LIMBO: Scalable Clustering of Categorical Data

Periklis Andritsos, Panayiotis Tsaparas, Renée J. Miller, and Kenneth C. Sevcik
{periklis,tsap,miller,kcs}@cs.toronto.edu

University of Toronto, Department of Computer Science

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. 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 distance measure for categorical attribute values. We show how the LIMBO algorithm can be used to cluster both tuples and values. LIMBO handles large data sets by producing a memory bounded summary model for the data. We present an experimental evaluation of LIMBO, and we study how clustering quality compares to other categorical clustering algorithms. LIMBO supports a trade-off between efficiency (in terms of space and time) and quality. We quantify this trade-off and demonstrate that LIMBO allows for substantial improvements in efficiency with negligible decrease in quality.

1 Introduction

Clustering is a problem of great practical importance that has been the focus of substantial research in several domains for decades. It 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 $100K and $110K are more similar than $100K 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 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	$\mathbf{C}$	$\mathbf{D}$
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 the tuples in Table 1 into two clusters. Clustering $\mathbf{C}$ 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 $\mathbf{D}$, d_2 is equally 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 [20]. 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.

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 are the following.

• 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 sizes. LIMBO can also control the size of the model it builds to summarize the data.

• We use LIMBO to cluster both tuples (in relational and market-basket data sets) and attribute values. We define a novel distance between attribute values that allows us to quantify the degree of interchangeability of attribute values within a single attribute.

• We empirically evaluate the quality of clusterings produced by LIMBO relative to other categorical clustering algorithms including the tuple clustering algorithms IB, ROCK [13], and COOLCAT [4]; as well as the attribute value clustering algorithm STIRR [12]. We compare the clusterings based on a comprehensive set of quality metrics.

The rest of the paper is structured as follows. In Section 2, we present the IB method, and we describe how to formulate the problem of clustering categorical data within the IB framework. In Section 3, we introduce LIMBO and show how it can be used to cluster tuples. In Section 4, we present a novel distance measure for categorical attribute values and discuss how it can be used within LIBMO to cluster attribute values. Section 5 presents the experimental evaluation of LIMBO and other algorithms for clustering categorical tuples and values. Section 6 describes related work on categorical clustering and Section 7 discusses additional applications of the LIMBO framework.

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 paper. We also introduce the Information Bottleneck method, and we formulate the problem of clustering categorical data within this framework.

2.1 Information Theory basics

The following definitions can be found in any information theory textbook, e.g., [7]. Let T denote a discrete random variable that takes values over the set $\mathbf{T}$[1], and let $p(t)$ denote the probability mass function of T. The *entropy* $H(T)$ of variable T is defined by $H(T) = -\sum_{t\in\mathbf{T}} p(t)\log p(t)$. Intuitively, entropy captures the "uncertainty" of variable T; the higher the entropy, the lower the certainty with which we can predict its value.

Now, let T and A be two random variables that range over sets $\mathbf{T}$ and $\mathbf{A}$ respectively. The *conditional entropy* of A given T is defined as follows.

$$H(A|T) = \sum_{t\in\mathbf{T}} p(t) \sum_{a\in\mathbf{A}} p(a|t)\log p(a|t)$$

Conditional entropy captures the uncertainty of predicting the values of variable A given the values of variable T. The *mutual information*, $I(T;A)$, quantifies the amount of information that the variables convey about each other. Mutual information is symmetric, and non-negative, and it is related to entropy via the equation $I(T;A) = H(T) - H(T|A) = H(A) - H(A|T)$.

Relative Entropy, or *Kullback-Leibler (KL)* divergence, is an information-theoretic measure of the difference between two probability distributions. Given two distributions p and q over a set $\mathbf{T}$, the relative entropy is defined as follows.

$$D_{KL}[p\|q] = \sum_{t\in\mathbf{T}} p(t)\log\frac{p(t)}{q(t)}$$

2.2 Clustering using the IB Method

In categorical data clustering, 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 $\mathbf{A}$. The data can then be conceptualized as an $n \times d$ matrix M, where each tuple $t \in \mathbf{T}$ is a d-dimensional row vector in M. Matrix entry $M[t,a]$ is 1, if tuple t contains attribute value a, and zero otherwise. Each tuple contains one value for each attribute, so each tuple vector contains exactly m 1's.

Now let T, A be random variables that range over the sets $\mathbf{T}$ (the set of tuples) and $\mathbf{A}$ (the set of attribute values) respectively. We normalize matrix M so that the entries of each row sum up to 1. For some tuple $t \in \mathbf{T}$, the corresponding row of the normalized matrix 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.[2] A similar formulation can be applied in the case of *market-basket* data, where each tuple contains a set of values from a single attribute [1].

A *k-clustering* $\mathbf{C}_k$ of the tuples in $\mathbf{T}$ partitions them into k clusters $\mathbf{C}_k = \{c_1, c_2, c_3, ..., c_k\}$, where each cluster $c_i \in \mathbf{C}_k$ is a non-empty subset of $\mathbf{T}$ such that $c_i \cap c_j = \emptyset$ for all i, j, $i \neq j$, and $\cup_{i=1}^{k} c_i = \mathbf{T}$. 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.

[1] For the remainder of the paper, we use italic capital letters (e.g., T) to denote random variables, and boldface capital letters (e.g., $\mathbf{T}$) to denote the set from which the random variable takes values.

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

	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

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

$$p(c) = \sum_{t \in c} p(t) = \frac{|c|}{n} \quad \text{and} \quad p(a|c) = \frac{1}{p(c)} \sum_{t \in c} p(t)p(a|t)$$

We seek clusterings of the elements of $\mathbf{T}$ such that, for $t \in c$, knowledge of the cluster identity, c, provides essentially the same prediction of, or information about, the values in $\mathbf{A}$ as does the specific knowledge of t. The mutual information $I(A;C)$ measures the information about the values in $\mathbf{A}$ provided by the identity of a cluster in $\mathbf{C}$. The higher $I(A;C)$, the more informative the cluster identity is about the values in $\mathbf{A}$ contained in the cluster. Tishby, Pereira and Bialek [20], define clustering as an optimization problem, where, for a given number k of clusters, we wish to identify the k-clustering that maximizes $I(A;C_k)$. Intuitively, in this procedure, the information contained in T about A is "squeezed" through a compact "bottleneck" clustering C_k, which is forced to represent the "relevant" part in T with respect to A. Tishby et al. [20] prove that, for a fixed number k of clusters, the optimal clustering $\mathbf{C}_k$ partitions the objects in $\mathbf{T}$ so that the average relative entropy $\sum_{c \in \mathbf{C}_k, t \in \mathbf{X}} p(t,c) D_{KL}[p(a|t) \| p(a|c)]$ is minimized.

Finding the optimal clustering is an NP-complete problem [11]. Slonim and Tishby [18] 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 $t \in \mathbf{T}$ is assigned to its own cluster. Due to the one-to-one mapping between $\mathbf{C}_n$ and $\mathbf{T}$, $I(A;C_n) = I(A;T)$. 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 the 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 $\mathbf{A}$ decreases; that is, $I(A;C_{\ell-1}) \leq I(A;C_\ell)$. 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 $\delta I(c_i, c_j) = I(A;C_\ell) - I(A;C_{\ell-1})$. We can also view the information loss as the increase in the uncertainty. Recall that $I(A;C) = H(A) - H(A|C)$. Since $H(A)$ is independent of the clustering $\mathbf{C}$, maximizing the mutual information $I(A;C)$ is the same as minimizing the entropy of the clustering $H(A|C)$.

For the merged cluster $c^* = c_i \cup c_j$, we have the following.

$$p(c^*) = p(c_i) + p(c_j) \tag{1}$$

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

Tishby et al. [20] show that

$$\delta I(c_i, c_j) = [p(c_i) + p(c_j)] D_{JS}[p(A|c_i), p(A|c_j)] \tag{3}$$

where D_{JS} is the *Jensen-Shannon (JS)* divergence, defined as follows. Let $p_i = p(A|c_i)$ and $p_j = p(A|c_j)$ and let $\bar{p} = \frac{p(c_i)}{p(c^*)} p_i + \frac{p(c_j)}{p(c^*)} p_j$. Then, the D_{JS} distance is defined as follows.

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

The D_{JS} distance defines a metric and it is bounded above by one. 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 clustering $\mathbf{C}_\ell$.

This approach considers all attribute values as a single 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 can show that in applying the Information Bottleneck method to relational data, considering all attributes as a single random variable is equivalent to considering each attribute independently [1].

In the model of the data described so far, every tuple contains one value for each attribute. However, this is not the case when we consider market-basket data, which 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. In the case of market-basket data, a tuple t_i contains d_i values. Setting $p(t_i) = 1/n$ and $p(a|t_i) = 1/d_i$, if a appears in t_i, we can define the mutual information I(T;A) and proceed with the Information Bottleneck method to clusters the tuples.

3 *LIMBO* Clustering

The Agglomerative Information Bottleneck algorithm suffers from 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 that 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 sufficient statistics, that is the size of our summary model. This, together with an IB inspired notion of distance and a novel definition of summaries to produce the solution, makes our approach different from the one employed in the BIRCH clustering algorithm for clustering numerical data [21]. In BIRCH a heuristic threshold is used to control the accuracy of the summary created. In the experimental section of this paper, we study the effect of such a threshold in LIMBO.

3.1 Distributional Cluster Features

We summarize a cluster of tuples in a *Distributional Cluster Feature (DCF)*. We will use the information in the relevant DCFs 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 earlier. 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

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

where $p(c)$ is the probability of cluster c, and $p(A|c)$ is the conditional probability distribution of the attribute values given the cluster c. We will often use $DCF(c)$ and c interchangeably.

If c consists of a single tuple $t \in \mathbf{T}$, $p(t) = 1/n$, and $p(A|t)$ is computed as described in Section 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 2. 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

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

where $p(c^*)$ and $p(A|c^*)$ are computed using Equations 1, and 2 respectively. 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 for merging the corresponding clusters c_1 and c_2. The distance $d(c_1, c_2)$ is computed using Equation 3. The information loss depends only on the clusters c_1 and c_2, and not on the clustering $\mathbf{C}$ in which they belong. Therefore, $d(c_1, c_2)$ is a well-defined distance measure.

The DCFs can be stored and updated incrementally. The probability vectors are stored as sparse vectors, reducing the amount of space considerably. Each DCF provides a summary of the corresponding cluster which is sufficient for computing the distance between two clusters.

3.2 The DCF tree

The DCF tree is a height-balanced tree as depicted in Figure 1. Each node in the tree contains

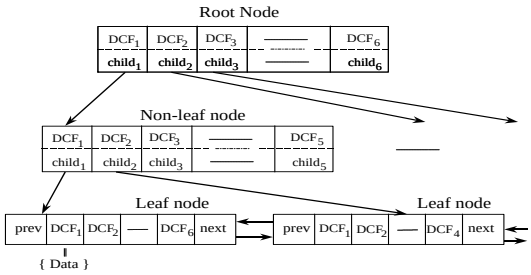

Fig. 1. A DCF tree with branching factor 6.

at most B entries, where B is the *branching factor* of the tree. All node entries 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 DCFs that are produced by merging the DCFs 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 in the tree, the DCF tree embodies a compact representation where the data is summarized by the DCFs of the leaves.

3.3 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 DCFs of the tree leaves are merged 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 3.1. 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 in the cluster c or not.

In our space-bounded algorithm, an input parameter S indicates the maximum space bound. Let E be the maximum size of a DCF entry (note that sparse DCFs 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 DCFs in the leaf node that are farthest apart and we use them as seeds for the new leaves. The remaining DCFs, 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 $d(c,t)$ with the minimum distance of any two DCF entries in the leaf. If $d(c,t)$ 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.

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 DCFs 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 $p(A|c)$, and probability $p(c)$. We employ the *Agglomerative Information Bottleneck (AIB)* algorithm to cluster the DCFs in the leaves and produce clusterings of the DCFs. We note that any clustering algorithm is applicable at this phase of the algorithm.

Phase 3: Associating tuples with clusters. For a chosen value of k, Phase 2 produces k DCFs that serve as *representatives* of k clusters. In the final phase, we perform a scan over the data set and assign each tuple to the cluster whose representative is closest to the tuple.

3.4 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, 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 for a split 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 (both h and U are bounded).

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 DCFs that represent the clusters.

4 Intra-Attribute Value Distance

In this section, we propose a novel application 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 $\tilde{\mathbf{A}} = \{a.DN, a.S, a.G, g.Cr,\ g.T, g.C\}$. 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 $\mathbf{T}$ that contain v, and 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 shows an example of a table when A' is the director attribute.

director	a.DN	a.S	a.G	g.Cr	g.T	g.C	p(d)
Scorsese	1/2	0	0	0	1/2	0	1/6
Coppola	1/2	0	0	0	1/2	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. 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.[3]

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 define a distance measure between tuples as the sum of the distances of the individual attributes. 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, which in turn could help in improving the classification of movie tuples. 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 $v_1, v_2 \in A'$, produces a new value $v_1 \vee v_2$, 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 [9]. 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. [12] proposed STIRR, an algorithm that clusters attribute values. STIRR does not define a distance measure between attribute values and, furthermore, produces just two clusters of values.

5 Experimental Evaluation

In this section, we perform a comparative evaluation of the LIMBO algorithm on both real and synthetic data sets. with other categorical clustering algorithms, including what we believe to be the only other scalable information-theoretic clustering algorithm COOLCAT [3, 4].

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

5.1 Algorithms

We compare the clustering quality of LIMBO with the following algorithms.

ROCK Algorithm. ROCK [13] 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 is an agglomerative algorithm, so it 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. In our experiments, we use the implementation of Guha et al. [13].

COOLCAT Algorithm. The approach most similar to ours is the COOLCAT algorithm [3, 4], by Barbará, Couto and Li. The COOLCAT algorithm is a scalable algorithm that optimizes the same objective function 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. All remaining tuples of the data set are 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. [4].

STIRR Algorithm. STIRR [12] applies 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. 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.

LIMBO Algorithm. In addition to the space-bounded version of LIMBO as described in Section 3, we implemented LIMBO so that the accuracy of the summary model is controlled instead. If we wish to control the accuracy of the model, we use a threshold on the distance $d(c, t)$ to determine whether to merge $DCF(t)$ with $DCF(c)$, thus controlling directly the information loss for merging tuple t with cluster c. The selection of an appropriate threshold value will necessarily be data dependent and we require an intuitive way of allowing a user to set this threshold. Within a data set, every tuple contributes, on "average", $I(A;T)/n$ 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 $\tau(\phi)$. That is, $\tau(\phi) = \phi\frac{I(A;T)}{n}$. 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 (this is equivalent to setting $S = \infty$ in the space-bounded version of LIMBO). We discuss the effect of ϕ in Section 5.4.

To distinguish between the two versions of LIMBO, we shall refer to the space-bounded version as LIMBO$_S$ and the accuracy-bounded as LIMBO$_\phi$. Note that 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$_\phi$.

5.2 Data Sets

We experimented with the following data sets. The first three have been previously used for the evaluation of the aforementioned algorithms [4, 12, 13]. 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*.[4] It contains 435 tuples of votes from the U.S. Congressional Voting Record of

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

1984. Each tuple is a congress-person's vote on 16 issues and each vote is boolean, 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 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. 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.

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.

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 the number of classes in the data set, *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 class labels. Finally, these data sets have up to 10% erroneously entered values. Additional larger synthetic data sets are described in Section 5.6.

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 [14]. 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 [14] 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. [5] 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.

We have also applied LIMBO on Software Reverse Engineering data sets with considerable benefits compared to other algorithms [2].

5.3 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 quantitative measures of clustering performance.

Information Loss, (IL): We use the information loss, $I(A;T) - I(A;C)$ 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. We present IL as a percentage of the initial mutual information lost after producing the desired number of clusters using each algorithm.

[5] Following the approach of Gibson et al. [12], 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.

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

Category Utility, (CU): Category utility [15], 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. CU depends only on the partitioning of the attributes values by the corresponding clustering algorithm and, thus, is a more objective measure. Let $\mathbf{C}$ be a clustering. If A_i is an attribute with values v_{ij}, then CU is given by the following expression:

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

We present CU as an absolute value that should be compared to the CU values given by other algorithms, for the same number of clusters, in order to assess the quality of a specific algorithm.

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 following quality measures.

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

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

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

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

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

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

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

In our experiments, we report values for all of the above measures. For LIMBO and COOLCAT, numbers are averages over 100 runs with different (random) orderings of the tuples.

5.4 Quality-Efficiency trade-offs for LIMBO

In LIMBO, we can control the size of the model (using S) or the accuracy of the model (using ϕ). Both S and ϕ permit a trade-off between the expressiveness (information preservation) of the summarization and the compactness of the model (number of leaf entries in the tree) it produces. For large values of S and 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 both Phase 1 and Phase 2 of the algorithm. For small values of S and 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 now investigate this trade-off for a range of values for S and ϕ. We observed experimentally that the branching factor B does not significantly affect the quality of the clustering. We set $B = 4$, which results in manageable execution time for Phase 1. Figure 2 presents the execution times for LIMBO$_S$ and LIMBO$_\phi$ on the DS5 data set, as a function of S and ϕ, respectively. For $\phi = 0.25$ the Phase 2 time is 210 seconds (beyond the edge of the graph).

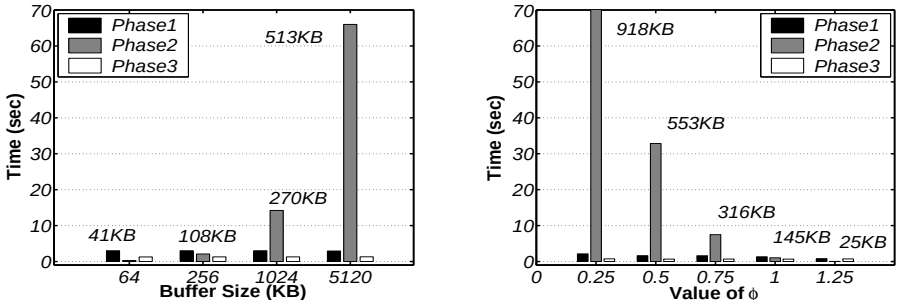

Fig. 2. LIMBO$_S$ and LIMBO$_\phi$ execution times (DS5)

The figures also include the size of the tree in KBytes. In this figure, we observe that for large S and small ϕ the computational bottleneck of the algorithm is Phase 2. As S decreases and ϕ increases the time for Phase 2 decreases in a quadratic fashion. This agrees with the plot in Figure 3, 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 equal to a few seconds for all values of S and ϕ. For $S \leq 256$KB and $\phi \geq 1.0$ the number of leaf entries becomes sufficiently small, so that the computational bottleneck of the algorithm becomes Phase 1. For these values the execution time is dominated by the linear scan of the data in Phase 1.

We now study the change in the quality measures for the same range of values for S and ϕ. In the extreme cases of $S = \infty$ and $\phi = 0.0$, we only merge identical tuples, and no information is lost in Phase 1. LIMBO then reduces to the AIB algorithm, and we obtain the same quality as AIB. Figures 4 and 5 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 $S \geq 256$KB. We observe that for $S \geq 256$KB and $\phi \leq 1.0$ we obtain clusterings of *exactly* the same quality as for $S = \infty$ and $\phi = 0.0$, that is, the AIB algorithm. At the same time, for $S = 256$KB and $\phi = 1.0$ the execution time of the algorithm is only a small fraction of that of the AIB algorithm, which was a few minutes.

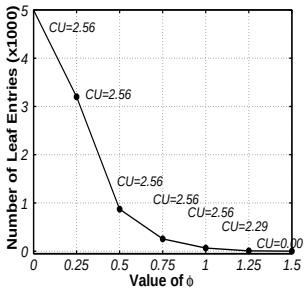

Fig. 3. LIMBO$_\phi$ Leaves (DS5)

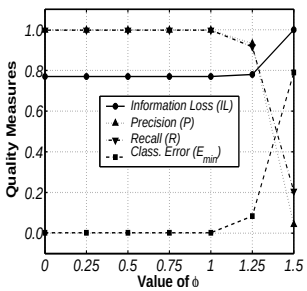

Fig. 4. LIMBO$_\phi$ Quality (DS5)

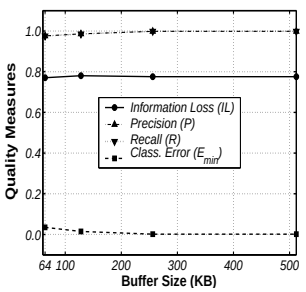

Fig. 5. LIMBO$_S$ Quality (DS5)

Similar trends were observed for all other data sets. There is a range of values for S, and ϕ, 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 4 shows the reduction in the number of leaf entries for each data set for LIMBO$_S$ and LIMBO$_\phi$. The parameters S and ϕ are set so that the cluster quality is almost identical to that of

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

Table 4. Reduction in Leaf Entries

AIB (as demonstrated in Table 6). These experiments demonstrate that in Phase 1 we can obtain significant compression of the data sets at no expense in the final 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. As a result, even for large values of ϕ and small values of S, LIMBO obtains essentially the same clustering quality as AIB, but in linear time.

Votes (2 clusters)						
Algorithm	$size$	IL**(%)**	P	R	E_{min}	CU
LIMBO$(^{\phi=0.0}_{S=\infty})$	384	**72.52**	**0.89**	**0.87**	**0.13**	**2.89**
LIMBO$_S(128KB)$	54	**72.54**	**0.89**	**0.87**	**0.13**	**2.89**
LIMBO$_\phi(1.0)$	23	**72.55**	**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$(^{\phi=0.0}_{S=\infty})$	8124	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
LIMBO$_S(128KB)$	54	**81.46**	**0.91**	**0.89**	**0.11**	**1.71**
LIMBO$_\phi(1.0)$	18	**81.45**	**0.91**	**0.89**	**0.11**	**1.71**
COOLCAT$(s = 1000)$	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 5. Results for real data sets

DS5 (n=5000, 10 attributes, 5 clusters)						
Algorithm	$size$	IL**(%)**	P	R	E_{min}	CU
LIMBO$(^{\phi=0.0}_{S=\infty})$	5000	**77.56**	**0.998**	**0.998**	**0.002**	**2.56**
LIMBO$_S(1024KB)$	232	**77.57**	**0.998**	**0.998**	**0.002**	**2.56**
LIMBO$_\phi(1.0)$	66	**77.56**	**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$(^{\phi=0.0}_{S=\infty})$	5000	**73.50**	**0.997**	**0.997**	**0.003**	**2.82**
LIMBO$_S(1024KB)$	236	**73.52**	**0.996**	**0.996**	**0.004**	**2.82**
LIMBO$_\phi(1.0)$	59	**73.51**	**0.994**	**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 6. Results for synthetic data sets

5.5 Comparative Evaluations

In this section, we demonstrate that LIMBO produces clusterings of high quality, and we compare against other categorical clustering algorithms.

Tuple Clustering Table 5 shows the results for all algorithms on all quality measures for the Votes and Mushroom data sets. For LIMBO$_S$, we present results for $S = 128K$ while for LIMBO$_\phi$, we present results for $\phi = 1.0$. We can see that both version of LIMBO have results almost identical to the quality measures for $S = \infty$ and $\phi = 0.0$, *i.e.*, the AIB algorithm. The *size* entry in the table holds the number of leaf entries 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 5 indicates, LIMBO's quality is superior to ROCK, 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. The

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

Table 7. Statistics for IL(%) and CU

results are shown in Table 6. 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 that is close to that of LIMBO. It is interesting to examine how COOLCAT behaves when we consider other statistics. In Table 7, 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 $S = 128$KB and $\phi = 1.0$, which are very similar to those for $S = \infty$. 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 is as high as 3.5%. The sample size was 1,000 for all runs. Table 7 indicates that LIMBO behaves in a more stable fashion over different runs (that is, different input orders). Notably, for the Mushroom data set, LIMBO's performance is exactly the same in all runs, while for Votes it exhibits a very low variance. This indicates that LIMBO is not particularly sensitive 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 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. 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.

In terms of computational complexity both LIMBO and COOLCAT include a stage that requires quadratic 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 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 leaves for LIMBO. Results for the Votes and Mushroom data sets are shown in Tables 8 and 9. LIMBO outperforms COOLCAT in all runs, for all quality measures even though execution time is essentially the same for both algorithms. 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 better with smaller sample size, while LIMBO remains essentially unaffected.

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. (Due to lack of space the choice of k is discussed in detail in [1].) The total information loss was 61%. Figure 6 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.

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 8. LIMBO vs COOLCAT on Votes

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 9. LIMBO vs COOLCAT on Mushroom

LIMBO accurately characterize the structure of the web graph. Authorities are clustered in three distinct clusters. Authorities in the same cluster share many hubs, while the those in different clusters have very few hubs in common. The three different clusters correspond to different viewpoints on the issue of abortion. The first cluster consists of "pro-choice" pages. The second cluster consists of "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 [5], despite having no apparent relation to the abortion query. A complete list of the results can be found in [1].[7]

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 4, 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 venues in the data set; 815 are database venues, and 396 are theory venues.[8] Results for $S = 5$MB and $\phi = 1.0$ are shown in Table 10. 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 ($S = 5$MB)	16	**94.02**	**0.90**	**0.89**	**0.12**
LIMBO ($\phi = 1.0$)	47	**94.01**	**0.90**	**0.90**	**0.11**
STIRR	-	98.01	0.56	0.55	0.45

Table 10. Bib clustering using LIMBO & 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 7 and 8 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 ones 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 in

[7] Available at: http://www.cs.toronto.edu/∼periklis/pubs/csrg467.pdf

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

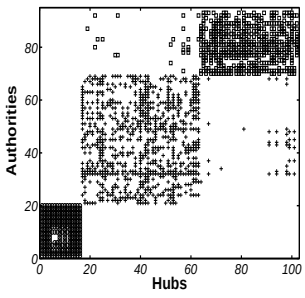

Fig. 6. Web data clusters

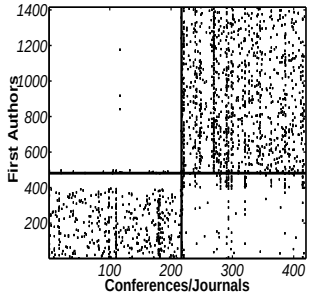

Fig. 7. LIMBO clusters

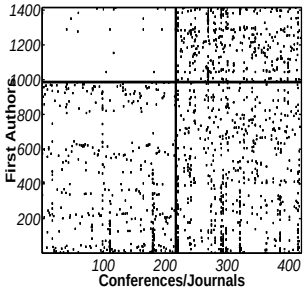

Fig. 8. STIRR clusters

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 venues.

5.6 Scalability Evaluation

In this section, we study the scalability of LIMBO, and we investigate how the parameters affect its execution time. We study the execution time of both LIMBO$_S$ and LIMBO$_\phi$. 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. 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 9 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 10 plots the number of leaves as a function of ϕ.[9] We observe that for the same range of values for ϕ ($1.0 < \phi < 1.5$), LIMBO produces a manageable DCF tree, with a small number of leaves, leading to fast execution time in Phase 2. Furthermore, in all our experiments 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.

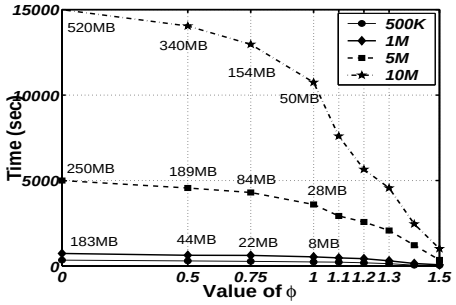

Fig. 9. Phase 1 execution times

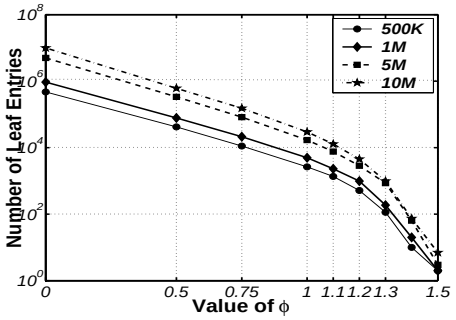

Fig. 10. Phase 1 leaf entries

Thus, for $1.0 < \phi < 1.5$, we have a DCF tree with manageable size, and fast execution time for Phase 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 11 shows the execution time for LIMBO$_S$ and LIMBO$_\phi$ on the data sets we consider. In this figure, we observe that execution

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

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 S and ϕ, and it was the same *across* the data sets (except for IL 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$_S$, with a buffer size of $5MB$, and LIMBO$_\phi$, with $\phi = 1.2$. Figure 12 shows the execution time as a function number of attributes, for different data set sizes. In all cases, execution time increased linearly. Table 11 presents the quality results for all values of

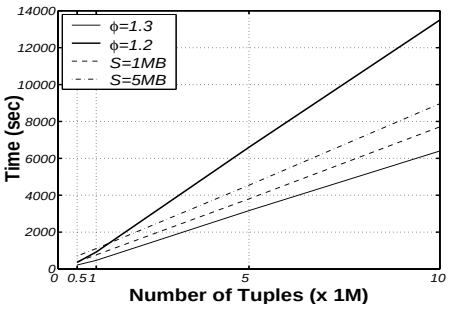

Fig. 11. Execution time (m=10)

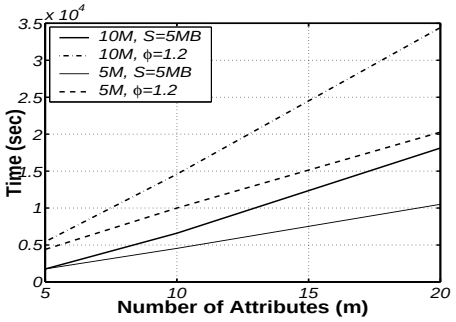

Fig. 12. Execution time

m for both LIMBO algorithms. The quality measures are essentially the same for different sizes of the data set.

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 11. LIMBO$_S$ and LIMBO$_\phi$ quality

Finally, we varied the number of clusters from $k = 10$ up to $k = 50$ in the $10M$ data set, for $S = 5MB$ and $\phi = 1.2$. As expected from the analysis of LIMBO in Section 3.4, the number of clusters affected only Phase 3. Recall from Figure 2 in Section 5.4 that Phase 3 is a small fraction of the total execution time. Indeed, as we increase k from 10 to 50, we observed just 2.5% increase in the execution time for LIMBO$_S$ and just 1.1% for LIMBO$_\phi$,

6 Other Related Work

CACTUS, [10], 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 [20]. The Information Bottleneck method has been used in an agglomerative hierarchical clustering algorithm [18] and applied to the clustering of documents [19]. Recently, Slonim and Tishby [17] 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 over the data for different initial partitions. In the future, we plan to experiment with sIB in Phase 2 of LIMBO.

Finally, an algorithm that uses an extension to BIRCH [21] is given by Chiu, Fang, Chen, Wand and Jeris [6]. Their approach assumes that the data follows a multivariate normal distribution. The performance of the algorithm has not been tested on categorical data sets.

7 Conclusions and Future Directions

We have evaluated the effectiveness of LIMBO 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 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 tuples or values. With our space-bounded version of LIMBO (LIMBO$_S$), we can build a model in one pass over the data in a fixed amount of memory while still effectively controlling information loss in the model. These properties make LIMBO$_S$ amenable for use in clustering streaming categorical data [8] In addition, to the best of our knowledge, LIMBO is the only scalable categorical algorithm that is hierarchical. Using its compact summary model, LIMBO efficiently builds clusterings for not just a single value of k, but for a large range of values (typically hundreds). Furthermore, we are also able to produce statistics that let us directly compare clusterings. We are currently formalizing the use of such statistics in determining good values for k. Finally, we plan to apply LIMBO as a data mining technique to schema discovery [16].

References

[1] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik. Limbo: A linear algorithm to cluster categorical data. Technical report, UofT, Dept of CS, CSRG-467, 2003.

[2] P. Andritsos and V. Tzerpos. Software Clustering based on Information Loss Minimization. In *WCRE*, Victoria, BC, Canada, 2003.

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

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

[5] 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.

[6] 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.

[7] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley & Sons, 1991.

[8] D. Barbará. Requirements for Clustering Data Streams. *SIGKDD Explorations*, 3(2), Jan. 2002.

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

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

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

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

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

[14] J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. In *SODA*, SF, CA, 1998.

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

[16] R. J. Miller and P. Andritsos. On Schema Discovery. *IEEE Data Engineering Bulletin*, 26(3):39–44, 2003.

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

[18] N. Slonim and N. Tishby. Agglomerative Information Bottleneck. In *NIPS*, Breckenridge, 1999.

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

[20] 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.

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