evaluate any merits with respect to the current approach.

2. **Performance of LIMBO in high dimensions**: In this thesis, we experimentally argued that the problem of clustering categorical data becomes harder as dimensionality grows. We plan to study the properties of the function used to assess the

information loss between clustered objects and provide theoretical justifications of its properties.

3. **LIMBO Stability**: Data sets evolve and, as a consequence, their interdependencies and groupings change. One of the desired properties of cluster analysis is that the clustering remain stable whenever the data changes by only a small amount, that is when new data objects are inserted and existing ones change or are deleted. But, how stable is LIMBO? If the data set analyzed consists of daily transactions or daily software releases 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 in general 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, specifically LIMBO, 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.

4. **Comparison of Quality Measures**: Throughout this thesis we used a variety of measures in order to assess the results of LIMBO. We are planning to study the similarities and differences of the different quality measures. For instance, a variety of measures [KE00, MM01, AL99, LG95] have been proposed in the Reverse Engineering community besides MoJo and we are interested in exploring their usefulness in our work.

Bibliography

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

[ACN01] Sanjay Agrawal, Surajit Chaudhuri, and Vivek R. Narasayya. Materialized View and Index Selection Tool for Microsoft SQL Server 2000. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, page 608, Sanata Barbara, CA, USA, 21–24 May 2001.

[AFF+02] Periklis Andritsos, Ron Fagin, Ariel Fuxman, Laura M. Haas, Mauricio A. Hernandez, C. Ho, Anastasios Kementsietsidis, Renée J. Miller, Felix Naumann, Lucian Popa, Yannis Velegrakis, Charlotte Vilarem, and Ling-Ling Yan. Schema Management. *IEEE Data Engineering Bulletin*, 25(3): 32–38, September 2002.

[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 ACM SIGMOD International Conference on the Management of Data*, pages 94–105, Seattle, WA, USA, 1–4 June 1998.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[AHWY03] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Clustering Evolving Data Streams. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 81–92, Berlin, Germany, 9–12 September 2003.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 207–216, Washington, D.C., USA, 26–28 May 1993.

[AL97] Nicolas Anquetil and Timothy Lethbridge. File Clustering Using Naming Conventions for Legacy Systems. In *Proceedings of CASCON 1997*, pages 184–195, Toronto, Canada, 10–13 November 1997.

[AL99] Nicolas Anquetil and Timothy Lethbridge. Experiments with Clustering as a Software Remodularization Method. In *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE)*, pages 235–255, Atlanta, GA, USA, 6–8 October 1999.

[AL03] Marcelo Arenas and Leonid Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 15–26, San Diego, CA, USA, 9–12 June 2003.

[AM01] Periklis Andritsos and Renée J. Miller. Reverse Engineering Meets Data Analysis. In *Proceedings of the 9th International Workshop on Program Comprehension (IWPC)*, pages 157–166, Toronto, ON, Canada, 12–13 May 2001.

[AM03] Periklis Andritsos and Renée J. Miller. Using Categorical Clustering in Schema Discovery. In *IJCAI Workshop on Information Integration on the Web (IIWeb-03)*, page 211, Acapulco, Mexico, 2003.

[AMT04] Periklis Andritsos, Renée J. Miller, and Panayiotis Tsaparas. Information-Theoretic Tools for Structure Discovery in Large Data Sets. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 731–742, Paris, France, 13–18 June 2004.

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

[AT03] Periklis Andritsos and Vassilios Tzerpos. Software Clustering based on Information Loss Minimization. In *Proceedings of the 10th Working Conference on Reverse Engineering (WCRE)*, pages 334–344, Victoria, BC, Canada, 13–16 November 2003.

[AT04] Periklis Andritsos and Vassilios Tzerpos. Evaluating Value Weighting Schemes in the Clustering of Categorical Data. *Submitted for publication*, 2004.

[ATMS04] Periklis Andritsos, Panayiotis Tsaparas, Renée J. Miller, and Kenneth C. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT)*, pages 123–146, Heraklion, Greece, 14–18 March 2004.

[Bar02] Daniel Barbará. Requirements for Clustering Data Streams. *SIGKDD Explorations*, 3(2): 23–27, January 2002.

[BCL02a] Daniel Barbará, Julia Couto, and Yi Li. An Information Theory Approach to Categorical Clustering. *Submitted for Publication*, 2002.

[BCL02b] Daniel Barbará, Julia Couto, and Yi Li. COOLCAT: An Entropy-based Algorithm for Categorical Clustering. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKM)*, pages 582–589, McLean, VA, USA, 4–9 November 2002.

[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.

[BH98] Ivan T. Bowman and Richard C. Holt. Software Architecture Recovery Using Conway's Law. In *Proceedings of CASCON 1998*, pages 123–133, Toronto, ON, Canada, 30 November– 3 December 1998.

[BH99] Ivan T. Bowman and Richard C. Holt. Reconstructing Ownership Architectures to Help Understand Software Systems. In *Proceedings of the 7th International Workshop on Program Comprehension (IWPC)*, pages 28–37, Pittsburgh, PA, USA, 5–7 May 1999.

[BHB99] Ivan T. Bowman, Richard C. Holt, and Neil V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In *Proceedings of the 21st International Conference on Software Engineering (ICSE)*, pages 555–563, Los Angeles, CA, USA, 16–22 May 1999.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering*, pages 431–440, San Jose, CA, USA, 26 February–1 March 2002.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Large-scale Hypertextual Web Search Engine. *Computer Networks*, 30(1–7):107–117, 1998.

[BR99] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley-Longman, 1999.

[BRRT01] Allan Borodin, Gareth O. Roberts, Jeffrey S. Rosenthal, and Panayiotis Tsaparas. Finding Authorities and Hubs From Link Structures on the World

Wide Web. In *Proceedings of the 10th International World Wide Web Conference (WWW)*, pages 415–429, Hong Kong, China, 1–5 May 2001.

[CDF+82] Arvola Chan, Sy Danberg, Stephen Fox, Wen-Te K. Lin, Anil Nori, and Daniel Ries. Storage and Access Structures to Support a Semantic Data Model. In *Proceedings of the 8th International Conference on Very Large Data Bases (VLDB)*, pages 122–130, Mexico City, Mexico, 8-10 September 1982.

[CI90] Elliot J. Chikofsky and James H. Cross II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software*, 7(1):13–17, January 1990.

[CS90] Song C. Choi and Walt Scacchi. Extracting and Restructuring the Design of Large Systems. *IEEE Software*, 7(1): 66–71, January 1990.

[CT91] Tomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley & Sons, 1991.

[De 87] Paul De Bra. *Horizontal Decompositions in the Relatinal Database Model*. PhD thesis, Universiteit Antwerpen, 1987.

[DJ03] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., 2003.

[DJMS02] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining Database Structure; or, How to Build a Data Quality Browser. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 240–251, Madison, WI, USA, 3–6 June 2002.

[DM00] Gautam Das and Heikki Mannila. Context-Based Similarity Measures for Categorical Databases. In *Proceedings of the 4th European Conference on*

Principles of Data Mining and Knowledge Discovery (PKDD), pages 201–210, Lyon, France, 13-16 September 2000.

[DMM03] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-Theoretic Co-clustering. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 89–98, Washington, DC, USA, 24–27 August 2003.

[DP83a] Paul De Bra and Jan Paredaens. An Algorithm for Horizontal Decompositions. *Information Processing Letters*, 17: 91–95, 1983.

[DP83b] Paul De Bra and Jan Paredaens. Horizontal Decompositions for Handling Exceptions to Functional Dependencies. *Advances in Database Theory*, 2: 123–144, 1983.

[DR00] Mehmet M. Dalkilic and Edward Robertson. Information Dependencies. In *Proceedings of the 19th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 245–253, Dallas, TX, USA, 15–17 May 2000.

[DS03] Franca Debole and Fabrizio Sebastiani. Supervised Term Weighting for Automated Text Categorization. In *Proceedings of the 2003 ACM Symposium on Applied Computing (SAC)*, pages 784–788, Melbourne, FL, USA, 9–12 March 2003.

[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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 226–231, Portland, OR, USA, 2–4 August 1996.

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

[EYS01] Ran El-Yaniv and Oren Souroujon. Iterative Double Clustering for Unsupervised and Semi-supervised Learning. In *Proceedings of the 12th European Conference on Machine Learning, (ECML)*, pages 121–132, Freiburg, Germany, 3-7 September 2001.

[Fis87] Douglas H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2: 139–172, 1987.

[GC85] Mark Gluck and James Corter. Information, Uncertainty, and the Utility of Categories. In *Proceedings of the 7th Annual Conference of the Cognitive Science Society (COGSCI)*, pages 283–287, Irvine, CA, USA, 1985.

[GIKS03] Luis Gravano, Panagiotis Ipeirotis, Nick Koudas, and Divesh Srivastava. Text Joins in an RDBMS for Web Data Integration. In *Proceedings of the 12th International World Wide Web Conference (WWW)*, pages 90–101, Budapest, Hungary, 20–24 May 2003.

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

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[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.

[GL00] Michael W. Godfrey and Eric H. S. Lee. Secrets from the Monster: Extracting Mozilla's Software Architecture. In *Proceedings of the 2nd International Symposium on Constructing Software Engineering Tools (CoSET 2000)*, Limerick, Ireland, 5 June 2000.

[GMM+03] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3): 515–528, June 2003.

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

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

[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.

[HB85] David H. Hutchens and Victor R. Basili. System Structure Analysis: Clustering with Data Bindings. *IEEE Transactions on Software Engineering*, 11(8): 749–757, August 1985.

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

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

[HKPT99] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *The Computer Journal*, 42(2): 100–111, 1999.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 670–681, Hong Kong, China, 20–23 August 2002.

[HS75] Jeffrey A. Hoffer and Dennis G. Severance. The Use of Cluster Analysis in Physical Data Base Design. In *Proceedings of the 1st International Conference on Very Large Data Bases (VLDB)*, pages 69–86, Framingham, MA, USA, 1975.

[HS95] Mauricio A. Hernández and Salvatore J. Stolfo. The Merge/Purge Problem for Large Databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 127–138, San Jose, CA, USA, 22–25 May 1995.

[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.

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

[Hul84] Richard Hull. Relative Information Capacity of Simple Relational Database Schemata. In *Proceedings of the 3rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 97–109, Waterloo, ON, Canada, 2–4 April 1984.

[Ioa03] Yannis E. Ioannidis. The History of Histograms (Abridged). In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 19–30, Berlin, Germany, 9–12 September 2003.

[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.

[KE00] Rainer Koschke and Thomas Eisenbarth. A Framework for Experimental Evaluation of Clustering Techniques. In *Proceedings of the 8th International Workshop on Program Comprehension (IWPC)*, pages 201–210, Limerick, Ireland, 10-11 June 2000.

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

[Kos00] Rainer Koschke. *Atomic Architectural Component Recovery for Program Understanding and Evolution*. PhD thesis, Institute for Computer Science, University of Stuttgart, 2000.

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

[Kri86] Klaus Krippendrof. *Information Theory: Structural Models for Qualitative Data*. Quantitative Applications in the Social Sciences. Sage Publications, 1986.

[LG95] Arun Lakhotia and John M. Gravley. Toward Experimental Evaluation of Subsystem Classification Recovery Techniques. In *Proceedings of the 2nd Working Conference on Reverse Engineering (WCRE)*, pages 262–269, Toronto, ON, Canada, 14-16 July 1995.

[LM98] Huan Liu and Hiroshi Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[Lut02] Rudi Lutz. Recovering High-Level Structure of Software Systems Using a Minimum Description Length Principle. In *Proceedings of the 13th Irish Conference on Artificial Intelligence and Cognitive Science (AICS)*, pages 61–69, Limerick, Ireland, 12–13 September 2002.

[MA03] Renée J. Miller and Periklis Andritsos. On Scehma Discovery. *IEEE Data Engineering Bulletin*, 26(3): 341–47, September 2003.

[Mai80] David Maier. Minimum Covers in Relational Database Model. *Journal of the ACM*, 27(4): 664–674, October 1980.

[MHH+01] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan amd C.T. Howard Ho, Ronald Fagin, and Lucian Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 30(1): 78–83, March 2001.

[MM01] Brian S. Mitchell and Spiros Mancoridis. Comparing the Decompositions Produced by Software Clustering Algorithms Using Similarity Measurements. In *Proceedings of the International Conference on Software Maintenance (ICSM)*, pages 744–753, Florence, Italy, 6-10 November 2001.

[MMCG99] Spiros Mancoridis, B.S. Mitchell, Y. Chen, and E.R. Gansner. Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures. In *Proceedings of the International Conference on Software Mainte-*

nance (ICSM), pages 50–66, Oxford, England, UK, 30 August–3 September 1999.

[MMM93] Ettore Merlo, Ian McAdam, and Renato De Mori. Source code informal information analysis using connectionist models. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1339–1344, Chambéry, France, 28 August–3 September 1993.

[MOTU93] Hausi A. Müller, Mehmet A. Orgun, Scott R. Tilley, and James S. Uhl. A Eeverse Engineering Spproach to Subsystem Structure Identification. *Journal of Software Maintenance: Research and Practice*, 5: 181–204, December 1993.

[MS03] Dharmendra S. Modha and W. Scott Spangler. Feature Weighting in k-Means Clustering. *Machine Learning*, 52(3): 217–237, September 2003.

[NCWD84] Shamkant B. Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems (TODS)*, 9(4): 680–710, December 1984.

[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.

[NR89] Shamkant B. Navathe and Minyoung Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 440–450, Portland, OR, USA, 31 May–2 June 1989.

[OMM+02] Liadan O'Callaghan, Adam Meyerson, Rajeeve Motwani, Nina Mishra, and Sudipto Guha. Streaming-Data Algorithms For High-Quality Clustering. In

Proceedings of the 18th International Conference on Data Engineering, pages 685–696, San Jose, CA, USA, 26 February–1 March 2002.

[PA04] Stratos Papadomanolakis and Anastassia Ailamaki. AutoPart: Automating Schema Design for Large Scientific Databases Using Data Partitioning. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM)*, pages 383–392, Santorini Island, Greece, 21-23 June 2004.

[Par72] David L. Parnas. On the Criteria to be used in decomposing Systems into Modules. *Communications of the ACM*, 15: 1053–1058, December 1972.

[PF03] Christopher R. Palmer and Christos Faloutsos. Electricity Based External Similarity of Categorical Attributes. In *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining, (PAKDD)*, pages 486–500, Seoul, Korea, 30 April-2 May 2003.

[PVM+02] Lucian Popa, Yiannis Velegrakis, Renée J. Miller, Mauricio Hernáandez, and Ronald Fagin. Translating Web Data. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 598–609, Hong Kong, China, 20–23 August 2002.

[RD02] Ravishankar Ramamurthy and David J. DeWitt. A Case for Fractured Mirrors. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 430–441, Hong Kong, China, 20–23 August 2002.

[RH01] Vijayshankar Raman and Joseph M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*, pages 381–390, Roma, Italy, 11–14 September 2001.

[SB88] Gerard Salton and Chris Buckley. Term-weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5): 513–523, 1988.

[SB02] Sunita Sarawagi and Anuradha Bhamidipaty. Interactive Deduplication using Active Learning. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 269–278, Edmonton, AB, Canada, 23–26 July 2002.

[Sch91] Robert W. Schwanke. An Intelligent Tool for Re-engineering Software Modularity. In *Proceedings of the 13th International Conference on Software Engineering (ICSE)*, pages 83–92, Austin, TX, USA, 13–17 May 1991.

[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.

[SE00] Sunita Sarawagi-(Editor). *Special Issue on Data Cleaning*. IEEE Data Engineering Bulletin, Volume 23(4), December 2000.

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

[SF93] Iztok Savnik and Peter A. Flach. Bottom-up Induction of Functional Dependencies from Relations. In *Proceedings of the AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 174–185, Washington, DC, USA, 11–12 July 1993.

[SF00] Iztok Savnik and Peter A. Flach. Discovery of Mutlivalued Dependencies from Relations. *Intelligent Data Analysis Journal*, 4(3–4): 195–211, 2000.

[SP89] Robert W. Schwanke and Michael A. Platoff. Cross References are Features. In *Proceedings of the 2nd International Workshop on Software Configuration Management (SCM)*, pages 86–95, Princeton, NJ, USA, 24 October 1989.

[ST99] Noam Slonim and Naftali Tishby. Agglomerative Information Bottleneck. In *Advances in Neural Information Processing Systems 12 (NIPS-12)*, pages 617–623, Denver, CO, USA, 29 November–4 December 1999.

[Tal99] Luis Talavera. Feature Selection as a Preprocessing Step for Hierarchical Clustering. In *Proceedings of the 16th International Conference on Machine Learning (ICML)*, pages 389–397, Bled, Slovenia, 27–30 June 1999.

[TH99] Vassilios Tzerpos and Richard C. Holt. MoJo: A Distance Metric for Software Clusterings. In *Proceedings of the 6th Working Conference on Reverse Engineering (WCRE)*, pages 187–193, Atlanta, GA, USA, 6–8 October 1999.

[TH00] Vassilios Tzerpos and Richard C. Holt. ACDC: An Algorithm for Comprehension-Driven Clustering. In *Proceedings of the 7th Working Conference on Reverse Engineering (WCRE)*, pages 258–267, Brisbane, Australia, 23–25 November 2000.

[TPB99] Naftali Tishby, Fernando C. Pereira, and William Bialek. The Information Bottleneck Method. In *Proceedings of the 37th Annual Allerton Conference on Communication, Control and Computing*, pages 368–387, Urban-Champaign, IL, USA, 22–24 September 1999.

[WGR01] Catharine M. Wyss, Chris Giannella, and Edward L. Robertson. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependen-

cies from Relation Instances. In *Proceedings of the 3rd International Conference on Data Warehousing and Knowledge Discovery (DaWaK)*, pages 101–110, Munich, Germany, 5–7 September 2001.

[WT03] Zhihua Wen and Vassilios Tzerpos. An Optimal Algorithm for MoJo Distance. In *Proceedings of the Eleventh International Workshop on Program Comprehension (IWPC)*, pages 227–235, Portland, OR, USA, 10–11 May 2003.

[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.

[Xia04] Chenchen Xiao. Software Clustering Using Static and Dynamic Data. Master's thesis, Department of Computer Science, York University, in preparation, 2004.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pages 103–114, Montreal, QB, Canada, 4–6 June 1996.

Appendix A

List of Symbols

List of Symbols	
Symbol	Meaning
$\mathcal{D}$	Database
n	Number of tuples
m	Number of attributes
d	Number of all attribute values
d_i	Number of attribute values of attribute i, $1 \leq i \leq m$
k	Number of clusters, $k \leq n$
X	Random variable of objects to be clustered
$\mathbf{X}$	Set from which X takes its values
Y	Random variable of the feature space (values)
$\mathbf{Y}$	Set from which Y takes its values
C	Random variable of clusters
$\mathbf{C}$	Set from which C takes its values
T	Random variable of the tuples
$\mathbf{T}$	Set of tuples from which T takes its values
V	Random variable of all the attribute values
$\mathbf{V}$	Set of all attributes values from which V takes its values

Symbol	Meaning
A	Random variable of all the attributes
$\mathbf{A}$	Set of all attributes from which A takes its values
V_i	Random variable of attribute value i, $1 \leq i \leq m$
$\mathbf{V}_i$	Set of attributes values from which V_i takes its values
c_i	A non-empty cluster, $1 \leq i \leq k$
c^*	New cluster after merging clusters c_i and c_j
$\delta I(c_i, c_j)$	Information loss between clusters c_i and c_j
M	Matrix of the initial data set
O	Matrix of the frequency of attribute values in their corresponding attributes
S	Maximum buffer size for the DCF tree
E	Maximum buffer size of a DCF entry
B	Branching factor of the DCF tree
ϕ	Parameter controlling information loss incurred when DCF entries are merged
ϕ_T	The value of ϕ used in tuples clustering
ϕ_V	The value of ϕ used in attribute value clustering
ϕ_A	The value of ϕ used in attribute clustering
ϕ	Parameter controlling information loss incurred when DCF entries are merged
LIMBO$_\phi$	LIMBO version using *phi*
LIMBO$_S$	LIMBO version using S
A'	Random variable of the attribute of interest in Intra-Attribute value clustering
$\mathbf{A}'$	The set of values of A'
$\tilde{\mathbf{A}}$	Set of remaining attribute in Intra-Attribute value clustering, $\tilde{\mathbf{A}} = \mathbf{A} \setminus \mathbf{A}'$
IL	Information Loss
CU	Category Utility
E_{min}	Min Classification Error

Symbol	Meaning
P	Precision
R	Recall
C_T	Set of tuple clusters
C_V	Set of attribute value clusters
C_V^D	Set of duplicate groups of attribute values
C_V^{ND}	Set of non-duplicate groups of attribute values
A^D	Set of attributes expressed over the members of C_V^D
F	Matrix of the attributes in A^D expressed over the members of C_V^D
ψ	Parameter used in the ranking of functional dependencies
$\mathcal{RAD}$	Relative Attribute Duplication
$\mathcal{RTR}$	Relative Tuple Reduction
$TF.IDF$	Weighting Scheme based on *Term Frequency-Inverse Document Frequency*
MI	Weighting Scheme based on *Mutual Information*
LDS	Weighting Scheme based on *Linear Dynamical Systems*
$w(v)$	Weight of value v