Indexing Uncertain Categorical Data *

Sarvjeet Singh Chris Mayfield Sunil Prabhakar Rahul Shah Susanne Hambrusch
Department of Computer Science, Purdue University
West Lafayette, IN 47907, USA
{sarvjeet, cmayfiel, sunil, rahul, seh}@cs.purdue.edu

Abstract

Uncertainty in categorical data is commonplace in many applications, including data cleaning, database integration, and biological annotation. In such domains, the correct value of an attribute is often unknown, but may be selected from a reasonable number of alternatives. Current database management systems do not provide a convenient means for representing or manipulating this type of uncertainty. In this paper we extend traditional systems to explicitly handle uncertainty in data values. We propose two index structures for efficiently searching uncertain categorical data, one based on the R-tree and another based on an inverted index structure. Using these structures, we provide a detailed description of the probabilistic equality queries they support. Experimental results using real and synthetic datasets demonstrate how these index structures can effectively improve the performance of queries through the use of internal probabilistic information.

1. Introduction

Uncertainty is prevalent in many application domains. Consider for example a data cleaning application that automatically detects and corrects errors [18]. In such an application, there often exists more than one reasonable choice for the correct value of an attribute. Relational database systems, however, do not allow the modeling or storage of this uncertainty directly. Instead, the application is forced either to use a complex model for the data (allowing multiple values, thereby significantly complicating the application and queries), or to pick one of the alternative values (e.g. the most likely choice) to store in the database [28]. While the second option is commonly employed, it results in significant loss of information and lower quality of data. An alternative that allows the application to store the uncertainty in the cleansed value directly is highly desirable.

Data cleansing applications often result in uncertainty in the "cleaned" value of an attribute. Many cleansing tools provide alternative corrections with associated likelihood. For example, data collected from sensors is notoriously imprecise. As part of an ongoing project at Purdue University, the movement of nurses is being tracked in order to study their behavior and effectiveness of current practices. Nurses carry RFID tags as they move about a hospital. Numerous readers located around the building report the presence of tags in their vicinity. The collected data is stored centrally in the form "Nurse 10 in Room 5 at 10:05 am." Each nurse carries multiple tags. The variability in the detection range of readers and the presence of interfering objects makes it impossible to position nurses accurately. Thus the application may not be able to identify with certainty a single location for the nurse at all times. A similar application is discussed in [18].

In the context of automatic data integration, deep web data in the form of dynamic HTML pages can be used to generate relational data [23]. This is a challenging problem and often the mapping from data in the web page to an attribute in the corresponding tuple is unclear. For example, it may be known that the page contains prices for data items, and the web page contains a set of numeric values. It is challenging for a program to determine which value maps to the price for a given item with accuracy. Instead, existing algorithms generate multiple candidates for the value of an attribute, each with a likelihood or probability of being the correct value. Again, due to the lack of support for storing such uncertainty, current applications have to build their own complex models for managing the uncertainty, or just choose the most likely value. Similar issues arise in the domain of integrating unstructured text information with structured databases, such as automatic annotation of customer relationship management (CRM) databases [7], and email search databases.

In summary, there are many applications for which the data exhibits uncertainty in attribute values. Support for such data has been proposed through the development of

*This work was supported by NSF grants IIS 0242421, IIS 0534702, IIS 0415097, AFOSR award FA9550–06–1–0099 and ARO grant DAAD19–03–1–0321

uncertain relational data models [2, 10, 20, 25, 28]. The ORION project [25] is a recent effort aimed at developing an advanced database management system with direct support for uncertain data. The current version of the system, which is developed as an extension of PostgreSQL, supports the storage and querying of uncertain attributes. As with traditional data, there is a need for efficient execution of queries over uncertain data. Existing database index structures are not directly applicable for uncertain data. Much of the recent interest in uncertain data management has focused on the development of models for representing uncertainty and query processing semantics [10, 11]. Indexing support for uncertain data has only been developed for real-valued attributes [9]. These index structures are inapplicable for categorical uncertain data.

This paper addresses the problem of indexing uncertain categorical data represented as a set of values with associated probabilities. We propose two different index structures. We show that these structures support a broad range of probabilistic queries over uncertain data, including the typical equality, probability threshold, and top-K queries. Our index structures can also be used for queries that are only meaningful for uncertain data such as distribution similarity queries. However, due to lack of space, only equality based queries are discussed in the paper. The new indexes are shown to provide efficient execution of these queries with good scalability through experimental validation using real and synthetic data. The contributions of this paper are: i) the development of two index structures for uncertain categorical data; and ii) the experimental evaluation of these structures with real and synthetic data.

2. Data Model and Problem Definitions

Under the categorical uncertainty model [2], a relation can have attributes that are allowed to take on uncertain values. For the sake of simplicity, we limit the discussion to relations with a single uncertain attribute, although the model makes no such restriction. The focus of this paper is on uncertain attributes that are drawn from categorical domains. We shall call such an attribute an *uncertain discrete attribute* (UDA)[1]. Let $R.a$ be a particular attribute in relation R which is uncertain. $R.a$ takes values from the categorical domain D with cardinality $|D| = N$. For a regular (certain) relation, the value of an attribute a for each tuple, $t.a$, would be a single value in D, i.e., $t.a \in D$. In the case of an uncertain relation, $t.a$ is a probability distribution over D instead of a single value. Let $D = \{d_1, d_2, ..., d_N\}$, then $t.a$ is given by the probability distribution $Pr(t.a = d_i)$ for all values of $i \in \{1, ..., N\}$. Thus, $t.a$ can be represented by a probability vector $t.a = \langle p_1, p_2, ..., p_N \rangle$ such that[2] $\sum_{i=1}^{N} p_i = 1$. In many cases, the probability vector is sparse and most p_is are zeros. In such cases, we may represent $t.a$ by a set of pairs $\{(d, p)|(Pr(t.a = d) = p) \wedge (p \neq 0)\}$. Hereafter we denote a UDA by u instead of $t.a$ unless noted otherwise. Also, we denote $Pr(u = d_i)$ by $u.p_i$.

Table 1(a) is for a CRM application with UDA attribute `Problem`. The `Problem` field is derived from the `Text` field in the given tuple using a text classifier. A typical query on this data would be to report all the tuples which are highly likely to have a brake problem (i.e., `Problem = Brake`). Table 1(b) shows a table from a personnel planning database where `Department` is uncertain field. Again, one might be interested in finding employees which are highly likely to be placed in the `Shoes` or `Clothes` department. Formally we define UDA as follows.

Definition 1 *Given a discrete categorical domain $D = \{d_1, .., d_N\}$, an uncertain discrete attribute (UDA) u is a probability distribution over D. It can be represented by the probability vector $u.P = \langle p_1, ..., p_N \rangle$ such that $Pr(u = d_i) = u.p_i$.*

Semantically, we assume that the uncertainty is due to lack of knowledge of the exact value. However, the actual value of attribute is just one of the given possibilities. With this interpretation, we define the semantics of operators on UDAs. Given an element $d_i \in D$, the equality of $u = d_i$ is a probabilistic event. The probability of this equality is given by $Pr(u = d_i) = p_i$. The definition can be extended to equality between two UDAs u and v under the independence assumption as follows:

Definition 2 *Given two UDAs u and v, the probability that they are equal is given by $Pr(u = v) = \sum_{i=1}^{N} u.p_i \times v.p_i$.*

This definition of equality is a natural extension of the usual equality operator for certain data. As with the regular equality operator, this uncertain version can be used to define operations such as joins over uncertain attributes. Example uses of this operator are to compute the probability of pairs of cars having the same problem, or of two employees working for the same department. Analogous to the notion of equality of value is that of distributional similarity. Distribution similarity is the inverse of distributional divergence, which can be seen as a distance between two probability distributions. We consider the following distance functions between two distributions:

L_1: $L_1(u, v) = \sum_{i=1}^{N} |u.p_i - v.p_i|$. This is the Manhattan distance between two distributions.

[1] In this paper, we use the term *discrete* to mean discrete categorical data. The alternative to this is discrete numeric data, on which some more operations can be defined, is not the focus of the paper.

[2] We wish to note that the sum can be < 1 in the case of missing values, and our model can also handle this case without any changes. In this paper, we assume that the sum is 1.

Make	Location	Date	Text	Problem
Explorer	WA	2/3/06	$\cdots$	{(Brake, 0.5), (Tires, 0.5)}
Camry	CA	3/5/05	$\cdots$	{(Trans, 0.2, (Suspension, 0.8)}
Civic	TX	10/2/06	$\cdots$	{(Exhaust, 0.4), (Brake, 0.6)}
Caravan	IN	7/2/06	$\cdots$	{(Trans, 1.0)}

(a)

Employee	Department
Jim	{(Shoes, 0.5),(Sales, 0.5)}
Tom	{(Sales, 0.4), (Clothes, 0.6)}
Lin	{(Hardware, 0.6), (Sales, 0.4) }
Nancy	{(HR, 1.0)}

(b)

Table 1. Examples of Uncertain Relations

L_2: $L_2(u,v) = \sqrt{\sum_{i=1}^{N}(u.p_i - v.p_i)^2}$. This is the Euclidean distance between two distributions.

$KL(u,v)$: $KL(u,v) = \sum_{i=1}^{N} u.p_i \log(u.p_i/v.p_i)$. This is Kullback-Leibler (KL) divergence based on cross entropy measure. This measure comes from information theory. Unlike the above two, this is not a metric. Hence it is not directly usable for pruning search paths but can be used for clustering in an index [26].

Divergence functions such as KL which tend to compare the probability values by their ratios are also important in equality based indexing. Since each probability value in the computation of equality probability is multiplied by a scaling factor, it is meaningful to consider ratios. If UDA u has a high equality probability with UDA q, and $KL(u,v)$ is small, then v is also likely to have a high equality probability with q. This principle is used to cluster UDAs for efficiently answering queries.

There is one major distinction between the notions of distributional similarity and equality between two UDAs. Two distributions may be exactly similar but can have less probability of being equal than two unequal distributions. For example, consider the case where two UDAs u and v have the same vector: $\langle 0.2, 0.2, 0.2, 0.2, 0.2\rangle$. In this case, $Pr(u = v) = 0.2$. However, if $u = \langle 0.6, 0.4, 0, 0, 0\rangle$ and $v = \langle 0.4, 0.6, 0, 0, 0\rangle$, the probability of equality, $Pr(u = v) = 0.48$, is higher even though they are very different in terms of distributional distance.

Having defined the model and primitives, we next define the basic query and join operators. We define equality queries, queries with probabilistic thresholds and queries which give top-k most probable answers. For each of these queries we can define a corresponding join operator.

Definition 3 *Probabilistic equality query (PEQ): Given a UDA q, and a relation R with a UDA a, the query returns all tuples t from R, along with probability values, such that the probability value $Pr(q = t.a) \geq 0$.*

Often with PEQ there are many tuples qualifying with very low probabilities. In practice, only those tuples which qualify with sufficiently high probability are likely to be of interest. Hence the following queries are more meaningful: (1) equality queries which use probabilistic thresholds [2], and (2) equality queries which select k tuples with the highest probability values.

Definition 4 *Probabilistic equality threshold query (PETQ): Given a UDA q, a relation R with UDA a, and a threshold τ, $\tau \geq 0$. The answer to the query is all tuples t from R such that $Pr(q = t.a) \geq \tau$.*

An example PETQ for the data in Table 1(b) determines which pairs of employees have a given minimum probability of potentially working for the same department. In a medical database with an uncertain attribute for possible diagnoses, a PETQ query can be used to identify patients that have similar problems. Analogous to PETQ, we define the top-k query PEQ-top-k, which returns the k tuples with the highest equality probability to the query UDA. Such a query can determine the k patients that are most similar to a given patient in terms of their likely diseases. In our indexing framework, the top-k queries are executed essentially using threshold queries. This is achieved by dynamically adjusting the threshold τ to the kth highest probability in the current result set, as the index processes candidates.

Similar to probabilistic equality-based queries, we can define all of the above queries with distributional similarity. Given a divergence threshold, τ_d, the tuples which qualify for query with UDA q are those whose distributional distance with q is at most τ_d. These are called *distributional similarity threshold queries* (DSTQ).

Definition 5 *DSTQ: Given a UDA q, a relation R with UDA a, a threshold τ_d, and a divergence function F, DSTQ returns all tuples t from R such that $F(q, t.a) \leq \tau_d$.*

There is again a similar notion for DSQ-top-k. The distributional distance can be any of the divergence functions (L_1, L_2, KL) defined above. An example application of a DSTQ is to find similar documents (e.g. web pages) in collections of documents. Although the focus of this paper is on probabilistic equality queries, it is straightforward to adapt our framework of indexing to distributional similarity queries. In addition, distributional distance is a key concept used for clustering in one of our indexes.

We can extend the select query operators above to join operators. Given two UDAs u and v, and a probability threshold τ, u joins with v if and only if $Pr(u, v) \geq \tau$.

Thus, given two relations R and S both having UDA a, we can define threshold equality join:

Definition 6 *Given two uncertain relations R, S both with UDAs a, b, respectively, relation $R \bowtie_{R_a=S_b,\tau} S$ consists of all pairs of tuples r, s from R, S respectively such that $Pr(r.a = s.b) \geq \tau$. This is called probabilistic equality threshold join (PETJ).*

This definition may also be extended to define PEJ-top-k, DSTJ, and DSJ-top-k joins. We wish to note here that joining does introduce new correlations between the resultant tuples and they are no longer independent of each other. Our model only includes the selection based on thresholds. Tracking dependencies requires keeping track of lineage and is not considered in our paper.

Although this paper addresses the general case of categorical uncertainty, it should be noted that for the special case of totally ordered categorical domains, e.g., $D = \{1, .., N\}$, additional inequality probabilistic relations and operators can be defined between two UDAs. For example, we can define $Pr(u \geq v)$, and $Pr(|u - v| < c)$. The notion of probabilistic equality can be slightly relaxed to allow a window within which the values are considered equal. The techniques require to index these queries are discretized versions of those in [9].

3. Index structures

In this section, we describe our index structures to efficiently evaluate queries and joins defined in the previous section. We develop two types of index structures: (1) Inverted index based structures, and (2) R-tree based structures. Although both structures have been explored for indexing set attributes [21, 22], the extension to the case of uncertain data with probabilities attached to members is not straight-forward. Experimental results show there is no clear winner between these two index structures. Section 4 discusses the advantages and disadvantages of each structure with respect to performance, depending on the nature of data and queries.

3.1. Probabilistic Inverted Index

Inverted indexes are popular structures in information retrieval [1]. The basic technique is to maintain a list of lists, where each element in the outer list corresponds to a domain element (i.e. the words). Each inner list stores the ids of documents in which the given word occurs, and for each document, the frequencies at which the word occurs. Traditional applications assume these inner lists are sorted by document id. We introduce a probabilistic version of this structure, in which we store for each value in a categorical domain D a list of tuple-ids potentially belonging to

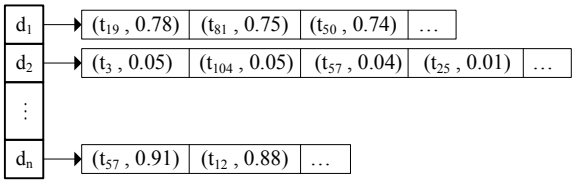

Figure 1. Probabilistic Inverted Index

D. Along with each tuple-id, we store the probability value that the tuple may belong to the given category. In contrast to the traditional structure, these inner lists are are sorted by descending probabilities. Depending on the type of data, the inner lists can be long. In practice, these lists (both inner or outer) are organized as dynamic structures such as B-trees, allowing efficient searches, insertions, and deletions.

Figure 1 shows an example of a probabilistic inverted index. At the base of the structure is a list of categories storing pointers to lists, corresponding to each item in D that occurs in the dataset. This is an inverted array storing, for each value in D, a pointer to a list of pairs. In the list $d_i.list$ corresponding to $d_i \in D$, the pairs (tid, p) store tuple-ids along with probabilities, indicating that tuple tid contains item d_i with probability p. That is, $d_i.list = \{(tid, p)|Pr(tid = d_i) = p > 0\}$. Again, we sort these lists in order of descending probabilities.

We first describe the insert and delete operations which are relatively more straightforward than search. To insert/delete a tuple (UDA) tid in the index, we add/remove the tuple's information in tuple-list. To insert it in the inverted list, we dissect the tuple into the list of pairs. For each pair (d, p), we access the list of d and insert pair (tid, p) in the B-tree of this list. To delete, we search for tid in the list of d and delete it.

Next we describe search algorithms to answer the PETQ query given a UDA q and threshold τ. Let $q = \langle (d_{i_1}, p_{i_1}), (d_{i_2}, p_{i_2}), ..., (d_{i_l}, p_{i_l}) \rangle$ such that $p_{i_1} \geq p_{i_2} \geq ... \geq p_{i_l}$. We first describe the brute force inverted index search which does not use probabilistic information to prune the search. Next we shall describe three heuristics by which the search can be concluded early. These methods search the tuples in decreasing probability order, stopping when no more tuples are likely to satisfy the threshold τ. These optimizations are especially useful when the data or query is likely to contain many insignificantly low probability values. The three methods differ mainly in their stopping criteria and searching directions. Depending on the nature of queries and data, one may be preferable over others.

Inv-index-search. This follows the brute-force inverted index based lookup. For all pairs (d_{i_j}, p_{i_j}) in q, we retrieve all the tuples in the list corresponding to each d. Now, from these candidate tuples we match with q to find out which of these qualify more than the threshold. This is a very simple

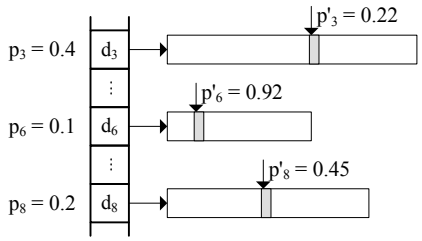

Figure 2. Highest-prob-first Search for $q = \langle (d_3, 0.4), (d_8, 0.2), (d_6, 0.1) \rangle$**.**

method, and in many cases when these lists are not too big and the query involves fewer d_{i_j}, this could be as good as any other method. However, the drawback of this method is that it reads the entire list for every query.

Highest-prob-first. Here, we simultaneously search the lists for each d_{i_j}, maintaining in each $d_{i_j}.list$ a current pointer of the next item to process (see Figure 2). Let p'_{i_j} be the probability value of the pair pointed by the current pointer in this list. At each step, we consider the most promising tuple-id. That is, among all the tuples pointed by current pointers, move forward in that list of d_j where the next pair (tid, p'_{i_j}) maximizes the value $p'_{i_j} p_{i_j}$. The process stops when there are no more promising tuples. This happens when the sum of all current pointer probabilities scaled by their probability in query q falls below the threshold, i.e. when $\sum_{j=1}^{l} p'_{i_j} p_{i_j} < \tau$. This works very well for top-k queries when k is small.

Row Pruning. In this approach, we employ the naive inverted index search but only consider lists of those items in D whose probability in query q is higher than threshold τ. It is easy to check that a tuple, all of whose items have probability less than τ in q, can never meet the threshold criteria. For processing top-k using this approach, we can start examining candidate tuples as we get them and update the threshold dynamically.

Column Pruning. This approach is orthogonal to the row pruning. We retrieve all the lists which occur in the query. Each of these lists is pruned by probability τ. Thus, we ignore the part of the lists which have probability less than the threshold τ. This approach is more conducive to top-k queries.

Note that the above methods require a random access for each candidate tuple. If the candidate set is significantly larger than the actual query answer, then this may result in too many I/Os. We also use no-random-access versions of these algorithms. Nevertheless, we first argue the correctness of our stopping criteria. This applies to all three of the above cases.

Lemma 1 *Let the query* $q = \{(d_{i_j}, p_{i_j}) | 1 \leq j \leq l\}$ *and threshold* τ*. Let* p'_{i_j} *be probability values such that* $\sum_{j=1}^{l} p_{i_j} p'_{i_j} < \tau$*. Then, any tuple* tid *which does not occur in any of the* $d_{i_j}.list$ *with probability at least* p'_{i_j}*, cannot satisfy the threshold query* (q, τ)*.*

Proof: For any such tuple tid, $tid.p_{i_j} \leq p'_{i_j}$. Hence, $\sum_{j=1}^{l} p_{i_j} tid.p_{i_j} < \tau$. Since q only has positive probability values for indices i_j's, $Pr(q = tid) < \tau$. □

In many cases, the random access to check whether the tuple qualifies performs poorly as against simply joining the relevant parts of inverted lists. Here, we use rank-join algorithms with early-out stopping [12, 17]. For each tuple so far encountered in our search, we maintain its lack parameter – the amount of probability value required for the tuple, and which lists it could come from. As soon as the probability values of required lists drop below a certain boundary such that a tuple can never qualify, we discard the tuple. If at any point the tuple's current probability value exceeds the threshold, we include it in the result set. The other tuples remain in the candidate set. A list can be discarded when no tuples in the candidate set reference it. Finally, once the size of this candidate set falls below some number (predetermined or determined by ratio to already selected result) we perform random accesses for these tuples.

3.2. Probabilistic Distribution R-tree (PDR-tree)

In this subsection, we describe an alternative indexing method based on the R-tree [15]. In this index, each UDA u is stored in a page with other similar UDAs which are organized as a tree. The tree-based approach is orthogonal to the inverted index approach where each UDA is shredded and indexed by its components. Here, the entire UDA is stored together in one of the leaf pages of the tree.

Conceptually, we can consider each UDA u as a point in high-dimensional space R^N. These points are clustered to form an index. A major distinction with the regular R-tree is that the queries for uncertain data have very different semantics. They are equivalent to hyperplane queries on the N-dimensional cube. Thus a straight-forward extension of the R-tree or related structures is inefficient due to the nature of queries and the curse of dimensionality (as the number of dimensions – the domain size – can be very large).

We now describe our structure and operations by analogy to the R-tree. We design new definitions and methods for *Minimum Bounding Rectangles* (MBR), the area of an MBR, the MBR boundary, splitting criteria and insertion criteria. The concept of distributional clustering is central to this index. At the leaf level, each page contains several UDAs (as many as fit in one block) using the aforementioned *pairs* representation. Each list of pairs also stores the number of pairs in the list. The page stores the number of UDAs contained in it. Figure 3 shows an example of a PDR-tree index.

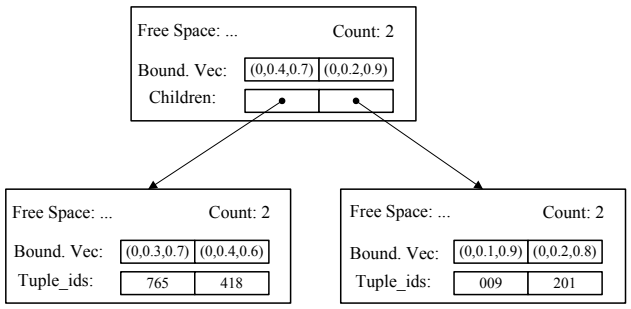

Figure 3. Probabilistic Distribution R-tree

Each page can be described by its MBR boundaries. The MBR boundary for a page is a vector $v = \langle v_1, v_2, ..., v_N \rangle$ in R^N such that v_i is the maximum probability of item d_i in any of the UDA indexed in the subtree of the current page. We maintain the essential pruning property of R-trees; if the MBR boundary does not qualify for the query, then we can be sure that none of the UDAs in the subtree of that page will qualify for the query. In this case, for good performance it is essential that we only insert a UDA in a given MBR if it is sufficiently tight with respect to its boundaries. This will be further explained when we discuss insertion. There are several measures for the "area" of an MBR, the simplest one being the L_1 measure of the boundaries, which is $\sum_{i=1}^{N} v_i$. Our methods are designed to minimize the area of any MBR. Next, we describe how insert, split and PETQ are performed.

Insert(u). To insert a UDA into a page, we first update its MBR information according to u. Next, from the children of the current page we pick the best page to accommodate this new UDA. The following criteria (or combination of these) are used to pick the best page: (1) Minimum area increase: we pick a page whose area increase is minimized after insertion of this new UDA; (2) Most similar MBR: we use distributional similarity measure of u with MBR boundary. This makes sure that even if a probability distribution fits in an MBR without causing an area increase, we may not end up having too many UDAs which are much smaller in probability values. Minimizing this will ensure that we do not hit too many non qualifying UDAs when a query accepts (doesn't prune) an MBR. Even though an MBR boundary is not a probability distribution in the strict sense, we can still apply most divergence measures described in Section 2.

Split( ). There are two alternative strategies to split an overfull page: *top-down* and *bottom-up*. In the top-down strategy, we pick two children MBRs whose boundaries are distributionally farthest from each other according to the divergence measures. With these two serving as the seeds for two clusters, all other UDAs are inserted into the closer cluster. An additional consideration is to create a balanced split, so that two new nodes have a comparable number of objects. No cluster is allowed to contain more that $3/4$ of the total elements. In the bottom-up strategy, we begin with each element forming an independent cluster. In each step the closest pair of clusters (in terms of their distributional distance) are merged. This process stops when only two clusters remain. As with the top-down approach, no cluster is allowed to contain more than $3/4$ of the total elements.

PETQ(q, τ). Given the structure, the query algorithm is straightforward. We do a depth-first search in the tree, pruning by MBRs. Let $\langle\langle \cdot, \cdot \rangle\rangle$ denote the dot-product of two vectors. For a node c, let $c.v$ denote its MBR boundary vector. If an MBR qualifies for the query, i.e., if $\langle\langle c.v, q \rangle\rangle \geq \tau$, our search enters the MBR, else that branch is pruned. At the leaf level, we evaluate each UDA in the page against the query and output the qualifying ones. For top-k queries, we need to upgrade the threshold probability dynamically during the search. An efficiency improvement over the raw depth-first search is to greedily select that child node c first for which $\langle\langle c.v, q \rangle\rangle$ is the maximum. This way we can upgrade our threshold quickly by finding better candidates at the beginning of the search which in turn results in better pruning. The following lemma proves the correctness of the pruning criteria.

Lemma 2 *Consider a node c in the tree. If $\langle\langle c.v, q \rangle\rangle < \tau$ then no UDA stored under the subtree of c qualifies for the threshold query (q, τ).*

Proof: Consider any UDA u stored in the subtree of c. Since an MBR boundary is formed by taking the pointwise maximum of its children MBR boundaries, we can show by induction that $u.p_i \geq c.v.p_i$ and $q_i \geq 0$ for any i, $\langle\langle u, q \rangle\rangle < \langle\langle c.v, q \rangle\rangle < \tau$. Thus, u cannot qualify. □

Compression techniques. An issue that was overlooked earlier is the description of MBR boundaries. Note that an MBR boundary may be described in terms of $|D|$ floating-point values. This may be space inefficient if the data domain is large. Consider the case when $|D| = 1000$ and page size is $8K$. The description of an MBR boundary may not just fit in a page. This results in a small constant fan-out for the index structure. The MBR description does not need to be precise and can be stored in approximate form. Thus, we can apply some lossy compression techniques. With this, the length of the representation of an MBR becomes variable. These variable length objects are packed appropriately. The compression technique needs to make sure that pruning correctness is not compromised. Hence the lossy representation of an MBR boundary vector must be an over-estimation of the actual values. There are two orthogonal approaches to this compression:

Set-Signature based approach: In this case, we define a function $f : D \rightarrow C$ where $|C| < |D|$. Thus C is

the compressed domain. In a given compressed distribution $Pr(c_i) = \max\{Pr(d_j)|f(d_j) = c_i\}$. This approach is akin to that taken by signature trees for set-values attributes [21]. Good correlation detection and clustering methods ensure meaningful f and $|C|$.

Discretized-overestimation: This reduces the number of bits required to represent each p_i in a UDA. Say we allow 2 bits (instead of 4 bytes) to represent each p_i. Then, we essentially approximate p_i by multiple of 0.25 which is greater than p_i. For example, a value of 0.62 will be mapped to 0.75 and can be represented in 2 bits by representing the multiplier 3. When considering more slabs, we may be able to code each multiplier using an optimal number of bits as per its frequency and achieve entropy coding. This also substantially reduces the size of the MBR boundary description.

4. Experimental Evaluation

In this section we present the experimental evaluation of the proposed index structures using real and synthetic datasets. The real dataset is generated by text clustering/categorization of customer service constraints for a major cell phone service provider in the context of CRM databases. The base data consists of 100,000 text documents consisting of complaints, responses, and ensuing communications between customers and service representatives. The dataset CRM1 consists of probability values generated by automatic categorization of the text into 50 categories. Dataset CRM2 is generated by unsupervised fuzzy clustering of the text [19, 24]. Each tuple has a fuzzy membership among 50 clusters.

The synthetic datasets are generated to simulate varying degrees of correlation and sparsity. The Uniform dataset has 5 items and the probability of each item is chosen randomly for all tuples. The Pairwise dataset also has 5 elements but the individual tuples have only 2 non-zero items with roughly equal probabilities. In addition, the total number of item combinations is restricted to 5. Both these datasets have $10k$ tuples. These two datasets represent the two extreme possible scenarios that our algorithms can face.

The dataset Gen3 used for studying scalability with domain size is also generated synthetically. Initially, a number of item groups are picked at random from the domain. The size of the item groups, which determines the fill factor (expected number of non-zero items in a tuple), is distributed geometrically. The expected group size was varied from 3 (in domain size 10) to 10 (in domain size 500). The item probabilities inside a group are chosen randomly.

All experiments are conducted with page size of 8 KB. We measure the number of I/O operations performed for processing queries. We test both equality threshold (PETQ) and PETQ-top-k queries. Multiple thresholds and values for k are considered in order to produce queries with varying selectivities. All graphs shown below report the number of I/O operations for executing queries. In order to simulate the effect of buffering, all experiments are conducted with a buffer manager that allocates 100 blocks to each query. A clock replacement algorithm is used to manage the buffer pool. Most graphs show how performance (measured in disk I/Os on the y-axis) is affected by the selectivity of the queries (shown as a percentage on the x-axis).

4.1. Results

Divergence Measures. The first experiment studies the relative performance of the three distribution similarity measures, L_1, L_2, and KL. The results for the CRM1 dataset are shown in Figure 4. The x-axis shows the query selectivity and the y-axis shows the number of disk I/O per query. For low selectivities, the KL measure clearly outperforms L_1 which in turn outperforms L_2. For high selectivities, all three perform similarly for top-K queries while the trend for threshold queries remains the same. The superior performance of KL was observed consistently in all our experiments. Consequently, we do not present the performance of L_1 and L_2 in the remainder of this section. We can also observe that for a given selectivity, the performance of top-k queries is poorer than that of threshold queries by roughly a constant factor. This is because a top-k query needs to explore more tuples in order to guarantee that the selected top-k tuples do indeed give the largest probabilities. This relative behavior of top-k queries versus threshold queries was observed in all our experiments.

Synthetic Data. In this experiment we compare the performance of the two index structures for synthetic datasets: Uniform and Pairwise. The results are shown in Figure 5. The x-axis shows the query selectivity (as a percentage), and the y-axis shows the number of disk I/O per query. For the Uniform dataset, the performance of the inverted index is clearly inferior to that of the PDR-tree. Because each data item included nonzero probabilities in many categories, evaluating the query results in accessing large numbers of lists in the inverted index structure. For the Pairwise dataset, the inverted index yields a much better performance than for the Uniform data. However, the PDR-tree continues to outperform the inverted index even in this case.

Real Datasets. This experiment compares the performance of the two index structures for the two real datasets, CRM1 and CRM2. The results for CRM1 are shown in Figure 6 and those for CRM2 are shown in Figure 7. The overall relative performance of is the same as that for the synthetic datasets. That is, the PDR-tree significantly outperforms the inverted index. Since CRM1 is classification-based data using a training set, it exhibits less uncertainty

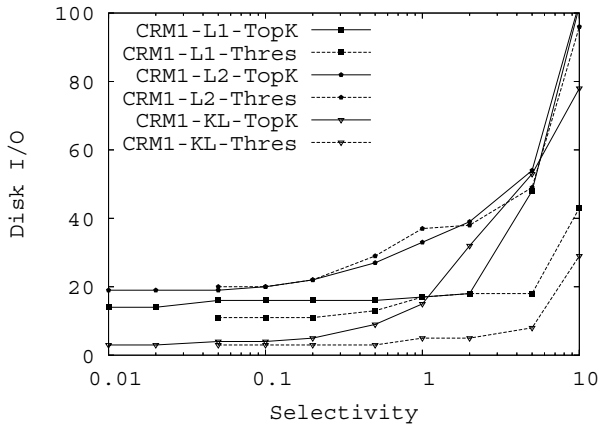

Figure 4. L1 vs L2 vs KL (PDR-tree)

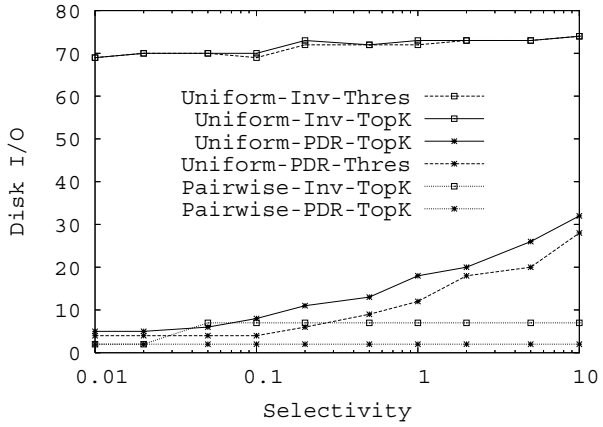

Figure 5. Inverted Index vs PDR-tree (synth)

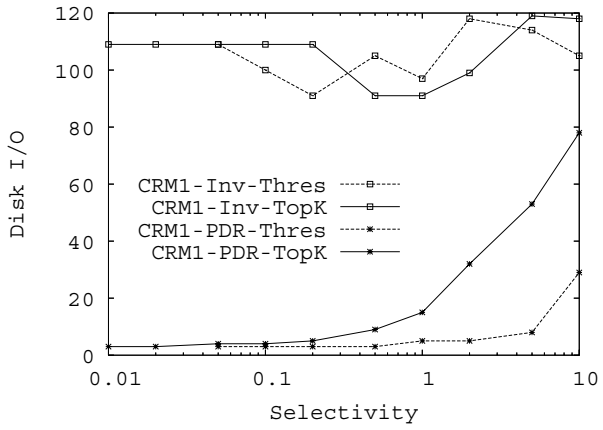

Figure 6. Inverted Index vs PDR-tree (CRM1)

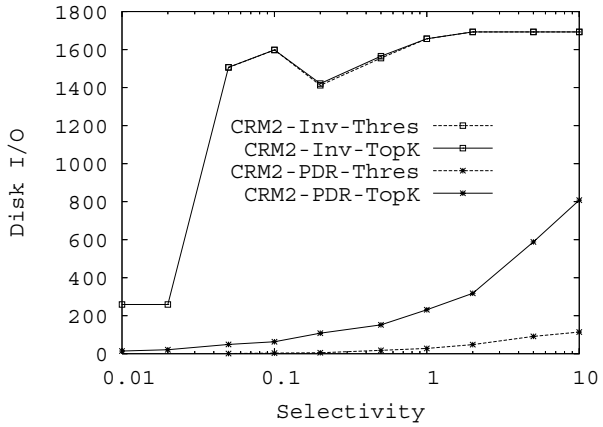

Figure 7. Inverted Index vs PDR-tree (CRM2)

that CRM2 which is based on unsupervised clustering. Consequently, CRM1 is a sparse dataset while CRM2 is more dense. As a result, the performance for CRM1 is about 10 times better than that for CRM2.

Dataset Size. This experiment studies the scalability of the index structures as the size of the dataset is increased. The test is run using the CRM2 data by indexing differing numbers of tuples. Figure 8 shows the results. The x-axis plots the number of tuples in thousands, and the y-axis plots the number of disk I/O per query. As expected, the inverted index scales linearly with dataset size, while the PDR-tree scales sub-linearly.

Domain Size. We now explore the impact of the domain size on index performance. In order to test this behavior, we generate another dataset, Gen3, for which we vary the number of items in the domain from 5 to 500. The number of non-zero entries is in the range of 3 to 10. The results are shown in Figure 9. As the domain size increases, the inverted index improves in performance. This can be attributed to the reduction in the average length of each list as the number of lists increases with domain size (since there is one list for each value in the domain). The charts for the PDR-tree show an initial increase followed by a decrease as the domain size increases. We believe this behavior is related to the data generation process. In particular, the relative number of non-zero entries at both ends of our experimental space are smaller than in the middle. This increase in the relative number of non-zero entries in the middle of the range results in poorer clustering for the PDR-tree.

PDR Split Algorithm. The final experiment studies the relative performance of the *top-down* and *bottom-up* strategies for the split algorithm of the PDR-tree. Figure 10 shows the results with the Uniform dataset. We find that the top-down alternative gives worse performance than the bottom-up alternative. The performance of top-down is caused by outliers in the data that result in poor choices for the initial cluster seeds. A similar relative behavior was observed for the other datasets including the real data.

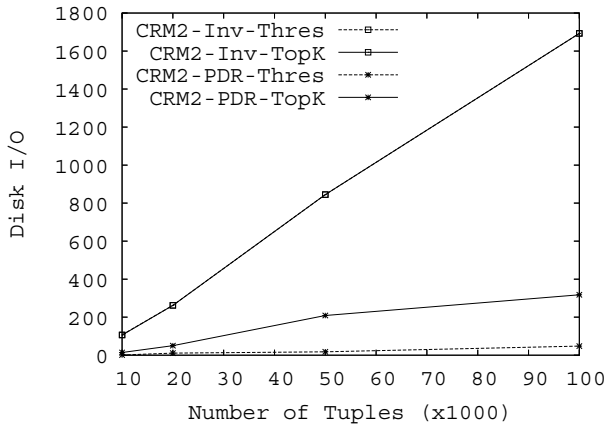

Figure 8. Scalability with Size of Data

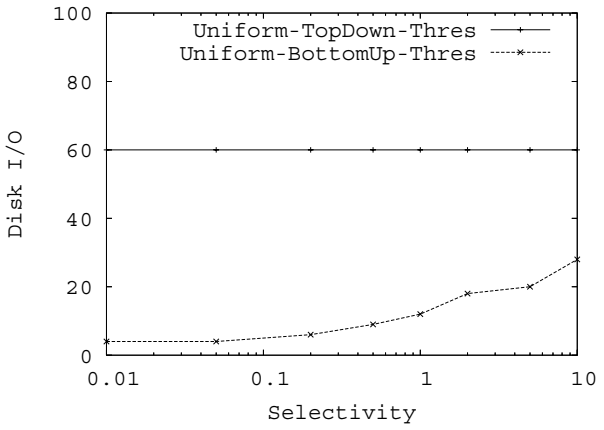

Figure 10. Top-down vs Bottom-up Approach

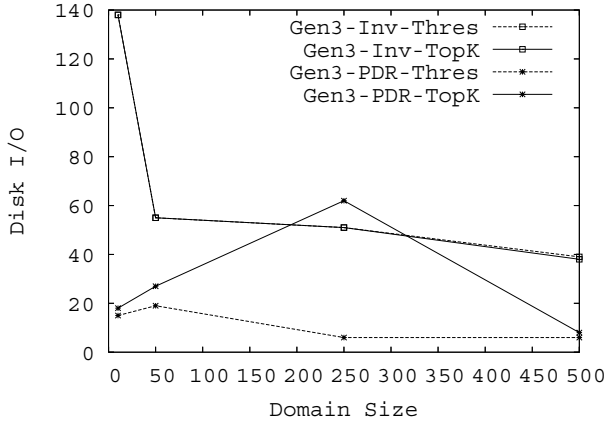

Figure 9. Scalability with Domain Size

5. Related Work

There has been a great deal of work on the development of models for representing uncertainty in databases [27]. An important area of uncertain reasoning and modeling deals with fuzzy sets [6, 14]. Recent work on indexing fuzzy sets is not immediately related to our work as we assume a probabilistic model [3, 4, 5, 16]. Another related area deals with probabilistic modeling of uncertainty which is the focus of this paper. The vast majority of this work has focused on *tuple*-uncertainty. Tuple-uncertainty refers to the degree of likelihood that a given tuple is a member of a relation. This is often captured simply as a probability value attached to the tuple. For example, the results of a text search predicate can be considered to be a relation where the various tuples (representing documents) have some degree of match with the predicate. This match can be treated as a probabilistic value [10]. Tuple uncertainty does not capture the type of uncertainty addressed in this paper where the value of a given attribute within a tuple is uncertain, whereas the tuple is definitely part of the relation.

Another form of data uncertainty is *attribute*-uncertainty wherein the value of an attribute of a tuple is uncertain while other parts of the tuple have precise values. Work on representing uncertainty in attribute values, has been studied earlier [2, 8]. As with most work on tuple-uncertainty, the issue of indexing uncertain attribute data has received little attention. In [9] index structures for real-valued uncertain data are developed. The proposed index structures, are only applicable for continuous numeric domains. These indexes could be modified to work for discrete domains such as integers, but they are inapplicable for general categorical data.

Burdick *et al.* consider the problem of OLAP over uncertain data [7]. They model the uncertainty from text annotators as tuple uncertainty and support aggregation queries over this data. Similar to our work, the example data used in their work is also taken from the CRM domain. Our model differs from theirs in that they limit the classification of the text to one class at a time (e.g. Brake is an attribute of the relation with probability values stored for each tuple), whereas we capture all reported problem areas (i.e. Brake, Transmission, etc. are elements of our uncertain categorical domain). Hence we represent the uncertainty as attribute uncertainty whereas they model it using tuple uncertainty. In their work, the value of interest within the domain (e.g. Brake) is predetermined, while we make no assumptions about the value of interest.

Indexing for set valued attributes has been extensively considered in the literature. Faloutsos developed the notion of signature files to index sets [13]. Indexing set-valued attributes in databases has been considered by Mamoulis [22]. Mamoulis *et al.* also applied indexing for computing join queries over set-valued indexes [21]. Our indexing problem is a generalization of the set model where we have

probability values in addition to the sets. To the best of our knowledge, ours is the first work to address the problem of indexing uncertain categorical data.

6. Conclusion

This paper considered an extension to traditional database management systems that supports uncertainty in data values. In particular, we focused on indexing techniques for categorical uncertain data. Since such uncertainty can be considered an extension of set-values attributes, we proposed the extensions of signature trees and inverted indexes for this problem. Both index structures were shown to have good scalability with respect to dataset and domain size. Experimental results showed that each of these structures performed efficiently, but the nature of the data and query parameters appeared to determine their relative performance. In future work, we shall consider the extension of these indexing techniques for more general uncertain attributes and also develop better techniques depending on the domain specific data needs.

References

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[2] D. Barbará, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992.

[3] P. Bosc and M. Galibourg. Indexing principles for a fuzzy data base. *Information Systems*, 14(6), 1989.

[4] P. Bosc and O. Pivert. About projection-selection-join queries addressed to possibilistic relational databases. *IEEE Transactions on Fuzzy Systems*, 13(1), 2005.

[5] B. Boss and S. Helmer. Index structures for efficiently accessing fuzzy data including cost models and measurements. *Fuzzy Sets and Systems*, 108(1), 1999.

[6] M. Boughanem, F. Crestani, and G. Pasi. Management of uncertainty and imprecision in multimedia information systems: Introducing this special issue. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 11(1):1–4, 2003.

[7] D. Burdick, P. Deshpande, T. Jayram, R. Ramakrishnan, and S. Vaithyanathan. OLAP over uncertain and imprecise data. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2005.

[8] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[9] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2004.

[10] N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2004.

[11] A. Das Sarma, O. Benjelloun, A. Halevy, and J. Widom. Working models for uncertain data. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2006.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. ACM Symp. on Principles of Database Systems*, 2001.

[13] C. Faloutsos. Signature files. In *Information Retrieval: Data Structures & Algorithms*, pages 44–65. Prentice-Hall, 1992.

[14] J. Galindo, A. Urrutia, and M. Piattini. *Fuzzy Databases: Modeling, Design, and Implementation*. Idea Group Publishing, 2006.

[15] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[16] S. Helmer. Evaluating different approaches for indexing fuzzy sets. *Fuzzy Sets and Systems*, 140(1), 2003.

[17] I. Ilyas, W. Aref, and A. Elmagarmid. Supporting top-k join queries in relational databases. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2003.

[18] N. Khoussainova, M. Balazinska, and D. Suciu. Towards correcting input data errors probabilistically using integrity constraints. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, 2006.

[19] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proceedings of the 13th international conference on World Wide Web*, 2004.

[20] L. Lakshmanan, N. Leone, R. Ross, and V. Subrahmanina. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

[21] N. Mamoulis. Efficient processing of joins on set-valued attributes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2003.

[22] N. Mamoulis, D. Cheung, and W. Lian. Similarity search in sets and categorical data using signature tree. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2003.

[23] R. McCann, P. DeRose, A. Doan, and R. Ramakrishnan. Slic: On-the-fly extraction and integration of web data. Technical report, University of Illinois, 2006.

[24] C. Oh, K. Honda, and H. Ichihashi. Fuzzy clustering for categorical multivariate data. In *IFSA World Congress and 20th NAFIPS International Conference*, 2001.

[25] http://orion.cs.purdue.edu/, 2006.

[26] F. C. N. Pereira, N. Tishby, and L. Lee. Distributional clustering of english words. In *Meeting of the Association for Computational Linguistics*, 1993.

[27] S. Prabhakar. Tutorial on probabilistic queries and uncertain data. In *Intl. Conf. on Management of Data (COMAD)*, 2005.

[28] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. Conf. on Innovative Data Systems Research (CIDR)*, 2005.