LIMBO: A Linear Algorithm to Cluster Categorical Data

University of Toronto, Department of Computer Science

CSRG Technical Report 467

Periklis Andritsos Panayiotis Tsaparas Renée J. Miller Kenneth C. Sevcik

{periklis,tsap,miller,kcs@cs.toronto.edu}

March 6, 2003

Abstract

Clustering is a problem of great practical importance in numerous applications. The problem of clustering becomes more challenging when the data is categorical, that is, when there is no inherent distance measure between data values. In this work, 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. We use the IB method to cluster both tuples and attribute values. While the IB method has been applied before to cluster small data sets, LIMBO is the first scalable hierarchical clustering algorithm to use this method. We present the first study of how clustering quality in IB-based algorithms compares to other categorical clustering algorithms. LIMBO supports a tradeoff between computation time and clustering quality. It handles large data sets efficiently and it is not affected by the order of the tuples in the data set. LIMBO is a hierarchical algorithm that produces clusterings for a range of k values (where k is the number of clusters). We take advantage of this feature to examine heuristics for selecting good clusterings (with natural values of k) within this range.

1 Introduction

Clustering is a problem of great practical importance that has been the focus of substantial research in several domains for decades. As storage capacities grow, we have at hand larger amounts of data available for analysis, and mining. Clustering plays an instrumental role in this process. This trend has created a surge of research activity in the construction of clustering algorithms that can handle large amounts of data, and produce results of high quality.

Clustering is defined as the problem of partitioning data objects into groups, such that objects in the same group are similar, while objects in different groups are dissimilar. This definition assumes that there is some well defined notion of *similarity*, or *distance* between data objects. 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). The definition of distance allows us to define a *quality measure* for a clustering (*e.g.*, the mean square distance between each point and the centroid of its cluster). Clustering then becomes the problem of grouping together points such that the quality measure is optimized.

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, many 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 1. 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 predict the attribute values associated with objects of the cluster accurately. The quality measure of the clustering is then the mutual information of the clusters and

the attribute values. 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 1: An instance of the movie database

Consider partitioning of the tuples in Table 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 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), and similarly for the actor.

This intuitive idea was formalized by Tishby, Pereira and Bialek [17]. 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 paper, we consider the application of the IB method to the problem of clustering large data sets of categorical data.

In particular, in this paper, we make the following contributions.

- We formulate the categorical clustering problem within the Information Bottleneck framework, and define

dissimilarity between categorical data objects based on the IB method.

- We propose LIMBO, the first scalable hierarchical algorithm for clustering categorical data based on the IB method. As a result of its hierarchical approach, LIMBO allows us in a single execution to consider clusterings of various size.
- We examine heuristics based on information theoretic measures to determine the numbers of clusters into which a given data set can be most naturally partitioned.
- We empirically evaluate the quality of clusterings produced by LIMBO relative to other categorical clustering algorithms including IB, ROCK [11], STIRR [10], and COOLCAT [3]. We compare the clusterings based on several different quality metrics, and the algorithms with respect to scalability and efficiency.
- We use LIMBO to cluster tuples in both relational and market-basket data, and to cluster attribute values. The distance between attribute values with the IB method quantifies the degree of interchangeability of attribute values within a single attribute.

The rest of the paper is structured as follows. In Section 2, we present the IB method, and in Section 3, we describe how to apply IB to the problem of clustering categorical data. In Section 4, we introduce LIMBO, and Section 5 presents the experimental evaluation of LIMBO and other algorithms. Section 6 addresses the problem of identifying a natural number of clusters in a given data set. Section 7 describes related work on categorical clustering, and Section 8 concludes the paper.

2 Background

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

2.1 Information Theory basics

The following definitions can be found in any information theory textbook, e.g. [6]. 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 the minimum 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. 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, the *mutual information*, $I(X;Y)$, quantifies the amount of information that the variables hold about each other. The mutual information between two variables 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.

2.2 The Information Bottleneck Method

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, 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 $p(Y|X = x)$. 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 follow:

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

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

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

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 $I(Y;X)$ measures the information about the values in $\mathbf{Y}$ provided by the identity of a specific element of $\mathbf{X}$, $I(Y;C)$ measures the information about the values in $\mathbf{Y}$ provided by the identity of a cluster in $\mathbf{C}$. The higher $I(Y;C)$,

the more informative the cluster identity is about the values in $\mathbf{Y}$ contained in the cluster. In the formalization of Tishby, Pereira and Bialek [17], 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 $I(Y;C_k)$. 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. [17] 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 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.

Finding the optimal clustering is an NP-complete problem [9]. Slonim and Tishby [15] 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 $\mathbf{X}$, $I(Y;C) = I(Y;X)$; that is, the clusters in $\mathbf{C}_n$ contain the same information for 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, $I(Y;C_{\ell-1}) \leq I(Y;C_\ell)$. The clusters c_i and c_j to be merged are chosen to minimize the information loss in moving from clustering $\mathbf{C}_\ell$ to clustering $\mathbf{C}_{\ell-1}$. This information loss is given by $\delta I(c_i,c_j) = I(Y;C_\ell) - I(Y;C_{\ell-1})$. We can also view the information loss as the increase in the uncertainty. Recall that $I(Y;C) = H(Y) - H(Y|C)$. Since the value $H(Y)$ is independent of the clustering $\mathbf{C}$, maximizing the mutual information $I(Y;C)$ 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

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

$$p(c^*) = p(c_i) + p(c_j) \tag{2}$$

$$p(y|c^*) = \frac{p(c_i)}{p(c^*)} p(y|c_i) + \frac{p(c_j)}{p(c^*)} p(y|c_j) \ . \tag{3}$$

Tishby et al. [17] show that

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

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

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

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. 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 $\mathbf{C}_{\ell-1}$.

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.

3.1 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 $\mathbf{A}_i = \{A_i.v_1, A_i.v_2, \ldots, A_i.v_{d_i}\}$ so that identical values from different attributes are treated as distinct values. A tuple $t \in \mathbf{T}$ takes exactly one value from the set $\mathbf{A}_i$ for the i^{th} attribute. Let $\mathbf{A} = \mathbf{A}_1 \cup \cdots \cup \mathbf{A}_m$ denote the set of all possible attribute values. Let $d = d_1 + d_2 + \cdots + d_m$ denote the size of $\mathbf{A}$. 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. $M[t, a] = 1$, if tuple t contains attribute value a, and zero otherwise. Since every tuple contains one value for each attribute, each tuple vector contains exactly m 1's.

Now, let T and A be random variables that range over sets $\mathbf{T}$ and $\mathbf{A}$ respectively. Following the formalism of Section 2, we define $p(t) = 1/n$, and we normalize the matrix M so that the t^{th} row holds the conditional probability

distribution $p(A|t)$. Since each tuple contains exactly m attribute values, for some $a \in \mathbf{A}$, $p(a|t) = 1/m$ if a appears in tuple t, and zero otherwise. Table 2 shows the normalized matrix M for the movie database example.[1] 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.

	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 2: The normalized movie table

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 define a random variable for every attribute A_i. We will now show that, in applying the Information Bottleneck method to the case of relational data, considering all attributes together is equivalent to considering each attribute independently.

Let A_i be a random variable that ranges over the set $\mathbf{A}_i$. For some $a \in \mathbf{A}_i$, and some $t \in \mathbf{T}$ we use $p(a|t)$ to denote the conditional probability $p(A = a|t)$, and $p_i(a|t)$ to denote the conditional probability $p(A_i = a|t)$. Also let $p(a)$ denote $p(A = a)$, and $p_i(a)$ denote $p(A_i = a)$. Since each tuple takes exactly one value in each attribute, $p_i(a|t) = 1$, if a appears in t, and zero otherwise. From the definitions in Section **??** we have that $p(a|t) = \frac{1}{m} p_i(a|t)$, for all $1 \leq i \leq m$, $a \in \mathbf{A}_i$, and $t \in \mathbf{T}$. It follows that $p(a) = \frac{1}{m} p_i(a)$. Furthermore, let c denote a cluster, and let $|c|$ denote the number of tuples in c. Since $p(a|c) = \frac{1}{|c|} \sum_{t \in c} p(a|t)$, and $p_i(a|c) = \frac{1}{|c|} \sum_{t \in c} p_i(a|t)$ we have that $p(a|c) = \frac{1}{m} p_i(a|c)$. 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

$$H(A) \quad = \quad \frac{1}{m} \sum_{i=1}^{m} H(A_i) + \log m$$

[1] We use abbreviations for the attribute values. For example d.H stands for director.Hitchcock.

$$H(A|C_k) = \frac{1}{m}\sum_{i=1}^{m} H(A_i|C_k) + \log m$$
$$I(A;C_k) = \frac{1}{m}\sum_{i=1}^{m} I(A_i;C_k) \ .$$

Given that $C_n \equiv T$, the information loss for some clustering C can be expressed as

$$I(A;T) - I(A;C) = \frac{1}{m}\sum_{i=1}^{m}(I(A_i;T) - I(A_i;C)) \ .$$

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

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 tuples are sets of values of a single attribute, and each tuple may contain a different number of values. This is what distinguishes market basket from relational data where tuples contain one value from each of a fixed number of distinct attributes. In the case of market-basket data, the input to our problem is a set $\mathbf{T}$ of n tuples on a single attribute A, with domain $\mathbf{A}$. Tuple t_i contains d_i values. If d is the size of the domain $\mathbf{A}$, 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. $M[t,a] = 1$, if tuple t contains attribute value a, and zero otherwise. The vector for tuple t_i contains exactly d_i 1's.

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

$$p(t_i) = 1/n$$
$$p(a|t_i) = \begin{cases} 1/d_i & \text{if } a \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \ .$$

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

3.3 Intra-Attribute Value distance

In categorical data there is no inherent distance between attribute values. For example, in the movie database instance, if we are given the values "Scorsese" and "Coppola" it is not apparent how to access their similarity, or dissimilarity. In order to compare attribute values, we need to place these values within a *context*. In this case the context is defined by the actors each director has worked with, and the genre of the films that it has directed. We will now describe how to use the IB method to formalize these ideas and define a dissimilarity measure between values of the same attribute. The idea is that two attribute values are similar if the context in which they appear is similar. The context is defined by 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_i be the attribute of interest, and let $\mathbf{A}_i$ denote the set of values of attribute A_i. Also let $\tilde{\mathbf{A}} = \mathbf{A} - \mathbf{A}_i$ denote the set of attribute values for the remaining attributes. Let A_i and $\tilde{A}$ be random variables that range over $\mathbf{A}_i$ and $\tilde{\mathbf{A}}$ respectively, and let $p(a|v)$ denote the distribution that variable $v \in A_i$ induces on values in $\tilde{\mathbf{A}}$. For some $v \in A_i$, and $a \in \tilde{\mathbf{A}}$, $p(a|v)$ is the fraction of the tuples in $\mathbf{T}$ that contain v, which also contain value a. For two values $v_1, v_2 \in A_i$, we define the distance between v_1 and v_2 to be the information loss $\delta I(v_1, v_2)$, which measures the information we lose 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 value $v_1 \vee v_2$.

Our definition of distance follows the underlying motivation of the work of Das and Mannila [7], where the similarity of two objects is measured by the degree of their interchangeability. We believe that our approach gives a natural quantification of the concept of interchangeability. In the movie database example, Scorsese and Coppola appear to be completely interchangable. The same holds for the actors Cary Grant and James Stewart[2].

Intra-Attribute Clustering. The definition of a distance measure between categorical values is in itself an important step towards imposing some structure on categorical data. We also consider the application of the intra-attribute distance to clustering of intra-attribute values. Given the joint distribution of random variables A_i and $\tilde{A}$ we can apply the Information Bottleneck method for clustering the values of attribute A_i. Intra-attribute values lend themselves

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

to clustering, because they do not appear together. Therefore, given two values $v_1, v_2 \in A_i$, $p(v_1 \vee v_2) = p(v_1) + p(v_2)$, where for some value v, $p(v)$ is the fraction of tuples in $\mathbf{T}$ that contain the value v. The distribution induced by $v_1 \vee v_2$ is the weighted sum of the distributions induced by v_1 and v_2, that is

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

4 Linear InforMation BOttleneck *(LIMBO)* Clustering

In Section 3, we described how we can express tuples as distributions over attribute values, and apply the Information Bottleneck method. The Agglomerative Information Bottleneck algorithm suffers from high computational complexity, namely $\mathcal{O}(n^2 \log n)$, which is prohibitive for large datasets. We now introduce the *Linear InforMation BOttleneck, (LIMBO)* algorithm that uses distributional summaries in order to deal with larger 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. We employ an approach similar to the one used in the *BIRCH* clustering algorithm for clustering numerical data [18]. However, we use an IB inspired notion of distance and a novel definition of summaries to produce the solution.

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.

Let $\mathbf{T}$ denote a set of tuples over a set $\mathbf{A}$ of attributes, and let T and A be the corresponding random variables, as described in Section 3. 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

$$DCF(c) = \Big(n(c), p(A|c)\Big)$$

where $n(c)$ is the number of tuples in c and $p(A|c)$ is the conditional probability of the attribute values given the cluster c. In the case that c consists of a single tuple $t \in \mathbf{T}$, $n(t) = 1$ and $p(a|t) = 1/m$ for all attribute values $a \in \mathbf{A}$ that appear in tuple t, and zero otherwise, where m is the number of attributes.

For larger clusters, the DCF is computed recursively as follows. Let c^* denote the cluster we obtain by merging two clusters c_1 and c_2. The DCF of the cluster c^* is equal to

$$DCF(c^*) = \Big(n(c_1) + n(c_2), p(A|c^*)\Big) \tag{4}$$

where for each $a \in \mathbf{A}$,

$$p(a|c^*) = \frac{n(c_1)}{n(c_1)+n(c_2)} p(a|c_1) + \frac{n(c_2)}{n(c_1)+n(c_2)} p(a|c_2)$$

We define the distance between two clusters c_1 and c_2, $d(c_1, c_2)$, to be the information loss $\delta I(c_1, c_2) = I(A;C) - I(A;C')$, where $\mathbf{C}$ and $\mathbf{C}'$ denote the clusterings before and after the merge of clusters c_1 and c_2. We note that the information loss is *independent* of the clusterings $\mathbf{C}$ and $\mathbf{C}'$, that is, the amount of information we lose depends only on the clusters c_1 and c_2, and not on the rest of the clustering. Therefore, $d(c_1, c_2)$ is a well-defined distance measure that does not depend on $\mathbf{C}$. From the discussion in Section 2 we have that

$$d(c_1, c_2) = \left(\frac{n(c_1)}{n} + \frac{n(c_2)}{n}\right) D_{JS}[p(a|c_1), p(a|c_2)] \ .$$

where n is the total number of tuples in the dataset.

The DCFs can be stored and updated incrementally. Each DCF provides a summary of the corresponding cluster which is sufficient for computing the distance between two clusters. The merging of two DCFs as defined in Equation 4 corresponds to the merging of the corresponding clusters. Similarly, the distance, $d(c_1, c_2)$, can be thought of as the distance between the DCFs, or equivalently between the clusters they represent.

4.2 The DCF tree

The DCF tree is a height-balanced tree as depicted in Figure 1. It is characterized by two parameters, the maximum branching factor allowed at each node and a threshold that determines a maximum distance between a tuple and a cluster into which it is merged. If the threshold is exceeded, the tuple becomes a singleton cluster.

All nodes of the tree store DCFs. At any point in the construction of the tree, the DCFs at the leaves define a clustering of the tuples seen so far. Each non-leaf node stores a DCF that is produced by merging the DCFs of its children. The DCF tree is built in B-tree-like dynamic fashion from the bottom up. As tuples of the data set are read, they are placed in the appropriate leaves. After the whole data set is read from the disk, the DCF

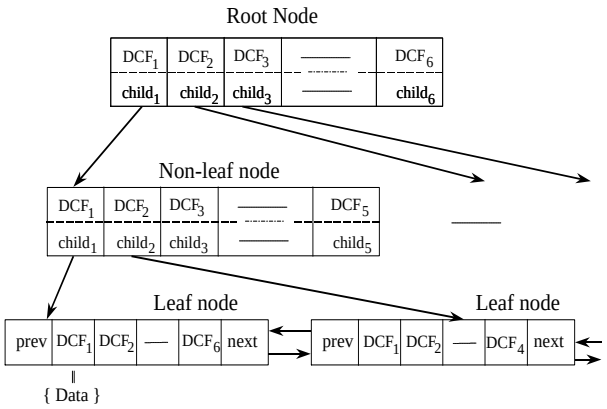


Figure 1: A DCF Tree with branching factor 6.

tree embodies a compact representation in which the data set is summarized by the information in the DCFs of the leaves.

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 leaves of the tree are merged to produce a specified 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 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 in the current clustering $\mathbf{C}$. At this point we need to decide whether the tuple t will be absorbed in the cluster c or not. If the distance $d(c,t)$, that is, the information loss incurred by merging t into c is less than the threshold, then we proceed with the merge, otherwise t forms a cluster on its own. If there is space for another entry in the leaf node, $DCF(t)$ is inserted and all DCFs in the path towards the root are updated using Equation (4). If there is no space, the leaf node has to be split. This is done in a manner similar to that used in BIRCH [18]. The

two DCFs in the node that have the greatest distance between them are selected as seeds for the two leaves and each of the remaining DCFs, along with $DCF(t)$, is placed in the leaf that contains the seed DCF to which it is closest.

When a leaf node is split, resulting in the creation of a new leaf node, its parent is updated, and a new entry is created that describes the newly inserted node. If there is space in the non-leaf node, we proceed with the addition of the entry, otherwise the non-leaf node must be split. This process continues upwards in the tree until the root is either updated or split itself. In the latter case, the height of the tree is increased by one.

Phase 2: Clustering. After the construction of the DCF tree, the leaf nodes hold the DCFs of a clustering $\tilde{\mathbf{C}}$ of the tuples in $\mathbf{T}$. In this phase, our algorithm will employ the *Agglomerative Information Bottleneck (AIB)* algorithm to cluster the DCFs in the leaves and produce a clustering $\mathbf{C}$ of the DCFs. 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.

Phase 3: Associating Tuple with Clusters. For a chosen value of k, Phase 2 produces k DCF that serve as *representatives* of k clusters. In the final phase, we perform a scan over the data set to assign each tuple to the cluster whose representative is closest to the tuple. This phase can also include an *outlier detection* procedure. When the distance from a tuple to the closest cluster is sufficiently large, the tuple can be deemed an outlier.

4.4 Threshold value

LIMBO uses a threshold value to control the decision to merge a new tuple into an existing cluster, or place it in a cluster by itself. This threshold limits the amount of information loss in our summary of the data set. It also affects the size of the DCF tree which in turn determines the computational cost of the AIB algorithm in Phase 2. Rather than using a threshold for the information loss, alternatively, we could select a threshold for the maximum size of the summary (as done in BIRCH [18]). In this work, we choose the former approach of bounding the information loss directly, in order to fully explore the tradeoff between cluster quality and efficiency.

As a guideline for setting the threshold value we adopt the following heuristic. On "average", every tuple contributes $I(A;T)/n$ to the mutual information $I(A;T)$. We define the clustering threshold to be a multiple ϕ of

this average and we denote the threshold by $\tau(\phi)$.

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

We can make a pass over the data, or use a sample of the data, to estimate $I(A;T)$. 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. In the extreme case $\phi = 0.0$, we prohibit any information loss in our summary. Empirically, we have shown that for values of ϕ close to 1.0 we obtain a concise and informative summarization. We discuss the effect of ϕ in Section 5.4.

4.5 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, m is the number of attributes and d is the total number of attribute values, L is the number of DCFs at the leaves of the DCF tree produced in Phase 1 of the algorithm, B is the branching factor of the DCF tree, and k is the specified 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, which has length $\mathcal{O}(\log_B n)$. For each node in the path, we perform at most B operations (distance computations, or updates), each taking time $\mathcal{O}(d)$. The time for handling splits is $\mathcal{O}(ndB)$ in total. Hence, the CPU cost of creating the DCF tree is $\mathcal{O}(dBn \log_B n)$.

- **Phase 2.** For the values of ϕ that produce clusterings of high quality (anywhere from 1.0 for small data sets to 1.5 for large data sets), 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. The CPU cost has complexity $\mathcal{O}(L^2 \log L)$ for running the AIB algorithm. In our experiments, $L \ll n$, so the CPU cost of this phase is low.

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

5 Experimental Evaluation

In this section we perform a comparative experimental evaluation of the LIMBO algorithm on both real and synthetic data sets. We also explore the effect of ϕ on the quality and efficiency of the algorithm. The section concludes with an efficiency evaluation of LIMBO. We now review the algorithms that we consider.

5.1 Algorithms

ROCK Algorithm. ROCK [11] assumes a similarity measure between tuples, and defines a *link* between two tuples whose similarity exceeds a threshold θ. The aggregate interconnectivity between two clusters is defined as the sum of links between their tuples. ROCK proceeds hierarchically, merging the two most interconnected clusters in each step. Thus, ROCK is not applicable to large data sets. We use the *Jaccard Coefficient* for the similarity measure as suggested in the original paper. 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, [11].

STIRR Algorithm. STIRR [10] iterates a *linear dynamical system* over multiple copies of a hypergraph of weighted attribute values, until a *fixed point* is reached. Each copy of the hypergraph contains two groups of attribute values, one with positive and another with negative weights, which define the two clusters. Handling more than two clusters involves a post-processing step not defined in the original paper. In our experiments, we use our own implementation. We report results for ten iterations. We augment tuples with an extra *tuple-id* attribute to track which tuples are placed in each of the clusters.

COOLCAT Algorithm. The approach most similar to ours is the COOLCAT algorithm [2, 3], by Barbará, Couto and Li. The COOLCAT algorithm uses the same quality measure for a clustering as our approach, namely the entropy of the clustering. It differs from our approach in that it relies on 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. It then places all remaining tuples of the data set 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. [3].

LIMBO Algorithm. We observed experimentally that the branching factor B does not affect significantly the quality of the clustering. We set $B = 4$, so that the DCF tree is of manageable size, and the effect on the creation of the tree is minimal. We explored a large range of values for ϕ. Larger values for ϕ delay leaf-node splits and create a smaller tree with a coarse representation of the data set. On the other hand, smaller ϕ values incur more splits but preserve a more detailed summary of the initial data set. In our implementation, we store DCFs as sparse vectors using a sparse vector library[3] resulting in significant savings in space.

5.2 Data Sets

We experiment with the following data sets. The first three have been previously used for the evaluation of the categorical clustering techniques mentioned above [3, 10, 11]. The synthetic data sets are also used for our scalability studies.

Congressional Votes: This relational data set was taken from the *UCI Machine Learning Repository.*[4] 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, either YES or NO. There is an additional attribute that gives the label for each tuple, Republican or Democrat. There are a total of 168 Republicans and 267 Democrats. There are 288 missing values that we treat as separate values.

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. It also contains an extra attribute that labels each mushroom as either poisonous or edible. There are 4,208 edible and 3,916 poisonous mushrooms in total. There are 2,480 missing values.

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.[5] We use this data to test our intra-attribute clustering algorithm.

[3] `http://www.genesys-e.org/ublas/`

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

[5] Following the approach of Gibson et al. [10] if the second author does not exist, then the name of the first author is copied instead.

Synthetic Data Sets. We produce synthetic data sets using a data generator available on the web.[6] 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 a number which is the number of clusters, *e*.g., DS5 or DS10. The data sets contain 5,000 tuples, and 10 Attributes, with domain sizes between 20 and 40 for each attribute. Three attributes participate in the rules the data generator uses to produce the cluster labels.

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 [12]. 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 [12] 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 tuples **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. [4] 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 by more than 10 hubs. The data set contains 93 authorities related to 102 hubs.

All data set are summarized in Tables 3.

5.3 Quality measures for Clustering

Clustering quality lies in the eye of the beholder; given two clusterings, determining the best clustering usually depends on subjective criteria. Consequently, we will use several quantitative measures of clustering performance.

We use the information loss, IL, $I(A;T) - I(A;C)$ for comparing clusterings. The lower the information loss, the better the clustering. With a clustering of low information loss, given a cluster, we can predict the attribute values of the tuples in the cluster with relatively high accuracy.

We also measured the value of the *Category Utility, CU* [13]. Category utility is defined as the difference between

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

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: Summary of the data sets used

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:

$$CU = \sum_{c \in \mathbf{C}} \frac{|c|}{n} \sum_i \sum_j [P(A_i = v_{ij}|c)^2 - P(A_i = v_{ij})^2]$$

Many data sets commonly used in testing clustering include a hidden from the algorithm variable, which specifies the class with which each tuple is associated. 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 performance measures from Information Retrieval, namely *Precision* and *Recall* [1] for the resulting clusters. 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 $\{c_1, \ldots, c_k\}$ output by a clustering algorithm. Consider an 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

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

where $\left| g_i \cap \overline{f(g_i)} \right|$ measures the number of tuples in class g_i that received the wrong label. We compute the *optimal* mapping between clusters and classes, that is, the mapping that minimizes the classification error. We use E_{min} to denote the classification error of the optimal mapping. Without loss of generality assume that the optimal mapping

assigns class g_i to cluster c_i. We define precision, P, and recall, R, 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 and, 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 average of the precision and recall of each cluster. 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 existing classes in the data set.

In our experiments, we report the values of information loss IL, category utility CU (where applicable), precision P, recall R, and classification error E_{min}. For LIMBO and COOLCAT all numbers are averages over 100 runs with different (random) orderings of the tuples.

5.4 Quality-Time tradeoffs for LIMBO

As previously discussed, the behavior of the LIMBO algorithm depends heavily upon the parameter ϕ. The value of ϕ offers a tradeoff between the compactness (number of leaf entries) of the DCF tree, and the expressiveness (information preserved) of the summarization it produces. For small values of ϕ, 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 Phase 2 of the algorithm. For large values of ϕ, we obtain a compact representation of the data set (small number of leaf entries), which results in faster execution time, at the expense of increased information loss.

We experimented with a range of values for ϕ in all data sets. In the extreme case of $\phi = 0.0$, we only merge identical tuples, and no information is lost. The LIMBO algorithm reduces to the AIB algorithm and we obtain the best clustering quality. For all data sets, for $\phi \leq 1.0$, we observe essentially no change (up the the third decimal digit) in the quality of the clustering. For $\phi = 1.0$, the number of leaf entries is much smaller, leading to very low execution times. Table 4 shows the reduction in the number of leaf entries for each data set when we set $\phi = 1.0$. Figures 2 and 3 show the reduction in leaf entries and the change in quality measures, respectively, for the DS5

Votes	Mushroom	DS5	DS10
94.71%	99.77%	98.71%	98.82%

Table 4: Leaf entries' reduction for $\phi = 1.0$

data set. In Figure 3, P and R, are very close for all ϕ except $\phi = 1.5$. Similar figures are presented for all other data sets; Figures 4 and 5 for Votes, Figures 6 and 7 for Mushroom and Figures 8 and 9 for DS10. These figures demonstrate that we can obtain significant compression of the data sets at no expense in the final clustering quality. The consistency of LIMBO can be attributed in part to the effect of Phase 3 which assigns the tuples to cluster representatives, and hides some of the information loss incurred in the previous phases. Thus it is sufficient for Phase 2 to discover k well separated representatives, in order for Phase 1 to produce a good clustering. As a result, even for large values of ϕ LIMBO may obtain exactly the same clustering as for $\phi = 0.0$.

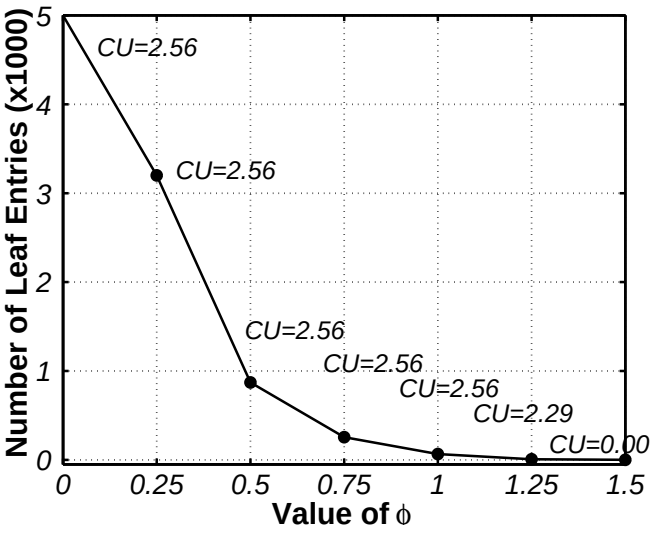


Figure 2: DS5 Leaf Entries

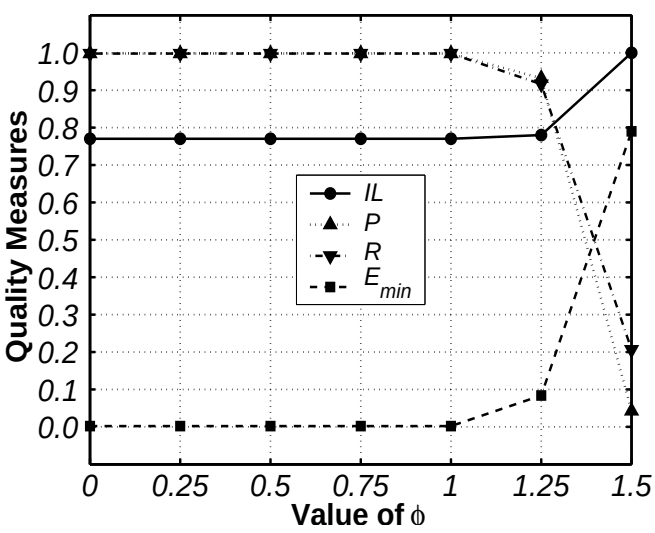


Figure 3: DS5 Quality

5.5 Comparative Evaluations

Our goal in this section is to demonstrate the quality of the clusterings LIMBO produces.

5.6 Comparison to Optimal Clustering

To get a better feel of how LIMBO performs on categorical data, we implemented a brute force algorithm (BF), that tries all possible clusterings and selects the one with the lowest information loss. Note that BF is computationally

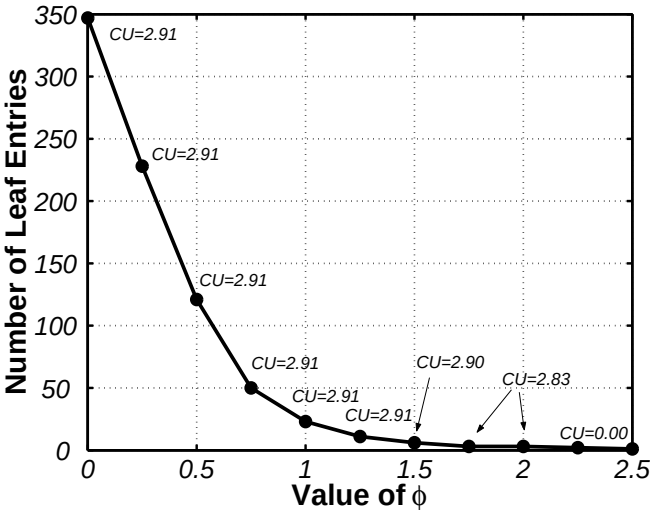


Figure 4: Votes Leaf Entries

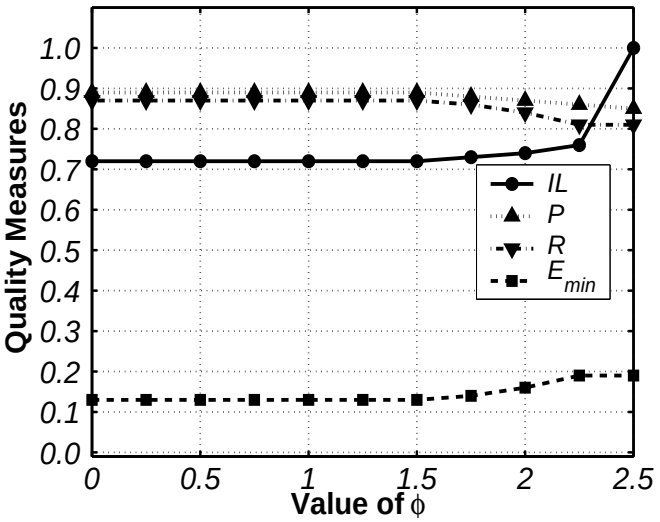


Figure 5: Votes Quality

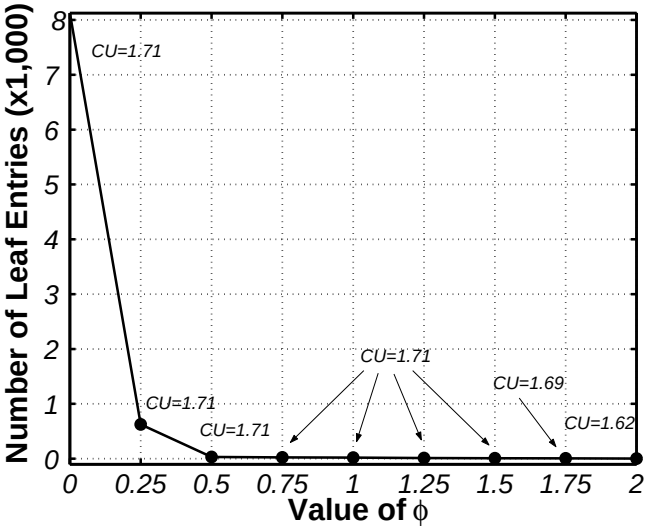


Figure 6: Mushroom Leaf Entries

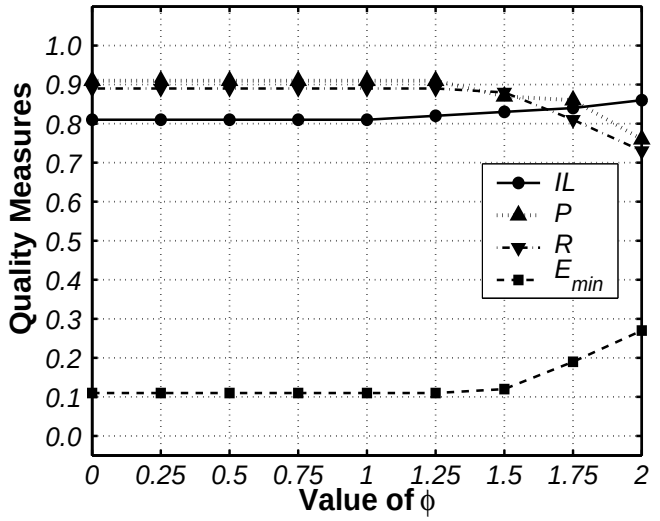


Figure 7: Mushroom Quality

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, $2 \leq k < n$ [7]. The data sets were produced using the synthetic data generator and they each contain 5 attributes, each one of them with a domain of size 3. We compared the information loss of the Brute Force algorithm against that of LIMBO for $\phi = 0.0$ (i.e. starting with each tuple in its own cluster). Results are given in Table 5.

With the sole exceptions of $(7, 2)$ and $(8, 4)$, for all other combinations of n, k in Table 5, LIMBO produces a clustering with information loss equal to the minimum information loss of any k-clustering of the n tuples. These

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

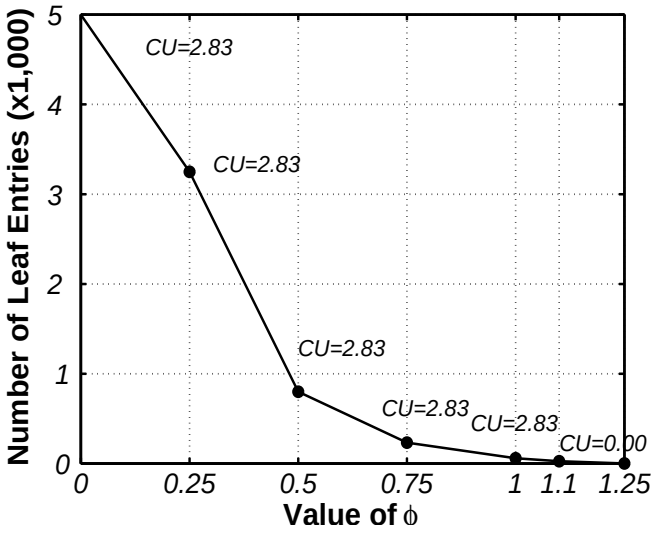


Figure 8: DS10 Leaf Entries

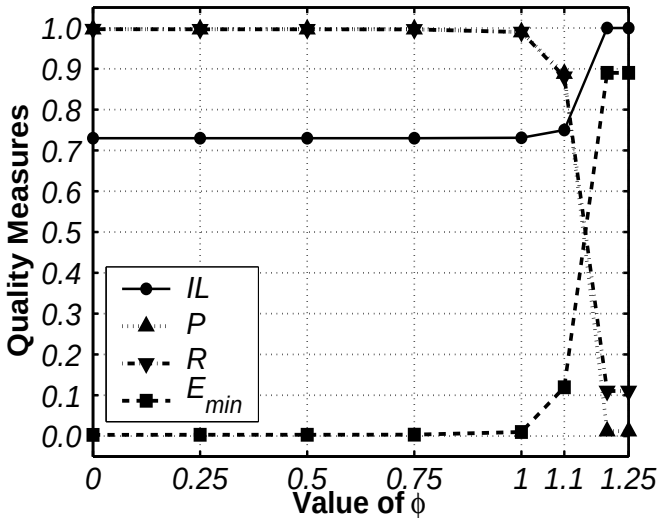


Figure 9: DS10 Quality

results are indicative of the ability of the LIMBO algorithm to find an optimal, or nearly optimal clustering for a fixed k.

5.6.1 Relational Data

Table 6 shows the results for all algorithms on all quality measures for the Votes and Mushroom data sets. For LIMBO we present results for $\phi = 1.0$, which are essentially the same as for $\phi = 0.0$. The *size* entry in the table holds the number of leaves for LIMBO, and the sample size for COOLCAT. For the Votes data set we use the whole data set as sample, while for Mushroom we use 1,000 tuples. As Table 6 indicates, LIMBO's quality is superior to ROCK, STIRR and COOLCAT, in both data sets. In terms of IL, 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 two classes. We do not present STIRR's results here, since it is designed for producing just two clusters. The results are shown in Table 7. We observe again that LIMBO has the lowest information loss and produces nearly optimal results with respect to precision and recall.

Our experiments show that ROCK is very sensitive to the threshold value θ and in many cases, the clustering produces one giant cluster that includes tuples from most classes. This results in poor precision and recall. The

n, k	**Clusterings**	**BF**	**LIMBO**
7,2	63	70%	71%
7,3	301	44%	44%
7,4	350	20%	20%
7,5	140	16%	16%
7,6	21	7%	7%
8,2	127	65%	65%
8,3	966	42%	42%
8,4	1701	35%	37%
8,5	1050	21%	21%
8,6	266	14%	14%
8,7	28	6%	6%

Table 5: Information Loss of Brute Force and LIMBO ($\phi = 0.0$)

STIRR algorithm produces clusterings of poor quality in both data sets, a fact that might be due to the choice of uniform initial weights for the attribute values. However, prior knowledge is required to define initial weights for the attribute values.

Comparison with COOLCAT. COOLCAT exhibits on average clustering quality that is close to that of LIMBO. However, it is interesting to examine how COOLCAT behaves when we consider other statistics. In Table 8, we present statistics for 100 runs of COOLCAT and LIMBO on different orderings of the Votes and Mushroom data sets. For the Votes data set, COOLCAT exhibits worst case information loss as high as 95.31%, and variance 12.25 percentile units. For all runs, we use the whole data set as the sample. For the Mushroom data set, the situation is better, but still the variance is as high as 3.5 percentile units. The sample size was 1,000 for all runs. On the other hand, Table 8 indicates that LIMBO behaves in a more stable fashion. Notably, for the Mushroom data set, LIMBO's performance is exactly the same in all runs, while for Votes it exhibits very low variance, indicating that LIMBO is not particularly sensitive to the order of the tuples. 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.

Votes (2 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 1.0$)	23	**72.26**	**0.89**	**0.87**	**0.13**	**2.91**
COOLCAT	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
STIRR	-	97.00	0.66	0.64	0.36	0.30
Mushroom (2 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 1.0$)	18	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
COOLCAT	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
STIRR	-	83.00	0.66	0.64	0.36	1.01

Table 6: Results for real data sets

After detailed examination we found that the runs with maximum information loss for the Votes data set correspond to cases where an outlier was selected as the initial representative. The Votes data set contains three such tuples, which are far from all other tuples, and they are naturally picked as representatives.[8] 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. As tuples are inserted to the clusters, the representatives "move" closer to the inserted tuples, thus making the algorithm sensitive to the ordering of the data set.

LIMBO and COOLCAT use similar approaches and they both include a crucial stage which requires quadratic computational complexity. 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 experiment with both algorithms having the same input size for

[8] A very recent version of COOLCAT includes a step designed to avoid such selections [2]. We plan to implement this step for our future experiments.

DS5 (n=5000, 10 attributes, 5 clusters)						
Algorithm	*size*	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 1.0$)	66	**76.66**	**0.998**	**0.998**	**0.002**	**2.56**
COOLCAT	125	77.02	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 ($\phi = 1.0$)	59	**72.70**	**0.997**	**0.997**	**0.003**	**2.83**
COOLCAT	125	73.32	0.979	0.973	0.026	2.74
ROCK ($\theta = 0.0$)	-	78.00	0.830	0.818	0.182	2.13

Table 7: Results for synthetic data sets

this phase, *i.e.*, we made COOLCAT's sample size, equal to LIMBO's number of leaves. Results for the Votes and Mushroom data sets are shown in Tables 9 and 10. LIMBO outperforms COOLCAT in all runs, for all measures. The two algorithms are closest in performance for the Votes data set with input size 23, and farthest apart for the Mushroom data set with input size 275. COOLCAT appears to perform better with smaller sample size, while LIMBO remains essentially unaffected.

5.6.2 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 6.) The total information loss was 61%. Figure 10 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 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 consists of "pro-choice" pages. The second cluster consists of

VOTES		Min	Max	Avg	Var
LIMBO ($\phi = 1.0$)	IL	71.98	72.79	72.17	0.035
	CU	2.85	2.94	2.91	0.0003
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	81.61	81.61	81.61	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 8: Statistics for IL(%) and CU

"pro-life" pages. The third cluster 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. A complete list of the results can be found in Tables 11, 12 and 13. In almost all cases the URLs and Titles of the web pages are indicative of their content.

5.6.3 Intra-Attribute Clustering

We now present results for the application of LIMBO to the problem of the intra-attribute value clustering. For this experiment, we use the Bibliographic data set. We are interested in clustering the conferences/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, for the first experiment we set the random variable A to range over conferences/journals, while variable $\tilde{A}$ ranges over first and second authors, and the year of publication. There are 1,211 distinct conferences/journals in the data set; 815 are database papers, and 396 are theory papers.[9] Our objective is to reconstruct these two clusters. Results for $\phi = 0.5$ and $\phi = 1.0$ are shown in Table 14. LIMBO's results are, for both values of ϕ, superior to the results that STIRR produces with respect to all quality measures.

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

Sample Size = Leaf Entries = 347					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 0.1$)	**72.22**	**0.89**	**0.87**	**0.13**	**2.91**
COOLCAT	74.15	0.86	0.84	0.15	2.63
Sample Size = Leaf Entries = 23					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 1.0$)	**72.26**	**0.89**	**0.87**	**0.13**	**2.91**
COOLCAT	72.50	0.88	0.86	0.13	2.87

Table 9: LIMBO vs COOLCAT on Votes

The difference is especially pronounced in the P and R measures.

We now turn to the problem of clustering the first authors. Our goal is to partition authors into Theory and Data Bases researchers. We first apply a filtering step to remove conferences that appear in fewer than 5 tuples. 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 evaluate the results of LIMBO and STIRR based on the distribution of the papers that were written by first authors in each cluster. Figures 11 and 12 illustrate the clusters produced by LIMBO and STIRR, respectively. The x-axis in both figures represents conferences/journals, while the y-axis represents first authors. If an author has published a paper in a particular conference, 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 conferences/journals. Database publications lie on the left of the line, while theory ones 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 conferences/journals. The bottom half, corresponds to a set of database researchers with very few publications in theory conferences/journals. 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 in the database cluster due to their co-authors. STIRR, 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 conferences/journals.

Sample Size = Leaf Entries = 275					
Algorithm	$IL(\%)$	P	R	E_{min}	CU
LIMBO ($\phi = 0.3$)	**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 ($\phi = 1.0$)	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
COOLCAT	82.10	0.82	0.81	0.19	1.60

Table 10: LIMBO vs COOLCAT on Mushroom

5.7 Efficiency 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 use the DS10 data set to measure execution times for each phase separately, for $\phi \geq 0.25$. Results are shown in Figure 13 where for each value of ϕ we also include the size of the DCF tree in memory. In this figure we observe that for small ϕ the computational bottleneck of the algorithm is Phase 2. As ϕ increases, the time for Phase 2 decreases in a quadratic fashion. Time for Phase 1 also decreases, however, at a much slower rate. Phase 3, as expected, remains unaffected, and it is equal to 1.43 seconds for all values of ϕ. For $\phi \geq 0.75$ the number of leaf entries becomes sufficiently small, so that the computational bottleneck of the algorithm becomes Phase 1. This is the range of ϕ in which we are interested.

We now study in detail Phase 1, where the DCF tree is built. We experiment with large data sets of $10K$, $500K$ and $1M$ tuples, produced with the synthetic data set generator, having 10 attributes with 20 to 40 values each. As the graph in Figure 14 demonstrates, execution time for Phase 1 remains constant for values of ϕ up to 0.8. For $0.8 < \phi < 1.4$, the execution time drops significantly. This decrease is due to the reduced number of splits and the decrease in the DCF tree size. Figure 15 plots the number of leaves as a function of ϕ.[10] In the same plot we show the size of the tree. We observe that for the range of values for ϕ in which we are interested ($0.8 < \phi < 1.4$), LIMBO produces a manageable DCF tree, with small number of leaves, which means fast execution time for the

[10]The y-axis of Figure 15 has a logarithmic scale.

(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 11: Pro-Life Cluster of the Web data set

(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 12: 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 13: 'Cincinnati' Cluster of the Web data set

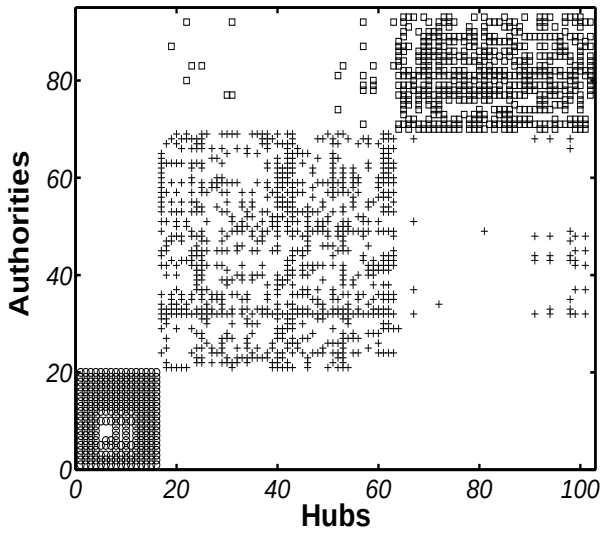


Figure 10: Clustering of Web Data

Algorithm	*Leaves*	$IL(\%)$	P	R	E_{min}
LIMBO ($\phi = 0.5$)	88	**94**	**0.91**	**0.91**	**0.09**
LIMBO ($\phi = 1.0$)	47	**94**	**0.90**	**0.90**	**0.11**
STIRR	-	98	0.56	0.55	0.45

Table 14: Conf/Jrnl clustering using LIMBO and STIRR

AIB algorithm in Phase 2. Furthermore, we observed 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%.

From the above observations and the discussion in Section 5.4, it appears that, for every data set, there exists a range of ϕ values, for which the decrease in clustering quality is negligible, and at the same time, the size of the DCF tree is small enough, so that Phase 1 is sped up significantly, and the input for Phase 2 can be handled efficiently. For example, for DS10, this range is the interval $[0.75, 1.0]$.

In our final experiment, we study the execution time for all three phases of LIMBO as a function of the size of the data set. We consider three data sets of sizes $100K$, $500K$, and $1M$, each containing 15 clusters. The first two data sets are samples of the $1M$ data set. The left graph in Figure 16 shows the execution time for LIMBO on these three data sets, for $\phi = 1.0$, and $\phi = 1.1$. In this figure, we observe that execution time scales in a near-linear fashion with respect to the size of the data set. Furthermore, Table 15 shows that the clustering quality of LIMBO

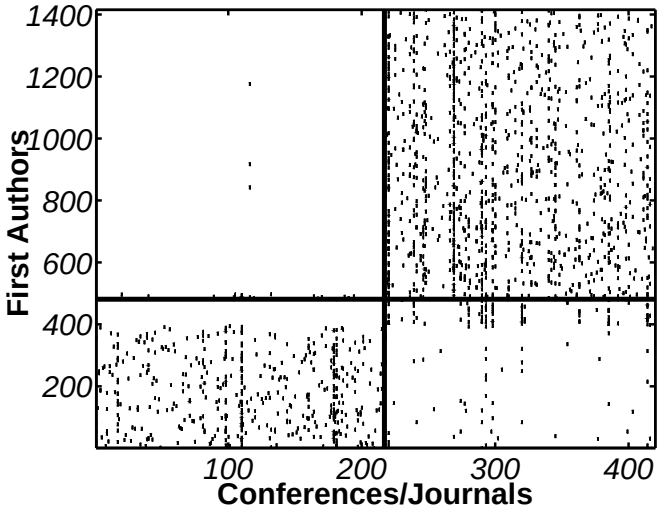


Figure 11: LIMBO clusters of first authors

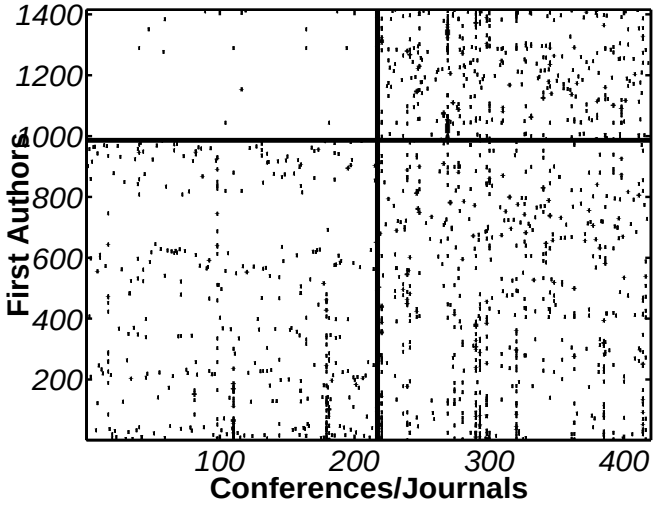


Figure 12: STIRR clusters of first authors

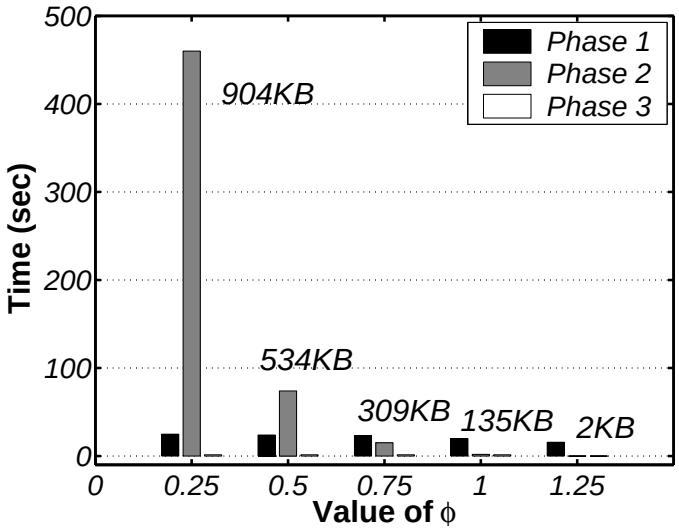


Figure 13: Execution time over ϕ for DS10

for this range of ϕ values remains unaffected, and it is almost the same across data sets. We run LIMBO again on the three data sets, picking values for ϕ within this range, such that the number of leaves at the end of Phase 1 is approximately the same for all three data sets. The right graph in Figure 16 shows that in this case the execution time scales linearly.

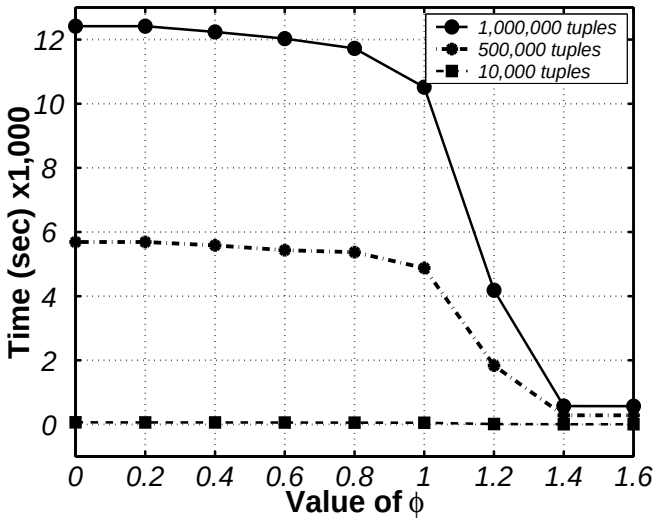


Figure 14: Execution times of Phase 1

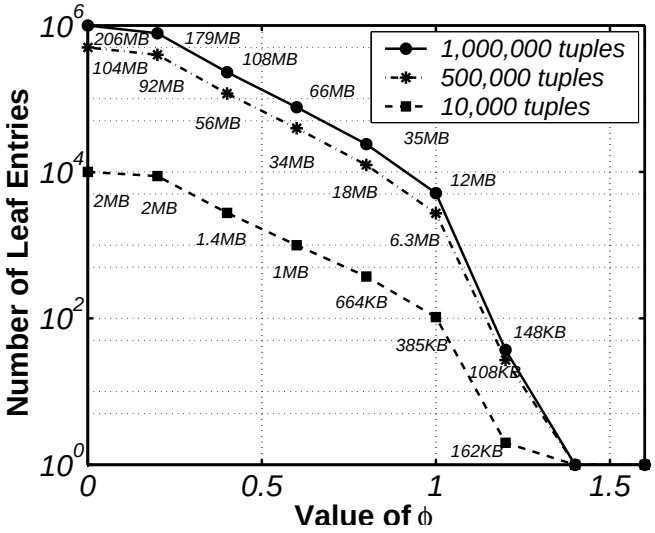
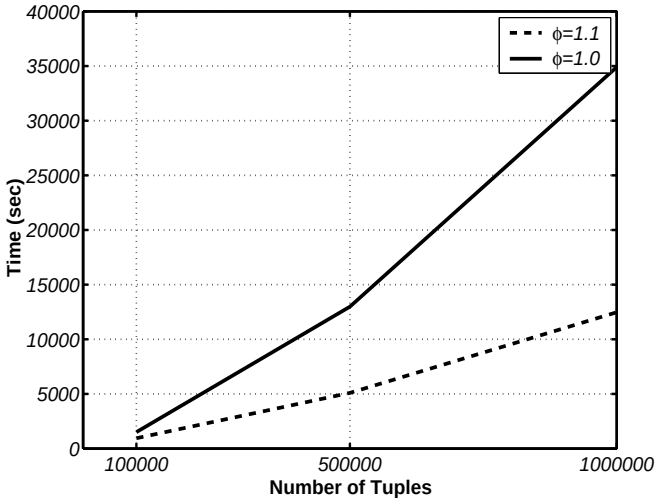


Figure 15: Number of leaves of Phase 1



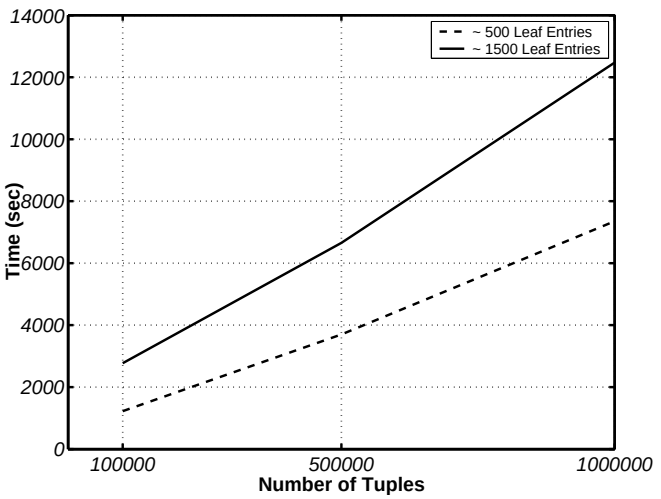


Figure 16: Execution times and Size for large data sets

6 Estimating K

Identifying automatically 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, $\delta I(A;C_k) = I(A;C_{k+1}) - I(A;C_k)$. Thinking in reverse, $\delta I(A;C_k)$ captures the amount of information we gain if we break up a cluster in two, to move

	$IL(\%)$	P	R	E_{min}	CU
$100K$	76	0.99	0.99	0.01	2.61
$500K$	75	0.99	0.99	0.01	2.62
$1M$	75	0.99	0.99	0.01	2.62

Table 15: Quality for large data sets for $\phi \in [1.0, 1.1]$

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 $\delta I(A; C_k)$ becomes sufficiently small. This has also been discussed by Slonim and Tishby [15].

The mutual information $I(A; C)$ 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 $H(C|A)$. Intuitively, $H(C|A)$ captures the *purity* of the clustering. For a clustering $\mathbf{C}$ with very low $H(C|A)$, 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 $H(C|A)$, the purer the clusters. $H(C_k|A)$ is minimized for $k = 1$, where $H(C_k|A) = 0$. An appropriate value for k is when $H(C_k|A)$ is sufficiently low, and $k > 1$. Furthermore, the value $\delta H(C_k|A) = H(C_{k+1}|A) - H(C_k|A)$ gives the increase in purity when merging two clusters to move from a clustering of size $k+1$ to one of size k. High values of $\delta H(C_k|A)$ mean that the two merged clusters are similar. Low values imply that the two merged clusters that are dissimilar. The latter case suggests $k+1$ as a candidate value for the number of clusters.

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 $k = 1$. While we run the AIB algorithm in Phase 2, we keep track of the mutual information $I(A; C_k)$, and the conditional entropy $H(C_k|A)$. Observing the behavior of these two measures, will hopefully provide us with candidate values for k, for which we run Phase 3 of LIMBO. Our case is best illustrated with the Web data set. Figure 17 presents the plots for $H(C_k|A)$, $I(C_k|A)$ (left), and $\delta H(C_k|A)$, $\delta I(A; C_k)$ (right) for the Web data. From the plots we can conclude that for $k = 3$, both $\delta I(A; C_k)$ and $H(C_k|A)$ are sufficiently close to zero, which means that the clustering is both pure and informative. This becomes obvious when looking at the clustering of Web data in Figure 10. Note that $\delta H(C_2|A)$

is very low, which means that producing a 2-clustering will result in merging two dissimilar clusters. Similar curves for the Votes are given in Figure 18 and for the Mushroom data set in Figure 19.

For the Votes data set, the $\delta I(A; C_k)$ curve clearly suggests $k = 2$ as the right number of clusters. The $H(C_k|A)$ measure is not equally informative since it increases steadily for increasing k. For $k = 6$, $\delta H(C_k|A)$ 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 [2].

For the mushroom data set, the $\delta I(A; C_k)$ curve does not give a clear indication of what the right number of clusters is. The $H(C_k|A)$ and $\delta H(C_k|A)$ 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, $\delta H(C_k|A_i)$ is close, or equal to zero for $k = 2$. 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 $k = 8$.

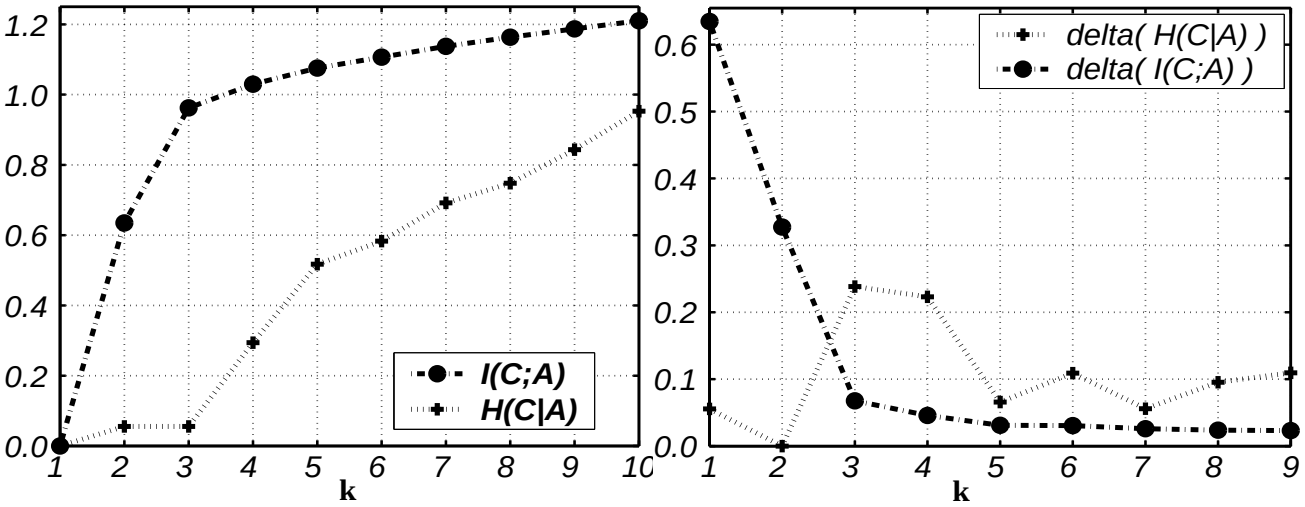


Figure 17: $I(C; A)$, $H(C|A)$, and $\delta I(C; A)$, $\delta H(C|A)$ for Web Data

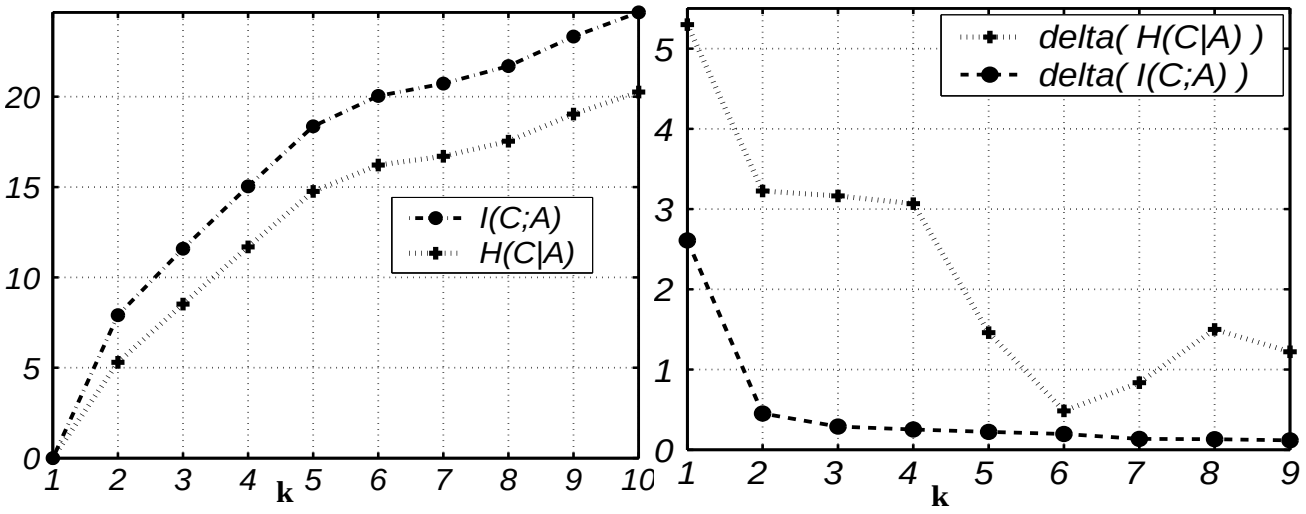


Figure 18: $I(C;A)$, $H(C|A)$, and $\delta I(C;A)$, $\delta H(C|A)$ for Votes

7 Other Related Work

CACTUS, [8], by Ghanti, Gehrke and Ramakrishnan, uses summaries of information constructed from the data set that are sufficient for discovering clusters. The algorithm defines attribute value clusters with overlapping cluster-projections on any attribute. This makes the assignment of tuples to clusters unclear.

Our approach is based on the *Information Bottleneck (IB) Method*, introduced by Tishby, Pereira and Bialek [17]. The Information Bottleneck method has been used in an agglomerative hierarchical clustering algorithm [15] and applied to the clustering of documents [16]. Recently, Slonim and Tishby [14] introduced the *sequential Information Bottleneck, (sIB)* algorithm, which reduces the running time relative to the agglomerative approach. However, it depends on an initial random partition and requires multiple passes of the data for different initial partitions. In the future, we plan to experiment with sIB in Phase 2 of LIMBO.

Finally, another algorithm that uses an extention to BIRCH [18] is given by Chiu, Fang, Chen, Wand and Jeris [5]. Their approach assumes that the data follows a multivariate normal distribution. The performance of the algorithm has not been tested on categorical data sets.

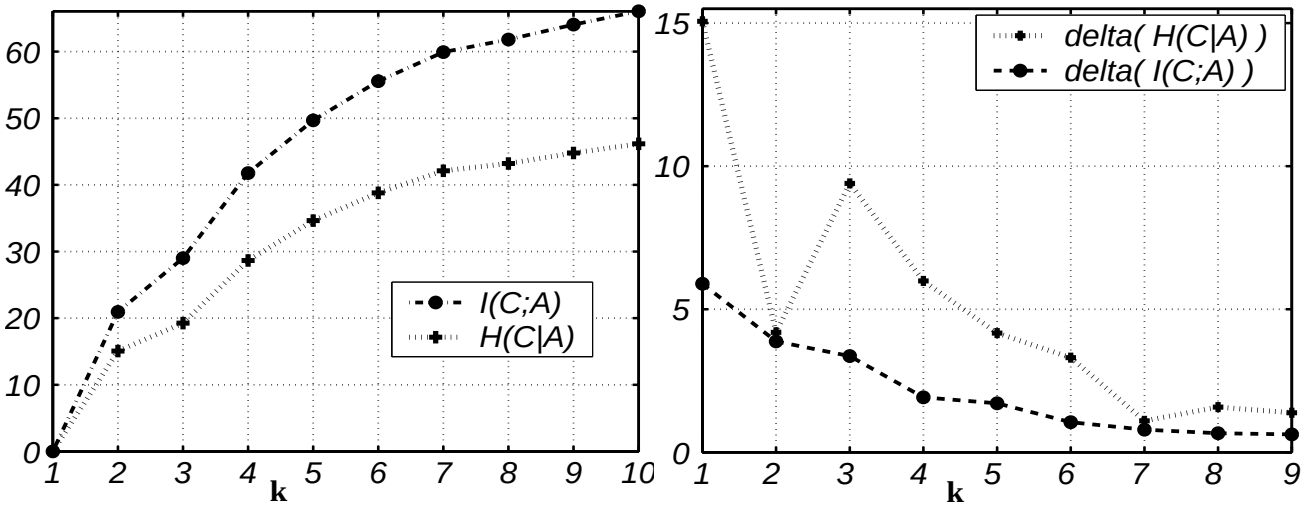


Figure 19: $I(C;A)$, $H(C|A)$, and $\delta I(C;A)$, $\delta H(C|A)$ for Mushroom

8 Conclusions

In this paper, we have described and evaluated the LIMBO algorithm for clustering categorical data sets. LIMBO's hierarchical nature, together with its merging threshold ϕ, make it capable of producing a range of clusterings of different sizes even in very large data sets. Our experiments show that LIMBO's execution time scales in an almost linear fashion with the size of the data set, and that this performance is achieved at no expense in clustering quality.

Compared to several other categorical clustering algorithms, LIMBO shows superior cluster quality both with respect to direct measures of clustering quality, and with respect to consistency with meaningful pre-specified classifications Our technique can be used for both relational and market basket data, and for attribute value clustering. Finally, we describe some information theoretic measures for selecting an appropriate value for the number of clusters.

In the future, we are interested in extending LIMBO so that it uses a memory buffer of fixed size, and it handles both numerical and categorical attributes. Finally, we plan to apply it as a data mining technique to schema discovery and management.

References

[1] R. B.-Yates and B. R.-Neto. *Modern Information Retrieval.* Addison-Wesley-Longman, 1999.

[2] D. Barbará, J. Couto, and Y. Li. An Information Theory Approach to Categorical Clustering. In *Submitted for Publication.*

[3] D. Barbará, J. Couto, and Y. Li. COOLCAT: An entropy-based algorithm for categorical clustering. In *CIKM*, McLean, VA, 2002.

[4] A. Borodin, G. O. Roberts, J. S. Rosenthal, and P. Tsaparas. Finding authorities and hubs from link structures on the World Wide Web. In *WWW-10*, Hong Kong, 2001.

[5] T. Chiu, D. Fang, J. Chen, Y. Wang, and C. Jeris. A Robust and Scalable Clustering Algorithm for Mixed Type Attributes in Large Database Environment. In *KDD*, San Francisco, CA, 2001.

[6] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley & Sons, New York, NY, USA, 1991.

[7] G. Das and H. Mannila. Context-Based Similarity Measures for Categorical Databases. In *PKDD*, Lyon, France, 2000.

[8] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS: Clustering Categorical Data Using Summaries. In *KDD*, San Diego, CA, 1999.

[9] M. R. Garey and D. S. Johnson. *Computers and intractability; a guide to the theory of NP-completeness.* W.H. Freeman, 1979.

[10] D. Gibson, J. M. Kleinberg, and P. Raghavan. Clustering Categorical Data: An Approach Based on Dynamical Systems. In *VLDB*, New York, NY, 1998.

[11] S. Guha, R. Rastogi, and K. Shim. ROCK: A Robust Clustering Algorithm for Categorical Atributes. In *ICDE*, Sydney, Australia, 1999.

[12] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. In *SODA*, San Francisco, CA, 1998.

[13] M. A. Gluck and J. E. Corter. Information, Uncertainty, and the Utility of Categories. In *COGSCI*, Irvine, CA, USA, 1985.

[14] N. Slonim, N. Friedman, and N. Tishby. Unsupervised Document Classification using Sequential Information Maximization. In *SIGIR*, Tampere, Finland, 2002.

[15] N. Slonim and N. Tishby. Agglomerative Information Bottleneck. In *NIPS-12*, Breckenridge, CO, 1999.

[16] N. Slonim and N. Tishby. Document Clustering Using Word Clusters via the Information Bottleneck Method. In *SIGIR*, Athens, Greece, 2000.

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

[18] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient Data Clustering Method for Very Large Databases. In *SIGMOD*, Motreal, QB, 1996.