ProTDB: Probabilistic Data in XML*

Andrew Nierman H. V. Jagadish

Electrical Engineering and Computer Science
University of Michigan
1301 Beal Ave.
Ann Arbor, MI 48109-2122 USA
{andrewdn, jag}@eecs.umich.edu

Abstract

Whereas traditional databases manage only deterministic information, many applications that use databases involve uncertain data. This paper presents a Probabilistic Tree Data Base (ProTDB) to manage probabilistic data, represented in XML.

Our approach differs from previous efforts to develop probabilistic relational systems in that we build a probabilistic XML database. This design is driven by application needs that involve data not readily amenable to a relational representation. XML data poses several modeling challenges: due to its structure, due to the possibility of uncertainty association at multiple granularities, and due to the possibility of missing and repeated sub-elements. We present a probabilistic XML model that addresses all of these challenges. We devise an implementation of XML query operations using our probability model, and demonstrate the efficiency of our implementation experimentally.

We have used ProTDB to manage data from two application areas: protein chemistry data from the bioinformatics domain, and information extraction data obtained from the web using a natural language analysis system. We present a brief case study of the latter to demonstrate the value of probabilistic XML data management.

Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002

1 Introduction

Traditional databases allow for the storage and retrieval of large amounts of data, but do not make any concessions for uncertainty in the data. In many domains, it is difficult, if not impossible, to state all information with 100% certainty. Scientific research, for example, is subject to a great deal of uncertainty and error that cannot be modeled by traditional database systems. Error-prone experimental machinery, polluted samples, and simple human error are a few of the many possible sources of this uncertainty.

With the recent importance of the web, and the many textual (and HTML encoded) sources of information that it makes available, information extraction has become a hot area. The idea is to use natural language analysis tools to create structured representations of free-form text documents. This information extraction is an error-prone endeavor: even the best systems can only hope to be right part of the time. Therefore, it is appropriate to treat this extracted information as uncertain.

We would like to model such uncertainty effectively. We would like to insert data into a database even if it is not known with certainty, along with an appropriate indication of the level of uncertainty associated with it. When queried, these certainty levels should be returned with the query results, giving an indication of how likely an element is to satisfy a particular query. Towards this end, probabilistic relational algebras and databases [4–7, 9, 11, 12, 19, 21, 25] have been proposed by several researchers.

Much of the data in the types of applications where uncertainty is an issue, such as web data and scientific data, are not easy to represent in a relational model, even ignoring issues of uncertainty. The flexibility of a semi-structured model is critical, and XML suggests itself as a natural representation choice. Additionally, uncertain information can frequently be represented much more succinctly in XML than in competing relational representations.

*Work supported in part by NSF under grant IIS-0002356.

Managing probabilistic XML data raises many challenges. XML is structured, so we must assign and interpret probabilities naturally for structurally related elements. Element probabilities can occur at multiple levels, and nested probabilities within a subtree must be considered. Also, XML supports incomplete information gracefully, so we should not insist on having complete probability distributions. A central contribution of this paper is the definition of a framework for probabilistic XML data management introduced in Sections 4 and 5. A property of our model is that probabilities default to 1 when not specified, and all results obtained in this case are identical to those in an equivalent deterministic system.

In Section 6 we show that the results produced are always probabilistically meaningful in light of our probability model. We develop efficient algorithms to evaluate queries against a probabilistic database. These algorithms provide the basis for the implementation of ProTDB, which we have constructed on top of TIMBER [17] a (non-probabilistic) XML database.

We populate our probabilistic XML database and present results from our implementation in Section 7. We show that the overhead of maintaining and manipulating probabilistic information in an XML database is small, compared with the cost of evaluating the same queries against a non-probabilistic XML database.

But first, we present motivating examples in Section 2 and then discuss prior work in Section 3.

2 Application Scenarios

Life is uncertain. This suggests, besides eating dessert first [13], that we manage uncertain data in our databases. In this section we consider two quite distinct applications that require probabilistic data management: information extraction and scientific data management. While these are the two specific applications that we have studied in some depth, the same general probabilistic requirements can be found in a broad range of applications.

2.1 Information Extraction

Vast amounts of information are available to us in the form of plain text. Such information can be hard to query. Traditional information retrieval can at best retrieve documents containing specified keywords, leaving the final information extraction task to a human. Recently, however, there have been major strides in automatically extracting information from text documents. Quite often, this takes the form of directly answering (simple) questions. This extracted information invariably has some form of certainty or probability associated with it. Our extension of this approach, more suited to complex querying, is to extract this information, store it in a suitable (semi-) structured form, and use this representation to evaluate queries.

```
> What is the name of the President of the United States?
George Bush (0.7); George W. Bush (0.4); Bill Clinton (0.2)

> How old is Bill Clinton?
55 (0.3)
```

Figure 1: Question Answering for Information Extraction

For information extraction, we use a question-answering system, NSIR [22, 23], under development at the University of Michigan. Given a question, NSIR sends off an appropriate query to a standard search engine, parses the retrieved documents to identify which phrases are likely to be of a type compatible with the question, and uses these to determine the answer. Since there are many possible sources of error in the process, NSIR makes available multiple possible answers, along with a probability for each. By repeatedly asking NSIR appropriate follow-up questions (see Figure 1 for a stylized sample of the dialog with NSIR), we populated a probabilistic XML database, similar to the small sample fragment shown in Figure 4. This sample makes use of ProTDB constructs whose formal definition will have to wait until Section 5.

2.2 Scientific Data Management

Scientific data is frequently obtained from experiments that suffer from various forms of experimental error. We have collaborated over the past two years with experimental biochemists working in the area of proteomics, and focus our discussion towards the specifics of this case.

A central operation in proteomics is to understand the differentiated production of proteins by organisms under different conditions. There is substantial scope for experimental inaccuracy in technologies used to measure protein quantities. Even greater difficulties arise in actually identifying individual proteins – there are a variety of tools available with varying degrees of reliability, and varying cost to run. There exists the possibility that some proteins have been completely misidentified. In other words, there are many sources of uncertainty, and effective modeling of this uncertainty can be crucial to the subsequent analysis tasks.

We expect that most readers of this paper will be able to connect easier with an example involving information extraction from web news articles. To conserve space, we will say no more about the applicability of our work to proteomics.

3 Related Work

In light of concerns such as those discussed above, there is a large body of work that has studied the representation of probabilistic data in relational databases. Also, information retrieval (IR) has traditionally dealt with issues of probability and ranking; so we briefly

Tuple	President	Age	Spouse
t_1	George W. Bush, [.4, .5] George Bush, [.2, .7]	54, [.2, .25] 55, [.3, .45] 56, [.1, .1] 77, [.1, .15]	Laura Welch, [.5, .6] Barbara Pierce, [.15, .2]
t_2	Bill Clinton, [1, 1]	55, [.2, .5]	NULL

Figure 2: ProbView's "Conceptual" Probabilistic Relation (non-first normal form)

outline recent work in IR that extends classical techniques to semi-structured documents, rather than flat text files.

Probabilistic Relational Databases:

The modeling of uncertain data has been considered in the relational context by several researchers [4–7, 9, 11, 12, 19, 21, 25]. Crucial design decisions for a probabilistic relational database are the unit with which probabilities will be associated (whether at the database, table, tuple, or attribute level), whether the resulting database is to remain in first normal form (1NF), and what probabilistic dependencies are allowed (or assumed).

Applications attach probabilities with attribute values more frequently than with tuples, so associating probabilities with attributes is often more natural than associating the probabilities with tuples. In fact, there can be information loss if attribute probabilities are represented in terms of tuple probabilities (as will be shown shortly). In addition, given multiple probabilistic attributes, representation of probabilities at the tuple level causes a combinatorial explosion of tuples (as we will see in Figures 2 and 3). On the other hand, there is no way to keep a relation in first normal form if attributes can take multiple values with associated probabilities. Unfortunately, non-first normal form data models often lead to complicated algebras and querying mechanisms.

Previous researchers have adopted many different approaches with regards to the design decisions (and tradeoffs) listed above. Notable work that maintains 1NF include [5, 6, 25]. Notable work that uses non-1NF formulations include [4, 11, 12]. ProbView [19] is an attempt to fuse the best of the two options.

ProbView represents data in first normal form, but it assumes that the initial data input is in a non-first normal form. ProbView never operates on this non-first normal form other than to transform it into its annotated 1NF model through the use of a linear program. These two representations are shown in Figures 2 and 3, for the information extraction domain introduced in Section 2.1.

As can be seen in Figure 3, there is a large amount of redundant storage, even for this simple example[1].

[1]In contrast, our XML representation (an example fragment is shown in Figure 4) for probabilistic data mirrors the more succinct structure seen in Figure 2.

President	Age	Spouse	Lower Bound	Upper Bound
George W. Bush	54	Laura Welch	0	.25
George W. Bush	54	Barbara Pierce	0	.2
George W. Bush	55	Laura Welch	0	.4
George W. Bush	55	Barbara Pierce	0	.2
George W. Bush	56	Laura Welch	0	.1
George W. Bush	56	Barbara Pierce	0	.1
George W. Bush	77	Laura Welch	0	.15
George W. Bush	77	Barbara Pierce	0	.15
George Bush	54	Laura Welch	0	.25
George Bush	54	Barbara Pierce	0	.2
George Bush	55	Laura Welch	0	.4
George Bush	55	Barbara Pierce	0	.2
George Bush	56	Laura Welch	0	.1
George Bush	56	Barbara Pierce	0	.1
George Bush	77	Laura Welch	0	.15
George Bush	77	Barbara Pierce	0	.15
Bill Clinton	55	NULL	.2	.5

Figure 3: ProbView's Annotated 1NF Probabilistic Relation

The second problem is related to the loss of information when converting from the base probabilistic relation to the annotated 1NF relation. In Figure 2 we see that Bill Clinton was a president (with 100% certainty) but his current age can only be bounded by $.2 <= Prob(Age_{BillClinton} = 55) <= .5$. After the transformation to ProbView's annotated 1NF (Figure 3), there is a probability range of [.2, .5] attached to the entire tuple `<Bill Clinton, 55, NULL>`. If a subsequent selection query asks for the names of presidents, Bill Clinton will be returned with probability in the range [.2, .5]. This range should be [1, 1] since we knew his name with certainty, while the uncertainty was associated with his age, not his name.

Several approaches utilize a non-first normal form model [9, 11, 12]. The approach followed in [9] generalizes ProbView to permit a tuple to be a complex value (where attributes can actually contain relations). Since this approach uses the same non-first normal form to 1NF conversion process that ProbView uses, it suffers from the same drawbacks as discussed above. In [11, 12] nested relations are used, and an associated probabilistic algebra is developed for this data model. Nested relations bear some resemblance to XML, however, XML is more flexible than nested relations, in allowing greater heterogeneity of structure and incompleteness of information.

Information Retrieval:

With the recent popularity of semi-structured models (and XML in particular), researchers in the IR community have been exploring extensions of the traditional IR techniques and relevance ranking mechanisms in order to apply these concepts to structured documents [2, 3, 8, 10, 16, 20, 24]. Rather than returning entire documents based on term occurrence, these techniques often leverage structure to return only the most relevant section/s of a document. A combination of XML querying mechanisms and IR relevance ranking is used.

XIRQL [10] is a recent probabilistic query language for information retrieval in XML documents. XIRQL develops an algebra that implements the querying capabilities found in XPath [15], and performs the appropriate probabilistic manipulations. This approach is presented in the context of information retrieval, and the overall system includes various query (and associated matching) abstractions in order to simplify the user's interactions with the system.

The role of uncertainty in structured XML documents is also discussed in [24]. In this approach, the probabilities arise due to imprecise matches of query values with data values (rather than uncertainty with the data itself). Also all probabilities are assumed to be independent.

4 Representation Considerations

XML data is structured, and this structure is variable (at least within limits). These features of XML allow for a more natural representation of uncertain information. But they also raise difficult questions with respect to defining a well-founded probability model. We discuss some of the issues below.

4.1 Multiple Granularity

Most relational probabilistic models are only able to associate probabilities with individual tuples – the notion of a tuple is the central "atom" in the relational model, and a probability associated with a tuple is naturally interpreted to mean the probability of the tuple being a member of the corresponding relation (set). In the case of XML, we may have probabilities associated with elements – we have to interpret what exactly this means given that elements can nest under other elements, and more than one of these elements may have an associated probability. We will deal with this issue in the next subsection.

It is also possible to associate probabilities with attribute values of elements, although not directly. XML restricts attributes to have a unique single value. Our approach is to modify the schema in XML to make any attribute into a subelement. These new elements can then be handled by our probabilistic system.

4.2 Structure and Probabilities

Given that a probability is being associated with an element, what is the fact with which we are associating a probability? Presumably it is the probability that the element "exists" in some collection. XML elements are not objects, and may not have any real world correspondence. Also, there is no natural notion of a collection in XML that corresponds to a relation in RDBMS. It is more appropriate to treat an element, including the sub-tree rooted at it, as a structured "tuple" of values. In other words, the existential probability associated with an element should be the probability that the state of the world includes this element and the sub-tree rooted at it.

Formally, the notion of context can be folded in by treating each node not as an independent fact, but rather each node as dependent upon its root to node chain. Each probability in the source XML document is assigned conditioned on the fact that the parent element exists (this happens to be a very natural way of assigning probabilities in all of the applications that we have explored). The single probability, $Prob = p$, that is entered for each element is really a shorthand for a conditional probability table where the element's conditional probability equals p when the parent exists, and the element's probability equals 0 when the parent does not exist.

Consider a chain $A \rightarrow B \rightarrow C$ from the root node A. The source XML document would contain the probabilities $Prob(C|B)$, $Prob(B|A)$, and $Prob(A)$, associated with the nodes C, B, and A, respectively[2]. In order to calculate $Prob(B)$ (in response to a query) we can use Bayes' formula to state $Prob(B|A) = \frac{Prob(A|B) \times Prob(B)}{Prob(A)}$. But, in our data model, a parent must exist with a probability of 1.0 if its child exists with certainty. So, $Prob(A|B) = 1.0$ and after some simple algebra we get $Prob(B) = Prob(B|A) \times Prob(A)$. Similarly, we get $Prob(C) = Prob(C|B) \times Prob(B|A) \times Prob(A)$. All values on the right hand side of these equations can be obtained from the source XML document. In general, the probability of an element e can be found by multiplying the conditional probabilities found in the source XML, along the path from e to the root, and again, each of these conditional probabilities will be available in the source XML file.

4.3 Dependence

Probabilities in an ancestor-descendant chain are related probabilistically, in the manner described above, through conditioning. By default, all other probabilities could be assumed to be independent. Clearly, there are some situations in which this independence assumption does not hold true. While it is possible to assert arbitrary dependence information between any pair (or larger set) of nodes, the general case very quickly leads to computational intractability. Moreover, as our work with example applications has shown, there exist important applications where such arbitrary dependence information is not required.

However, there is one important case of dependence that turns out to be useful, and efficient, to model in many scenarios – mutual exclusion. It is often the

[2] A is the root, so the initial assignment is not conditioned upon any parent element of A.

case that some data item is known to have a single unique value – this value may be unknown, and may be specified probabilistically, but is nonetheless unique. For instance, the age of a person in years is a unique non-negative integer at any point in time: we may say that George Bush is either 55 years old or 56 years old, but the probability that he is both 55 years old and 56 years old should be equal to 0.

4.4 Incompleteness

An important characteristic of XML databases is that they handle incomplete information gracefully. For instance, consider a database with information regarding customers maintained by a business. Suppose we do not have the telephone number in this database for some customer C. This missing number is interpreted as "we have no knowledge of any telephone number for customer C" rather than as "customer C does not have a telephone". Nonetheless, most queries are meaningfully answered despite this lack of complete knowledge. When queried for "customers with telephone number = 123", the database returns exactly the set of customers for whom the telephone number is known to be 123. Customer C is not included in the returned set, even though there exists the possibility that C has a telephone number of 123 after all. When queried for "customers with telephone number not equal to 123", customer C is not returned either. This concept is carried through consistently even in more complex situations. For instance, items purchased by customer C will not be included in sales aggregated by area code.

With probabilistic XML, we would like to continue to handle incompleteness in the same spirit. In fact, in many applications we examined with a need for probabilistic data management, the probability specifications were frequently incomplete. As such, we do not have any requirement that the sum of probabilities of disjoint events add to one. Of course, even with this liberal management of probabilities, the probabilities must lie between 0 and 1, and disjoint probabilities must not add up to more than 1. In order to calculate the probability of negated events when using incomplete probability specifications, we make the additional assumption that the probabilities that are given are correct, and the "slack" probability applies to some unspecified value/s. In other words, we can assume that $Prob(\neg A) = 1 - Prob(A)$.

5 Probability Representation Model

5.1 Simple Constructs

The obvious first step to take is to introduce a probability attribute, `Prob`, that takes a value between 0 and 1 (inclusive). It specifies the probability of a particular element existing in the XML database (at the specified location).

Consider line 46 of Figure 4. It states that the probability of the given chief of state (the chief of state corresponding to Bill Clinton) having an age of 55 is equal to 0.3. If an element does not contain the `Prob` attribute (such as the `countryName` element on line 54), then its probability is assumed to be equal to 1.

It is frequently important to express a probability distribution for an element. Similarly, there may be relationships between (sibling) element value probabilities. These are captured by means of a probabilistic construct called `Dist` (short for distribution). A typical `Dist` element may have multiple `Val` (value) elements as children, each with an associated probability. The probability distribution of each child element is conditioned on the facts recorded in the ancestor chain of the element. Also, the `Dist` construct can record dependencies between its values. Possible distribution types include `mutually-exclusive` and `independent`[3].

In Figure 4, observe the mutually exclusive `Dist` construct (line 29) for the ages of George Bush. We can also have distributions and values over non-leaf nodes, such as the mutually exclusive `Dist` construct (line 19) pertaining to the two possible chiefs of state for the United States. Both of these `ChiefOfState` values are actually entire subtrees (one with a probability of 0.5, lines 20-42 in Figure 4, and one with a probability of 0.2, lines 43-47). In contrast, lines 23-26 indicate that the `name` element has an `independent` distribution, and could have a value of George W. Bush (with a conditional probability of 0.4), George Bush (with a conditional probability of 0.7), or both (with a conditional probability of 0.28).

5.2 DTD Adaptation

To accommodate these probabilistic constructs, the DTD of the source XML document will have to be adapted. To enable the use of the `Prob` attribute, for each element:
`<!ELEMENT elementName ...>`, we must also define the attribute
`<!ATTLIST elementName Prob CDATA "1.0">` in the DTD.
We define the initial `Dist` and `Val` elements as follows:

```
<!ELEMENT Dist (Val+)>
<!ATTLIST Dist type (independent |
mutually-exclusive) "independent">
<!ELEMENT Val (#PCDATA)>
<!ATTLIST Val Prob CDATA "1">
```

To enable the use of distributions (`Dist`) for all of the elements, we modify each element definition, and possibly the `Val` definition as well. If an element, `curElement` is a leaf element (it contains only text, or `#PCDATA`) then its definition in the DTD will be:

[3]In the future, we intend to explore other dependence types, as well as functionally stated distributions such as Gaussians.

```
<countries>
  <country Prob='0.9'>
    <countryName>United States</countryName>
    <coordinates Prob='.9'>
          <latitude>
            <direction>North</direction>
            <degrees Prob='.8'>38</degrees>
            <minutes>00</minutes>
          </latitude>
          <longitude>
            <direction>West</direction>
            <degrees>97</degrees>
          </longitude>
    </coordinates>
    <government>
        <independenceDay Prob='.85'>07/04/1776
        </independenceDay>
        <chiefOfState>
          <Dist type="mutually-exclusive">
            <Val Prob='.5'>
              <title Prob='0.75'>President</title>
              <name>
                <Dist>
                  <Val Prob='.4'>George W. Bush</Val>
                  <Val Prob='.7'>George Bush</Val>
                </Dist>
              </name>
              <age>
                <Dist type="mutually-exclusive">
                  <Val Prob='.2'>54</Val>
                  <Val Prob='.35'>55</Val>
                  <Val Prob='.1'>56</Val>
                  <Val Prob='.15'>77</Val>
                </Dist>
              </age>
              <spouse>
                <Dist type="mutually-exclusive">
                  <Val Prob='.5'>Laura Welch</Val>
                  <Val Prob='.2'>Barbara Pierce</Val>
                </Dist>
              </spouse>
            </Val>
            <Val Prob='.2'>
              <title Prob='0.65'>President</title>
              <name>Bill Clinton</name>
              <age Prob='.3'>55</age>
            </Val>
          </Dist>
        </chiefOfState>
    </government>
  </country>

  <country>
    <countryName>Uruguay</countryName>
    ...
  </country>
</countries>
```

Figure 4: A Fragment of an XML Document for Input to the ProTDB System

`<!ELEMENT curElement (#PCDATA)>`
and it will be changed to:
`<!ELEMENT curElement (#PCDATA|Dist)>`
If `curElement` is not a leaf, we need to make two changes to the DTD:

1. Change the element definition from:
   `<!ELEMENT curElement (prev-def)>` to:
   `<!ELEMENT curElement ((prev-def)| Dist)>`
   (where `prev-def` is the original definition of `curElement`)

2. and add this element's previous definition to the `Val` construct:
   `<!ELEMENT Val (X)>` to be:
   `<!ELEMENT Val (X | (prev-def))>`

The above modifications to a regular DTD, for non-probabilistic data, will enable the creation of XML files that conform to our probabilistic data model.

6 Query Evaluation

Since all probability information is encoded in XML, it is accessible through normal XML query mechanisms. However, manipulation of these probabilities is non-trivial, and not something that a typical user would wish to do directly. Rather, we would like to permit the user to issue queries in the normal manner, as if the queries were against a deterministic database (with equivalent schema) – the user can ignore the additional `Dist` and `Val` elements when querying the data. We should then produce query responses that take probabilistic information into account.

The query responses produced can be restricted to only those with probability above some threshold, or not. They can be sorted in descending order of probability, or not. We may wish to suppress probability information in the result or expose it. Arguments can be advanced in favor of any of the design choices mentioned above, and possibly others as well. One or the other of these choices may be most appropriate in specific application scenarios.

Fortunately, all such choices are naturally implemented as a simple post-processing step after the answer to the query has been computed by the system, but before the answer is presented to the user. The computational cost in this post-processing step is typically very small when compared to the cost of query processing itself. As such, we can focus on the same core probabilistic query processing functionality, geared to producing the "most informative" answer, with the expectation that this answer may be further processed before presentation to the user.

The core query operation in XML is to find matches of a specified query tree pattern (such as the ones in Figure 5) in the data tree/s. Previous work [18] has shown that a single such pattern-match can capture multiple XPath [15] expressions within a single-block

XQuery [14] expression. In the next few subsections we describe, by example, how this pattern-match specification can be executed against a probabilistic XML database.

6.1 Single Node Queries

The query pattern in Figure 5(a) will match a data element with tag `independenceDay` and with a value of "`07/04/1776`". Such an element occurs in line 16 of the source data (Figure 4), with a `Prob` value of 0.85. This probability is conditioned upon the existence of its parent, which in turn is conditioned on its parent, and so on up the ancestor chain. Thus, the probability of the element on line 16 existing (and hence matching the given query pattern) equals $Prob(independenceDay = 07/04/1776) = Prob(independenceDay = 07/04/1776|government) \times Prob(government|country) \times Prob(country|countries) \times Prob(countries) = 0.85 \times 1.0 \times 0.9 \times 1.0 = 0.765$.

There is one result in the result set, and the probability that this result matches the query is equal to 0.765 as shown in Figure 6(a). Notice that the probability of the `independenceDay` node existing in this result has been updated to 1.0, from 0.85. In the result, we update each node's probability to be the probability of that node, given that the query matches. In this simple case, it is obvious that $Prob(independenceDay = 07/04/1776|independenceDay = 07/04/1776) = 1.0$. This node probability updating is also done in the other matching data trees in Figures 6(b, c, d, d'). This approach of conditioning node probabilities upon the matching query is detailed in Section 6.8.

6.2 Conjunctive Queries

Consider the pattern trees in Figures 5(b) and 5(c). Both of these query pattern trees involve conjunctions of constraints. The query pattern tree in Figure 5(b) asks for "chiefs of state with an age equal to 55", and the pattern tree in Figure 5(c) asks for "chiefs of state with a name equal to George Bush, and an age equal to 55". The matching data trees and their associated probability calculations for these queries are shown in Figures 6(b), 6(b'), and 6(c). Notice that some portions of the matching data trees are "grayed out" in the figure. The portions of the data trees that are mapped to the nodes in the probabilistic version of the query pattern tree are colored black, while the "non mapped" nodes that are descendants of some matching node are colored gray. The user may choose to return only the matching (black) nodes, or the entire matching subtree (both the black and gray nodes) . This second option is more typical in the applications we have pursued.

We start with an intuitive description of the probability calculations required for query pattern tree in Figure 5(c). Similarly to the single node case in the previous section, we will need to walk up the tree and multiply probabilities along node to root paths. When considering the first data tree match (shown in Figure 6(c)), the individual probabilities for each separate condition would be calculated as follows:

1. $Prob(name = GeorgeBush) = 0.7 \times 0.5 \times 0.9 = 0.315$ (factoring in the `name` value probability at line 25 of Figure 4, the `chiefOfState` value probability at line 20, and the `country` node probability at line 2).

2. $Prob(age = 55) = 0.35 \times 0.5 \times 0.9 = 0.1575$.

3. $Prob(chiefOfState) = 0.5 \times 0.9 = 0.45$.

In order to calculate the probability of the conjunctive event, $(name = GeorgeBush) \wedge (age = 55) \wedge chiefOfState$ (as in Figure 6(c)), we need to consider the common factors in the three computations above, and factor out the node to root paths that are shared. In particular the ancestor path from the `chiefOfState` node to the root is shared in common between all three computations – counting this path only once results in the correct probability assessment $Prob((name = GeorgeBush) \wedge (age = 55) \wedge chiefOfState) = 0.7 \times 0.35 \times 0.5 \times 0.9 = 0.11025$. In general, to calculate any simple conjunctive query[4] we simply walk up the ancestor chains of each node in the query, multiply in the current conditional probability, and stop traversing a given ancestor chain if we encounter a node we have already factored into the result.

6.3 Adding Disjunction

In order to compute the probability of any general query we must be able to handle disjunctions as well. After standard logic manipulations, we can convert a query tree matching function to a formula $\mathcal{F}$ in disjunctive normal form, where $\mathcal{F} = \mathcal{C}_1 \wedge \ldots \wedge \mathcal{C}_n$, and each $\mathcal{C}_i$ is a sequence of conjunctions. Due to the standard set-based notion of factoring out intersections of events[5], we can convert $\mathcal{F}$ into a series of additions and subtractions that will only involve the computation of probabilities of conjunctions. Based on the previous section, we know how to compute the probability for each of these individual conjunctive formulas.

6.4 Probabilistic Accumulation

In a probabilistic context, some queries implicitly require the accumulation of probabilities. For example, a query for "chiefs of state with an age $\leq$ 56" could

[4]By "simple" we mean a conjunctive query that contains only equality conditions upon the tag names and content, no negation, and no mutually exclusive dependencies in the data. Extensions for these types of queries will be outlined in subsequent sections.

[5]For example, $Prob(\mathcal{C}_1 \vee \mathcal{C}_2 \vee \mathcal{C}_3) = Prob(\mathcal{C}_1) + Prob(\mathcal{C}_2) + Prob(\mathcal{C}_3) - Prob(\mathcal{C}_1 \wedge \mathcal{C}_2) - Prob(\mathcal{C}_1 \wedge \mathcal{C}_3) - Prob(\mathcal{C}_2 \wedge \mathcal{C}_3) + Prob(\mathcal{C}_1 \wedge \mathcal{C}_2 \wedge \mathcal{C}_3)$.

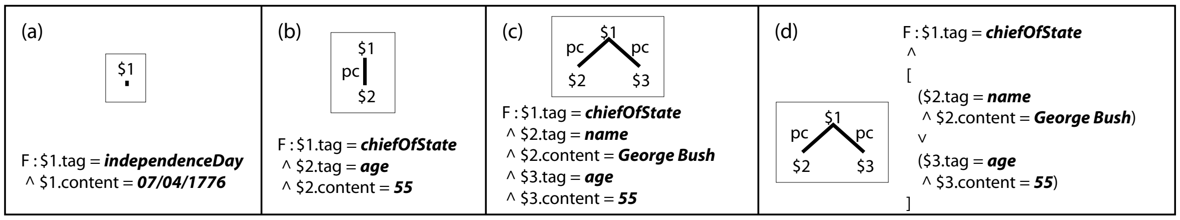
(a)
$1
F : $1.tag = independenceDay
^ $1.content = 07/04/1776
(b)
$1
pc
$2
F : $1.tag = chiefOfState
^ $2.tag = age
^ $2.content = 55
(c)
$1
pc
pc
$2
$3
F : $1.tag = chiefOfState
^ $2.tag = name
^ $2.content = George Bush
^ $3.tag = age
^ $3.content = 55
(d)
$1
pc
pc
$2
$3
F : $1.tag = chiefOfState
^
[
($2.tag = name
^ $2.content = George Bush)
v
($3.tag = age
^ $3.content = 55)
]

Figure 5: Four pattern trees.

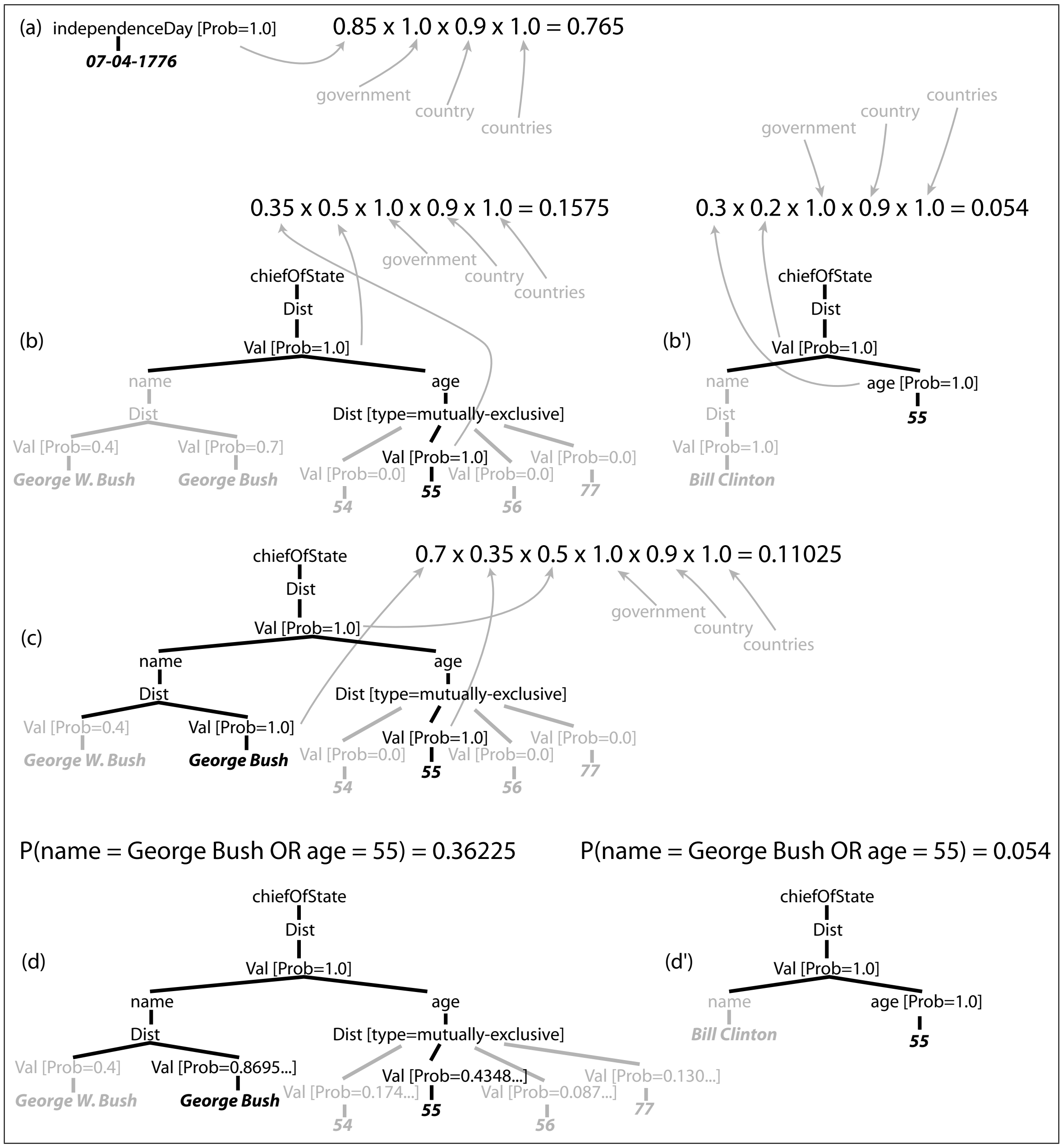

Figure 6: Data trees matching the query pattern trees in figure 5.

yield four separate data tree matches for our data in Figure 4 (corresponding to the matching ages on lines 30, 31, and 32 and on line 46). But the results would be more meaningful if there was one resulting data tree (corresponding to President Bush) for the matching values on lines 30, 31, and 32, and another resulting data tree (corresponding to President Clinton) for the matching value on line 46. The probability calculation for the second of these result trees is the same as in Figure 6(b'). The calculation for the first of these result trees is only slightly more complex: we calculate the probability of the disjunction of these three conditions.

In general, we group multiple results by their common ancestor. The probability of the resulting tree is computed as the disjunction of each individual tree's query formula.

The need for this probabilistic accumulation is obvious when we utilize inequalities such as: $<, >, \leq, \geq$. But, it may even be needed with simple equality queries when there are duplicate values for a given element (or when there are duplicate element names, and the query is over element names and not their actual content).

6.5 Handling Negation

Negation could be handled in the same manner as inequality queries in the previous section. For example, a query for "chiefs of state with an age $\neq 77$" could result in exactly the same computation as discussed in the previous section for the "President Bush" subtree. But, this is not the most computationally effective strategy: the number of disjunctive conditions is based upon the number of possible values for an element, which could be quite large. Instead we take advantage of the special property of negation, namely that $Prob(\neg A) = 1 - Prob(A)$.

If there exist some non-negated conjuncts in f we perform a transformation so that we only need to calculate the conjunctive probabilities of non-negated conjuncts (as in Section 6.2). As an example, we can compute $Prob(A \wedge B \wedge \neg C)$ as $Prob(A \wedge B) - Prob(A \wedge B \wedge C)$. The equation is derived by recognizing that $Prob(A \wedge B \wedge C) + Prob(A \wedge B \wedge \neg C) = Prob(A \wedge B)$.

6.6 Handling Mutual Exclusion

Given a conjunction of events, it is quite simple to handle mutual exclusion. If the conjunction contains two non-negated events that are mutually exclusive, then the probability of this conjunction is equal to 0. For example, a conjunctive query might require that the president's age be equal to both 54 and 55, and results for this query should have a probability of 0 (if these values are mutually exclusive). Detection of mutual exclusion between two nodes may involve checking all ancestors of these two nodes up to the document root (since mutual exclusion can be between entire sub-trees, and not just simple values).

6.7 Computing Result Probabilities for General Queries

Given the procedures outlined previously, we can calculate the probability of any query result, where the query can contain conjunctive and disjunctive constraints, negated conditions, and mutually exclusive data elements.

Theorem 1 (Probabilistic Computation) *Given our model's constraints on conditional probabilities, independence, and mutual exclusion, the above procedures correctly compute the probability for a Boolean formula F of node predicates, given that the allowable Boolean connectives are $\wedge$, $\vee$, and $\neg$. Allowable predicates include (element.tag = value) as well as (element.content θ value), where $\theta \in \{=, \neq, <, \leq, >, \geq\}$.*

6.8 Computing Node Probabilities

As mentioned previously, we update each node's probability in the result to be the probability of that node, assuming that the query matches (the entire tree probability is simply the probability that the query matches). Given that the previous methods for computing the probability of a result tree are in place, computing this updated probability for each node in the result is quite easy. The probability we wish to show is the probability of a node, given that the overall tree matching formula F is true. So, for each node $node_i$ in the result, we need to compute $Prob(node_i|F)$. But $Prob(node_i|F) = \frac{Prob(node_i \wedge F)}{Prob(F)}$ by the definition of conditional probability. We know how to compute the probability of both the numerator and denominator based on the previous sections.

As an example, the pattern tree in Figure 5(d), results in the two matches shown in Figure 6(d) and 6(d') when applied to the sample data. Working with Figure 6(d), we see that the overall tree probability will be equal to[6]:

$Prob(F) =$
$Prob((name = GeorgeBush) \vee (age = 55)) =$
$Prob(name = GeorgeBush) + Prob(age = 55) - Prob((name = GeorgeBush) \wedge (age = 55)) =$
$(0.7 \times 0.5 \times 0.9) + (0.35 \times 0.5 \times 0.9) - (0.7 \times 0.35 \times 0.5 \times 0.9) =$
$0.315 + 0.1575 - 0.11025 = 0.36225$

These single node and conjunctive node computations are the same as those shown in Section 6.2. Now, we show an example of the updating of a single node's probability in the output. The probabilistic computation required to update the `name` node in Figure 6(d)

[6]As a notational convenience, we simply use *name* and *age* in the equations, to refer to two specific nodes in the source data. In actuality, internal node IDs are used, and there is no ambiguity in the matching. Also, we simplify the probabilistic computation by removing the `chiefOfState` node from the formula F (although the result probabilities will be the same).

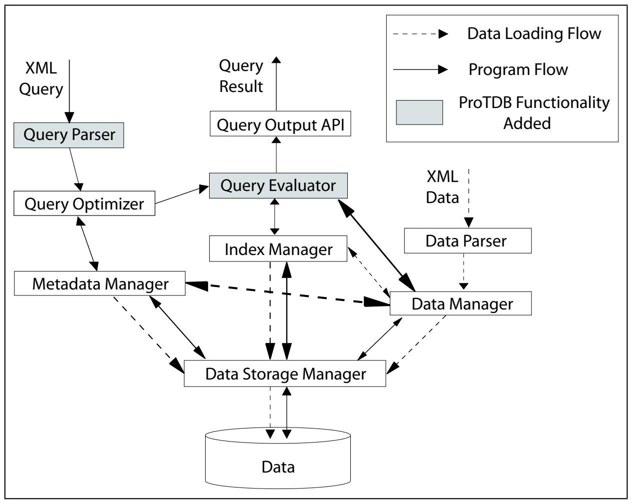

Figure 7: The Basic System Architecture.

(corresponding to `name = ‘‘George Bush’’`) is given as:

$Prob(name = GeorgeBush|F) =$
$Prob((name = GeorgeBush)|(name = GeorgeBush) \vee (age = 55)) =$
$\frac{Prob((name=GeorgeBush)\wedge((name=GeorgeBush)\vee(age=55)))}{Prob((name=GeorgeBush)\vee(age=55))} =$
$\frac{Prob(name=GeorgeBush)}{Prob((name=GeorgeBush)\vee(age=55))} =$
$\frac{0.315}{0.36225} = 0.869565\ldots$

All of the node probabilities in Figure 6 have been calculated as the conditional probability of the node given the query condition.

6.9 Deterministic Default

Even in a probabilistic database, not all data will be probabilistic. Our model permits the free intermixture of both deterministic and probabilistic data. Any element with a missing probability attribute can be assumed to have a default probability value of 1. With this default assumption, we can establish the following theorem (the proof is omitted for lack of space):

Theorem 2 (Consistency) *For every TAX query q, its execution against a deterministic database produces exactly the same results as the execution of q against an equivalent probabilistic database with all probabilities set to 1.*

7 Experimentation

We wish to evaluate our system both in terms of computational overhead (how much additional time is required when comparing our system to a non-probabilistic XML database system), and in terms of the general “usefulness” of our system.

In many ways, “usefulness” is a rather ill-defined metric. “Usefulness” (quality) of the results, and “usefulness” (applicability) of the overall system are both important. Both of these senses of usefulness require a qualitative assessment, in contrast to the purely quantitative assessment of computational costs. But, if the system is not useful, then the fact that it performs quickly is decidedly less exciting. To highlight the efficacy of our approach we contrast basic sample queries, and the corresponding results, from both our probabilistic system and the underlying non-probabilistic system.

7.1 System Design

Figure 7 shows the basic architecture of the ProTDB system. The architecture of this system is exactly the same as that of the Timber [17] native XML database upon which it is based. All modules, other than the two shaded boxes, are identical to the corresponding modules in Timber. The query evaluator is enhanced with the basic probabilistic machinery discussed in Section 6. The parser module is changed so that a normal user query (issued as if it were against non-probabilistic data with an equivalent schema) is transformed to account for the new `Dist` and `Val` elements.

The system is implemented in Microsoft Visual C++ on a Pentium 4 machine running at 1.5 GHZ with 256 MB RAM, and running the Windows 2000 Professional operating system.

7.2 Experimental Design

We extracted geopolitical information from the web using the question answering system to produce an XML file similar to that found in Figure 4, except that our file had over 3,000,000 nodes and occupied over 200 megabytes[7].

7.3 Performance Results

Figure 8 shows query times for queries with a selectivity of .05% (50 “country” subtrees are returned out of the 100,000 country subtrees in the input XML file). We issue conjunctive and disjunctive queries where the root of the pattern tree is a country node, and there are n conjuncts or disjuncts that are descendants of this country node. We show results for n equal to 2, 4, and 6 conjuncts/disjuncts. The overhead incurred by the probabilistic computations is minimal.

Figure 8 shows the query times for queries with a selectivity of 5% (5000 “country” subtrees are returned). We see that the computational overhead for our probabilistic manipulations is reasonable, for a small number of query conditions.

7.4 Quality Assessment

To appreciate the benefits of a probabilistic database system, consider the two results (shown in Figure 9(a)), for the query: “Who is the current head

[7]Some sub-trees in the extracted document were replicated in order to produce a sufficiently large probabilistic document.

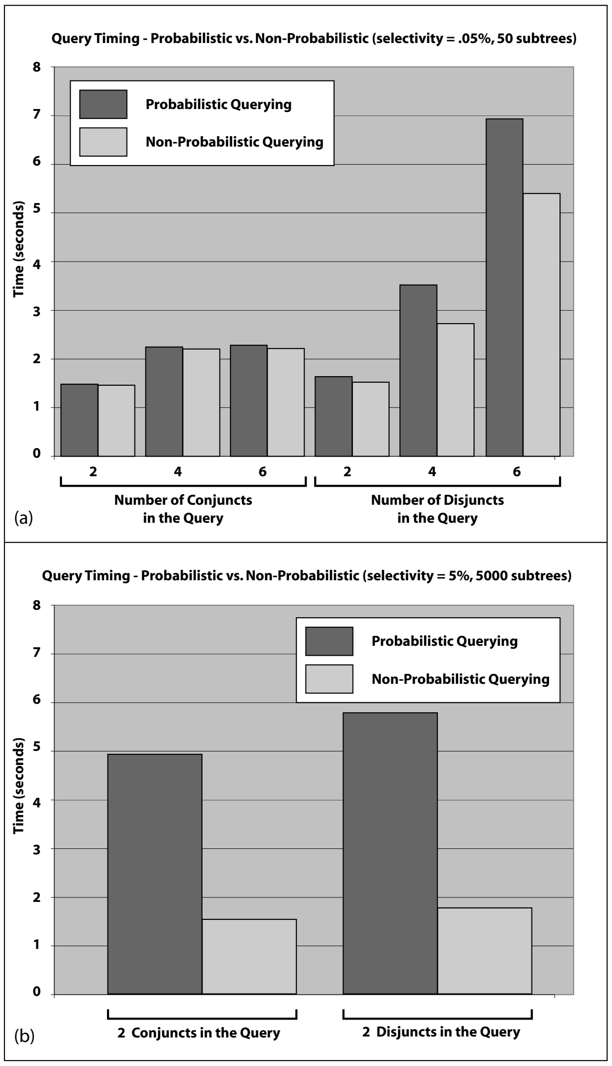

Figure 8: Timing Results

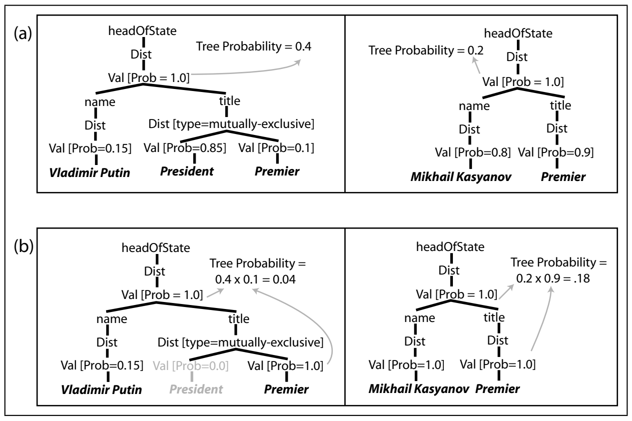

Figure 9: Querying for the head of state of Russia

of state of Russia?" This is a question to which the real answer may not be obvious to the average reader of this paper. (Vladimir Putin is the chief of state but not the head of state). The CIA World Factbook [1] lists each of the following individuals: Mikhail Kasyanov, Aleksey Kudrin, Aleksey Gordeyev, Viktor Khristenko, Ilya Klebanov, and Valentina Matviyenko as heads of state of Russia.

We see in Figure 9(a) that Vladimir Putin actually has a higher probability of matching the "head of state of Russia" query, than Mikhail Kasyanov (an actual head of state) does. Our probabilistic system allows the user to see all the results, ranked by probability, where a deterministic approach would simply display one result, display all of them (unranked), or use some threshold to remove low probability data during the QA phase. But even this final option is not desirable. In Figure 9(b), we see that if the user has some domain knowledge (that a head of state of Russia is likely to have the title of "Premier") then they can get the correct result, with the highest ranking, by incorporating this constraint into the query. If the "Kasyanov" result was pruned based on a threshold, it would have been unavailable for such querying.

8 Conclusion

We have presented an overview of the ProTDB probabilistic XML database system currently under development at the University of Michigan. The probabilistic constructs have been chosen with care to obtain reasonable expressive power while allowing efficient probabilistic computation. The model also allows convenient intermixing of probabilistic and non-probabilistic data.

Initial experiments with probabilistic information extracted from the web are positive in showing both efficiency of operation and the inherent value of these probabilistic manipulations.

Finally, our probabilistic model and querying mechanisms result in correct probability computations (based on the dependencies specified in the model) for any query, or combination of queries. This is important for many applications, and is an area where many previous approaches have failed.

9 Acknowledgments

We would like to thank Jason Atkins for his work on representing probabilistic data in TIMBER. His initial work was the impetus for the model presented in this paper.

Dragomir Radev performed the information extraction from the web and provided us with probabilistic data for input into our system.

Also, we would like to thank Laks Lakshmanan and Lise Getoor for carefully reading drafts and offering thoughtful, detailed comments.

References

[1] `http://www.cia.gov/cia/publications/factbook/`.

[2] Initiative for the evaluation of XML retrieval. `http://qmir.dcs.qmw.ac.uk/INEX/`.

[3] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet Wiener. The Lorel query language for semistructured data. *Journal of Digital Libraries*, November 1996.

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

[5] Roger Cavallo and Michael Pittarelli. The theory of probabilistic databases. In *Proceedings of the 13th International VLDB Conference*, pages 71–81, 1987.

[6] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems*, 21(3):339–369, September 1996.

[7] Debabrata Dey and Sumit Sarkar. PSQL: A query language for probabilistic relational data. *Data & Knowledge Engineering*, 28:107–120, 1998.

[8] Daniel Egnor and Robert Lord. Structured information retrieval using XML. In *Informal Proceedings of the SIGIR Workshop on XML and Information Retrieval*, 2000.

[9] Thomas Eiter, Thomas Lukasiewicz, and Michael Walter. Extension of the relational algebra to probabilistic complex values. In *Proceedings of the International Symposium on Foundations of Information and Knowledge Systems*, pages 94–115, 2000.

[10] Norbert Fuhr and Kai Großjohann. XIRQL: A query language for information retrieval in XML documents. In *Proceedings of the 24th International ACM SIGIR Conference*, New Orleans, Louisiana, September 2001.

[11] Norbert Fuhr and Thomas Rölleke. A probabilistic NF2 relational algebra for integrated information retrieval and database systems. In *Proceedings of the 2nd World Conference on Integrated Design and Process Technology*, pages 17–30, 1996.

[12] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems*, 15(1):32–66, 1997.

[13] Sol Gordon and Harold Brecher. *Life is Uncertain... Eat Dessert First!: Finding the Joy You Deserve.* Dell Books, April 1996.

[14] W3C Working Group. XML path language (XPath). `http://www.w3.org/TR/xpath`.

[15] W3C Working Group. XQuery: An XML query language. `http://www.w3.org/TR/xquery/`.

[16] Yoshihiko Hayashi, Junji Tomita, and Gen'ichiro Kikui. Searching text-rich XML documents with relevance ranking. In *Informal Proceedings of the SIGIR Workshop on XML and Information Retrieval*, 2000.

[17] H. V. Jagadish, Shurug Al-Khalifa, Laks Lakshmanan, Andrew Nierman, Stylianos Paparizos, Jignesh Patel, Divesh Srivastava, and Yuqing Wu. TIMBER: A native XML database. Technical report, University of Michigan, April 2002. `http://www.eecs.umich.edu/db/timber/`.

[18] H. V. Jagadish, Laks Lakshmanan, Divesh Srivastava, and Keith Thompson. TAX: A tree algebra for XML. In *Proceedings of the DBPL Conference*, Marino, Rome, Italy, 2001.

[19] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. ProbView: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

[20] Mounia Lalmas. Dempster-Shafer's theory of evidence applied to structured documents: modelling uncertainty. In *Proceedings of the 20th International ACM SIGIR Conference*, Philadelphia, Pennsylvania, July 1997.

[21] Michael Pittarelli. An algebra for probabilistic databases. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):293–303, 1994.

[22] Dragomir Radev, Kelsey Libner, and Weiguo Fan. Getting answers to natural language queries on the web. *Journal of the American Society for Information Science and Technology*, 2002.

[23] Dragomir R. Radev, Weiguo Fan, Hong Qi, and Amardeep Grewal. Probabilistic question answering from the web. In *Proceedings of the 11th International World Wide Web Conference*, 2002.

[24] Anja Theobald and Gerhard Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *Proceedings of the EDBT Conference*, Prague, Czech Republic, March 2002.

[25] Esteban Zimanyi. Query evaluations in probabilistic relational databases. *Theoretical Computer Science*, 171:179–219, 1997.