Incorporating Constraints in Probabilistic XML

Sara Cohen[*] Benny Kimelfeld[†] Yehoshua Sagiv[†]

The Selim and Rachel Benin School of Engineering and Computer Science
The Hebrew University of Jerusalem, Edmund J. Safra Campus, Jerusalem 91094, Israel
{sara,bennyk,sagiv}@cs.huji.ac.il

ABSTRACT

Constraints are important not just for maintaining data integrity, but also because they capture natural probabilistic dependencies among data items. A *probabilistic XML database* (PXDB) is the probability sub-space comprising the instances of a *p-document* that satisfy a set of constraints. In contrast to existing models that can express probabilistic dependencies, it is shown that query evaluation is tractable in PXDBs. The problems of sampling and determining well-definedness (i.e., whether the above subspace is nonempty) are also tractable. Furthermore, queries and constraints can include the aggregate functions *count, max, min* and *ratio.* Finally, this approach can be easily extended to allow a probabilistic interpretation of constraints.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.4 [**Database Management**]: Systems—*Query processing*; G.3 [**Mathematics of Computing**]: Probability and Statistics

General Terms

Algorithms

Keywords

Probabilistic databases, probabilistic XML, constraints, sampling probabilistic data

1. INTRODUCTION

In applications that involve analysis and management of databases, uncertainty is often an inherent property of the data. Examples include medical information that is based on statistics and (imprecise) examinations, and image analysis. As another example, *screen-scraping*, used to automatically derive data from Internet sites, naturally gives rise to uncertainties—both due to the error-prone nature of the task, as well as to the possible unreliability of data sources on the Web. As a running example in this paper, we will consider data about an academic department, that may have been derived by screen-scraping a university Web site.

In a *probabilistic database* [1,7,8,11–16,19–21] uncertainty is an integral part of the database, usually by means of associating probabilities (or probabilistic events) with the different data items. Formally, the result is a probability distribution over ordinary databases. Consequently, traditional database concepts can be naturally applied in a manner that is aware of uncertainty, by using the formal foundations of probability theory. For example, the result of applying a query to a probabilistic database includes a probability measure for each of the answers, namely, the probability of obtaining that answer when querying a random instance.

In the probabilistic relational model of [7,8], tuples are associated with probabilities and are assumed to be independent of each other. In [8], it is shown that query evaluation is efficient for a sub-fragment of the *hierarchical conjunctive queries* (which are a severely limited form of queries) and is #P-hard (which is deemed highly intractable [18,23]) for all other conjunctive queries. These results have been extended to relational aggregate queries [20]. Computing *probabilistic maximal joins* has been studied in [15].

P-documents[1] form a probabilistic model for XML and were presented in [16]. In a p-document, probabilities are associated with document edges. The probability of a node is dependent only on the probabilities of its ancestors. For example, in Figure 1, a p-document describing an academic department is depicted. The probability that David is a Ph.D. student of Lisa (see the middle of the tree) is dependent on the nodes above David, but independent of all other data items in the document. *Probabilistic trees*, another model for probabilistic XML data, were presented in [1,21]. In this model, probabilistic events are associated with nodes. This allows complex probabilistic dependencies between nodes to be expressed explicitly.

Within the current state-of-art models, [13] draws a clear tradeoff between the efficiency of query evaluation and the ability to model probabilistic dependencies. On the one hand, an efficient algorithm for evaluating *twig queries* [3]

[*]The research of this author was supported by The Israel Science Foundation (Grant 1032/05).

[†]The research of this author was supported by The Israel Science Foundation (Grant 893/05).

PODS'08, June 9–12, 2008, Vancouver, BC, Canada.
Copyright 2008 ACM 978-1-60558-108-8/08/06 ...$5.00.

[1] In [13], "p-document" is a more general concept. In the notation of [13], our model of p-documents is $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$.

with projection is given in [13,14] for p-documents.[2] The situation for the probabilistic-tree model of [1,21], on the other hand, is quite different. In [13], it is proved that query evaluation for non-trivial Boolean tree queries is #P-complete in the probabilistic-tree model. Note that the "monotonic" queries that are considered in [21] do not allow projection. Thus, typically, either query evaluation is intractable or the model has limited expressiveness.

Bayesian networks (cf. [17]) are commonly used for describing probabilistic dependencies. However, it is not clear how natural constraints and queries can be efficiently translated into Bayesian networks. Furthermore, it is unlikely that there are similar tractability results to those described in this paper, because determining (or approximating) the probability of very simple events (i.e., inference) in Bayesian networks is intractable [5,6].

In this paper, we present PXDBs, a novel way of modeling probabilistic XML. In our model, a probabilistic database consists of both a p-document, as well as a set of *constraints*. Constraints are important, first from the traditional database point of view, since they are a mechanism for maintaining data integrity. Even more significantly, constraints are important also from the probabilistic aspect because they implicitly capture natural probabilistic dependencies among data items. In addition, constraints can be derived immediately from user-knowledge about real-world requirements, and as such, are expected to be easy to formulate. This paper is the first to consider combining probabilistic data with a rich language of constraints. (In the past, e.g. [20], only key constraints were considered.)

To demonstrate the type of constraints that we can express, consider again the p-document of Figure 1. Suppose that we have the real-world constraint that a department chair must be a full professor (and not an assistant professor), and that a department has only one chair. In addition, the university may have a policy saying that an assistant professor guides at most one Ph.D. student. All of these constraints can be expressed in our model. Now, whether or not David is a Ph.D. student of Lisa depends not only on his ancestor nodes, but also on Lisa's position (which, in turn, depends on the position of Mary). Even more intricate probabilistic dependencies may be expressed by our model, e.g., by using constraints that are in themselves probabilistic, e.g., "with a probability of 95%, at least 80% of the chairs are full professors." Note that the probabilistic dependencies are in no case explicitly stated, but are simply implied by the constraints.

Although the nodes of our probabilistic XML have complex interdependencies, we show that three pivotal tasks can be performed efficiently. The first is that of testing *well-definedness* of a PXDB, i.e., non-emptiness of the probability sub-space (of the p-document) comprising the documents that satisfy the constraints. The second task is *query evaluation.* The third task is *sampling,* i.e., randomly generating a document in the distribution of a given PXDB, where the probability of generating each document is that of being generated by the PXDB at hand. This task is straightforward in all models described above, but not in our model.

Our contributions can be summarized as follows. First, a new model for probabilistic XML is presented, which can declaratively capture real-world probabilistic dependencies. Second, we present a natural, rich language for expressing constraints and queries. This language is based on recursively composing twig patterns and comparisons involving the aggregate functions *count, max, min* and *ratio*. We show that all three tasks can be performed efficiently for this language. The algorithm for evaluating formulae of this language is intricate and quite long to describe. A sketch of the algorithm appears in Section 5. All details appear in the long version of this paper [4].

For clarity of presentation, our results are first stated in a natural, but limited setting. Later, we discuss how to generalize our framework in several directions, mainly extensions of the language of constraints and queries. In particular, our model can allow a probabilistic interpretation of constraints. Finally, we discuss applying the concepts in this paper to other probabilistic models (in addition to p-documents).

2. DETERMINISTIC DATA MODEL

2.1 Trees

We use trees that are directed and unordered. Given a tree t, the set of nodes and the set edges are denoted by $\mathcal{V}(t)$ and $\mathcal{E}(t)$, respectively. Note that $\mathcal{E}(t) \subseteq \mathcal{V}(t) \times \mathcal{V}(t)$. We use $root(t)$ to denote the root of t. If $(n_1, n_2) \in \mathcal{E}(t)$, then n_2 is a *child* of n_1, which in turn is the *parent* of n_2. A *leaf* of t is a node without any children.

Suppose that there is a path from node n_1 to node n_2. We say that n_2 is a *descendant* of n_1, whereas n_1 is an *ancestor* of n_2. Note that every node is both a descendant and an ancestor of itself. If $n_1 \neq n_2$, then n_2 is a *proper descendant* of n_1, which in turn is a *proper ancestor* of n_2.

If t' and t are trees such that $\mathcal{V}(t') \subseteq \mathcal{V}(t)$ and $\mathcal{E}(t') \subseteq \mathcal{E}(t)$, then t' is a *subtree* of t. If t' contains the root of t, then it is a *root subtree* (abbr. *r-subtree*). If n is a node of t, then t^n_Δ denotes the subtree of t that is rooted at n and induced by all the descendants of n (i.e., it contains all those descendants and the edges connecting them).

2.2 Documents

In this paper, data are unranked XML documents. Formally, we assume that $\mathbf{\Sigma}$ is an infinite set of *labels.* An *XML document* (or just *document* for short) is a tree with a label of $\mathbf{\Sigma}$ attached to each node. We use d to denote documents, and u, v and w to denote their nodes. The label of a node v is denoted by $label(v)$.

Figure 2 depicts a document with data about a university. Nodes are represented by their labels. The direction of the edges is from top to bottom.

2.3 Patterns

A *twig pattern* (or *pattern* for short) is a tree T with *child* and *descendant* edges. Each node n of the pattern has a predicate $cond_n : \mathbf{\Sigma} \to \{\mathbf{true}, \mathbf{false}\}$. We use n and m to denote nodes of patterns (in order to make a clear distinction from nodes of documents).

For example, seven patterns appear in the top part of Figure 1. (The meaning of the rounded rectangles will be clarified in the next section.) Note that child and descendant edges are depicted by single and double lines, respectively. There are three types of predicates in this figure. The predicate "$*$" is simply **true**. A predicate of the form "$= x$" requires equality to x. Finally, a predicate of the form "$\sim *x$" means that x should be a suffix of the given label.

[2] The algorithm of [13] is for a more general model, but the above assumption of independence is still made.

A *match* of a pattern in a document is a mapping from the nodes of the pattern to those of the document, such that all the constraints of the pattern are satisfied. Formally, a match of the pattern T in the document d is a mapping $\varphi{:}\mathcal{V}(T) \to \mathcal{V}(d)$ that satisfies the following.

1. Roots are matched, i.e., $\varphi(\mathit{root}(T)) = \mathit{root}(d)$.
2. For all nodes $n \in \mathcal{V}(T)$, the label of $\varphi(n)$ *satisfies* cond_n; that is, $\mathit{cond}_n(\mathit{label}(\varphi(n))) = \textbf{true}$.
3. For all child edges $(n_p, n_c) \in \mathcal{E}(T)$, the node $\varphi(n_p)$ is the parent of $\varphi(n_c)$ in d.
4. For all descendant edges $(n_a, n_d) \in \mathcal{E}(T)$, the node $\varphi(n_a)$ is a proper ancestor of $\varphi(n_d)$ in d.

$\mathcal{M}(T, d)$ denotes the set of all the matches of a pattern T in a document d.

2.4 Queries and Selectors

A query is constructed by applying a projection to a pattern. Formally, a *projection sequence* for a pattern T is of the form $(n_1, \ldots, n_k)$, where the n_i are nodes of T. A *query* has the form $Q = \pi_X\, T$, where T is a pattern and X is a projection sequence for T. The *result* of applying Q to a document d, denoted by $Q(d)$, consists of all the tuples t that are obtained by applying X to the matches of $\mathcal{M}(T, d)$, that is,

$$Q(d) = \{(\varphi(n_1), \ldots, \varphi(n_k)) \mid \varphi \in \mathcal{M}(T, d)\}\,.$$

By a slight abuse of notation, we do not distinguish between a sequence of length 1 and its single element. For example, we write $\pi_n\, T$ instead of $\pi_{(n)}\, T$, and we view the result of $\pi_n\, T$ as a set of nodes of T. A query of the form $\pi_n\, T$ (i.e., a projection on a single node) is called a *selector*, since it selects nodes of documents.

Example 2.1. *Consider the patterns in the top part of Figure 1 and the document d of Figure 2. A selector $\pi_n\, T$ is represented by surrounding n with a rounded rectangle. The selector S_{dep} selects the departments under the root of d, and the result is a single node. The selector S_{chr} selects member nodes where the person is both a professor and a chair. S_{mem} selects member nodes that are ancestors of professors; for d, the result is all the member nodes. Finally, S_{st} selects the name nodes that are children of nodes labeled with "ph.d. st.;" for d, these are the name nodes of David and Nicole.* □

2.5 Constraints

Our constraints are based on selectors and the *count* aggregate function. They are defined as follows.

Let U be a set of nodes of a document d. The aggregate function $\textsc{Cnt}$ returns the size of U, namely, $\textsc{Cnt}(U) = |U|$. In particular, if S is a selector, then $\textsc{Cnt}(S(d))$ is the number of nodes of d that are selected by S. We are now ready to define constraints.

Definition 2.2 (Constraints). *A* constraint *is of the form $\forall S(\textsc{Cnt}(S_1)\, \theta_1\, N_1 \to \textsc{Cnt}(S_2)\, \theta_2\, N_2)$, where*

- *S, S_1 and S_2 are selectors,*
- *θ_1 and θ_2 are numerical-comparison operators from* $\{=, \neq, <, \leq, >, \geq\}$, and
- *N_1 and N_2 are integers.*

Given a document d, the quantifier $\forall S$ iterates over all the nodes that are selected by S. For each selected node v, both S_1 and S_2 are applied to the subtree d^v_Δ. Formally, a document d *satisfies* $\forall S(\textsc{Cnt}(S_1)\, \theta_1\, N_1 \to \textsc{Cnt}(S_2)\, \theta_2\, N_2)$ if the following formula is true.

$$\forall v \in S(d)(\textsc{Cnt}(S_1(d^v_\Delta))\, \theta_1\, N_1 \to \textsc{Cnt}(S_2(d^v_\Delta))\, \theta_2\, N_2)$$

We use $\mathcal{C}$ to denote a finite set of constraints. If d satisfies every constraint of $\mathcal{C}$, then we write $d \models \mathcal{C}$.

Example 2.3. *The constraints $C_1, \ldots, C_4$ of Figure 1 are defined by means of the patterns and selectors shown in the top part of that figure. In this example, we use a shorthand notation as follows. First, a pattern T (e.g., in C_3) stands for the comparison $\textsc{Cnt}(\pi_{root(T)}\, T) > 0$, namely, there exists a match of T. Second, the symbol $*$ represents the pattern that consists of one node with the predicate* **true**.

The constraint C_1 says that a department cannot have more than one chair. According to C_2, a department with 3 or more professors must have a chair. C_3 requires a member to be a full professor in order to be a chair. Finally, C_4 says that an assistant professor can supervise at most one Ph.D. student.

When checking satisfaction of C_2 and C_4, we map the root of S_{mem} to department nodes and to the root of d, respectively. The document d of Figure 2 satisfies each of $C_1, \ldots, C_4$. Note that if Mary was not a chair (i.e., the corresponding position node did not have a chair child), then d would violate C_2. Similarly, if Lisa was an assistant (rather then full) professor, then d would violate C_4. □

3. PROBABILISTIC DATABASES

Probabilistic XML is a probability distribution over a space of documents. In this section, we define the model of *probabilistic XML databases (PXDB)*. A PXDB consists of a p-document and a set of constraints. We first define these two components.

3.1 P-Documents

A *probabilistic document* (or *p-document*, for short) defines a probability distribution over documents, regardless of any constraint. We follow the model of [13, 14, 16] where a p-document is a tree $\tilde{\mathcal{P}}$ that has two types of nodes. *Ordinary* nodes are regular XML nodes (with a label) and they may appear in random documents. *Distributional* nodes are only used for defining the probabilistic process of generating random documents (but they do not occur in those documents). A distributional node specifies a probability distribution over the subsets of its children.

We denote by $\mathcal{V}^{ord}(\tilde{\mathcal{P}})$ and $\mathcal{V}^{dst}(\tilde{\mathcal{P}})$ the (disjoint) sets of ordinary and distributional nodes of $\tilde{\mathcal{P}}$, respectively. A distributional node is neither the root nor a leaf of $\tilde{\mathcal{P}}$. We use the tilde sign to denote that $\tilde{\mathcal{P}}$ is a probability space.

A random document of a p-document $\tilde{\mathcal{P}}$ is generated by a two-step procedure. In the first step, we generate a random r-subtree R of $\tilde{\mathcal{P}}$ by applying the following top-down process, starting at the root. If we are at an ordinary node, we simply proceed to its children. When reaching a distributional node, we randomly, and independently, choose a (possibly empty) subset of its children and proceed to each chosen child. We delete the unchosen children and all their

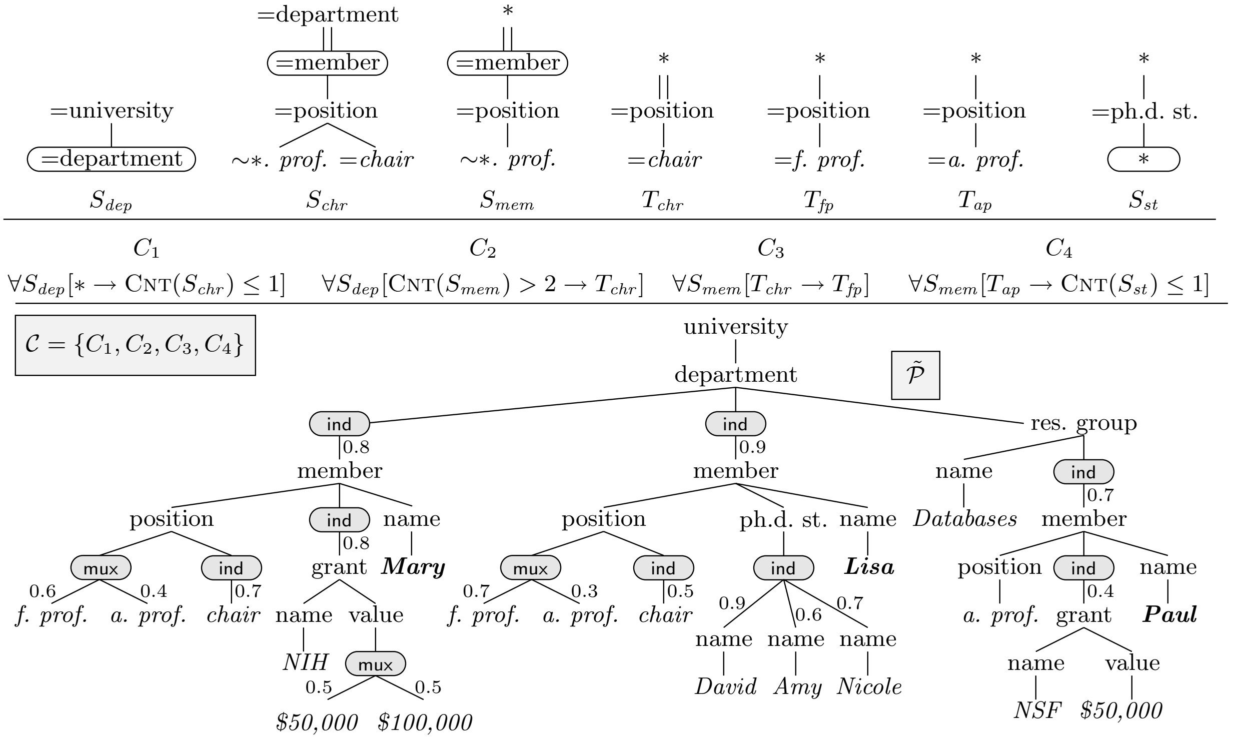

Figure 1: A PXDB $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$ describing information obtained by screen-scraping a university Web site

descendants. The result of the top-down process is a random r-subtree R of $\tilde{\mathcal{P}}$. Note that R is not a document, because it contains distributional nodes of $\tilde{\mathcal{P}}$. So, in the second step, we remove all the distributional nodes. If we remove the parent of an ordinary node u, then the new parent of u is its lowest ancestor among all the ordinary nodes.[3]

There are two types of distributional nodes. An ind node has children that are probabilistically independent of each other, while the children of a mux node are *mutually exclusive,* that is, at most one child can be chosen. Note that a distributional node can have distributional children. Hence, we can obtain more complex types of distributions by composing hierarchies of distributional nodes.

Let $v_1, \ldots, v_k$ be the children of a node $u \in \mathcal{V}^{dst}(\tilde{\mathcal{P}})$. For each child v_i, the p-document $\tilde{\mathcal{P}}$ specifies the probability that v_i exists given that u exists. We denote this probability by $\tilde{\mathcal{P}}(u, v_i)$ and assume that it is a rational number in $[0, 1]$. Note that $\tilde{\mathcal{P}}(u, v_i)$ is part of the description of the p-document $\tilde{\mathcal{P}}$.[4] If u is a mux node, then we also assume that $\sum_{i=1}^{k} \tilde{\mathcal{P}}(u, v_i) \leq 1$.

Example 3.1. *Consider the p-document $\tilde{\mathcal{P}}$ shown in the middle part of Figure 1. Ordinary nodes are represented by their labels. Distributional nodes are shown as rounded rectangles. $\tilde{\mathcal{P}}$ specifies that Mary is a chair with probability* 0.7*. Furthermore, she is either a full professor (with probability* 0.6*) or an assistant professor (with probability* 0.4*), but not both. In fact, she must be either a full or an assistant professor because there is a probability of 0 that she is neither. The document d of Figure 2 belongs to the probability distribution of $\tilde{\mathcal{P}}$.* □

Consider a p-document $\tilde{\mathcal{P}}$. We use $\mathcal{P}$ (i.e., without the tilde sign) to denote the random variable that represents a

[3] Note that two different r-subtrees R (obtained from two different random processes) may yield the same document. This follows from the fact that a distributional node can have a distributional child.

[4] Our complexity analysis assumes that $\tilde{\mathcal{P}}(u, v_i)$ is given as two integers: the numerator and the denominator.

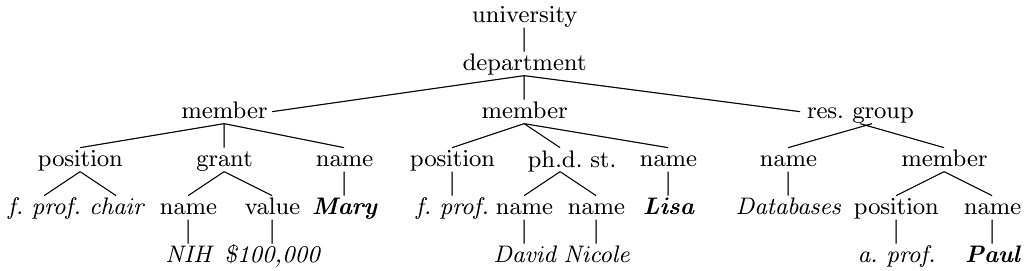

Figure 2: A random instance d of the PXDB $\tilde{\mathcal{D}}$ of Figure 1

document of the probability space $\tilde{\mathcal{P}}$ (i.e., a document chosen according to the distribution of $\tilde{\mathcal{P}}$). For example, if $\mathcal{C}$ is a set of constraints, then $\Pr(\mathcal{P} \models \mathcal{C})$ is the probability that a random document of $\tilde{\mathcal{P}}$ satisfies all the constraints of $\mathcal{C}$.

EXAMPLE 3.2. *Consider the p-document $\tilde{\mathcal{P}}$ of Figure 1. The random variable $\mathcal{P}$ contains the Ph.D. student Amy with probability 0.54, which is easily computed by multiplying the probabilities on the path from the root to Amy.* □

3.2 PXDBs

A PXDB is defined in terms of a p-document and a set of constraints. Informally, the PXDB is the sub-space comprising all the documents that satisfy the constraints. Next, we give the formal definition.

We say that a p-document $\tilde{\mathcal{P}}$ is *consistent* with a set $\mathcal{C}$ of constraints if $\Pr(\mathcal{P} \models \mathcal{C}) > 0$. A PXDB is a probability space defined by a pair $(\tilde{\mathcal{P}}, \mathcal{C})$, such that $\tilde{\mathcal{P}}$ is a p-document, $\mathcal{C}$ is a set of constraints and $\tilde{\mathcal{P}}$ is consistent with $\mathcal{C}$. The PXDB $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$ comprises all documents d of $\tilde{\mathcal{P}}$, such that $d \models \mathcal{C}$ and $\Pr(\mathcal{P} = d) > 0$. The probability distribution of $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$ is that of $\tilde{\mathcal{P}}$ conditioned on satisfying $\mathcal{C}$. That is, the probability of a document d of $\tilde{\mathcal{D}}$ is $\Pr(\mathcal{P} = d \mid \mathcal{P} \models \mathcal{C})$.

EXAMPLE 3.3. *Figure 1 depicts the PXDB $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$, where $\mathcal{C} = \{C_1, \ldots, C_4\}$. Examples 3.1 and 2.3 describe $\tilde{\mathcal{P}}$ and $\mathcal{C}$, respectively.* □

$\mathcal{D}$ denotes the random variable that represents a document of the probability space $\tilde{\mathcal{D}}$. For all documents d (by the definition of conditional probability),

$$\Pr(\mathcal{D} = d) = \begin{cases} \frac{\Pr(\mathcal{P} = d)}{\Pr(\mathcal{P} \models \mathcal{C})} & \text{if } d \models \mathcal{C}, \\ 0 & \text{otherwise.} \end{cases}$$

EXAMPLE 3.4. *To illustrate the dependencies that exist among the nodes of a PXDB, let us consider again the event from Example 3.2 (i.e., the Ph.D. student Amy appears in a random document). But now the probability space is $\tilde{\mathcal{D}}$ rather than $\tilde{\mathcal{P}}$ (both are depicted in Figure 1). Due to C_4, this event depends on two other events: whether Lisa is an assistant or a full professor, and whether she has other Ph.D. students. In turn, C_3 implies that the former event depends on whether Lisa is a chair. But due to C_1, Lisa cannot be a chair if Mary is. Whether Mary is a chair depends on her rank (i.e., assistant or full professor) and, due to C_2, also on whether Paul exists as a member (otherwise, there are fewer than 3 members, and hence, the department does not necessarily have a chair). So, there does not seem to be any obvious way of computing the probability that Amy appears in $\mathcal{D}$.* □

The above example shows that the probabilistic dependencies among data items of a PXDB go far beyond the tree structure of the p-document. In particular, they involve nodes that are not related by the ancestor-descendant relationship.

In Section 7, we consider additional types and interpretations of constraints that extend the capability to model real-world situations. Two *different* examples are: “80% of the chairs are full professors,” and “with probability 0.8, all chairs are required to be full professors.”

4. COMPUTATIONAL PROBLEMS

Before defining three computational problems, we explain the relevant complexity measure.

We use the notion of *data complexity* in order to determine tractability of the problems at hand. This means that queries and constraints are assumed to be fixed while the input consists of the p-document. Note, however, that the numerical operands in the CNT functions (used in the constraints) may be large. Consequently, “polynomial data complexity” might not be a practical yardstick of efficiency. Therefore, we apply the following twist, as done in [20]. In addition to the p-document, the input contains a *numerical specification* of the constraints. The numerical specification simply gives the number N in every comparison $\textsc{Cnt}(S)\,\theta\,N$ (while S and θ remain fixed).

The first problem that we study is deciding whether a PXDB is well-defined, namely, whether a given p-document is consistent with a set of constraints. More generally, the *constraint-satisfaction* problem (defined below) is computing the probability that a p-document satisfies a set of constraints.

PROBLEM 1 (CONSTRAINT SATISFACTION). *For a set $\mathcal{C}$ of constraints,* CONSTRAINT-SAT$\langle\mathcal{C}\rangle$ *denotes the problem of determining the probability $\Pr(\mathcal{P} \models \mathcal{C})$, where the input is a p-document $\tilde{\mathcal{P}}$ and a numerical specification of $\mathcal{C}$.*

The next problem is query evaluation. Over probabilistic databases, the usual approach (e.g, [7, 8, 14]) is to compute the probability of each tuple. Consider a PXDB $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$ and a query Q. Let $\mathcal{A}$ be the set of all tuples t, such that $t \in Q(d)$ for some document d of $\tilde{\mathcal{D}}$. The *result* of applying Q to $\tilde{\mathcal{D}}$, denoted by $Q(\tilde{\mathcal{D}})$, is the mapping $\Phi : \mathcal{A} \to [0, 1]$, such that for all $t \in \mathcal{A}$, it holds that $\Phi(t) = \Pr(t \in Q(\mathcal{D}))$.

PROBLEM 2 (QUERY EVALUATION). *For a query Q and a set $\mathcal{C}$ of constraints,* EVAL$\langle Q, \mathcal{C}\rangle$ *denotes the problem of computing $Q(\tilde{\mathcal{D}})$, where $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$. Note that the input is a numerical specification of $\mathcal{C}$ and a p-document $\tilde{\mathcal{P}}$ that is consistent with $\mathcal{C}$.*

Sampling is “the act, process, or technique of selecting a representative part of a population for the purpose of determining parameters or characteristics of the whole population” (*Merriam-Webster Online Dictionary*). In our setting, given a PXDB $\tilde{\mathcal{D}}$, sampling is a randomized method that generates a document d of $\tilde{\mathcal{D}}$. This method should properly simulate $\tilde{\mathcal{D}}$ in the sense that for all documents d, the probability of generating d is equal to $\Pr(\mathcal{D} = d)$.

PROBLEM 3 (SAMPLING). *For a set $\mathcal{C}$ of constraints,* SAMPLE$\langle\mathcal{C}\rangle$ *is the problem of randomly generating a document of $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$, such that every d of $\tilde{\mathcal{D}}$ is generated with probability $\Pr(\mathcal{D} = d)$. The input is a numerical specification of $\mathcal{C}$ and a p-document $\tilde{\mathcal{P}}$ that is consistent with $\mathcal{C}$.*

5. THE EVALUATION ALGORITHM

In this section, we show that the constraint-satisfaction and query-evaluation problems are tractable. We do so by solving these problems for *c-formulae* that are defined below.

We consider a query $Q = \pi_X T'$ and a set of constraints $\mathcal{C}$ that are fixed, whereas the input is a p-document $\mathcal{P}$ and a numerical specification of $\mathcal{C}$. We assume that $Q = \pi_X T'$ is a

Boolean query, namely, X is empty, written as $X = ()$. It is not difficult to reduce the general case (i.e., X is nonempty) to Boolean queries by extending the notion of labels.

If there is a match of T' in d (i.e., $\mathcal{M}(T', d) \neq \emptyset$), then we write $d \models T'$. Evaluating the Boolean query Q over a document d is deciding whether $d \models T'$; if so, the answer is **true**, and otherwise, it is **false**. Hence, the evaluation of Q over a PXDB $\tilde{\mathcal{D}}$ amounts to computing the probability $\Pr(\mathcal{D} \models T')$. By the definition of conditional probability,

$$\Pr\left(\mathcal{D} \models T'\right) = \Pr\left(\mathcal{P} \models T' \mid \mathcal{P} \models \mathcal{C}\right) = \frac{\Pr\left(\mathcal{P} \models \mathcal{C} \wedge T'\right)}{\Pr\left(\mathcal{P} \models \mathcal{C}\right)} .$$

It thus suffices to first compute $\Pr(\mathcal{P} \models \mathcal{C})$ and then $\Pr(\mathcal{P} \models \mathcal{C} \wedge T')$ in order to solve the constraint-satisfaction and query-evaluation problems. We do this by means of *c-formulae* that generalize constraints and Boolean queries, and can express $\mathcal{C}$ and $\mathcal{C} \wedge T'$.

Intuitively, a new c-formula is built from existing ones in three steps. First, we *augment* a pattern T by attaching a c-formula to every node. (When mapping a node n of T to a node v of a document d, the subtree d_Δ^v must satisfy the c-formula attached to n.) Second, we convert the augmented pattern to an *s-formula,* which is a generalized selector. Third, we use one or more s-formulae in order to create a new c-formula, which is a generalized constraint. Next, we give the mutually recursive definition of augmented patterns, s-formulae and c-formulae; we denote them by αT, σ and γ, respectively. The term *formula* refers to either an s-formula or a c-formula.

Definition 5.1 (Formulae). *There are five cases.*

1. *The basic c-formulae are* **true** *and* **false***.*
2. *If $\gamma_1, \dots, \gamma_m$ are c-formulae, then $\gamma_1 \wedge \cdots \wedge \gamma_m$ is a c-formula.*
3. *If T is a pattern and α is a function that maps every node of T to a c-formula, then αT is an augmented pattern.*
4. *If αT is an augmented pattern and $n \in \mathcal{N}(T)$, then $\pi_n \, \alpha T$ is an s-formula.*
5. *If $N \in \mathbb{Z}$, $\theta \in \{=, \neq, <, \leq, >, \geq\}$ and $\sigma_1, \dots, \sigma_k$ are s-formulae, then $\gamma = \textsc{Cnt}(\sigma_1 \vee \cdots \vee \sigma_k) \, \theta \, N$ is a c-formula.*

We now define how to evaluate augmented patterns, s-formulae and c-formulae over a document d. An augmented pattern has an associated set of matches. The result of an s-formula σ, denoted by $\sigma(d)$, is a set of nodes of d. A c-formula γ gets a Boolean value, and $d \models \gamma$ denotes that this value is **true**. In the following definition, the order of items corresponds to that of Definition 5.1.

Definition 5.2 (Evaluation). *Let d be a document.*

1. *If $\gamma \in \{\mathbf{true}, \mathbf{false}\}$, then $d \models \gamma$ iff $\gamma = \mathbf{true}$.*
2. *If $\gamma = \gamma_1 \wedge \cdots \wedge \gamma_m$ is a c-formula, then $d \models \gamma$ iff $d \models \gamma_i$ for all $1 \leq i \leq m$.*
3. *If αT is an augmented pattern, then $\mathcal{M}(\alpha T, d)$ is the subset of all matches $\varphi \in \mathcal{M}(T, d)$, such that for all $n' \in \mathcal{V}(T)$, it holds that $d_\Delta^{\varphi(n')} \models \alpha(n')$.*
4. *If $\sigma = \pi_n \, \alpha T$ is an s-formula, then σ selects the set of nodes $\sigma(d) = \{\varphi(n) \mid \varphi \in \mathcal{M}(\alpha T, d)\}$.*
5. *If $\gamma = \textsc{Cnt}(\sigma_1 \vee \cdots \vee \sigma_k) \, \theta \, N$, then $d \models \gamma$ evaluates to $|\sigma_1(d) \cup \cdots \cup \sigma_k(d)| \, \theta \, N$.*

5.1 Queries and Constraints as C-Formulae

We now show how to express queries and constraints as c-formulae. We use T_0 to denote the *trivial pattern*, which consists of a single node r that has the predicate **true**.

An augmented pattern αT and a c-formula γ are *congruent* if for all documents d, there is a match of αT in d (i.e., $\mathcal{M}(\alpha T, d) \neq \emptyset$) if and only if $d \models \gamma$. Given a c-formula γ, we can construct a congruent αT by taking the trivial pattern T_0 and attaching γ to its root. Similarly, given an augmented pattern αT, a congruent c-formula is $\textsc{Cnt}(\pi_r \, \alpha T) = 1$, where r is the root of αT. Hence, $\textsc{Cnt}(\pi_r \, \alpha T) = 0$ is an *anti-congruent* of αT, namely, $d \models \textsc{Cnt}(\pi_r \, \alpha T) = 0$ if and only if $\mathcal{M}(\alpha T, d) = \emptyset$. So, the negation of a c-formula γ is obtained by converting γ to a congruent αT and then constructing an anti-congruent of αT. It thus follows that c-formulae are closed under conjunction, negation and disjunction.

From now on, we view every pattern T as an augmented one; if α is not explicitly given, then it maps every node of T to the c-formula **true**.

A Boolean query $Q = \pi_{()} \, T$ is expressed as a c-formula by taking the congruent of T. Now, consider a constraint $\forall S(\textsc{Cnt}(S_1) \, \theta_1 \, N_1 \rightarrow \textsc{Cnt}(S_2) \, \theta_2 \, N_2)$ and let $S = \pi_n \, T$. We express this constraint as a c-formula by first converting T into an augmented pattern αT, as described next, and then taking an anti-congruent of αT. To obtain αT, we attach **true** to all the nodes of T, except for n which gets the c-formula $\textsc{Cnt}(S_1) \; \theta_1 \; N_1 \wedge \textsc{Cnt}(S_2) \; \bar{\theta}_2 \; N_2$, where $\bar{\theta}_2$ is the complement of θ_2 (e.g., if θ_2 is $<$ then $\bar{\theta}_2$ is $\geq$).

It follows from this discussion that if c-formulae can be evaluated efficiently over p-documents, then both constraint satisfaction and query evaluation can be solved in polynomial time. Evaluation of c-formulae is the subject of the next section.

5.2 Evaluating C-Formulae

The following theorem shows that c-formulae can be evaluated efficiently over p-documents.

Theorem 5.3. *Consider a c-formula γ. Computing the probability $\Pr(\mathcal{P} \models \gamma)$, where the input is a p-document $\tilde{\mathcal{P}}$ and a numerical specification of γ, is in polynomial time.*

By the discussion above, we derive the following corollary.

Corollary 5.4. *For all queries Q and finite sets of constraints $\mathcal{C}$, both* $\textsc{Constraint-Sat}\langle\mathcal{C}\rangle$ *and* $\textsc{Eval}\langle Q, \mathcal{C}\rangle$ *can be solved in polynomial time.*

In the remainder of this section, we describe the algorithm for evaluating c-formulae, i.e., the proof of Theorem 5.3. Since the evaluation algorithm is involved, we will only sketch the main ideas. The complete proof is given in [4].

Let γ be a c-formula and $\tilde{\mathcal{P}}$ be a p-document. The goal is to compute $\Pr(\mathcal{P} \models \gamma)$. Our strategy is to apply a recursion that reduces the problem of computing $\Pr(\mathcal{P} \models \gamma)$ to problems of computing probabilities of the form $\Pr(\mathcal{P}' \models \gamma')$, where $\mathcal{P}'$ is a proper subtree of $\mathcal{P}$ (and γ' is a c-formula).

Note that this approach was also used in [13,14] for the task of evaluating Boolean queries (rather than c-formulae), that is, computing the probability $\Pr(\mathcal{M}(T', \mathcal{P}') \neq \emptyset)$ for a fixed pattern T' and a given p-document $\tilde{\mathcal{P}}'$. However, evaluating c-formulae is far more intricate.

To make the discussion clearer, we start by considering a restricted case, where γ is of the form $\textsc{Cnt}(\pi_n\, \alpha T) = N$, namely, γ is a c-formula with a single conjunct, the argument of $\textsc{Cnt}$ has only one disjunct and θ is the equality operator. Later on, we briefly discuss the general case (i.e., when these restrictions are not assumed).

In order to straightforwardly apply a recursive reduction it is helpful if if the c-formula γ and the p-document $\tilde{\mathcal{P}}$ satisfy the following conditions:

1. $\pi_n\, \alpha T$ satisfies the following.
   (a) No condition is attached to the root of T, i.e., $cond_{root(T)} = \mathbf{true}$.
   (b) No c-formula is attached to the root of T, i.e., $\alpha(root(T)) = \mathbf{true}$.
   (c) γ counts a non-root element, that is, $root(T) \neq n$.
   (d) $root(T)$ has exactly one child.
   (e) The single edge that emanates from $root(T)$ is a child edge.
2. $root(\tilde{\mathcal{P}})$ has exactly one child which is ordinary.

If the above conditions hold, then we do the following reduction. Let T' and $\tilde{\mathcal{P}}'$ be the pattern and p-document that are obtained from T and $\tilde{\mathcal{P}}$, respectively, by removing the root. Let γ' be the c-formula $\textsc{Cnt}(\pi_n\, \alpha T') = N$. Then

$$\Pr(\mathcal{P} \models \gamma) = \Pr\left(\mathcal{P}' \models \gamma'\right) . \qquad (1)$$

Clearly, the conditions above may not hold. We first consider the case where Condition 1 is violated, and show how to transform $\Pr(\mathcal{P} \models \gamma)$ into a sum $\sum_{i=1}^{q} \pm \Pr(\mathcal{P} \models \gamma_i')$, such that each γ_i' satisfies Condition 1. Afterwards, we discuss violations of Condition 2.

Suppose that Condition 1 does not hold. In this case we transform $\Pr(\mathcal{P} \models \gamma)$ into a sum $\sum_{i=1}^{q} \pm \Pr(\mathcal{P} \models \gamma_i')$, such that each γ_i' satisfies Condition 1. Initially, the sum consists of only the original probability $\Pr(\mathcal{P} \models \gamma)$. Afterwards, we repeatedly apply *transformations* to an operand $\pm \Pr(\mathcal{P} \models \gamma_i')$ of the sum, in order to replace it with a new sum. We repeat this step until none of the transformations that we consider can be applied to any operand. Our transformations are devised such that upon termination, all the c-formulae γ_i' of $\sum_{i=1}^{q} \pm \Pr(\mathcal{P} \models \gamma_i')$ will satisfy Condition 1.

We need eight different transformations [4]. Due to lack of space, we discuss only one, called, *ExCR* (extracting c-formulae from roots). Furthermore, we describe this transformation only for the simple case where γ_i' is of the form $\textsc{Cnt}(\pi_n\, \alpha' T') = 0$. This transformation is applicable if $\alpha'(root(T')) \neq \mathbf{true}$, that is, Condition 1(b) is violated. Let α'' be the same mapping as α', except that it maps $root(T')$ to $\mathbf{true}$. The transformation ExCR replaces the addend $\Pr(\mathcal{P} \models \textsc{Cnt}(\pi_n\, \alpha' T') = 0)$ with the following equal sum.

$$\begin{aligned}&\Pr\left(\mathcal{P} \models \neg\alpha'(root(T'))\right) + \\ &\Pr\left(\mathcal{P} \models \alpha'(root(T')) \wedge \textsc{Cnt}(\pi_n\, \alpha'' T') = 0\right)\end{aligned}$$

Note that $\alpha'(root(T'))$ (i.e., the c-formula attached to the root of $\alpha' T'$) does not necessarily satisfy Condition 1(b). So, the transformation ExCR may have to be applied again to the addends of the above sum.

By repeatedly applying the transformations, we will end up with a summation of $\sum_{i=1}^{r} \pm \Pr(\mathcal{P} \models \gamma_i)$, such that every γ_i satisfies Condition 1. If Condition 2 holds, then the recursive reduction (described earlier) can be applied. We now consider the case where only Condition 2 is violated.

We consider again a c-formula γ of the (restricted) form $\textsc{Cnt}(\pi_n\, \alpha T) = N$, such that γ satisfies Condition 1. If the root of $\tilde{\mathcal{P}}$ has exactly one child v, such that v is distributional, then the subtree of $\tilde{\mathcal{P}}$ rooted at v is not a well-defined p-document. We handle this case similarly to [13,14] by introducing a dummy node as the parent of v (further details are omitted due to lack of space). Next, suppose that the root of $\tilde{\mathcal{P}}$ has multiple children $w_1, \ldots, w_k$. We partition $\tilde{\mathcal{P}}$ into two p-documents as follows. $\tilde{\mathcal{P}}_k$ is the subtree of $\tilde{\mathcal{P}}$ that consists of the root, the child w_k and all the descendants of w_k. $\tilde{\mathcal{P}}_{<k}$ comprises the root and the rest of the nodes of $\tilde{\mathcal{P}}$, i.e., it is obtained by removing w_k and its descendants. Clearly, a random document of $\tilde{\mathcal{P}}$ consists of two parts that are obtained from $\tilde{\mathcal{P}}_k$ and $\tilde{\mathcal{P}}_{<k}$. These two parts are probabilistically independent. Since γ satisfies Condition 1(c), $n \neq root(T)$. So, $\Pr(\mathcal{P} \models \gamma)$ can be computed as a sum of $N+1$ products,[5] namely,

$$\begin{aligned}&\sum_{i=0}^{N} \Pr\left(\mathcal{P}_{<k} \models \textsc{Cnt}(\pi_n\, \alpha T) = i\right) \times \qquad (2)\\ &\quad \times \Pr\left(\mathcal{P}_k \models \textsc{Cnt}(\pi_n\, \alpha T) = N - i\right) .\end{aligned}$$

The first term in each product is computed recursively by partitioning $\tilde{\mathcal{P}}_{<k}$ into two p-documents as described above. The second term is computed by applying Equation (1).

Our discussion above sketched the main ideas of our algorithm for the restricted form of γ defined above. We conclude the description of the algorithm with a few remarks on the details needed for more general forms of c-formulae. Some transformations introduce conjunctions of c-formulae and disjunctions of s-formulae (which are defined in Parts 2 and 5, respectively, of Definition 5.1). Therefore, the above description of the algorithm is an oversimplification due to the assumption about the restricted form of the c-formula γ. In particular, the handling of multiple children is more complicated than that described by Equation (2), because we need to evaluate a conjunction of c-formulae $\gamma^1 \wedge \cdots \wedge \gamma^l$ rather than just an atomic constraint $\textsc{Cnt}(\pi_n\, \alpha T) = N$ (see [4] for details). A second remark is that constraints with inequalities are handled by one of the transformations that replaces any θ comparison with equality.

To prove correctness of our algorithm, we show that the repeated execution of the transformations terminates. Moreover, for proving efficiency, we show the following two facts. First, there is a polynomial (in the size of $\tilde{\mathcal{P}}$ and the numerical specification) upper bound on the number of repeated applications (when starting with a given c-formula γ). Second, the result of applying the transformations (at each node of $\tilde{\mathcal{P}}$) has a polynomial size. The proof of efficiency also requires to show that there is a polynomial upper bound on the overall number of probabilities that need to be evaluated throughout the entire algorithm (i.e., including the recursive steps). This follows from the fact that *memoing* [24] is used to avoid an exponential blowup during the recursion (and from the second fact above).

[5] Note that we can assume that N is at most the number of (ordinary) nodes of $\mathcal{P}$, so this summation is efficient.

6. SAMPLING PXDBS

In this section, we present an efficient sampling algorithm for PXDBs. The formal result is the following.

Theorem 6.1. *Let $\mathcal{C}$ be a finite set of constraints. There is an efficient randomized algorithm for solving the problem* Sample$\langle\mathcal{C}\rangle$.

Throughout this section, we assume that $\mathcal{C}$ is a fixed set of constraints. The input consists of a p-document $\tilde{\mathcal{P}}$ and a numerical specification of $\mathcal{C}$. Furthermore, $\tilde{\mathcal{P}}$ is consistent with $\mathcal{C}$, and $\tilde{\mathcal{D}}$ is the PXDB $(\tilde{\mathcal{P}}, \mathcal{C})$. By $\mathcal{E}^{dst}(\tilde{\mathcal{P}})$ we denote the set of all edges $(v, u) \in \mathcal{E}(\tilde{\mathcal{P}})$, such that v is distributional. We assume that $\mathcal{E}^{dst}(\tilde{\mathcal{P}}) = \{(v_1, w_1), \ldots, (v_m, w_m)\}$.

In the rest of this section, we describe the algorithm of Figure 3, called $\mathsf{Sample}\langle\mathcal{C}\rangle(\mathcal{P})$, that constructs a sample with the appropriate probability. The underlying idea is rather simple. The loop of Line 4 iterates over $\mathcal{E}^{dst}(\tilde{\mathcal{P}})$. When processing the edge (v_i, w_i), the task is to randomly choose it for the construction of the sample. The main problem is to compute the probability of making this choice. Two factors determine this probability: the choices made thus far, and the probability (at this stage) of producing a random document that satisfies the constraints if we indeed choose (v_i, w_i).

To reflect the choices made thus far, we change the p-document as follows.[6] Let $\tilde{\mathcal{P}}_{i-1}$ be the p-document just before processing the edge (v_i, w_i). We obtain the p-document $\tilde{\mathcal{P}}_i$ by modifying $\tilde{\mathcal{P}}_{i-1}$ as follows. If we choose the edge (v_i, w_i), then we set $\tilde{\mathcal{P}}_i(v_i, w_i)$ to 1; otherwise, we set it to 0. Furthermore, if v_i is a mux node, then for each child w' of v_i, such that $w' \neq w_i$, we do the following. If we choose (v_i, w_i), then we set $\tilde{\mathcal{P}}_i(v_i, w')$ to 0; otherwise, we set it to $\tilde{\mathcal{P}}_{i-1}(v_i, w')/(1 - \tilde{\mathcal{P}}_{i-1}(v_i, w_i))$.

The subroutine Norm produces a new p-document by modifying $\tilde{\mathcal{P}}_{i-1}$ as described above. In Line 10, it is applied under the assumption that (v_i, w_i) is included in the construction of the sample, whereas the opposite is true in Line 19.

At the beginning of the ith iteration, $\tilde{\mathcal{P}}_{i-1}(v_i, w_i)$ is the prior probability of choosing the edge (v_i, w_i). However, for randomly deciding whether to include (v_i, w_i) in the generation of the sample, we have to use the posterior probability of choosing (v_i, w_i) given that the sample satisfies the constraints. Let p_i be this posterior probability. Note that $\Pr(\mathcal{P}_i \models \mathcal{C})$ is the posterior probability that the sample satisfies the constraints given that (v_i, w_i) is chosen. Hence, by Bayes' theorem, $p_i = \tilde{\mathcal{P}}_{i-1}(v_i, w_i) \cdot \Pr(\mathcal{P}_i \models \mathcal{C})/q_{i-1}$, where $q_{i-1} = \Pr(\mathcal{P}_{i-1} \models \mathcal{C})$ is the prior probability (at the beginning of the ith iteration) that the sample satisfies the constraints.

So, Lines 10–13 calculate the probability p_i of choosing (v_i, w_i). The actual random choice is made in Line 14. If (v_i, w_i) is chosen, then Lines 16–17 set the values of $\tilde{\mathcal{P}}_i$ and q_i. Otherwise, these values are set in Lines 19–20. Note that in the latter case, we calculate q_i from the values that have already been computed in Lines 11 and 12 where it is assumed that (v_i, w_i) is chosen.

In Lines 5–9, the loop handles two special cases where the prior probability of choosing the edge (v_i, w_i) is either 0 or 1. This is done not just for the sake of efficiency—it is essential for correctness.

Algorithm $\mathsf{Sample}\langle\mathcal{C}\rangle(\tilde{\mathcal{P}})$

```
1: let $\mathcal{E}^{dst}(\tilde{\mathcal{P}}) = \{(v_1, w_1), \ldots, (v_m, w_m)\}$
2: $\tilde{\mathcal{P}}_0 \leftarrow \tilde{\mathcal{P}}$
3: $q_0 \leftarrow \Pr(\mathcal{P}_0 \models \mathcal{C})$
4: for $i = 1$ to $m$ do
5:    if $\tilde{\mathcal{P}}_{i-1}(v_i, w_i) \in \{0, 1\}$ then
6:       $p_i \leftarrow \tilde{\mathcal{P}}_{i-1}(v_i, w_i)$
7:       $\tilde{\mathcal{P}}_i \leftarrow \tilde{\mathcal{P}}_{i-1}$
8:       $q_i \leftarrow q_{i-1}$
9:       continue
10:   $\tilde{\mathcal{P}}' \leftarrow \mathsf{Norm}(\tilde{\mathcal{P}}_{i-1}, v_i \rightarrow w_i)$
11:   $q' \leftarrow \Pr(\mathcal{P}' \models \mathcal{C})$
12:   $\hat{q}_i \leftarrow \tilde{\mathcal{P}}_{i-1}(v_i, w_i)$
13:   $p_i \leftarrow \hat{q}_i \cdot q'/q_{i-1}$
14:   toss $b_i \in \{\mathbf{true}, \mathbf{false}\}$ with prob. $p_i$ for true
15:   if $b_i$ then
16:      $\tilde{\mathcal{P}}_i \leftarrow \tilde{\mathcal{P}}'$
17:      $q_i \leftarrow q'$
18:   else
19:      $\tilde{\mathcal{P}}_i \leftarrow \mathsf{Norm}(\tilde{\mathcal{P}}_{i-1}, v_i \not\rightarrow w_i)$
20:      $q_i \leftarrow (q_{i-1} - q' \cdot \hat{q}_i)/(1 - \hat{q}_i)$
21: return the constant random variable $\mathcal{P}_m$
```

Figure 3: Sampling $\tilde{\mathcal{D}} = (\tilde{\mathcal{P}}, \mathcal{C})$

$\tilde{\mathcal{P}}_m$ is the last p-document to be created. For all i, the probability $\tilde{\mathcal{P}}_m(v_i, w_i)$ is either 0 or 1. Consequently, $\tilde{\mathcal{P}}_m$ is a distribution with only one document, namely, $\mathcal{P}_m$. This document is returned in Line 21.

The following theorem shows that the algorithm indeed samples $\tilde{\mathcal{D}}$. Due to lack of space, the proof is omitted. By Corollary 5.4, $\mathsf{Sample}\langle\mathcal{C}\rangle$ is efficient. Thus, Theorem 6.1 immediately follows.

Theorem 6.2. *For all documents d, the probability that* $\mathsf{Sample}\langle\mathcal{C}\rangle(\mathcal{P})$ *returns d is equal to* $\Pr(\mathcal{D} = d)$.

7. GENERALIZATIONS

We now discuss how the results of the previous sections can be generalized in various ways.

7.1 Generalizing to C-Formulae

In order to deal with queries and constraints as they were originally defined, we have to translate them to c-formulae. Obviously, the results of Section 5 about efficient solutions to the constraint-satisfaction and query-evaluation problems carry over to queries and constraints that are expressed as arbitrary c-formulae. Furthermore, the sampling algorithm and its proof of correctness are oblivious to the type of constraints (as long as we can efficiently compute the probability that the constraints are satisfied by a random instance). Consequently, our results on sampling generalize straightforwardly to the case of constraints expressed as c-formulae.

7.2 Additional Aggregate Functions

Next, we consider additional aggregate functions. Generally, an *aggregate function agg* maps a set of nodes to a rational number. Below, we naturally generalize the notion of c-formulae to *a-formulae* (that use arbitrary aggregate

[6] An alternative is to introduce new constraints instead of changing the p-document, but this is inefficient because it produces a set of constraints that is linear in the size of $\tilde{\mathcal{P}}$.

functions rather than just Cnt). In Definitions 5.1 and 5.2, $\textsc{Cnt}(\sigma_1 \vee \cdots \vee \sigma_k)\,\theta\,N$ is replaced with $\mathit{agg}(\sigma_1 \vee \cdots \vee \sigma_k)\,\theta\,R$, where R is a rational number. In addition, every occurrence of the term "c-formula" is replaced with "a-formula." We denote by $\mathbf{AF}^{\{agg_1,\ldots,agg_k\}}$ the set of all a-formulae that use aggregate functions from $\{agg_1,\ldots,agg_k\}$.

Naturally, every a-formula of $\mathbf{AF}^{\{agg_1,\ldots,agg_k\}}$ is a constraint. We say that a query $Q = \pi_X\,\alpha T$ *belongs* to the family $\mathbf{AF}^{\{agg_1,\ldots,agg_k\}}$ if T is a pattern, X is a projection sequence for T, and α maps every node of T to an a-formula of $\mathbf{AF}^{\{agg_1,\ldots,agg_k\}}$. Given a document d, the result $Q(d)$ consists of all the tuples that are obtained by applying the projection to the matches of $\mathcal{M}(\alpha T, d)$.

Next, we define the aggregate functions Min, Max, Sum, Avg and Ratio. For that, we assume that labels of $\mathbf{\Sigma}$ may be rational numbers. The Boolean function numeric(l) specifies whether a given label l is numeric. Like Cnt, these five aggregate functions are defined over subsets U of document nodes. We denote by $\mathit{Num}(U)$ the set $\{v \in U \mid \text{numeric}(\mathit{label}(v))\}$.

The aggregate functions Max and Min are defined as follows. Suppose first that $\mathit{Num}(U) \neq \emptyset$. Then $\textsc{Max}(U) = \max\{\mathit{label}(v) \mid v \in \mathit{Num}(U)\}$ and $\textsc{Min}(U) = \min\{\mathit{label}(v) \mid v \in \mathit{Num}(U)\}$. If $\mathit{Num}(U)$ is empty, then $\textsc{Max}(U)$ is $-\infty$ and $\textsc{Min}(U)$ is ∞.

The functions Sum and Avg are naturally defined as follows. $\textsc{Sum}(U) = \sum_{v \in \mathit{Num}(U)} \mathit{label}(v)$. If U is nonempty, then $\textsc{Avg}(U) = \textsc{Sum}(U)/\,\textsc{Cnt}(U)$; otherwise, if U is empty, then $\textsc{Avg}(U)$ is 0.

The *ratio* function, denoted by Ratio, is a special type of an aggregate function that can be used for representing important constraints of the form "at least 40% of all professors (in each department) have an active grant" (such a constraint can be derived, e.g., from published statistics of the university). Formally, this function gets as input a set U of nodes and a subset U' of U, and it returns $|U'|/|U|$ (if U is empty, then the result is 0).

An a-formula that uses Ratio is built from a disjunction $\sigma_1 \vee \cdots \vee \sigma_k$ and another a-formula γ, and it has the form $\textsc{Ratio}(\sigma_1 \vee \cdots \vee \sigma_k, \gamma)\theta R$. Given a document d, the a-formula $\textsc{Ratio}(\sigma_1 \vee \cdots \vee \sigma_k, \gamma)\,\theta\,R$ is **true** if $r\,\theta\,R$ is satisfied, where r is the fraction of the nodes n from $\sigma_1(d) \cup \cdots \cup \sigma_k(d)$ that satisfy $d^n_\Delta \models \gamma$.

The following theorem shows that our results can be extended to constraints that use the Max, Min and Ratio functions, in addition to Cnt. The numerical specification includes all the rational numbers that appear in the a-formula.

Theorem 7.1. *For all sets of constraints $\mathcal{C}$ and queries Q that belong to $\mathbf{AF}^{\{\textsc{Cnt},\,\textsc{Max},\,\textsc{Min},\,\textsc{Ratio}\}}$, the three problems* Constraint-Sat$\langle\mathcal{C}\rangle$, Eval$\langle Q,\mathcal{C}\rangle$ *and* Sample$\langle\mathcal{C}\rangle$ *can be solved efficiently.*

Next, we show that similar results do not exist for the aggregate functions Sum and Avg. In particular, the following proposition shows that for very simple c-formulae of $\mathbf{AF}^{\{\textsc{Sum}\}}$ and $\mathbf{AF}^{\{\textsc{Avg}\}}$, respectively, computing (or even approximating) the probability of satisfaction is intractable. The a-formula $\xi_{\Sigma\mathrm{all}}$ is the formula that verifies whether the sum of all the numerical values in the document equals a specific value, i.e., the a-formula[7] $\textsc{Sum}(* \vee *//*) = R$ (where R is given in the numerical specification). Similarly, $\xi_{\overline{\mathrm{all}}}$ verifies whether the average numerical value in the document equals the specified R, i.e, $\textsc{Avg}(* \vee *//*) = R$.

Proposition 7.2. *Deciding each of* $\Pr(\mathcal{P} \models \xi_{\Sigma all}) > 0$ *and* $\Pr(\mathcal{P} \models \xi_{\overline{all}}) > 0$*, given* $\tilde{\mathcal{P}}$ *and a numerical specification, is NP-complete.*

The proof of the above proposition is by a reduction from the problem Subset-Sum. Note that a direct corollary is that even approximating $\Pr(\mathcal{P} \models \xi_{\Sigma\mathrm{all}})$ and $\Pr(\mathcal{P} \models \xi_{\overline{\mathrm{all}}})$, under the notion of approximation defined in [7], is infeasible (unless NP $\subseteq$ P or NP $\subseteq$ BPP, depending on whether the approximation is deterministic or randomized, respectively).

7.3 Generalizing P-Documents

All our results can be generalized to p-documents that extend the *probabilistic instances* studied in [11, 12].[8] These p-documents are obtained by allowing an additional type of distributional nodes, called exp, that is defined as follows. An exp node v contains an explicit specification of the probability distribution over subsets of its children. That is, if $w_1,\ldots,w_k$ are the children of v, then the p-document specifies several subsets of $\{w_1,\ldots,w_k\}$ and the probability of choosing each one as the (exact) set of children of v. The probabilities specified for v add up to 1.

The *probabilistic trees* of [1, 21] form another model of probabilistic XML. In [13], it is proved that except for trivial cases, every query (i.e., twig pattern with projection) is intractable to evaluate in this model;[9] however, there is an efficient (multiplicative) approximation. We can show that if our model is extended with the features of probabilistic trees, then even approximating query evaluation becomes intractable. In particular, it is NP-complete to determine whether there is a nonzero probability of satisfying the constraint: "every node labeled with A has a child."

7.4 Probabilistic Constraints

We introduce *probabilistic constraints* as a natural method to model restrictions with some level of uncertainty. Intuitively, a probabilistic constraint is simply a standard constraint C, associated with a probability p_C, which indicates the likelihood with which the constraint must be satisfied.

As an example, consider constraint C_3, which states that a chair must be a full professor. While this may be a rule in most universities, it is likely to be violated by some. Thus, constraint C_3 may actually be true only with some probability p (say, 0.95). As another example, we can derive probabilistic constraints from published statistics. Suppose that we learn from a public statement that in 40% of the universities there are more than a hundred Ph.D. students. This can naturally be modelled as a probabilistic constraint.

We consider the probabilistic space defined by a PXDB that consists of a p-document and a set of probabilistic constraints. There are two natural ways to define this probabilistic space—each by a reduction to a PXDB with deterministic constraints. Under *strict negated compliance* (SNC), each probabilistic constraint C must be satisfied with probability p_C, and the negation of C must be satisfied with

[7] This is an abuse of notation—we use XPath expressions rather than s-formulae.

[8] We consider only tree instances with point probabilities (rather than general graphs with intervals of probabilities, which cannot be captured in the setting of this paper).

[9] In [13], this model is denoted by $\mathsf{PrXML}^{\{\mathsf{cie}\}}$.

probability $1-p_C$. Under *weak negated compliance* (WNC), C must be *imposed* with probability p_C (and otherwise, it is disregarded and hence, can either be satisfied or not).

While SNC may be a more intuitive semantics, it could lead to a probability space of PXDBs that is not well-defined. For example, consider the following two probabilistic constraints. First, a full professor has at least one Ph.D. student with probability 0.7. Second, a full professor has at most 15 Ph.D. students with probability 0.9. Under SNC, the probability space is not well-defined, because there is a nonzero probability (i.e., $0.03 = (1 - 0.7) \times (1 - 0.9)$) that a random document must satisfy the negation of both constraints, which is impossible. For WNC, the probability space is well-defined, as long as the conjunction of all constraints is satisfiable (similarly to the case of deterministic constraints). In particular, for the above example, the probability space is well-defined with respect to WNC.

Importantly, we can extend all our results to probabilistic interpretations of constraints, for both WNC and SNC (when well-defined). Other semantics, e.g., that allow for probabilistic hierarchies of constraints, can also be defined and manipulated efficiently. Further details are omitted due to lack of space.

8. CONCLUSION AND FUTURE WORK

PXDBs provide a natural, database-driven methodology for an effective representation of probabilistic data. Although p-documents by themselves have limited capabilities, adding constraints makes it possible to introduce rather complex dependencies among probabilistic data. These complex dependencies notwithstanding, we can efficiently solve the constraint-satisfaction, query-evaluation and sampling problems for a rich language, namely, c-formulae with the aggregate functions count, max, min and ratio.

In [13], we have studied various combinations and natural extensions of models of probabilistic XML that were proposed in the literature [1, 11, 12, 14, 16, 21]. The focus of [13] is on expressiveness of the models, and tractability of evaluating queries that are formulated as patterns with projection. This work has also developed, implemented and optimized a query processor for one of the models.

Among the models studied in [13], there is a clear tradeoff between the efficiency of query evaluation and the ability to model probabilistic dependencies. In particular, efficient evaluation is possible only when distributional nodes are independent of each other. In [13], it is also shown that approximation techniques overcome this inherent tradeoff in the following sense. Query evaluation can be efficiently (multiplicatively) approximated over the most expressive model (among those considered in [13]).

The model of PXDBs takes a completely different approach to surmounting the limitation entailed by the above tradeoff. It does so by describing probabilistic dependencies in terms of a fixed set of constraints rather than many specific relationships among distributional nodes (as done in [1, 21]). The latter can describe cases that cannot be handled by the former, but the former is easier to formulate and much more succinct (hence, the assumption that the set of constraints is fixed is realistic). Furthermore, as mentioned in Section 7, even queries with negation (which is a restricted form of count) do not admit an efficient approximate (let alone exact) evaluation in the model of [1, 21].

Our plan for future work includes three topics. First, we will study the practical aspects of our evaluation algorithm. We expect that the optimization techniques (e.g., [22, 24]) that are used in [13] would lead to a scalable implementation with a practical level of performance. Second, we will investigate how to extend the constraint language, e.g., by adding *variables* (thus, modeling constraints in the spirit of [2,9,10]). Third, we will explore whether our approach can be adapted to the relational probabilistic model of [7, 8, 20].

9. REFERENCES

[1] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *EDBT*, 2006.

[2] N. Bidoit and D. Colazzo. Testing XML constraint satisfiability. *Electr. Notes Theor. Comput. Sci.*, 174(6), 2007.

[3] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: optimal XML pattern matching. In *SIGMOD*, 2002.

[4] S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilistic XML (extended version). Can be found in the second author's home page (`http://www.cs.huji.ac.il/~bennyk`), 2008.

[5] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artif. Intell.*, 42(2-3), 1990.

[6] P. Dagum and M. Luby. Approximating probabilistic inference in bayesian belief networks is NP-hard. *Artif. Intell.*, 60(1), 1993.

[7] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, 2004.

[8] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.

[9] W. Fan and L. Libkin. On XML integrity constraints in the presence of DTDs. *J. ACM*, 49(3), 2002.

[10] W. Fan and J. Siméon. Integrity constraints for XML. *J. Comput. Syst. Sci.*, 66(1), 2003.

[11] E. Hung, L. Getoor, and V. S. Subrahmanian. Probabilistic interval XML. In *ICDT*, 2003.

[12] E. Hung, L. Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *ICDE*, 2003.

[13] B. Kimelfeld, Y. Kosharovski, and Y. Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD*, 2008.

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

[15] B. Kimelfeld and Y. Sagiv. Maximally joining probabilistic data. In *PODS*, 2007.

[16] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *VLDB*, 2002.

[17] J. Pearl. Bayesian networks: A model of self-activated memory for evidential reasoning. In *CogSci*, 1985.

[18] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM J. Comput.*, 12(4), 1983.

[19] C. Re, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, 2007.

[20] C. Re and D. Suciu. Efficient evaluation of HAVING queries on a probabilistic database. In *DBPL*, 2007.

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

[22] H. Tamaki and T. Sato. OLD resolution with tabulation. In *ICLP*, 1986.

[23] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 21(2), 1992.

[24] D. S. Warren. Memoing for logic programs. *Commun. ACM*, 35(3), 1992.