Efficient Query Evaluation Over Probabilistic XML With Long-Distance Dependencies

Asma Souihli

Institut Télécom; Télécom ParisTech; CNRS LTCI
Paris, France
first.last@telecom-paristech.fr

Advisor: Pierre Senellart

ABSTRACT

We address the problem of querying probabilistic semistructured databases in view of the tradeoff between the efficiency of evaluation and the ability to model probabilistic dependencies between elements of the tree. We introduce, through a discussion of several challenges, the ProApproX query processor over probabilistic XML as a first step towards building a full-fletched probabilistic semistructured data management system. ProApproX aims at efficient data querying of a comparatively larger subset of the XPath query language than was processed by related systems, through techniques of exact calculations or efficient approximations of the result probability. This paper describes PhD work carried out at Télécom ParisTech under the guidance of Pierre Senellart.

1. INTRODUCTION

Uncertainty comes along with data generated by imprecise automatic tasks such as data extraction and integration, data mining, or natural language processing. A measure of this uncertainty may be induced from the trustworthiness of the resources, the quality of the data mapping procedure, etc. In many of these tasks, information is described in a semistructured manner because representation by means of a hierarchy of nodes is natural, especially when the source (e.g., XML or HTML) is already in this form. One possible way, among the most natural, to represent this uncertainty is through probabilistic databases.

P-documents [3] are probabilistic XML trees with ordinary and distributional nodes. The latter define the process of generating a random XML instance following the specified distribution at the level of each node. The model is then a compact and complete representation of a probabilistic space of documents (i.e., a finite set of possible worlds, each with a particular probability). Such probabilistic space can be queried with classical tree query languages. A Boolean query over a p-document returns a probability, the probability that the query is true. A proper system for managing probabilistic XML information is expected to efficiently calculate the probability of a given query in an exact way whenever possible, or to approximate it (e.g., through Monte Carlo methods). Previous work by Kimelfeld, Kosharovsky, and Sagiv [15] has shown that, in general, computation is intractable under data complexity, and approximation is intractable under query-and-data complexity. Nevertheless, several restrictions make the problem easier to handle and many optimizations can reduce the running time cost.

The objective of this PhD research is to find formal models for the representation and efficient querying of probabilistic databases, and to build corresponding prototype systems. Previously studied probabilistic models, p-documents in particular, are a basis for the work. Particular aspects of interest include:

- management of the various forms of uncertainty;
- efficient evaluation of queries, exact or approximate;
- indexing and physical representation of probabilistic data.

We start by presenting a cursory review of related work and then go into the definition of probabilistic XML. Section 4 introduces, through a number of observations, the idea behind ProApproX, a probabilistic query engine currently under development. We present in more detail the techniques used for query evaluation over probabilistic data in Section 5 and emphasize the difficulties related to their processing, starting from the most trivial ones. In ProApproX, we aim to process a wider range of queries (comparing to competing systems) over a wider range of models. The computation of more sophisticated queries, including negation for example or position predicates, requires some advanced preprocessing. We will be discussing the need for modeling rules behind the querying process in order to relate the semantics of such queries with the existence of uncertain candidates. Section 6 deals with more open issues related to optimization.

2. RELATED WORK

Managing probabilistic and uncertain data is a topic of much current interest among database researchers, and there has been a significant amount of work under the relational database scheme. Extensive literature and many systems and prototypes have been introduced, in particular MayBMS [4, 13], MYSTIQ [6], Trio [18], Orion (previously known as U-DBMS) [21]. In contrast, the semistructured model has received less attention.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EDBT/ICDT PhD 2011, March 25, 2011, Uppsala, Sweden.
Copyright 2011 ACM 978-1-4503-0696-6/11/03 ...$10.00.

The model we focus here is called probabilistic XML (PrXML); an in-depth study of its expressiveness is the work of Abiteboul, Kimelfeld, Sagiv, and Senellart [3], while query evaluation is investigated by Kimelfeld, Kosharovsky, and Sagiv in [15]. Subsequent research has extended the model with continuous probabilistic distributions [2], constraints [8], or possible trees of unbounded size [5]. Yet, at the practical level, a full-fletched probabilistic semistructured data management system is still missing.

Hollander and van Keulen [11] investigate the maturity of probabilistic relational databases for querying probabilistic semistructured data by mapping XML to relational data. The querying can then be managed using a system such as MayBMS, for which efficient query evaluation algorithms have been developed, both exact and approximate. The downside of the approach is that relational databases do not exploit the specific characteristics of tree-like data encoded into databases, that for instance makes tree-pattern queries over local models tractable [15].

Another direction is to natively process tree-pattern queries over XML data. This is what is done in the EvalDP algorithm proposed by Kimelfeld, Kosharovsky, and Sagiv [16, 15]. This linear-time bottom-up processing of the tree has first-rate performance, but is hard to generalize once we want to deal with less restricted queries and more expressive probabilistic models. Besides, the authors do not look at the actual building of a PDBMS, with all system issues (indexing, data storage, etc.) that this involves.

3. PROBABILISTIC XML

XML data is modeled as unranked, labeled, unordered trees. Formally, a probabilistic XML document $\tilde{\mathscr{P}}$ is a tree that consists of two types of nodes: *ordinary nodes* and *distributional nodes*. Distributional nodes are fictive nodes that specify how their children can be randomly selected. This general framework, studied in [3], is designed as a generalization of previously proposed models for PrXML. Given the criteria of dependency between elements, we distinguish two types of data representation models. In the *local dependency* model, choices made for different nodes are independent or locally dependent, i.e., a distributional node chooses one or more children independently from other choices made at other levels of the tree. In the *long-distance dependency* model, which ProApproX uses to process a query, the generation at a given distributional node might also depend on different conditions (i.e., probabilistic choices) related to other parts of the tree. This model, more general than the local dependency model, is based on one distributional node, *cie* (for *conjunction of independent events*): nodes of this type are associated with a conjunction of independent (possibly negated) random Boolean variables $e_1 \dots e_m$ called *events*; each event has a global and independent probability $p(e_i)$ that e_i is true. Note that different *cie* nodes share common events, i.e., choices can be correlated. See Figure 1 (left) for an example of such a p-document. We investigate the idea of building a generalized system that can handle long-distance dependencies between different elements of the XML database as well as local dependencies. As noted in [3], local dependencies (*ind* and *mux* distributional nodes) can be tractably translated into *cie* nodes. We appeal to these theoretical studies to perform efficient translations when implementing the ProApproX query engine.

4. THE PROAPPROX SYSTEM

We describe here briefly the system we build in the context of this PhD research, ProApproX, *a system built on top of an XML native DBMS that uses the computational techniques of a probabilistic relational DBMS.*

Most previous works put a number of restrictions on the types of queries that can be processed and on the kind of data itself to make the problem easier to tackle. Though it made identification of an important number of tractable cases possible, it is hard to see how to generalize or extend these algorithmic models in order to handle more sophisticated cases. In particular, the EvalDP algorithm [15] succeeds in computing efficiently the exact probability of a twig (tree-pattern) query with projection only over p-documents with local dependencies. The algorithm evaluates the query directly on the tree with a bottom-up traversal. This approach is restricted to a limited range of queries: projection queries without joins, negation, position predicates etc. This is mainly because in the probabilistic context, the semantics of a query with more elaborate conditions may require special interpretation to define the evaluation process. For instance, when we look for a node B that does not have a child C, the query processor has to understand also that a node B having an uncertain C is a candidate answer for this query, with the probability of not having C. Moreover, the algorithm is sensitive to the length of the query as it needs to evaluate all its subsets to reach the final exact answer. This means that proper indexing and optimizations are essential for the scalability of the algorithm.

Starting from these observations, we study the possibility of designing a solution built on top of an existing XML query engine to benefit from its performance. We will surely need to modify the initial query in order to capture the probability lineage related to each answer (match), as the native DBMS sees the database as an ordinary deterministic one.

The authors of EvalDP emphasize that there is a clear and natural tradeoff between the efficiency of evaluation and the ability to model probabilistic dependencies between the elements of the tree. In this PhD work, we want to support a wider range of query language over probabilistic XML databases and choose to deal with the most general model for expressing dependencies. In front of the tradeoff, we choose expressiveness, but from a practical point of view, we struggle to produce good querying performances by making use of a number of well-defined heuristics and approximation algorithms when the exact computation happens to be hard. Many other important issues are taken into consideration. A complementary indexing on top of the related to the native XML DBMS might be of benefit when querying this special structure.

5. QUERYING P-DOCUMENTS

A Boolean query over a probabilistic document returns the probability that the query is true, or, in other words, the sum of probabilities of all possible worlds in which the query is true. One fundamental observation, only valid in positive, or *locally monotone* [19], query languages, is that the probability of a query over a p-document $\tilde{\mathscr{P}}$ is the probability that one of the matches of the query over the underlying deterministic document is present in a possible world of $\tilde{\mathscr{P}}$. However, since matches are not independent, the probability of a query is not the sum of the probabilities of all matches. We first

explain how we deal with such positive queries, and then how to handle negation.

Over a p-document, we run positive XPath queries (without position predicates or aggregation, up to now) over a p-document with long-distance dependencies, extract the conjunction of the *prob* attribute of the nodes appearing in a subtree matched by the query, and return the set of all possible answers (i.e., the set of all possible matches). We can then see a match as a clause defined by a conjunction of literals. For the algorithm computing the probability of this query, the input is obviously a disjunction of the matches, or in other words, the disjunction of conjunctive clauses. The probability $\Pr(F)$ of a formula F in disjunctive normal form (DNF) denotes the probability that a random, independent, truth assignment to the variables satisfies F (this truth assignment is chosen following the independent probability of the event variables). If there are n Boolean variables, then there are 2^n possible assignments. The problem of counting the exact number of solutions to a DNF is known to be $\#P$-hard which implies that there is no polynomial time algorithm for the exact solution if $P \neq NP$. Yet, we want to show in this paper that many "easy" cases can be detected, and that even if a formula is a hard one, many approximation algorithms, guided by a number of heuristics, can be deployed to build an efficient query processor.

Figure 1 presents a simplified fragment of a probabilistic XML document describing a given Wikipedia article as a merge of all its (uncertain) revisions. The example is reproduced (with elaboration) from [1]. The events coming with the revisions are interpreted as a value of trust or a reputation of the contributor or the probability this particular revision is correct conditioned by the fact that the author is reliable. Clearly, probabilities in this database are related to the quality of information. The distribution of the $\Pr(e_i)$'s could be user-defined or inferred with a trust inference algorithm.

From The Initial Query To The Lineage Query.

Consider the tree pattern query (TPQ)

Q_1 : `/article[title='Roger Waters']//contributor`

over this tree that looks for the list of contributors to the revisions of the article titled "Roger Waters". The corresponding probabilistic lineage for every possible path matching this query is the following two conjunctions of independent events:
$c_1 :< e_1 >$
$c_2 :< e_2 \wedge e_3 >$

To retrieve the lineage, we need to process a different query Q_1' that goes trough a tree pattern matching the query Q_1 and returns the concatenation of all the *prob* attributes met at each level. In ProApproX, we implemented a collection of translators such that a given query is dispatched to its adequate translator. The lineage query for Q_1 is then expressed with the XQuery language and has the following shape:

```
for $a in article
for $b in $a/title/text() [.='Roger Waters']
for $c in $a//contributor
let $leaves:=($b,$c)
let $atts:=(for $i in $leaves return $i/ancestor-or-self::*/
   attribute(prob))
return <match> {
   distinct-values(for $att in $atts return string($att)) } </match>
```

We explain later on why it is difficult to design a unique and generalized translator for arbitrary XPath queries.

Simple And Tree-Pattern Queries.

The probability related to the previous tree pattern query Q_1 is then simply the disjunction of c_1 and c_2. Note here that the two clauses are independent as there is no correlation between their probabilities (realizations), thus, the query is tractable and presents a *PTIME* plan for its computation. We start from the following basic *PTIME* plans for computing the probability of a formula:

- if ϕ and ψ are independent, then:
  $\Pr(\phi \wedge \psi) = \Pr(\phi) \cdot \Pr(\psi)$
  $\Pr(\phi \vee \psi) = 1 - (1 - \Pr(\phi)) \cdot (1 - \Pr(\psi))$
- if ϕ and ψ are inconsistent (mutually exclusive), then:
  $\Pr(\phi \vee \psi) = \Pr(\phi) + \Pr(\psi)$.

Now, consider a formula $F = \phi \vee \psi \vee \varphi$ where $\phi = x \wedge y$, $\psi = x \wedge z$, and $\varphi = t$. The first and the second clauses are dependent by sharing a common variable or event x in the conjunction of their conditions. Despite that non of the three *PTIME* basic computation plans is possible, we still can apply the inclusion-exclusion principle[1], and compute the exact probability of the answer because the number of clauses is small. Now, we want to simplify the computation process as much as possible: we can divide the three clauses into two sets $S_1 = \{\phi, \psi\}$ and $S_2 = \{\varphi\}$, develop the *sieve* (*inclusion-exclusion*) principle on ϕ and ψ to have the probability of S_1 then apply a *PTIME* plan for the equivalent formula $F = S_1 \vee S_2$:

Given:
$\Pr(\phi \wedge \psi) = \Pr(x \wedge y \wedge z) = \Pr(x) \cdot \Pr(y) \cdot \Pr(z)$,
$\Pr(\phi) = \Pr(x) \cdot \Pr(y), \Pr(\psi) = \Pr(y) \cdot \Pr(z)$,
we have:
$\Pr(S_1) = \Pr(\phi) + \Pr(\psi) - \Pr(\phi \cap \psi)$
$F = \Pr(S_1 \vee S_2) = 1 - ((1 - \Pr(S_1)) \cdot (1 - \Pr(S_2)))$

Consider a second simple-pattern query (SPT), i.e., a tree-pattern query without branching, over the tree of Figure 1.

Q_2 : `./article//p/text()`

looks for a text node sibling of a paragraph node. The corresponding probabilistic formula lineage will be:

$$F = (e_1) \vee (e_2 \wedge e_3) \vee (e_2 \wedge e_1 \wedge \neg e_4)$$

F is the prototypical $\#P$-hard query as the events of the clauses overlap (c_1 dependent of c_3 and c_2 dependent of c_3). Here, computing the probability of F by applying the inclusion-exclusion principle is still possible since the cardinality of F in terms of clauses is relatively small. When the number of clauses is much larger and a *PTIME* plan is not possible, a specific heuristic will call a polynomial-time function *redo(F)* that restructures the formula into a disjunction of sets following the pseudo-algorithm:

redo(DNF Φ of size n) returns a set S

[1]For dependent probabilistic events $S_1 \dots S_n$ the inclusion-exclusion principle becomes:

$$\Pr(\bigcup_{i=1}^{n} S_i) = \sum_{k=1}^{n} (-1)^{k-1} \sum_{\substack{I \subset \{1,\dots,n\} \\ |I|=k}} \Pr(S_I), \text{ where } S_I := \bigcap_{i \in I} S_i.$$

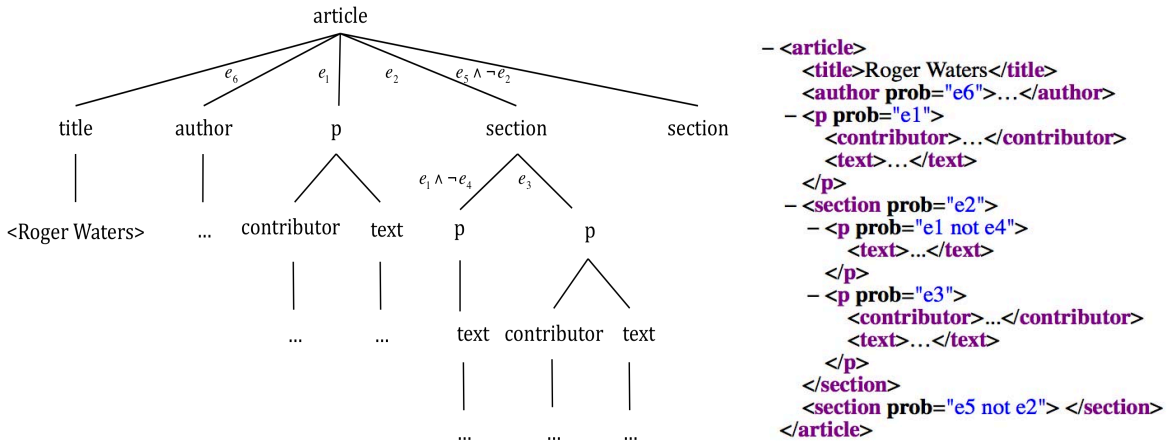

Figure 1: A tree representation of a probabilistic document and its XML encoding as stored for ProApproX

1. build a first subset S_1 of k clauses $\in \Phi$ such that:
   $\forall C, C' \in S_1 : C$ and C' are independent
   and
   $\forall C \in S_1, \forall C' \in \Phi \setminus S_1 : C$ and C' are independent.

2. if $|S_1| < |\Phi|$ then minimally partition the rest of clauses of $\Phi \setminus S_1$ into m subsets $(2 \leq m \leq (n-1)/2)$ such that ~~$\forall S_i, S_j$: S_i, S_j are independent.~~
   $S \leftarrow \{S_1, ..., S_m\}$

We can apply an adequate strategy having the following outline:

1. For the first set of independent event, apply the appropriate *PTIME* plan to compute the exact $\Pr(S_1)$.

2. For each remaining subsets $S_2 ... S_n$, if $|S_i|$ is lower then a given threshold k (say, 5 or 6), apply the sieve principle which can take time exponential in k. Else, run an absolute or relative ϵ-approximation. Here, we have a choice between additive or multiplicative approximation techniques [15]. In ProApproX, we implemented a multiplicative approximation algorithm, with a number of optimizations, that leads to very good accuracy, which could be thought as a good choice for an efficient convergence in case S_i has a very small probability. Nevertheless, the convergence guarantee [15] that is obtained from Hoeffding's inequality [10] requires a running time growing in $O(n^3 \ln n)$ in the number of clauses. In the case of a high probability, the additive Monte Carlo approximation would be the best choice for estimating the current S_i as it converges very rapidly for big values. To summarize the reasoning, we can set an example heuristic:

   - Compute the probability of the subset S_i following a *PTIME* plan if its clauses are independent
   - If $\Pr(S_i)$ is relatively high (say $\geq 10^{-3}$), and $|S_i| > 15$: run an additive approximation
   - If $\Pr(S_i)$ is relatively high (say $\geq 10^{-3}$), and $|S_i| \leq 15$: run a multiplicative approximation
   - If $\Pr(S_i)$ is relatively low (say $< 10^{-3}$), and $|S_i| \leq 15$: run a multiplicative approximation.
   - If $\Pr(S_i)$ is relatively low (say $< 10^{-3}$), and $|S_i| > 15$, which means that we are in the very hard case, where a large number of elements is highly correlated (which is unlikely to happen frequently): run an additive approximation or apply a *heuristic for the worst case* (see further) for the set S_i.

   Note that we do not have access to $\Pr(S_i)$ itself, since this is the value we want to compute. But we can get rough estimates of it, e.g., by using standard probabilistic inequalities or by sampling.

3. Obtaining the exact probability of S_1 and the exact or approximated value of each remaining subset, we simply apply a *PTIME* plan for a DNF of independent clauses and get the final probability.

The worst case is when most of the matches are highly correlated, i.e., the smallest subset that we can build has a cardinality that is very close to the initial (big) set of answers; a scenario that is very infrequent in practice. Even in such a case, one can relax the latter strategy and find a simple approximation (*heuristic for the worst case*). We exemplify with an inspiring technique deployed in the SPROUT query engine of MayBMS [12] for running upper- and lower-bound approximations for a hard query: We may choose to build the subsets such as every S_i minimally contains a number of independent clauses, even if we obtain a number of subsets that are dependent. The computing of every $\Pr(S_i)$ is then polynomially possible, the lower bound can be chosen as $\max(\Pr(S_i))$ and the upper-bound as $\min(1, \sum_{i=1}^{n} \Pr(S_i))$.

Throughout the computation, we maintain all inequalities and probabilistic guarantees, in order to present to the user in the end the computed probability and its confidence interval.

Tree-Pattern Queries With Join. A user-query that involves a join condition between two nodes needs an additional level in the semantics of the translator. Nevertheless, we can just extend the translator to handle the join condition and to return the correct query that gives the correct lineage. An example query that present a join predicate over the

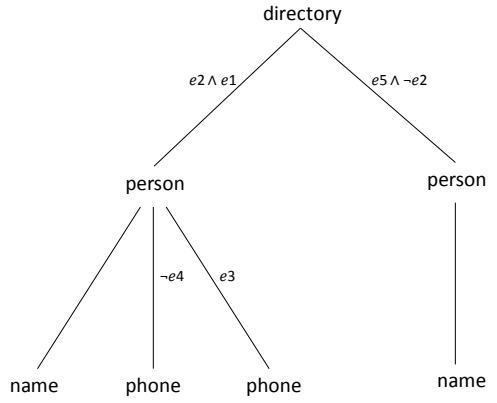

Figure 2: A PrXML tree with the *probability of occurrence* semantics

p-document of Figure 1 is

$$Q_3 : \texttt{./article[author=.//contributor]}.$$

The query looks for articles where authors are contributors in the revisions as well. Since the query language is still positive, the lineage of such a query is still a DNF, as a given answer would simply cover a subtree, similarly as for TPQs.

Negated Queries. Negated queries require a different translator. As illustrated previously, the translated query must return a lineage that takes into consideration probabilistic, and thus uncertain, nodes.

The process of data integration in some contexts (e.g., integrating a number of directories [22]) may need to express the uncertainty about finding an occurrence as a result of a statistical analysis over the input set of data to be merged. Figure 2 is a snippet from a probabilistic semistructured database that stores information about people and probabilities of finding this information. A negated query Q_4: `//person[not(phone)]` looks for person instances without phone numbers. The expected translator should return a lineage query that looks for the two possible patterns in this example and return the sequences:
$< e_2 \wedge e_1 \wedge \neg e_4 \wedge e_3 >$
$< e_5 \wedge \neg e_2 >$
but the correct probability should follow a disjunction of a slightly different clauses. In fact, the probability that the first person instance does not include a phone node is the realization of the event $e_2 \wedge e_1$ and that none of the siblings phones appears, i.e.: $\Pr(Q_4) = (e_2 \wedge e_1 \wedge e_4 \wedge \neg e_3) \vee (e_5 \wedge \neg e_2)$.

In the general case when the conditions to negate are themselves conjunctions, the obtained lineage is not in DNF any more. To apply the previously presented evaluation strategy, we can convert the resulting lineage to DNF, incurring possibly an exponential running time cost in the process. The design of a fully working translator for negated queries and the study of the complexity of the evaluation task is still work in progress.

6. FURTHER ISSUES

We now discuss some further issues that are relevant to this PhD research but are still partly or fully open.

Refining The Approximations. When computing the probability of a DNF is hard, we run an absolute or relative approximation to estimate the correct answer, given a fixed error factor ϵ ($0 \leq \epsilon < 1$) and a probabilistic guarantee δ. In ProApproX, we give the possibility to personalize the number of samples needed for additive and multiplicative approximations, in one of the following three ways [20]:

- by calculating, using Hoeffding's inequality, a suitable number of trials given a tolerated error under a probabilistic guarantee δ;
- by empirically stopping the sampling once the estimated probability has not deviated more than a given value over a given number of consecutive trials;
- by giving a fixed number of trials.

When a user let the system run the adequate strategy for computation, a number of heuristics are implemented to lead to the suitable algorithm in view of the hardness of the query. In this paragraph, we discuss the case when the approximation of a DNF lineage formula (or a DNF subset formula of F) is relatively hard. We are considering the case where the number of satisfying assignments to F is exponentially small with respect to the total (huge) number of possible assignments. From an approximation point of view, we would need exponentially many samples form the set of possible assignments to get a given number of successes. Instead of picking assignments uniformly at random and testing each clause, we could sample from the set of assignments that satisfy at least one clause (but not uniformly). Many strategies for *sampling only the important space* were introduced from which we cite the algorithms of Karp, Luby, and Madras in [14]. Also, when the number of literals per clause is constant, Ajtai and Wigderson provide an $(\epsilon, \delta = 0)$ approximation algorithm which runs in time polynomial in $(1/\delta)$ times the length of the formula.

Indexing. In the case of probabilistic databases, an appropriate indexing could be of major benefit, as we go through a translated and more complex version of the initial query. In PEPX [17], Li, Shao, and Chen introduce a physical model for PrXML databases that stores the descendent probabilistic lineage at each node instead of just storing the probability related to a node at a given level of the tree. The authors prove through a number of experiments that the design does not affect the expressiveness and can improve query evaluation performance. This is the kind of data storage strategies that we need to set up to improve the performance of our translated lineage queries.

7. DISCUSSION AND CONCLUSIONS

Throughout the different sections, we gathered and discussed a number of strategies to resolve issues related to managing probabilistic semistructured databases, especially with respect to query processing. We started by raising the question of how to model probabilistic dependencies and achieve efficiency of query evaluation. Our choice was based on electing the most expressive model that joins both local and global dependencies. At the same time, we aim for a querying solution that, at least, has "some" performance that will not be affected by the expressiveness level of the model.

We presented the broad outlines of an algorithm that processes each query following its "complexity category". From a "complexity category" to another, the performance of the algorithm naturally varies. As ProApproX is built on top of a native XML query engine, the first step of the processing (Section 5) features efficient, index-based, XQuery processing. Afterwards, the probability computing is delegated, by a heuristic plan, to the most likely best strategy.

Currently, we are experimenting an early ProApproX prototype to compare its performance with EvalDP. We use the same dataset and queries as in [15]. First results show that we are able to perform faster evaluation with acceptable precision guarantees, for most of these queries. We also noticed that for some query, the implementation of the EvalDP algorithm leads to a wrong value, which may be attributed to the relatively high intricacy of implementation of this dynamic programming algorithm. We stress again that our system allows us to process a wider range of queries (especially, join queries), over a wider range of models (long-distance dependencies). A second heuristic plan and a number of optimizations and "special" indexing features are essential for the scalability of ProApproX. Nevertheless, further experiments are under progress to bring a performance appraisal.

We also want to deal with aggregate queries. The result of a query that make uses of aggregate functions is a set of possible values (for each possible document), each with its probability. A theoretical study of such aggregate queries is given in [2], which also introduce continuous distributions (e.g., data provided from sensors [7, 9]).

8. REFERENCES

[1] T. Abdessalem, M. L. Ba, and P. Senellart. A probabilistic XML merging tool. In *Proc. EDBT*, 2011. Demonstration.

[2] S. Abiteboul, T.-H. H. Chan, E. Kharlamov, W. Nutt, and P. Senellart. Aggregate queries for discrete and continuous probabilistic XML. In *Proc. ICDT*, 2010.

[3] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *VLDB J.*, 18(5):1041–1064, 2009.

[4] L. Antova, C. Koch, and D. Olteanu. Query language support for incomplete information in the MayBMS system. In *Proc. VLDB*, 2007.

[5] M. Benedikt, E. Kharlamov, D. Olteanu, and P. Senellart. Probabilistic XML via Markov chains. *Proc. VLDB Endowment*, 3(1):770–781, 2010.

[6] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D. Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *Proc. SIGMOD*, 2005.

[7] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proc. SIGMOD*, 2003.

[8] S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilistic XML. *ACM Trans. Database Syst.*, 34(3), 2009.

[9] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. VLDB*, 2004.

[10] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. American Statistical Association*, 58(301):13–30, 1963.

[11] E. Hollander and M. van Keulen. Storing and querying probabilistic XML using a probabilistic relational DBMS. In *Proc. MUD*, 2010.

[12] J. Huang. Design and implementation of the SPROUT query engine for probabilistic databases. Master's thesis, University of Oxford, 2009.

[13] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *Proc. SIGMOD*, 2009.

[14] R. M. Karp, M. Luby, and N. Madras. Monte-Carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

[15] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic XML. *VLDB J.*, 18(5):1117–1140, 2009.

[16] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *Proc. VLDB*, 2007.

[17] T. Li, Q. Shao, and Y. Chen. PEPX: a query-friendly probabilistic XML database. In *Proc. CIKM*, 2006.

[18] M. Mutsuzaki, M. Theobald, A. de Keijzer, J. Widom, P. Agrawal, O. Benjelloun, A. D. Sarma, R. Murthy, and T. Sugihara. Trio-One: Layering uncertainty and lineage on a conventional dbms. In *Proc. CIDR*, 2007. Demonstration.

[19] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic XML data. In *Proc. PODS*, 2007.

[20] P. Senellart and A. Souihli. Un système de gestion de données XML probabilistes. In *Proc. BDA*, 2010. Conference without formal proceedings (demonstration).

[21] S. Singh, C. Mayfield, S. Mittal, S. Prabhakar, S. E. Hambrusch, and R. Shah. Orion 2.0: native support for uncertain data. In *Proc. SIGMOD*, 2008.

[22] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *Proc. ICDE*, 2005.