Information-Theoretic Tools for Mining Database Structure from Large Data Sets

Periklis Andritsos
University of Toronto
periklis@cs.toronto.edu

Renée J. Miller
University of Toronto
miller@cs.toronto.edu

Panayiotis Tsaparas
Univ. of Rome, "La Sapienza"
tsap@dis.uniroma1.it

ABSTRACT

Data design has been characterized as a process of arriving at a design that maximizes the information content of each piece of data (or equivalently, one that minimizes redundancy). Information content (or redundancy) is measured with respect to a prescribed model for the data, a model that is often expressed as a set of constraints. In this work, we consider the problem of doing data redesign in an environment where the prescribed model is unknown or incomplete. Specifically, we consider the problem of finding structural clues in an instance of data, an instance which may contain errors, missing values, and duplicate records. We propose a set of information-theoretic tools for finding structural summaries that are useful in characterizing the information content of the data, and ultimately useful in data design. We provide algorithms for creating these summaries over large, categorical data sets. We study the use of these summaries in one specific physical design task, that of ranking functional dependencies based on their data redundancy. We show how our ranking can be used by a physical data-design tool to find good vertical decompositions of a relation (decompositions that improve the information content of the design). We present an evaluation of the approach on real data sets.

1. INTRODUCTION

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

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

In this work, we consider the problem of mining a data instance for structural clues that can help in identifying a better data design. Our work is in the spirit of several recent approaches that propose tools to help an analyst in the process of understanding and cleaning a database [21, 10]. However, while these approaches focus on providing summaries that help in the process of integration [21] or querying [10], we focus on summaries that reveal information about the data design and its information content.

To approach this problem, it is important to understand what makes a data design good. Data design has been characterized as a process of arriving at a design that maximizes the information content of each piece of data (or equivalently, one that minimizes redundancy). Information content (and redundancy) is measured with respect to a prescribed model for the data, a model that is often expressed as a set of constraints or dependencies. In their recent work, Arenas and Libkin presented information-theoretic measures for comparing data designs [6]. Given a schema and a set of constraints, the information content of a design is precisely characterized. Their approach has the benefit that it permits two designs for the same database to be compared directly.

However, to characterize the information content, it was necessary to have a prescribed model. Consider the following example.

	$Ename$	$City$	Zip
t_1	Pat	Boston	02139
t_2	Pat	Boston	02138
t_3	Sal	Boston	02139

Figure 1: Examples of Duplication and Redundancy

Clearly, there is duplication of values in this instance. However, what we consider to be redundant will depend on the constraints expressed on the schema. If the functional dependency $Ename \rightarrow City$ holds, then the value `Boston` in tuple t_2 is redundant given the presence of tuple t_1. That is, if we remove this value, it could be inferred from the information in the first tuple. However, the value `Boston` in the third tuple is not redundant. If we lose this value, we will not know the city of `Sal`. So while the value `Boston` is duplicated in t_3, it is not redundant. But if we change the constraints and instead of $Ename \rightarrow City$, we have the dependency $Zip \rightarrow City$, then the situation is reversed. Given t_1, the value `Boston` is redundant in t_3, but not in t_2.

Understanding redundancy is at the heart of database design. Redundancy occurs naturally because it reflects the intuitive thinking process of humans. Humans are naturally associative thinkers and naturally tend to aggregate and bring all the information they have together in their minds since this facilitates the processing of information. However, this is not a good way to store information in a computer. Normalization is the systematic process, during database design, that is used to separate information about different entities or objects of interest into different tables. This allows us to avoid

SIGMOD 2004 June 13-18, 2004, Paris, France.

some of the data redundancy that would occur naturally when associating different types of information together.

However, it is not obvious how to apply normalization or how to define the information content of a database in an environment where the given schema and constraints may be incorrect or incomplete. In this work, we consider this problem. At their core, our techniques find duplicate values. However, unlike techniques based on counting (for example, frequent item-set mining [2]), we use information-theoretic clustering techniques to identify and summarize the information content of duplicate values.

In our approach, rather than viewing the data as being inconsistent or incomplete with respect to a given schema, we consider the schema to be potentially inconsistent or incomplete with respect to a given data instance. Our contributions are the following.

- We propose a set of information-theoretic tools that use clustering to discover duplicate records, sets of correlated attribute values, and groups of attributes that share such values.
- We provide a set of efficient algorithms that can be used to identify duplication in large, categorical data sets. Our algorithms generate compact summaries that can be used by an analyst to identify errors or to understand the information content of a data set.
- We present several applications of our summaries to the data quality problems of duplicate elimination and the identification of anomalous values. We also present applications to the data design problems of horizontally partitioning an integrated or overloaded relation and to ranking functional dependencies based on their information content. For the latter application, we show how our techniques can be used in combination with a dependency miner to help understand and use discovered dependencies.

The rest of the paper is organized as follows. In Section 2, we present related work and in Section 3, we introduce some basic concepts from information theory. In Section 4, we draw the relationship between the existence of duplicate information and clustering. In Section 5, we describe LIMBO [5], the scalable information-theoretic clustering algorithm that we use, and in Section 6 the tools that use this algorithm to derive structural clues. In Section 7, we introduce a novel algorithm that ranks functional dependencies that hold in an instance. Section 8 presents the experimental evaluation of all our methods and Section 9 concludes our work and discusses additional applications.

2. RELATED WORK

Our work finds inspiration in two independent lines of work. The first on data quality browsers, such as Potter's Wheel [21] and Bellman [10]. The second on the information-theoretic foundation of data design [6, 8].

Data browsers aim to help an analyst understand, query, or transform data. In addition to sophisticated visualization techniques (and adaptive query or transformation processing techniques [21]), they employ a host of statistical summaries to permit real-time browsing. In our work, we consider the generation of summaries that could be used in a data browser for data (re)design. These summaries complement the summaries used in Bellman, where the focus is on identifying co-occurrence of values across different relations (to identify join paths and correspondences between attributes of different relations). Instead, we present a robust set of techniques for identifying and summarizing various forms of duplication within a relation.

Arenas and Libkin provide information-theoretic measures for comparing data designs [6]. Given a schema and a set of constraints, the information content of a design is precisely characterized. The measures used are computational infeasible and they rely on having a clean instance that conforms to a fixed (given) set of constraints. Our techniques are based on an efficient information-theoretic clustering approach. Because we are using unsupervised learning, we are able to create informative summaries even without an accurate model (set of constraints) for the data. We show how our techniques can be used to identify good (or more accurately, better) designs.

Constraint or dependency mining is a related field of study where the goal is to find all dependencies that hold on a given instance of the data (that is, all dependencies that are not invalidated by the instance). Such approaches include the discovery of functional [24, 15, 28] and multi-valued [25] dependencies. Our work complements this work by providing a means of characterizing the redundancy captured by a dependency. We have found that constraint miners reveal hundreds or thousands of potential (approximate) dependencies when they are run on large, real data sets. Our work helps a data analyst understand and quickly identify interesting dependencies within these large sets.

The importance of automated data design and redesign tools has been recognized in reports on the state of database research. Yet, the advances that have been made in this area are largely limited to physical design tools that help tailor a design to best meet the performance characteristics of a workload [20, 3]. Cluster Analysis has been used for vertical partitioning [14, 17, 18], however these techniques partition the attributes of a relation based on their usage in workload queries and not the data. On the other hand, fractured mirrors [20] store different versions of the same database, which are combined during query optimization. This technique is also based on the usage of the attributes in queries.

Finally, our work complements work on duplicate elimination [4, 13, 23, 22]. We propose a technique to identify duplicates based on the information content of tuples. Our approach does not consider how to identify or use distance functions for measuring the error between values (which is the main focus of related work in this area). An interesting area for future work would be on how to combine these techniques.

3. INFORMATION THEORY BASICS

In what follows, we introduce some basic concepts from Information Theory. We can find the following definitions in any textbook of information theory, e.g., [7]. We denote with V a discrete random variable and with $\mathbf{V}$ the set from which V takes its values.[1] If $p(v)$ denotes the probability mass function of V, then the *entropy* $H(V)$ of V is defined as $H(V) = -\sum_{v \in \mathbf{V}} p(v) \log p(v)$. Intuitively, entropy is a measure of the "uncertainty" of variable V. When its value is high, the certainty with which we can predict the values of the random variable is low. If variable V takes on n states, then the maximum value of the entropy is $H_{max}(V) = \log(n)$ and is realized when each of the states is equiprobable, *i.e.*, $p(v) = 1/n$ for all $v \in \mathbf{V}$.

If V and T are two discrete random variables that takes their values from sets $\mathbf{V}$ and $\mathbf{T}$ respectively, then the *conditional entropy* of T given V is defined as follows.

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

Conditional entropy is a measure of the uncertainty with which we can predict the values of random variable T when the values of V are given.

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

Having defined entropy and conditional entropy, we may now introduce *mutual information*, $I(V;T)$. This measure captures the amount of information that the variables convey about each other. It is symmetric, non-negative, and its relation to entropy is given via the equation $I(V;T) = H(V) - H(V|T) = H(T) - H(T|V)$.

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

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

Intuitively, relative entropy $D_{KL}[p\|q]$ measures the error in an encoding that assumes distribution q, when p is the true distribution.

4. CLUSTERING AND DUPLICATION

In this section, we consider how information about tuples, and values within tuples, can be used to build structural information, independent of the constraints that hold on a relation.

Our techniques are designed to find duplication in large data sets. However, because we are considering real data sets which may contain errors or missing values, we will be looking for not only groups containing exact duplicates, but rather groups containing near duplicates or similar values. Hence, our techniques will be based on clustering. In clustering, we identify groups of data objects that are similar and where objects within different groups are dissimilar.

Schemas, like structured query languages that use them, treat data values largely as uninterpreted objects. This property has been called *genericity*, [1], and is closely tied to data independence, the concept that schemas should provide an abstraction of a data set that is independent of the internal representation of the data. That is, the choice of a specific data value (perhaps "Pat" or "Patricia") has no inherent semantics and no influence on the schema used to structure values. The semantics captured by a schema is independent of such choices. For query languages, genericity is usually formalized by saying that a query must commute with all possible permutations of data values (where the permutations may be restricted to preserve a distinguished set of constants) [1].

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

If we are seeking to identify duplication in a set of tuples, we need a measure of similarity that reflects the duplication in these tuples. However, even with such a measure, it is not obvious how to define the quality of the results. On the other hand, for humans there is an intuitive notion of the quality of a clustering. A good clustering is one where the clusters are *informative* about the objects they contain. Given a cluster, we should be able to predict the attribute values of the tuples in this cluster to the maximum possible.

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

To apply information-theoretic techniques, we will treat relations as distributions. For each tuple $t \in \mathbf{T}$ containing a value $v \in \mathbf{V}$, we will set the probability that v appears in tuple t as $1/m$. If we go back to the example of Figure 1, we consider a representation of its tuples as given in Figure 2. Each row corresponds to a tuple. There is a non-zero value in the row for each attribute value of the tuple and the values in a row sum up to one.

	Pat	Sal	$Boston$	02139	02138
t_1	1/3	0	1/3	1/3	0
t_2	1/3	0	1/3	0	1/3
t_3	0	1/3	1/3	1/3	0

Figure 2: Example of tuple representation

The representation we consider for the values of the same data set is given in Figure 3. Each row in the table of Figure 3 characterizes the occurrence of values in tuples and for each value there is a non-zero entry for the tuples in which it appears. Similar to the tuple represenation, each row sums up to one. More formal definitions about both representations are given in the following section.

	t_1	t_2	t_3
Pat	1/2	1/2	0
Sal	0	0	1
$Boston$	1/3	1/3	1/3
02139	1/2	0	1/2
02138	0	1	0

Figure 3: Example of value representation

Using such representations, we consider a number of novel clustering approaches for identifying duplication in tuples, values, and attributes. All of our techniques are founded on the IB method, which we introduce next.

5. CLUSTERING METHODOLOGY

In this section, we describe the *Information Bottleneck, (IB)*, method [27] and a scalable clustering algorithm based on IB.

5.1 *Information Bottleneck*

The intuitive idea of producing clusters that are informative about the objects they contain was formalized by Tishby, Pereira and Bialek [27]. 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. More formally, given a set of objects in set $\mathbf{V}$ expressed over set $\mathbf{T}$, we seek a clustering, $\mathbf{C}$, of $\mathbf{V}$ such that the mutual information $I(C;T)$ remains as large as possible, or otherwise the loss of information described by $I(V;T) - I(C;T)$ is minimum. The *IB* method is generic, imposing no semantics on specific data values.

Finding the optimal clustering is an NP-complete problem [12]. Slonim and Tishby [26], propose a greedy agglomerative approach, the *Agglomerative Information Bottleneck, (AIB)*, algorithm, for finding an informative clustering. If set $\mathbf{V}$ contains q objects, the algorithm starts with the clustering $\mathbf{C}_q$, in which each object $v \in \mathbf{V}$ is assigned to its own cluster. Due to the one-to-one mapping between $\mathbf{C}_q$ and $\mathbf{V}$, $I(C_q;T) = I(V;T)$. The algorithm then proceeds iteratively, for $q-k$ steps (k is the number of desired clusters), reducing the number of clusters in the current clustering by one in each iteration. At step $q-\ell+1$, two clusters c_i and c_j in the ℓ-clustering (the clustering of ℓ clusters) are merged in cluster c^*, to produce a new $(\ell-1)$-clustering $\mathbf{C}_{\ell-1}$. As the algorithm forms

clusters of smaller size, the information that the clustering contains about $\mathbf{T}$ decreases, which means that $I(C_{\ell-1};T) \leq I(C_\ell;T)$. The two clusters c_i and c_j to be merged are chosen such that the loss of information $\delta I(c_i,c_j) = I(C_\ell;T) - I(C_{\ell-1};T)$, in moving to the $(\ell-1)$-clustering, is minimum. After merging clusters c_i and c_j, the new cluster $c^* = c_i \cup c_j$ has [26]:

$$p(c^*) = p(c_i) + p(c_j) \quad (1)$$

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

Tishby et al. [27] show that

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

where D_{JS} is the *Jensen-Shannon (JS)* divergence, defined as follows. Let $p_i = p(T|c_i)$ and $p_j = p(T|c_j)$ 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 $\mathbf{C}_\ell$.

5.2 Scalable Clustering

The *AIB* algorithm suffers from high computational complexity, namely it is quadratic in the number of objects to be clustered, which does not make it appropriate for large sets. We therefore use a scalable version of AIB, called LIMBO (*scaLable InforMation BOttleneck*) [5]. LIMBO uses distributional summaries in order to deal with large data sets. This algorithm is similar in spirit to the BIRCH [29] clustering algorithm and is based on the idea that we do not need to keep whole clusters in main memory, but instead, just sufficient statistics to describe them.

The sufficient statistics are called *Distributional Cluster Features, (DCF)*s, and they will be used to compute the distance between two clusters or between a cluster and a tuple. Let $\mathbf{V}$ be the set of objects to be clustered expressed on the set $\mathbf{T}$, and let V and T be the corresponding random variables. Also let $\mathbf{C}$ denote a clustering of $\mathbf{V}$ and C be the corresponding random variable.

For a cluster $c \in \mathbf{C}$, the DCF of c is defined by the pair

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

where $p(c)$ is the probability of cluster c ,($p(c) = |c|/|\mathbf{V}|$), and $p(T|c)$ is the conditional probability distribution of the values in T given the cluster c. If c consists of a single object $v \in \mathbf{V}$, $p(v) = 1/|V|$ and $p(T|c)$ is computed as described in Section 5.1.

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

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

where $p(c^*)$ and $p(T|c^*)$ are computed using Equations 1 and 2, respectively. Finally, given two clusters c_1 and c_2, 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 after merging the corresponding clusters. $d(c_1,c_2)$ is computed using Equation 3.

The importance of DCFs lies in the fact that they 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. We use a tree data structure, termed DCF-tree. Our scalable 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 object (tuple, attribute value or attribute) with the DCF to which it is closest.

Phase 1: Insertion into the DCF tree. The objects to be clustered are read from disk one at a time and 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. After all objects are inserted in the tree, the DCF-tree embodies a compact representation in which the data set is summarized by the information in the DCFs of the leaves. This summarization is based upon a parameter ϕ which controls the accuracy of the model represented by the tree. More precisely we use the quantity $\phi \cdot \frac{I(V;T)}{|V|}$ as a threshold and merge DCFs at the leaf level of the tree that do not exceed it. Smaller values of ϕ result in more compact summarizations. For instance using $\phi = 0.0$, we only merge identical objects and LIMBO becomes equivalent to AIB.

Phase 2: Clustering. Upon the creation of the tree, we can apply AIB in a much smaller number of objects represented by the DCFs in the leaves.

Phase 3: Associating object 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 object o to the cluster c such that $d(o,c)$ is minimized.

In our approach, we shall use this more scalable clustering algorithm to find duplicate groups of tuples, attribute values and groups of attributes that can be considered similar because they contain such groups.

6. DUPLICATION SUMMARIES

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

The input to our problem here is the set of tuples $\mathbf{T}$ and the set $\mathbf{V} = \mathbf{V}_1 \cup \ldots \cup \mathbf{V}_m$, which denotes the set of all possible values. [2] Let d denote the size of set $\mathbf{V}$. We shall denote with V and T the random variables that range over sets $\mathbf{V}$ and $\mathbf{T}$, respectively.

6.1 Tuple Clustering

In tuple clustering, we find clusters of tuples that preserve the information about the values they contain as much as possible. We represent our data as a $n \times d$ matrix M, where $M[t,v] = 1$ if tuple $t \in \mathbf{T}$ contains attribute value $v \in \mathbf{V}$, and zero otherwise. Note that the vector of a tuple t contains m 1's. For a tuple $t \in \mathbf{T}$, defined over exactly m attribute values, we then define:

$$p(t) = 1/n \quad (4)$$

$$p(v|t) = \begin{cases} 1/m & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Intuitively, we consider each tuple t to be equi-probable and normalize matrix M so that the t^{th} row holds the conditional probability distribution $p(V|t)$. Given M, we can define mutual infor-

[2] We will use the terms *attribute values* and *values* interchangeably.

mation $I(T;V)$ and cluster the tuples in $\mathbf{T}$ into clusters $\mathbf{C}_T$ such that the information loss $I(C_T;V) - I(T;V)$ is minimum.

6.1.1 Duplicate Tuples

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

1. Set a ϕ_T value.
2. Apply Phase 1 to construct tuple summaries.
3. Using leaf DCFs with $p(c*) > 1/n$, apply Phase 3 to associate tuples of the initial data set with their closest summary, where proximity is measured by the information loss between the two.

Step 1 of the above procedure defines the accuracy of the representation of groups of tuples in the summaries at the leaf level of the DCF-tree. If $\phi_T = 0.0$ we merge identical tuples and the representation is exact. As we increase ϕ_T the summaries permit more errors in the duplicate values. Step 2 applies Phase 1 to produce summaries for ϕ_T, while Step 3 associates tuples with summaries that represent *groups of tuples* (more than one tuple). It is then natural to explore the sets of tuples associated with the same summaries to find candidate (almost) duplicate ones.

6.1.2 Horizontal Partitioning

A second application of tuple clustering is the horizontal partitioning of a table. Horizontal partitioning can be useful on tables that have been overloaded with different types of data [9]. For example, an order table originally designed to store product orders may have been reused to store new service orders. In horizontal partitioning, we are seeking to separate out different types of tuples based on the similarities in their attribute values. Specifically, we try to identify whether there is a natural clustering that separates out tuples having different characteristics.

For horizontal partitioning, we do a full clustering. That is, we apply Phase 1 to obtain a small set of summary DCFs. Here, we do not need to set, *a priori*, a threshold on the information loss ϕ, rather we can pick a number of leaves that is sufficiently large (for example, 100 leaves) and apply Phase 1 to obtain 100 summaries. We then apply AIB to these 100 leaves to obtain clusterings for all k values between 1 and 100. We use the following heuristic to identify good k values that may correspond to natural horizontal partitions. We do so by producing statistics that let us directly compare clusterings. These statistics include the rate of change in the mutual information of clusterings ($\delta I(C_k;V)$) and the rate of change in the conditional entropy of clusterings ($\delta H(C_k|V)$) as k varies from 1 to the number of leaf entries. The conditional entropy $H(C_k|V)$ captures the dissimilarity across clusters in the clustering C_k. By examining these derivatives, we are able to identify good clusterings among the different k values. Finally, for each such clustering, we may inspect the clustering to determine if the clustering corresponds to a natural semantic distinction between objects.

6.2 Attribute Value Clustering

As in tuple clustering, we can build clusters of attribute values that maintain as much information about the tuples in which they appear as possible. The parameter ϕ in this case will be denoted by ϕ_V and small values of it allow for the identification of almost perfectly co-occurring groups of attribute values.

A useful connection between tuple and attribute value clustering is drawn when the number of tuples is large. We can use a $\phi_T \neq 0.0$ value to cluster the tuples and then, attribute values can be expressed over the (much smaller) set of tuple clusters instead of individual tuples. Attribute value clustering can then be performed as described above. This technique is referred to as *Double Clustering* [11].

Contrary to tuple clustering, our goal here is to cluster the values represented in random variable V so that they retain information about the tuples in T in which they appear. We represent our data as a $d \times n$ matrix N, where $N[v,t] = 1$ if attribute value $v \in \mathbf{V}$ appears in tuple $t \in \mathbf{T}$, and zero otherwise. Note that the vector of a value v contains $d_v \leq d_i$ 1's, $1 \leq d_i \leq n$. For a value $v \in \mathbf{V}$, we define:

$$p(v) = 1/d \tag{6}$$

$$p(t|v) = \begin{cases} 1/d_v & \text{if } v \text{ appears in } t \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

Intuitively, we consider each value v to be equi-probable and normalize matrix N so that the v^{th} row holds the conditional probability distribution $p(T|v)$. Consider the example relation depicted in Figure 4. Figure 6 (left) shows the normalized matrix N for the relation in Figure 4. Together with N, we define a second matrix, O, which keeps track of the frequency of the attribute values in their corresponding attributes. O is defined as a $d \times m$ matrix were $O[v,A] = d_v$ if value v appears d_v times in attribute V. Intuitively, each entry of matrix $O[v,A]$ stores the support of a value v in attribute A of the relation. For our example, matrix O is given on the right-hand-side of Figure 6. Note that for a value v: $\sum_j O[v,A_j] = d_v$ and for an attribute A: $\sum_l O[v_l,A] = n$.

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

Figure 4: Duplication

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

Figure 5: No perfect correlation

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

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

Figure 6: Matrix N (left) and O (right)

Given matrix N, we can define mutual information $I(V;T)$ and cluster the attribute values in $\mathbf{V}$ into clusters $\mathbf{C}_V$ such that the loss of information $I(C_V;T) - I(V;T)$ is minimum. Intuitively, we seek groups of attribute values in $\mathbf{C}_V$ that retain the information about the tuples in which they appear. Such groups of values may contain duplicate values. We show how we can characterize the sets of attribute values in the clusters of $\mathbf{C}_V$ in the next subsection.

Set $\mathbf{V}$ may entail a large number of values and, thus, the AIB algorithm is infeasible. Thus, we perform the clustering using

LIMBO where the DCFs are extended in order to include information about matrix O. We define the *Attribute Distributional Cluster Features, (ADCF)*, of a cluster of values c^* as a triplet:

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

where $p(c^*)$ and $p(T|c^*)$ are defined as in Section 5.2 and $O(c^*) = \sum_{c \in c^*} O(c)$, *i.e.*, $O(c^*)$ is the sum of the corresponding rows of sub-clusters c^* represents.

In a similar fashion as in tuple clustering we use LIMBO to identify duplicate or almost duplicate values in the data set.

1. Set a ϕ_V value.

2. Apply Phase 1 to construct summaries of the attribute values.

3. Using leaf $ADCF$s with $p(c^*) > 1/d$, apply Phase 3 to associate attribute values of the initial data set with their closest summary, where proximity is measured by the information loss between the two.

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

In our example, performing clustering where we allow no loss of information during merges ($\phi_V = 0.0$), attribute values a and 1 are clustered as are values x and 2. These values have perfect co-occurrence in the tuples of the original relation. The clustering of values for $\phi_V = 0.0$ is depicted on the left-hand-side of Figure 7. The resulting matrix O of our example is depicted on the

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

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

Figure 7: Clustered Matrix N (left) and O (right)

right-hand-side of Figure 7. For the cluster $\{a,1\}$ of values the corresponding row of O becomes $(2,2,0)$, which means that the values of this cluster appear two times in attribute A and two times in attribute B. In general, O stores the cumulative counts of values inside the attributes of a relation. Both N and O contain important information and the next sub-section describes their use in finding duplicate and non-duplicate groups of values.

Before moving to the next sub-section, it is critical to emphasize the role of parameter ϕ_V. As already explained, ϕ_V is used to control the accuracy of the model represented in the leaves of the tree. Besides this, it plays a significant role in identifying "almost" perfect co-occurrences of values. To illustrate this consider the relation in Figure 5. This relation is the same as the one in Figure 4 except for value x in the second tuple. Constructing matrices N and O can be done as explained before. However, when trying to cluster with $\phi_V = 0.0$, our method does not place values x and 2 together since they do not exhibit perfect co-occurrence any more. This may be a result of an erroneous placement of x in the second tuple, or a difference in the representation among data sources that were integrated in this table. Moreover, the functional dependency $C \rightarrow B$ that holds in the relation of Figure 4 now becomes *approximate* in that it does not hold in all the tuples. To capture such anomalies, we perform clustering with $\phi_V > 0.0$, which allows for some small loss of information when merging $ADCF$ leaves in the $ADCF$-tree. Matrices N and O for $\phi_V = 0.1$ are depicted in Figure 8. Our notion of approximation is value-based. This is in

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

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

Figure 8: Matrix N (left) and O (right), ($\phi_V = 0.1$)

contrast to other methods that characterize approximation based on the number of tuples rather than values within tuples [15, 24]. For the data in Figure 8, our method with $\phi_V = 0.1$ determines that value x in the second tuple affects the perfect duplication of pair $\{2,x\}$ less than any other other value.

Tuple and Attribute Value clustering can be combined when the size of the input is large. Specifically, we can define the mutual information $I(T;V)$ and cluster the tuples in $\mathbf{T}$ into clusters represented by C_T. We define ϕ_T so that $C_T \ll T$ and we then use C_T to define $I(V;C_T)$ and to scale-up the clustering of attribute values.

6.3 Grouping Attributes

We have used information loss to define a notion of proximity for values. Based on this, we can define proximity for attributes based on the values they contain. We then cluster attributes using LIMBO. In this application of LIMBO, we control the information loss through a ϕ value denoted by ϕ_A. Typically, the number of attributes m is much less than the number of tuples n, so we use small values of ϕ_A.

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

• C_V^D denotes the set of duplicate groups of attribute values. A set of values c^D belongs to C_V^D if and only if there are at least two tuples t_i, t_j for which both $p(t_i|c^D) \neq 0$ and $p(t_j|c^D) \neq 0$, and at the same time there are at least two attributes A_x and A_y such that both $O[c^D, A_x] \neq 0$ and $O[c^D, A_y] \neq 0$.

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

In our example, it is easy to see from Figure 7 that $C_V^D = \Big\{\{a,1\},\{2,x\}\Big\}$ and $C_V^{ND} = \Big\{\{w\},\{z\},\{y\},\{p\},\{r\}\Big\}$. Now, from these groups, C_V^D contains "interesting" information in that it may lead to a grouping of the attributes such that attributes in the same group contain more duplicate values than attributes in different groups.

If $\mathbf{A}$ is the set of attributes and A the random variable that takes its values from this set, we only express the members of $\mathbf{A}$ over

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

Since set $\mathbf{A}$ usually includes a manageable number of attributes, we can use LIMBO with $\phi_A = 0.0$ and produce a full clustering of the attributes, *i.e.*, produce all clusterings up to $k = 1$. By performing an agglomerative clustering (in Phase 2) over the attributes, at each step we cluster together a pair that creates a group with the maximum possible duplication. For our example, Figure 9 depicts the table of attributes expressed over the set C_V^D as explained above and using the information in matrix O (the rows that correspond to the members of C_V^D). Note that we have the same matrix both for $\phi_V = 0$ and $\phi_V = 0.1$ and that in this example $\mathbf{A} = \mathbf{A}^D$. We name this matrix F. Normalizing rows of F so that they sum up to one, we can proceed with our algorithm and cluster the attributes. All the merges performed are depicted in the *dendrogram* given in Figure 10. A dendrogram is a tree structure that depicts the sequence of merges during clustering together with the corresponding values of distance (or similarity). The horizontal axis of our example shows the information loss incurred at each merging point. Initially, all attributes form singleton clusters. The first merge with the least amount of information loss occurs between attributes B and C and upon that, attribute A is merged with the previous cluster.

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

Figure 9: Matrix F

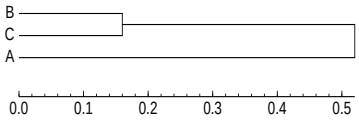

Figure 10: Attr. Cluster Dendrogram

Looking back at our example of Figure 4, we can see that attributes B and C contain more tuples with the duplicate group of values $\{2, x\}$ than A and B do with respect to the group of values $\{a, 1\}$.

In the next section, we show how to use our attribute clustering to rank a set of functional dependencies holding on an instance. Our ranking reveals which dependencies can best be used in a decomposition algorithm to improve the information content of the schema.

7. RANKING DEPENDENCIES

A desirable goal of structure discovery is to derive clues with respect to a potential decomposition of an integrated data set. To this end, we have presented tools for finding exact or approximate relationships among tuples, attribute values and attributes of a data set. However, as we pointed out, duplication is not the same as redundancy. To understand the relationship, we turn to work on mining for constraints (dependencies). There have been several approaches towards discovery of functional [24, 15, 28] and multivalued [25] dependencies. However, none of the approaches presents a characeterizaton of the resulting dependencies. In this section, we present a novel procedure that performs a ranking of the functional dependencies found to hold on an instance, based on the redundancy they represent in the initial relation. We motivate why

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

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

Figure 11: The FD-RANK Algorithm

decompositions over dependencies with a high rank produce better designs than other decompositions.

A good indication of the amount of duplication of the values in C_V^D in a cluster of attributes C_A is the entropy $H(C_V^D|C_A)$. The entropy captures how skewed the distribution of C_V^D in C_A is. Skewed distributions are expected to have higher duplication. The lower the entropy the more skewed the distribution. The following proposition shows that each step in the clustering of attributes minimizes the entropy.

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

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

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

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

The above result justifies the observation that if we scan the dendrogram of a full clustering of the attributes of $\mathbf{A}^D$, the sub-clusters that get merged first are the ones with the higher duplication. Upon the creation of the dendrogram, if we have a set of functional dependencies FD, we can rank them according to how much of the duplication in the initial relation is removed after their use in the decomposition. Given a functional dependency that contains attributes with high duplication, we may then say that the duplicate values in these attributes are redundant. The more redundancy a functional dependency removes from the initial relation the more interesting it becomes for our purposes. Knowing all values of information loss across all merges (in a sequence Q) of attribute sub-clusters, we can proceed with algorithm FD-RANK given in Figure 11 to rank the functional dependencies in FD.

Intuitively, if we have the sequence of all merges Q of the attributes in matrix F (the set C_A^D) with their corresponding information losses, we first initialize the rank of each dependency to be the maximum information loss realized during the full clustering procedure (Step 1.a). For the set of values that participate in a functional dependency (Step 1.b), we update its rank with the highest information loss of a merge where all attributes are merged and

this information loss is below a percentage, specified by ψ, of the maximum information loss (Step 1.c). At this point we can break ties among the functional dependencies that acquire the same ranking based on the number of participating attributes; we rank the ones with more attributes higher than others. Step 2 collapses two functional dependencies with the same antecedent and ranks, into a single functional dependency and, finally, Step 3 orders set FD in ascending order of their corresponding ranks.

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

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

8. EXPERIMENTAL EVALUATION

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

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

• *DB2 Sample Database:* This is a data set we constructed out of the small database that is pre-installed with IBM DB2.[3] We built a single relation after joining three of the tables in this database, namely tables `EMPLOYEE`, `DEPARTMENT` and `PROJECT`. The schema of the tables together with their key (the attributes separated by a line at the top of each box) and foreign key (arrows) constraints are depicted in Figure 12. The relational algebra expression we used to produce the single relation was (we use the initials of each relation):

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

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

• *DBLP Database:* This data set was created from the XML document found at `http://dblp.uni-trier.de/xml/`. This document stores information about different types of computer science publications. In order to integrate the information in a single relation, we chose to use IBM's schema mapping tool that permits the creation of queries to transform the information stored in XML format into relations [19]. We specified a target schema (the schema over which the tuples in the relation are defined) containing the 13 attributes depicted in Figure 13. We specified correspondences between the source XML schema and the attributes in Figure 13. The queries given by the mapping tool where used to create a relation that contained $50,000$ tuples and 57187 attribute values. Each tuple contains information about a single author and, therefore, if a particular publication involved more than one author, the mapping created additional tuples for each one of them. Moreover, the highly heterogeneous information in the source XML document (information regarding conference, journal publications, etc.) introduced a large number of `NULL` values in the tuples of the relation. We used this highly heterogeneous relation to demonstrate the strength of our approaches in suggesting a better structure than the target relation we initially specified.

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

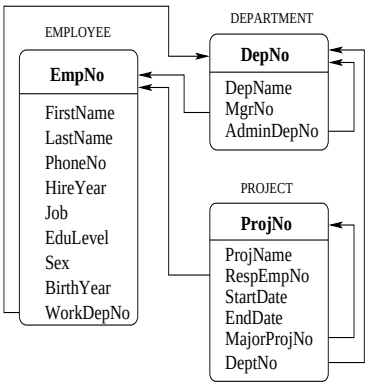

Figure 12: DB2 Sample

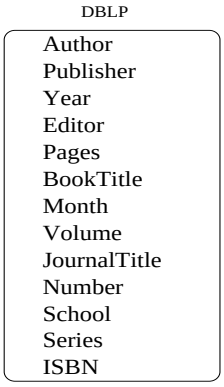

Figure 13: DBLP

Parameters. We observed experimentally that the branching factor of the DCF-tree, B, does not significantly affect the quality of the clustering [5]. We set $B = 4$, so that the Phase 1 insertion time is manageable (smaller values of B lead to higher insertion cost due to the increased height of the DCF tree). We explored a large number of values for ϕ [5]. Generally speaking larger values for ϕ (around 1.0) 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. The value $\phi = 0.0$ makes our method equivalent to the AIB, since only identical objects are merged together.

Functional Dependency Discovery. Our goal is not to rediscover functional dependencies, but rather provide a ranking of any existing set of them. For the purposes of our study we used FDEP [24], as the method to discover functional dependencies. Other methods could also be used.

FDEP first computes all maximal invalid dependencies by pairwise comparison of all tuples and from this set it computes the minimal valid dependencies. FDEP is the algorithm proposed by Savnik and Flach [24] and performs the second step using a depth-first search approach during which the set of maximal invalid dependencies is used to test whether a functional dependency holds and prune the search space.

After computing the functional dependencies using FDEP, we computed the minimum cover using Maier's algorithm [16].

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

• *Relative Attribute Duplication:* Given a set of n tuples, a set $C_A = \{A_1, A_2, \ldots, A_j\}$, with $j \geq 1$, of attributes and the restric-

tion t_{C_A} of tuples on the attributes of C_A (we assume bag semantics here), we define

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

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

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

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

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

Overall $\mathcal{RAD}$ and $\mathcal{RTR}$ offer two different measures of the extent to which values are repeated in the relation. A closer look at $\mathcal{RAD}$ reveals that this measure is more *width-sensitive*. From the definition of conditional entropy, the nominator of the fraction in $\mathcal{RAD}$ can be considered as the weighted entropy of the tuples in a particular set of attributes, where the weights are taken as the probability of this set of attributes. On the other hand, $\mathcal{RTR}$ is more *size-sensitive* in that it can quantify the duplication within different set of tuples taken over the same set of attributes.

8.1 Small scale experiments

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

8.1.1 Application of Tuple Clustering

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

Typographic, Notational and Schema Discrepancies. Such errors may be introduced when the same information is recorded differently in several data sources and then integrated into a single source. For example, this might be the case where the employee numbers are stored following different schemes (typographical or notational errors). On the other hand, this might also be the case where unknown values during integration are filled with `NULL` values in order to satisfy the common integrated schema (schema discrepancies). To identify this type of errors, we introduced tuples in the data set where some of the values in their attributes differ from the values in the corresponding attributes of their matching tuples in the data set. First, we fixed the value of ϕ_T to 0.1 and performed a study with various numbers of erroneous tuples and attribute values within them. Then, we fixed the number of erroneous tuples that we inserted to 5 and performed a study where the ϕ_T and the number of erroneous attribute values varied. We changed the same number of attribute values in each of the inserted tuples every time. The results of both experiments are given in Table 1. From this table, the strength of our method in determining groups of tuples that do not differ a lot is evident. For a small number of "dirty" tuples inserted, the table on the left indicates that our method fails to discover some approximate duplicates only when the number of attribute values on which they differ is more than half the number of attributes in the schema. The same table, shows that as the number of these duplicates increases the performance of the method deteriorates gracefully. The table on the right, where the number of inserted tuples is 5, shows that as the accuracy of the chosen model in the summaries decreases (larger ϕ_T values), the identification of approximate duplicates becomes more difficult, since in these cases more tuples are associated with the constructed summaries.

In general, any duplicates found using tuple clustering are presented to the user and an inspection of the suggested tuples reveals whether these are interesting ones, *i.e.*, duplicates corresponding to the same physical entities represented by the tuples. We should note again the effectiveness of Phase 3, which did not fail to identify the correct correspondences of tuples with their summaries in the leaf entries of the tree.

8.1.2 Application of Attribute Value Clustering

In this section, we present experiments on attribute value clustering. First, we found perfect correlations and then, by increasing ϕ_V approximate ones among the attribute values in the data set.

Value correlations. Using $\phi_T = 0.0$ (no clustering of tuples is performed), and $\phi_V = 0.0$ we first looked for perfect correlations among the values, that is, groups of attributes values that appear exclusively together in the tuples. Our clustering method successfully discovered such groups of values that make up the set C_V^D.

We should note here that although for $\phi_V = 0.0$ we do not expect to get anything more than the perfectly correlated sets of values, we believe that this information is critical in that it aligns our method with that of Frequent Itemset counting [2]. However, with higher values of ϕ_V, we are able to discover potential entry errors.

Value Errors. In this part of the experiments, we introduced errors similar to the ones in tuple clustering, however our goal here is to locate the values that are "responsible" for the errors in the tuple proximity. For better results, we may combine the results of tuple and attribute value clustering. We performed experiments for the same set of tuples that were artificially inserted when we performed tuple clustering, where we counted the number of correct placements of "dirty" values in the clusters of attribute values that appear almost exclusively together in the tuples. That is, we wanted to see if a dirty value was correctly clustered with the values it replaced. Results of these experiments are given in Table 2. Similar to tuple clustering, our method performs well even if the number of inserted tuples is quite large (relative to the size of the initial data set). The correct placement into the attribute value clusters, takes place when the number of altered values covers more than half of the attributes in this data set.

8.1.3 Attribute Grouping

Having information about duplicate values in C_V^D we built matrix F. The dendrogram that was produced for $\phi_V = 0.0$ and $\phi_A = 0.0$ is depicted in Figure 14. We remind the reader that the horizontal axis represents information loss. In this data set, the maximum information loss realized was 0.922. As indicated by the boxes, our attribute grouping has separated the attributes of the initial schemas to a large extent, with the only exception being attributes `EduLevel` and `StartDate`. From the dendrogram, we could also identify that pairs (`EmpNo, FirstName`), (`LastName, PhoneNo`), (`ProjNo, ProjName`) and

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

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

Table 1: DB2 Sample results of erroneous tuples, for $\phi_T = 0.1$ (left) and #Err. Tuples=5 (right)

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

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

Table 2: DB2 Sample results of erroneous values, for $\phi_T = 0.1$ (left) and #Err. Tuples=10 (right)

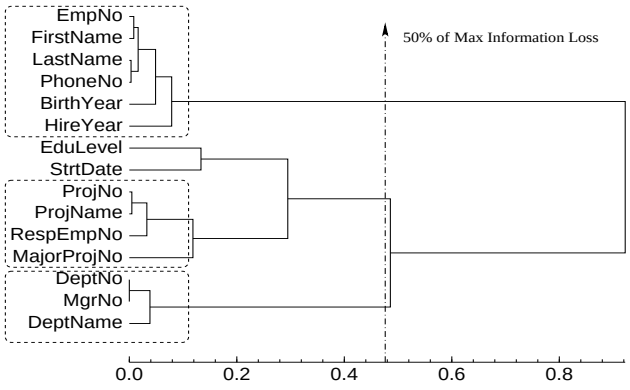

Figure 14: DB2 Sample Attribute Clusters

(`DeptNo, MgrNo`) exhibit the highest redundancy in the data set, a result that agrees with the data instance as well as our intuition.

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

8.1.4 Ranking of Functional Dependencies

Having the sequence of merged attributes, we used FD-RANK to identify which functional dependencies, if used in a decomposition, would help in the removal of high amounts of redundancy in the initial data set. FDEP initially discovered 106 functional dependencies, and the minimum cover consisted of 14 dependencies. The highest ranked dependencies, with $\psi = 0.5$ are given, in order of increasing rank, in the following list:

1. [DeptNo]→[DeptName,MgrNo]

2. [DeptName]→[MgrNo]

3. [EmpNo]→[BirthYear,FirstName,LastName,PhoneNo,HireYear]

4. [ProjNo]→[ProjName,RespEmpNo,StartDate,MajorProjNo]

FD	$\mathcal{RAD}$	$\mathcal{RTR}$
1.	0.947	0.922
2.	0.965	0.922
3.	0.924	0.878
4.	0.872	0.800

Table 3: $\mathcal{RAD}$ and $\mathcal{RTR}$ values for DB2 Sample

Finally, Table 3 shows the $\mathcal{RAD}$ and $\mathcal{RTR}$ values for the previous functional dependencies, if their attributes are used to project the tuples in the the initial relation. Table 3 shows that decompositions of the initial relation according to the ordered list of dependencies would favor the removal of considerable amounts of redundancy. Our ranking identifies dependencies with high redundancy (high $\mathcal{RAD}$ and $\mathcal{RTR}$ values). This is attributed to the fact that correlations of the corresponding attributes are high, however the attribute value clusters in C_V^D have lower support in the initial data set. This fact is also visible in the dendrogram, where the attributes of `Department` have a lower information loss than those of `Employee` and `Project` and according to Proposition 1, they are going to remove more redundancy.

8.2 Large scale experiments

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

The DBLP data set contains integrated information. The relation contains tuples of computer science publications that appeared as part of conference proceedings, journals, theses, etc. As we already argued, this type of information added anomalies due to the discrepancies between the source and the target. More specifically, most conference publications have their `Journal` attributes filled with `NULL` values. Some conference publications, though, appear as part of a `Series` publication, (like SIGMOD publications in the SIGMOD Record journal), and thus a direct projection on attributes that are known in conference (or journal) publications might lead in errors. A better approach would be to first horizontally partition the data set into a small number of groups with similar characteristics.

Before performing horizontal partitioning, we performed attribute grouping in order to identify which attributes would be most useful in such a partitioning. We used $\phi_T = 0.5$, which reduced the number of tuples to 1361 and then performed the attribute grouping with $\phi_A = 0.0$. The result of this grouping is depicted in Figure 15. From the dendrogram, we observe that a number of attributes demonstrate an almost perfect correlation. These are the

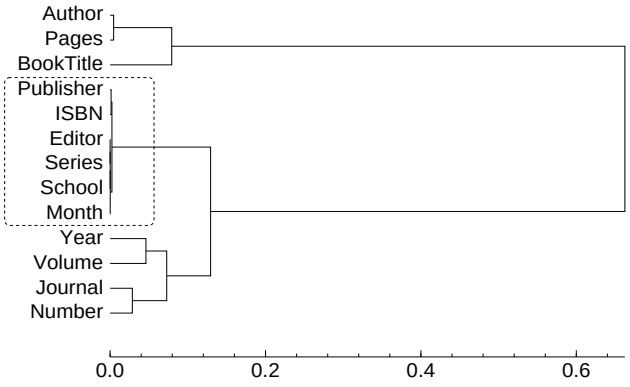

Figure 15: DBLP Attribute Clusters

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

Having a set of attributes with limited non-missing information, the horizontal partitioning produced unexpected results. More precisely, we performed all three Phases of our algorithm to cluster the tuples into 3 groups. The result contained a huge cluster of $49,998$ tuples and two clusters of one tuple in each. However, this result was very informative. All the tuples in the relation are almost duplicates on many attributes and NULL values forced them into the same summaries. Hence, our first observation here is that the six attributes with NULL values can be set aside in the analsysis without considerable loss of information about the tuples. At the same time, if our goal is the definition of a possible schema for the relation, the existence of a huge percentage of NULL values suggests that these attributes contain very large amounts of duplication and should be stored separately, before any horizontal partitioning.

After the previous observation, we projected the initial relation onto the attribute set {Author, Pages, BookTitle, Year, Volume, Journal, Number}. Then we performed a horizontal partitioning of the tuples. Using our heuristic for choosing k as described in Section 6.1.2, we determined that $k = 3$ was a natural grouping for this data. The loss of initial information after Phase 3 was 9.45%, indicating that the clusters are highly informative. The characteristics of the three clusters are given in Table 4. We now consider each cluster separately and due to lack of space we only report results of our attribute grouping and dependency ranking.

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

Table 4: Horizontal Partitions

Cluster 1: This horizontal partition contains all Conference publications where the BookTitle attribute was a non-NULL value in every tuple. Using $\phi_T = 0.5$ and $\phi_V = 1.0$ (given the number of attribute values), we performed the grouping of attributes and the result is given in Figure 16. This dendrogram of the attributes in C_A^D reveals that there is zero distance among the Volume, Journal and Number attributes. Indeed, these are attributes that exclusively contained NULL values in this cluster. In addition, we found almost zero distance between attributes Author and Pages, which happens due to an almost one-to-one mapping between their values (author tuples had unique Pages values in this cluster). Finally, BookTitle is closer to the previous attributes as conference titles are correlated with the authors. Having the sequence of attribute merges, we used FDEP to find functional dependencies that hold in c_1 and FD-RANK with $\psi = 0.5$ to rank them. There were 12 dependencies and the minimum cover contained 11. It should be noted that there was no functional dependency among Author, Pages and BookTitle. The top-two dependencies along with the $\mathcal{RAD}$ and $\mathcal{RTR}$ values of their attributes are given in Table 5. These numbers indicate the significant redundancy reduction we achieve when these dependencies are used in a decomposition. Although the dependencies that were ranked higher did not contain conference attributes, they are highly informative in that the NULL values in the attributes they cover indicate removal of more redundancy. On the other hand Author, Pages and BookTitle have large domains, which makes them less significant for redundancy reduction here.

FD	$\mathcal{RAD}$	$\mathcal{RTR}$
[Volume]→[Journal]	1.0	1.0
[Number]→[Journal]	1.0	1.0

Table 5: Ranked Dependencies for c_1.

Cluster 2: The second horizontal partition contains Journal publications where the Journal, Volume and Number attributes had non-NULL values. Again, using $\phi_T = 0.5$ and $\phi_V = 1.0$ (given the size of the attribute values) the dendrogram produced is depicted in Figure 17. The first observation is that all attributes in C_A^D are generally characteristics of journal publications. Upon that, we see that correlations appear among Journal, Volume, Number and Year, which is something natural to assume in such publications. For example, the SIGMOD Record journal appears once every quarter and the values of the Number attribute are 1 through 4. Finally, using the sequence of merges of the attributes in C_A^D we ranked the functional dependencies holding in this partition. FDEP discovered a set of 12 functional dependencies whose minimum cover contained 11 dependencies. Using FD-RANK with $\psi = 0.5$, the top-two ranked dependencies are given in Table 6 together with the $\mathcal{RAD}$ and $\mathcal{RTR}$ values of the attributes they contain. Note that both dependencies had the same rank. However, the first dependency has more attributes and is ranked at the top.

FD	$\mathcal{RAD}$	$\mathcal{RTR}$
[Author,Volume,Journal,Number]→[Year]	0.754	0.881
[Author,Year,Volume]→[Journal]	0.858	0.982

Table 6: Ranked Dependencies for c_2.

Cluster 3: The last horizontal partition was very small in size, compared to the previous two, and contained miscellaneous publications, such as Technical Reports, Theses, etc. It also contained a very small number of Conference and Journal publications that were written by a single author. The dendrogram produced based on the C_A^D set is given in Figure 18. Given the nature and the size of the cluster, the attribute associations are rather random and we did not find any functional dependencies in the partition, a fact suggesting that this relation does not have internal structure.

Finally, we should point out that the initial horizontal partitioning we used adds an additional benefit to our approach; although the initial relation defined on all 13 attributes contained hundreds

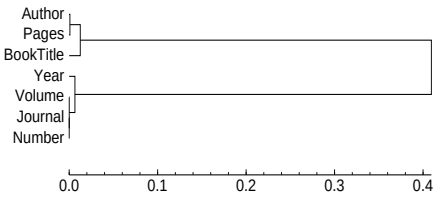

Figure 16: Cluster 1

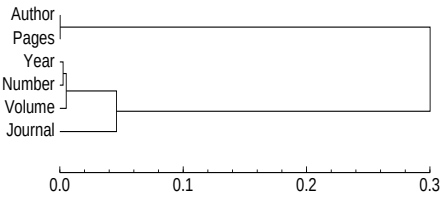

Figure 17: Cluster 2

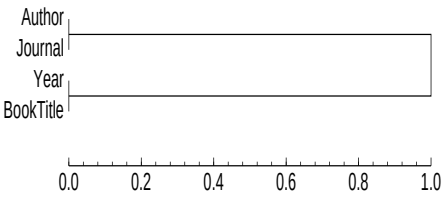

Figure 18: Cluster 3

of functional dependencies, mainly due to the attributes containing NULL values, the clusters we produced had a small number of dependencies (or none) defined on their attributes. This makes the understanding of their schema an easier task.

9. CONCLUSIONS

We have presented a novel approach to discover structure. Our approach defines schema discovery as a problem where the schema of a relation is inconsistent with respect to the data, rather than the opposite. We presented a set of information-theoretic tools based on clustering that discover duplicate, or almost duplicate, tuples and attribute values in a relational instance. From the information collected about the values, we then presented an approach that groups attributes based on the duplication of values. The groups of attributes with large duplication provide important clues for the redefinition of the schema of a relation. Using these clues, we introduced a novel approach to rank the set of functional dependencies that are valid in an instance. Our case studies demonstrated the effectiveness of our methods in discovering integration anomalies and alternative structural properties.

10. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD*, pages 207–216, Washington, D.C., USA, 1993.

[3] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Materialized View and Index Selection Tool for Microsoft SQL Server 2000. In *SIGMOD*, page 608, 2001.

[4] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating Fuzzy Duplicates in Data Warehouses. In *VLDB*, pages 586–597, Hong Kong, China, 2002.

[5] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik. LIMBO: Scalable Clustering of Categorical Data. In *EDBT*, pages 123–146, Heraklion, Greece, 2004.

[6] M. Arenas and L. Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. In *PODS*, pages 15–26, San Diego, CA, USA, 2003.

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

[8] M. M. Dalkilic and E. Robertson. Information Dependencies. In *PODS*, pages 245–253, Dallas, TX, USA, 2000.

[9] T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. John Wiley & Sons, Inc., 2003.

[10] T. Dasu, T. Johnson, S. Muthukrishnan, and V. Shkapenyuk. Mining Database Structure; or, How to Build a Data Quality Browser. In *SIGMOD*, pages 240–251, Madison, WI, USA, 2002.

[11] R. El-Yaniv and O. Souroujon. Iterative Double Clustering for Unsupervised and Semi-supervised Learning. In *ECML*, pages 121–132, Freiburg, Germany, 2001.

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

[13] M. A. Hernández and S. J. Stolfo. The Merge/Purge Problem for Large Databases. In *SIGMOD*, pages 127–138, San Jose, California, 1995.

[14] J. A. Hoffer and D. G. Severance. The Use of Cluster Analysis in Physical Data Base Design. In *VLDB*, pages 69–86, Framingham, MA, USA, 1975.

[15] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Toivonen. TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

[16] D. Maier. Minimum Covers in Relational Database Model. *Journal of the ACM*, 27(4):664–674, Oct. 1980.

[17] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. *TODS*, 9(4):680–710, 1984.

[18] S. B. Navathe and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. In *SIGMOD*, pages 440–450, Portland, OR, USA, 1989.

[19] L. Popa, Y. Velegrakis, M. Hernandez, R. J. Miller, and R. Fagin. Translating web data. In *VLDB*, pages 598–609, Hong Kong, China, Aug. 2002.

[20] R. Ramamurthy and D. J. DeWitt. A case for fractured mirrors. In *VLDB*, pages 430–441, Hong Kong, China, Aug. 2002.

[21] V. Raman and J. M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *VLDB*, pages 381–390, Roma, Italy, 2001.

[22] S. Sarawagi and A. Bhamidipaty. Interactive Deduplication using Active Learning. In *KDD*, pages 269–278, Edmonton, Canada, 2002.

[23] S. Sarawagi-(Editor). *Special Issue on Data Cleaning*. Bulletin of the Technical Committee on Data Engineering, Volume 23(4), December 2000.

[24] I. Savnik and P. A. Flach. Bottom-up induction of functional dependencies from relations. In *AAAI-93 Workshop: Knowledge Discovery in Databases*, pages 174–185, Washington, DC, USA, 1993.

[25] I. Savnik and P. A. Flach. Disocvery of Mutlivalued Dependencies from Relations. *Intelligent Data Analysis Journal*, 4(3):195–211, 2000.

[26] N. Slonim and N. Tishby. Agglomerative Information Bottleneck. In *NIPS-12*, pages 617–623, Breckenridge, CO, 1999.

[27] 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.

[28] C. Wyss, C. Giannella, and E. Robertson. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances. In *DaWaK*, pages 101–110, Munich, Germany, 2001.

[29] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient Data Clustering Method for Very Large Databases. In *SIGMOD*, pages 103–114, Montreal, QB, 1996.