Un Système de gestion de données XML probabilistes*

Pierre Senellart · Asma Souihli
Institut Télécom; Télécom ParisTech
CNRS LTCI, 46 rue Barrault 75634 Paris, France
[first.last]@telecom-paristech.fr

Résumé

Cette proposition de démonstration porte sur un système de gestion de données probabilistes semi-structurées. Le système présenté repose sur une généralisation des modèles de représentation de données incertaines en XML proposés dans la littérature et permet une interrogation efficace des données dans un sous-ensemble du langage de requêtes XPath, moyennant des techniques de calculs exacts ou d'approximations du résultat.

Mots-clefs : XML, données probabilistes, approximations.

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 semi-structured model, 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. In this paper, we describe an implementation of a system dealing with models of probabilistic XML expressing uncertainties about the stored information.

P-documents [2] are 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. The main idea behind the system is to efficiently calculate the probability of a given query in an exact way whenever possible, or to approximate it using Monte Carlo methods. Previous work [4] 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 tackle and many optimizations reduce the running time cost. Thus, if the data has only *local dependencies* and if the query is a tree-pattern without value joins, the EvalDP algorithm from [4] yields a polynomial-time computation of query probability.

In the system demonstrated in this paper, we aim at covering a wider range of data model and queries and we explain how to compute and approximate the probability of a query using a simplified but efficient process: rewriting the initial query into one that returns every match on the underlying deterministic document as a sequence of events (labeled probabilities related to the probabilistic nodes of the p-document). These sequences are then deployed to perform exact computation or approximations. Though there has been a number of theoretical studies of probabilistic XML [2, 4], this work is the first towards a full-fletched database management system for querying probabilistic XML data. We start by introducing an example probabilistic tree and then go into the definition of probabilistic XML and probabilistic queries. In a third section we explain the basics of the proposed system detailing the different evaluation methods, their advantages and their drawbacks. We establish, through empirical experiments, an efficient policy about delegating the computation task between different algorithms. We describe the demonstration scenario in Section 5.

*This research was funded by the Dataring project of the French ANR.

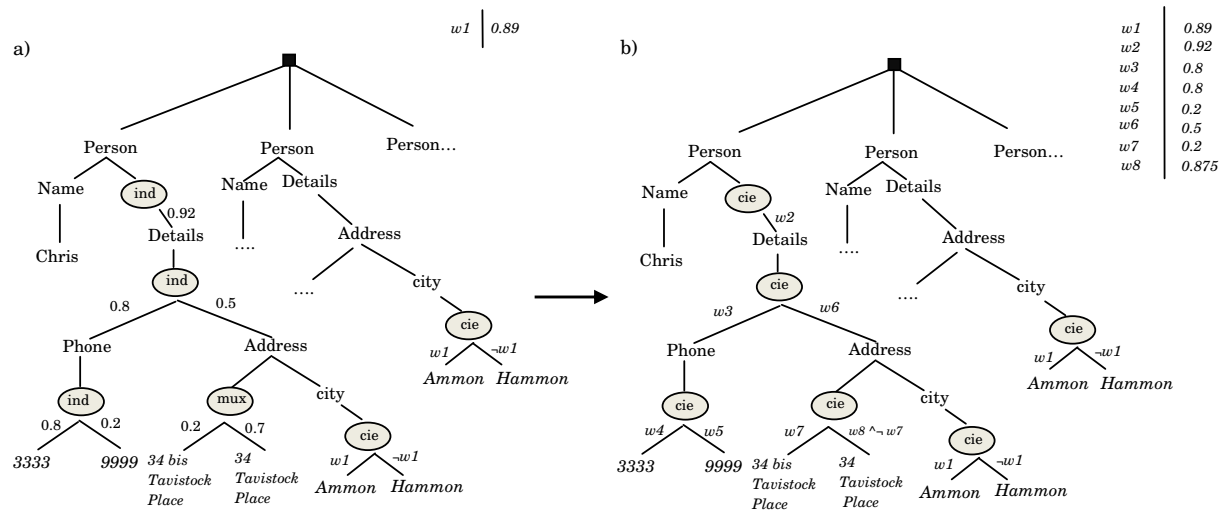

Figure 1: Translation from PrXML$^{\{ind,mux,cie\}}$ to PrXML$^{\{cie\}}$

2 Example of Application

resulting from a number of data integration operations. We refer to the purpose of van Keulen, de Keijzer, and Alink in [5] about integrating data shared in open environments using a probabilistic XML scheme. The example of Figure 1a shows a portion of a document resulting from integrating information about persons gathered from several directories. Information about the person "Chris" may be hidden in some directories or public in some others (older or newer), which is modeled by the probability 0.92 to find details about that person. In these directories, "Chris" may be attached his first phone number (with probability 0.8), his second (with probability 0.2, independently), or both (with the probability of the independent disjunction). Yet, there are directories that assign him exclusively a first variant of his address (with probability 0.2), while others mention a second variant (with probability 0.7, mutually exclusive of the first case), or do not present any details about the address (with probability 0.1). Thus, the *mux* node expresses that children are selected in a mutually exclusive way, while *ind* nodes specify that they are chosen independently of each other.

Information about Chris's city is depicted following a general rule in the database indicating that 89% of the available directories spell the name of this same city with a "H". A given directory (a possible world of the probabilistic document) only uses one of these spellings. Such long-distance dependencies are expressed in the document with the help of *cie* nodes: "Hammon" is used if the event w_1 is realized, whereas "Ammon" is used if $\neg w_1$ is realized. The probability of the latter is the complement of the probability of w_1, i.e., 11%. We present in more detail in the next section the different types of distributional nodes in a probabilistic XML document.

3 Probabilistic Data and Queries

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 [2] is designed as a generalization of previously proposed models for PXML. 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. The model may hold three types of distributional nodes, each with a different way of describing its local probability distribution: (i) *ind*: children of the node are chosen independently of one

another, according to their probabilities; (ii) *mux*: children of the node are chosen in a mutually exclusive way, depending of their probabilities, that must sum up to one or less; (iii) *exp*: the distribution of all possible choices of children is explicitly given, i.e., each subset of the set of the children is associated with a probability. Use of *exp* nodes has less practical interest. To simplify, we shall ignore them in the following, but extension of the proposed algorithms to support *exp* nodes is relatively straightforward.

In the *long-distance dependency* model, 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 is based on one distributional node, *cie* (conjunction of independent events): nodes of this type are associated with a conjunction of independent random Boolean variables $w_1 ... w_m$ called events; each event has a probability $p(w_i)$ that w_i is true. Note that different conjunctions can share common events. Consequently, *cie* nodes can correlate their choices.

Note that a given p-document $\tilde{\mathscr{P}}$ can use any combination of distributional nodes, and thus present local dependencies, long-distance dependencies, or both. Given a p-document $\tilde{\mathscr{P}}$, we denote by $\mathscr{P}$ the tree random variable obtained by following the random process for selecting children of each distributional node of the tree and keeping only the resulting ordinary nodes. We also define the *underlying deterministic document* of a p-document as the one obtained by removing all distributional nodes (linking ordinary nodes to their closest ordinary ancestors).

Querying P-documents. For queries, we consider *tree-pattern* queries. For instance, given the p-document of Figure 1a, we can look for the addresses of Chris with the tree-pattern query `//Person[Name='Chris']//Address` in XPath notation. We also allow value joins such as in the query `//Person[Name=.//city]` to get all persons whose name is the same as that of their city. The semantics of such queries is standard over deterministic documents. We define the match of a query as the minimal subtree containing all nodes of a document that are mapped to a node of the query.

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, due to the positive nature of the query language, is that the probability of a query over a p-document $\tilde{\mathscr{P}}$ is the probability that one of the match of the query over the underlying deterministic document remains in $\mathscr{P}$. However, since the matches are not independent, the probability of a query is not the sum of the probabilities of all matches.

4 A Probabilistic XML System

Our system evaluates queries over a given probabilistic XML document either with an exact calculation when some form of independence between query matches is detected, or by running the approximation algorithms elsewhere. We compare the approach to the exact EvalDP algorithm [4] (whose implementation has been kindly provided by the authors) that is very efficient but does not support either long-distance dependencies or value joins. EvalDP computes efficiently the probability of a query by means of dynamic programming, and achieves high accuracy of results. Approximation techniques are an effective way to handle long-distance dependencies and value joins, if they can be made to run fast. This is exactly the purpose of our system, dealing effectively with local or long-distance dependencies and tree-queries with joins.

As noted in [2], *ind* and *mux* nodes can be tractably translated into *cie* nodes. We appeal to these theoretical studies to perform efficient translations, as encoding matches with conjunctions of all events involved simplify the computation and approximation processes.

Encoding the Matches. Consider the query `//Person[Name='Chris']//Address/text()` over the example of Figure 1. The right side of the figure shows the translation of the tree to a document with only *cie* nodes. We can actually discard the *cie* nodes and integrate each event w_i as an attribute attached to its corresponding XML node. Obviously, we can rewrite a given query in order to get the concatenation of all events appearing along a match, e.g., the address query could be rewritten to return all possible matches: $\langle w_2, w_6, w_7 \rangle$, $\langle w_2, w_6, w_8, \neg w_7 \rangle$. We run approximations directly over these event conjunctions. To run the

rewritten query and get the different matches, the system features efficient, index-based, XQuery processing using Saxon as a Java class library.

Additive Approximation. The simplest way to implement an additive Monte Carlo approximation technique, as mentioned in [4], is to generate random instances from the p-document, evaluate the query over them, and get the approximated probability as the proportion of samples that make the query true. We implemented this technique but performances were very low due to the cost of generating a sample of a potentially very large database, and the resulting I/O operations. *Additive approximation* is reconsidered by drawing random values for the w_i's belonging to the path of each mapping, in this way getting rid of the instance generation. This gives much more acceptable running time. However, because of the very nature of additive approximation, the convergence is slow for low values of probabilities.

Multiplicative Approximation. The probability that a query Q is true in a random instance $\mathscr{P}$ can be described by: $\Pr(\mathscr{P} \models Q) = \sum_{i=1}^{n} \Pr(m_i \lhd \mathscr{P}) \times \Pr\left(\bigwedge_{j=1}^{i-1} \neg(m_j \lhd \mathscr{P}) \mid m_i \lhd \mathscr{P}\right)$ where $\Pr(m_i \lhd \mathscr{P})$ is the probability that a match m_i of Q remains in $\mathscr{P}$. The first term is easy to compute by just gathering the probabilities of all events involved in the match; the second term can be approximated by conducting biased draws to considerate the probability that none of the preceding matches exists in the random document $\mathscr{P}$, given that a current match m_i appears in that same document. The evaluation leads to a very good accuracy. Nevertheless, the convergence guarantee [4] that is obtained from Hoefdding's inequality requires a running time growing in $O(n^3 \ln n)$ in the number of matches to the query (which is, itself, potentially exponential in the size of the query). However, the very case where there are a high number of matches are, empirically, mostly those for which the probability of the query is high, and consequently where additive approximation gives good results. Therefore, our system yields processing of queries with high number of matches to the additive approximation algorithm and with low number of matches to the multiplicative approximation.

Exact Computation. Of course, a query that only involves independent matches does not require a complex process; this case often appears in practice. It suffices to note the independence between patterns to the query and to compute the result based on the principle of inclusion-exclusion. Exploring the set S of patterns to the query, we are able to detect *independence up to intersection* by simply verifying the following property (given that n is the size of S, and m_k is a match in S): *patterns to the query are independent up to intersection iff* $\bigcap_{i=1}^{n} m_i = m_u \cap m_v$ *for all* $1 \leq u < v \leq n$. It suffices then to compute the global result based on the principle of inclusion-exclusion, e.g., for two independent mappings m_1 and m_2 to a given query Q and a random document $\mathscr{P}$: $\Pr(\mathscr{P} \models Q) = 1 - (\Pr(\neg m_1 \wedge \neg m_2)) = 1 - (\Pr(\neg m_1) \times Pr(\neg m_2)) = 1 - (1 - Pr(m_1)) \times (1 - Pr(m_2))$.

Optimizations. Several optimizations are implemented, applied mainly to the *multiplicative approximation*. A first preprocessing removes mappings that are always true or false, the second goes along with the main treatment and decides to stop the computation and returns the result when the contribution of remaining mappings is considered to be very low (given a sorted input of mappings according to their probability in the tree).

In order to compare the performance of our system with EvalDP, we reconsider the same dataset and queries as in [4]. First results show that we perform a much faster evaluation for most of them. We implemented an extension of an XPath parser that generate the corresponding query which yields the sequence of events for every matching pattern. We also noticed that for some query, the implementation of the EvalDP algorithm leads to a wrong value. Finally, we stress that our system allows us to process a wider range of queries, over a wider range of models.

5 Demonstration Scenario

We describe here the structure of the user interface for interacting with the system, as we intend to demonstrate it.

Input: The user can select a p-document from a default collection (including the dataset from [4] and an extension of the example of Figure 1), or upload an external one. She can specify an XPath query or choose to run a preset query from a proposed list related to the current internal p-document. Regarding

the evaluation method, settings can be custom-made: the user can choose to run one or more algorithms (naïve evaluation, EvalDP, independent exact computation, approximations), or run the default configuration that picks an adequate method for the given query as explained in Section 4. We also give the possibility to personalize the number of samples needed for additive and multiplicative approximations, in one of the following three ways:

- by calculating, using Hoeffding's inequality [3], a suitable number of trials given a tolerated error under a probabilistic guarantee;
- 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.

Output: Different results are displayed, namely the probability of the query for each run method, running times, and probabilistic guarantees given by Hoeffding's inequality for approximation techniques. Additionally, the user can choose to plot the evolution of the estimated probability of approximation techniques and corresponding error intervals, to get a better grasp on the precision obtained by these techniques.

6 Conclusion and Perspectives

We presented a first step and a scenario towards a probabilistic DBMS using XML technology capable of efficiently querying discrete probabilistic data models. 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 [1], which also introduce continuous distributions. Our implementation could be extended to these directions and could also move to a distributed framework to manage probabilistic data, i.e., in a open file sharing environment or in the case of data integration [5]. Update operations also belong to future implementation perspectives.

References

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

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

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

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

[5] M. van Keulen, A. de Keijzer, and W. Alink. A probabilistic XML approach to data integration. In *ICDE*, pages 459–470, 2005.