Agrégation de documents XML probabilistes[*]

Serge Abiteboul[†] T.-H. Hubert Chan[‡] Evgeny Kharlamov[†§]
Werner Nutt[§] Pierre Senellart[¶]

Les sources d'incertitude et d'imprécision des données sont nombreuses. Une manière de gérer cette incertitude est d'associer aux données des annotations probabilistes. De nombreux modèles de bases de données probabilistes ont ainsi été proposés, dans les cadres relationnel et semi-structuré. Ce dernier est particulièrement adapté à la gestion de données incertaines provenant de traitement automatiques. Un important problème, dans le cadre des bases de données probabilistes XML, est celui des requêtes d'agrégation (count, sum, avg, etc.), qui n'a pas été étudié jusqu'à présent. Dans un modèle unifiant les différents modèles probabilistes semi-structurés étudiés à ce jour, nous présentons des algorithmes pour calculer la distribution des résultats de l'agrégation (qui exploitent certaines propriétés de régularité des fonctions d'agrégation), ainsi que des moments (en particulier, espérance et variance) de celle-ci. Nous prouvons également l'intractabilité de certains de ces problèmes.

Mots-clefs : XML, données probabilistes, agrégation, complexité, algorithmes

1 Introduction

The study of queries over imprecise data has generated much attention in the setting of relational databases [4, 9, 20, 28]. The Web (with HTML or XML data) in particular is an important source of uncertain data, for instance, when dealing with imprecise automatic tasks such as information extraction. A natural way to model this uncertainty is to annotate semi-structured data with probabilities. Some works have recently addressed queries over such imprecise hierarchical information [2, 14, 15, 17, 19, 22, 26, 27]. An essential aspect of query processing has been ignored in these works, namely aggregate queries. This is the problem we study here.

In this article, we consider *probabilistic XML documents*, described using the unifying model of *p-documents* [1, 17]. A p-document can be thought of as a probabilistic process that generates a random XML document. Some nodes, namely distributional nodes, specify how to perform this random selection. We consider three kinds of distributional operators: *cie*, *mux*, *det*, respectively for *conjunction of independent events* (based on conjunctive conditions of some probabilistic

[*]This research was supported by the Advanced ERC Grant Webdam on Foundations of Web Data Management, and by the Dataring project of the French ANR.

[†]INRIA Saclay – Île-de-France, 4 rue J. Monod, 91893 Orsay Cedex, France, first.last@inria.fr

[‡]Department of Computer Science, The University of Hong Kong, Pokfulam Road, Hong Kong, hubert@cs.hku.hk

[§]Free University of Bozen-Bolzano, Dominikanerplatz 3, 39100 Bozen, Italy, last@inf.unibz.it

[¶]Institut Télécom; Télécom ParisTech; CNRS LTCI, 46 rue Barrault, 75634 Paris Cedex 13, France, pierre.senellart@telecom-paristech.fr events), *mutually exclusive* (at most one node selected from a set of a nodes), and *deterministic* (all nodes selected). This model, introduced in [1, 17], captures a large class of models for probabilistic trees that had been previously considered. We consider queries that involve the standard aggregate functions count, sum, min, max, countd (count distinct) and avg. The focus here is on aggregation itself and not arbitrary query processing. We therefore focus our attention to aggregating *all* the leaves of a document. We briefly discuss the issue of reducing the evaluation of general aggregate queries to this problem.

A p-document is a (possibly compact) representation of a probabilistic space of (ordinary) documents, i.e., a finite set of possible documents, each with a particular probability. The result of the aggregate function is a single value for each possible document. Therefore, its result over a p-document is a random variable, i.e., a set of values, each with a certain probability. We investigate how to compute the distribution of this random variable. Such a distribution may be too detailed to present to a user. This leads us to consider some summaries of the distribution, especially its expected value and other probabilistic moments.

Our results highlight an (expectable) aspect of the different operators in p-documents: the use of *cie* (a much richer means of capturing complex situations) leads to a complexity increase. For documents with *cie* nodes, we show the problems are hard (typically NP- or $\mathsf{FP}^{\#\mathsf{P}}$-complete). This difficulty is yet another reason to consider expected values and other moments. For count and sum, we show how to obtain them in P. Unfortunately, we show that for min, max, countd, and avg, the problem of computing moments is also $\mathsf{FP}^{\#\mathsf{P}}$-complete. We present Monte-Carlo methods that allow tractable approximation of probabilities and moments of aggregate functions.

On the other hand, with the milder forms of imprecision, namely *mux* and *det*, the complexity is lower. Computing the distribution for count, min and max is in P. The result distribution of sum may be exponentially large, but the computation is still in P in both *input and output*. On the other hand, computing avg or countd is $\mathsf{FP}^{\#\mathsf{P}}$-complete. The good news is that we can compute expected values (and moments) for all of them in P.

Finally, we also provide results for a large class of aggregate queries where the aggregate function is based on the operator of a monoid and can be evaluated by a divide-and-conquer strategy. Examples of these functions are count, sum, min and max. We show how to use an algebraic structure of p-documents to evaluate monoid aggregate functions in general. Interesting non-monoid aggregate queries are rare but they exist, e.g. avg or countd, and are typically harder to compute. It should be noted that we don't know how to use the monoid properties in presence of the *cie* operator.

After presenting some preliminaries and the main investigated problems in Section 2, we discuss how to aggregate p-documents with *cie* nodes in Section 3. In Section 4, we present monoid aggregate functions and show how to compute distributions of these functions on p-documents with *mux* and *det* nodes by exploiting the structure of the documents. Then we continue with this model of p-documents and study complexity of distributions and moments in Section 5. Finally, we present related work and conclude in Section 6.

2 Preliminaries and Problem Definition

In this section we recall the model of probabilistic trees from [1], and formalize the problem we study in this paper.

Documents. We assume a countable set of *identifiers* $\mathcal{V}$ and *labels* $\mathcal{L}$, such that $\mathcal{V} \cap \mathcal{L} = \emptyset$. A *labeling* function, denoted θ, maps $\mathcal{V}$ to $\mathcal{L}$. Inspired by XML, we define a *document*, denoted

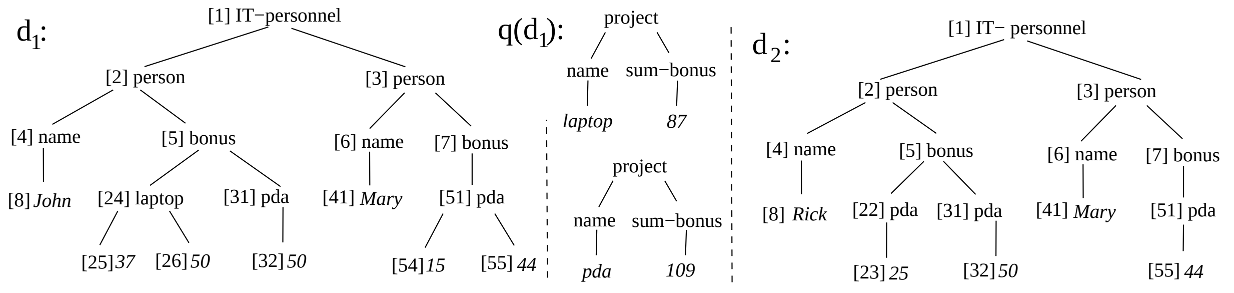

Figure 1: Documents d_1 and d_2 and aggregate query result $Q(d_1)$.

by d, as a finite, unordered[1], labeled tree where each node has a unique identifier v and a label $\theta(v)$. The set of all *nodes* and *edges* of d are denoted respectively by $\mathcal{V}(d)$ and $\mathcal{E}(d)$, where $\mathcal{E}(d) \subseteq \mathcal{V}(d) \times \mathcal{V}(d)$. We use the common notions *child* and *parent*, *descendant* and *ancestor*, *root* and *leaf* in the usual way. We denote the root of d by $\mathsf{root}(d)$ and the empty tree, that is, the tree with no nodes, by ϵ. A *forest*, denoted by F, is a set of documents.

Example 1. Consider the two example documents d_1 and d_2 in Figure 1. Identifiers appear inside square brackets before labels. Both documents describe the personnel of an IT department and bonuses that the personnel earned working on different projects. Document d_1 indicates `John` worked under two projects (`pda` and `laptop`) and got bonuses of `37` and `50` in the former project and `50` in the latter one.

Aggregate Functions. An *aggregate function* is a function that maps finite bags of values into some domain such as the rationals, the reals, or tuples of reals. For example,

- count and countd return the number of the elements and the number of distinct elements in a bag, respectively.
- min, max over a bag of elements from a linearly ordered set $(A, <)$ return, respectively, the minimal and maximal element in the bag. (One can generalize max to topK.)
- sum and avg over bags of rational numbers compute their sum and average, respectively.

Aggregate functions can be naturally extended to work on documents d: the result $\alpha(d)$ is $\alpha(B)$ where B is the bag of the labels of all leaves in d. This makes the assumption that all leaves are of the type required by the aggregate function, e.g., rational numbers for sum. We ignore this issue here and assume they all have the proper type (this can be ensured by a query that only selects some of the leaves, as detailed further). We extend the notion of aggregate function over documents to forests by considering the bag of leaves of all trees in the forest.

So-called *monoid* aggregate functions play an important role in our investigation, because they can be handled by a divide-and-conquer strategy (see [8]). Formally, a structure $(M, \oplus, \bot)$ is called an *abelian monoid* if $\oplus$ is an associative and commutative binary operation with $\bot$ as neutral element. If no confusion can arise, we speak of the monoid M. An aggregate function is a *monoid* one if for some monoid M and every $a_1, \ldots, a_n \in M$:

$$\alpha(\{\!|a_1, \ldots, a_n|\!\}) = \alpha(\{\!|a_1|\!\}) \oplus \cdots \oplus \alpha(\{\!|a_n|\!\}).$$

[1]This is a common simplification over the XML model; ordering children of a node does not significantly change the results.

It turns out that count, sum, min, max and topK are monoid aggregate functions. For sum, min, max: $\alpha(\{\!|a|\!\}) = a$ and $\oplus$ is the corresponding obvious operation. For count: $\alpha(\{\!|a|\!\}) = 1$ and $\oplus$ is $+$. For top2 over the natural numbers (and similarly for topK): $\alpha(\{\!|l|\!\}) = (l, 0)$ and $(l_1, l_2) \oplus (l_3, l_4) = (l_i, l_j)$, where $l_i \geq l_j$ and l_i, l_j are the top-2 elements in $\{\!|l_1, l_2, l_3, l_4|\!\}$.

It is easy to check that neither avg nor countd are monoid aggregate functions.

Aggregate Queries over Documents. An *aggregate query* is a query that uses aggregate functions.

Example 2. Continuing with Example 1, one can compute the *sum* of bonuses for *each* project that the personnel is involved in. In XQuery notation, this query Q can be written as follows:

```
for $x in distinct-values(//bonus/*/name())
return <project>
         <name> { $x } </name>
         <sum-bonus> { sum(//bonus/*[name()=$x]//*) } </sum-bonus>
       </project>
```

The query result $Q(d_1)$ is the forest of two documents presented in the middle of Figure 1.

When the query Q above is executed over a document, the variable `$x` is bound to a number of distinct values, which are computed in the outer loop. For each binding of `$x`, the expression $q =$ `//bonus/*[name()=$x]//*` is evaluated and the result is aggregated. In terms of XQuery, the XPath pattern q is *single-path*, that is, it does not require any branching. Our further investigations focus on single-path aggregate queries (the reasons will be discussed later while presenting aggregate queries over p-documents).

Let Q have n variables and an aggregate function α applied to a single-path pattern q. Then Q is computed in three steps: (i) find the matching n-tuples $\bar{a} = a_1, ..., a_n$ from the variables of Q to the labels of d; (ii) for each match $\bar{a}$, instantiate the variables in q with $\bar{a}$, resulting in a pattern $q_{\bar{a}}$ and compute the document $d(\bar{a})$ that is the fragment of d satisfying $q_{\bar{a}}$; (iii) for every $\bar{a}$, evaluate α over $d(\bar{a})$, and structure the resulting $(n+1)$-tuples $(\bar{a}, \alpha(d(\bar{a})))$ according to Q.

Continuing with Example 2: (i) there are two matches for `$x`: `laptop` and `pda`; (ii) evaluating q for them yields $d($`laptop`$)$ and $d($`pda`$)$, on the left of Figure 2; (iii) sum on these documents is $\mathsf{sum}(d(\texttt{laptop})) = 87$, $\mathsf{sum}(d(\texttt{pda})) = 109$. The query result is in Figure 1.

px-Spaces. A *probability space* over documents is an expression $(\mathcal{D}, \mathsf{Pr})$, where $\mathcal{D}$ is a set of documents and Pr maps each document to a probability with $\Sigma\{\mathsf{Pr}(d) \mid d \in \mathcal{D}\} = 1$. This definition is extended in a straightforward way to probability spaces $(\mathcal{F}, \mathsf{Pr})$ over forests, where $\mathcal{F}$ is a set of forests. If $\mathcal{F}$ is finite then $(\mathcal{F}, \mathsf{Pr})$ is called *px-space* and denoted by $\mathcal{S}$.

Using the documents from the previous example, we can construct $\mathcal{S} = (\{d1, d2, \ldots\}, \mathsf{Pr})$ a px-space with, say, $\mathsf{Pr}(d_1) = 0.047$ and $\mathsf{Pr}(d_2) = 0.008$, etc.

p-Documents. Following [1], we now introduce a very general syntax for representing compactly px-spaces, called *p-documents*. P-documents are similar to documents, with the difference that they have two types of nodes: *ordinary* and *distributional* ones. Distributional nodes are only used for defining the probabilistic process that generates random forests (but they do not actually occur in those ones). Ordinary nodes have labels and they may appear in random forests.

More precisely, we assume given a set $\mathcal{X}$ of Boolean random variables with some specified probability distribution Δ over them. A *p-document*, denoted by $\widehat{\mathcal{P}}$, is an unranked, unordered,

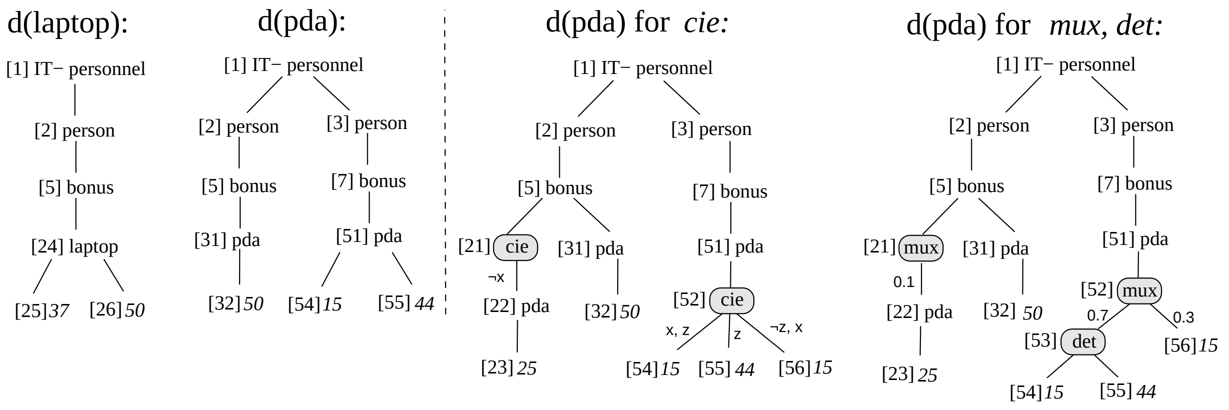

Figure 2: Non-aggregate query results for Q over d_1, $\widehat{\mathcal{P}}_1 \in \mathsf{PrXML}^{cie}$, $\widehat{\mathcal{P}}_2 \in \mathsf{PrXML}^{mux,det}$.

labeled tree. Each node has a unique identifier v and a label $\mu(v)$ in $\mathcal{L} \cup \{cie(E)\}_E \cup \{mux(\mathsf{Pr})\}_{\mathsf{Pr}} \cup \{det\}$ where $\mathcal{L}$ are labels of *ordinary* nodes, and the others labels are of *distributional* nodes. We consider three kinds of the latter labels: $cie(E)$ (for conjunction of independent events), $mux(\mathsf{Pr})$ (for mutually exclusive), and det (for deterministic). We will refer to distributional nodes labeled with these labels, respectively, as *cie*, *mux*, *det* nodes. If a node v is labeled with $cie(E)$, then E is a function that assigns to each child of v a conjunction $e_1 \wedge \cdots \wedge e_k$ of literals (x or $\neg x$, for $x \in \mathcal{X}$). If v is labeled with $mux(\mathsf{Pr})$, then Pr assigns to each child of v a probability with the sum equal to 1.

We require the leaves to be ordinary nodes[2]. There are two more kinds of distributional nodes considered in [1] that we briefly discuss further.

Example 3. Two p-documents are shown in Figure 3. The left one, $\widehat{\mathcal{P}}_1$, has only *cie* distributional nodes. For example, node n_{21} has the label $cie(E)$ and two children n_{22} and n_{24} such that $E(n_{22}) = \neg x$ and $E(n_{24}) = x$. The p-document on the right, $\widehat{\mathcal{P}}_2$, has only *mux* and *det* distributional nodes. Node n_{52} has the label $mux(\mathsf{Pr})$ and two children n_{53} and n_{56} such that $\mathsf{Pr}(n_{53}) = 0.7$ and $\mathsf{Pr}(n_{56}) = 0.3$.

We denote *classes of p-documents* by PrXML with a superscript denoting the types of distributional nodes that are allowed for the documents in the class. For instance, $\mathsf{PrXML}^{mux,det}$ is the class of p-documents with only *mux* and *det* distributional nodes, like $\widehat{\mathcal{P}}_2$ on Figure 3.

The *semantics of a p-document* $\widehat{\mathcal{P}}$, denoted by $[\![\widehat{\mathcal{P}}]\!]$, is a px-space over *random forests*, where the forests are denoted by $\mathcal{P}$ and are obtainable from $\widehat{\mathcal{P}}$ by a randomized three-step process.

1. We choose a valuation ν of the variables in $\mathcal{X}$. The probability of the choice, according to the distribution Δ, is $p_\nu = \prod_{x \text{ in } \widehat{\mathcal{P}}, \nu(x)=\mathsf{true}} \Delta(x) \cdot \prod_{x \text{ in } \widehat{\mathcal{P}}, \nu(x)=\mathsf{false}} (1 - \Delta(x))$.
2. For each *cie* node labeled $cie(E)$, we delete its children v such that $\nu(E(v))$ is false, and their descendants. Then, independently for each *mux* node v labeled $mux(\mathsf{Pr})$, we select one of its children v' according to the corresponding probability distribution Pr and delete the other children and their descendants, the probability of the choice is $\mathsf{Pr}(v')$. We do not delete any of the children of *det* nodes[3].

[2]In [1], the root is also required to be ordinary. For technical reasons explained in Section 4, we do not use that restriction here.

[3]It may seem that using *det* nodes is redundant, but actually they increase the expressive power when used together with *mux* and other types of distributional nodes [1].

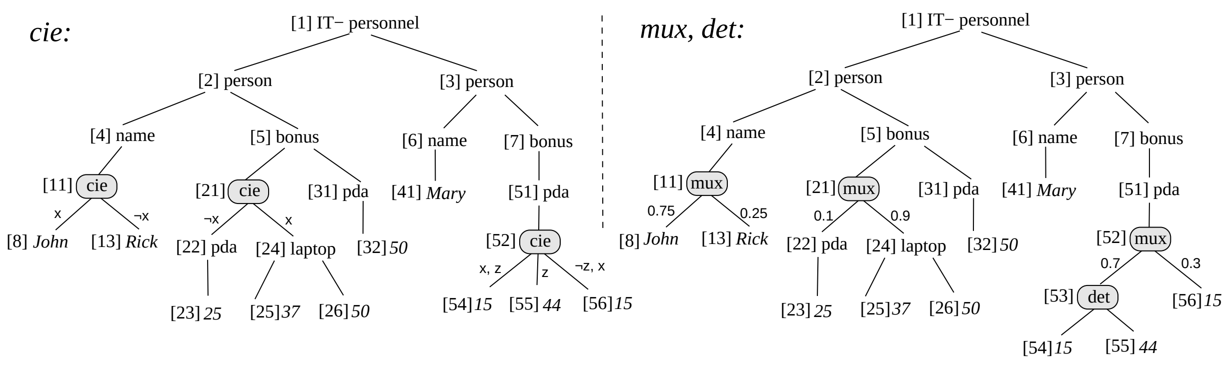

Figure 3: P-documents $\widehat{\mathcal{P}}_1 \in \mathsf{PrXML}^{cie}$, and $\widehat{\mathcal{P}}_2 \in \mathsf{PrXML}^{mux,det}$.

3. We then remove in turn each distributional node, connecting each ordinary child v of a deleted distributional node with its lowest ordinary ancestor v', or, if no such v' exists, we turn this child into a root.

The result of this third step is a random forest $\mathcal{P}$. The probability $\mathsf{Pr}(\mathcal{P})$ is defined as the product of p_ν, the probability of the variable assignment we chose in the first step, with all $\mathsf{Pr}(v')$, the probabilities of the choices that we made in the second step for the *mux* nodes.

Example 4. Applying the randomized process to $\widehat{\mathcal{P}}_2$ in Figure 3 one obtains documents from Figure 1: d_1 with $\mathsf{Pr}(d_1) \approx 0.047$ and d_2 $\mathsf{Pr}(d_2) \approx 0.008$. If one assumes that $\mathsf{Pr}(x) = .85$ and $\mathsf{Pr}(y) = .055$ then one obtain d_1 and d_2 with the same probabilities .047 and .008, respectively, from $\widehat{\mathcal{P}}_1$ by assigning $\{x/1, y/1\}$ and $\{x/0, y/1\}$, respectively. We notice that the semantics of both p-documents in Figure 3 is the same, that is, $\widehat{\mathcal{P}}_1$ and $\widehat{\mathcal{P}}_2$ represent the same px-space.

Remark. In [1] two more types of distributional nodes are considered. Firstly, *ind* nodes that select their children *independently* of each other according to some probability for each child. It has been shown in [1] that *ind* nodes can be captured by *mux* and *det*. Moreover, PrXML^{cie} is strictly more expressive than $\mathsf{PrXML}^{mux,det}$. Finally, one can consider *exp* nodes that *explicitly* specify a probability for each given subset of their children to be chosen. This kind of distributional node is a generalization of *mux* and *det*, and most of the results for these distributional nodes can be extended to *exp*. For simplicity, we shall not discuss this further.

It was shown in [17, 18] that query answering with projections is intractable for PrXML^{cie} ($\mathsf{FP}^{\#\mathsf{P}}$-complete) whereas it is polynomial for $\mathsf{PrXML}^{mux,det}$. In a similar vein, it will turn out that in a number of cases aggregate query answering is more difficult for PrXML^{cie} than for the restricted case of $\mathsf{PrXML}^{mux,det}$.

Aggregate Queries over px-Spaces and p-Documents. In the following, we restrict ourselves to queries of the form $\alpha(q)$, where q is a single-path. Note that in Example 2 the aggregate query `sum(//bonus/*[name()=$x]//*)` is of this form and that such queries can occur as parts of larger queries, as shown in that example. Moreover, we assume that our single-path queries do not contain variables. The single-path query `//bonus/*[name()=$x]//*` contains the variable `$x`, but when the query is executed, the variable is bound to a constant. Over the documents and p-documents in the examples the two possible constants are `pda` and `laptop`, which give rise to two constant-free queries $q_{\texttt{pda}}$ and $q_{\texttt{laptop}}$.

The application of such a query $\alpha(q)$ to a document d according to the XQuery semantics can be understood as computing first the subtree d' of d whose leaves are the nodes satisfying q and then $\alpha(d')$ (note that according to our definition $\alpha(d')$ is computed from the leaves of d').

Moving to px-spaces, we first generalize the application of aggregate functions from documents to px-spaces by defining $\alpha(\mathcal{S})$ as the probability distribution of the values of α over $\mathcal{S}$, that is,

$$\alpha(\mathcal{S}) := \Big\{ (c, p) \;\Big|\; c \text{ is in the range of } \alpha,\ p = \textstyle\sum_{d \in \mathcal{S},\, \alpha(d)=c} \mathsf{Pr}(d) \Big\}.$$

This is canonically extended to p-documents by defining $\alpha(\widehat{\mathcal{P}}) := \alpha([\![\widehat{\mathcal{P}}]\!])$, that is, applying α to the px-space generated by $\widehat{\mathcal{P}}$.

In a similar vein, if q is a single-path query, then $q(\mathcal{S})$ is the px-space whose elements are the images $q(d)$ of documents $d \in \mathcal{S}$ and where the probability of an element $q(d)$ is $\mathsf{Pr}(q(d)) = \sum_{d' \in \mathcal{S},\, q(d')=q(d)} \mathsf{Pr}(d')$. Combining aggregation and single path queries, we define the answer to an aggregate query like above as $\alpha(q)(\mathcal{S}) := \alpha(q(\mathcal{S}))$, that is, α is applied to the px-space $q(\mathcal{S})$. Again, this is extended canonically to p-documents.

Let $\widehat{\mathcal{P}}$ be in PrXML^{cie}. If q is single-path, we can apply it naively to $\widehat{\mathcal{P}}$, ignoring the distributional nodes. The result $q(\widehat{\mathcal{P}})$ is the subtree of $\widehat{\mathcal{P}}$ containing the original root and as leaves the nodes satisfying q. For example, applying q_{pda} to $\widehat{\mathcal{P}}_1$ in Figure 3 yields "d(pda) for *cie*" in Figure 2. Interestingly, one can show that for all single path queries q it holds that $[\![q(\widehat{\mathcal{P}})]\!] = q([\![\widehat{\mathcal{P}}]\!])$. In other words, we have a strong representation system. If $\widehat{\mathcal{P}}$ in $\mathsf{PrXML}^{mux,det}$, then $q(\widehat{\mathcal{P}})$ can be obtained analogously (see for example "d(pda) for *mux*, *det*" in Figure 2 as the result of applying q_{pda} to $\widehat{\mathcal{P}}_2$ in Figure 3). One can show that, again, we have a strong representation system.

For aggregate queries where the aggregation is performed over single-path patterns, we have isolated aggregation from query processing, which is the crux we use to solve the problem of aggregate query answering in presence of probabilities. In this paper, the focus is on the computation of $\alpha(q_{\bar{a}}(\widehat{\mathcal{P}}))$, that is, evaluation of aggregate functions and not of queries over p-documents.

The problems. Suppose we are given a px-space under the form of a p-document and we want to compute an aggregate function over it. (Recall that, in general, we would like to ask first a query, compute the leaves that are of interest and then apply the aggregate query on these leaves. We deal here with the simpler problem of computing the aggregate function on *all* the leaves.) The result of an aggregate function over a px-space is a random variable. Given an aggregate function α, we are interested in the following *problems*, where the input parameters are a p-document $\widehat{\mathcal{P}}$ with corresponding random forest $\mathcal{P}$ and possibly a number c:

Membership: Given c, is c in the carrier of $\alpha(\mathcal{P})$, i.e., is $\mathsf{Pr}(\alpha(\mathcal{P}) = c) > 0$?

Probability computation: Given c, compute $\mathsf{Pr}(\alpha(\mathcal{P}) = c)$.

Distribution computation: Find all c's such that $\mathsf{Pr}(\alpha(\mathcal{P}) = c) > 0$, and for each such c compute $\mathsf{Pr}(\alpha(\mathcal{P}) = c)$.

Moment computation: Compute $\mathbb{E}(\alpha(\mathcal{P})^k)$, where $\mathbb{E}$ is the expected value.

One may want to return to a user the entire distribution as the result (distribution computation problem). The *membership* and *probability computation* problems can be used to solve it. Computing the distribution may be too costly or the user may prefer a *summary* of the distribution. For example, a user may want to know its expected value $\mathbb{E}(\alpha(\mathcal{P}))$ and the variance

Var$(\alpha(r))$. In general the summary can be an arbitrary k-th moment $\mathbb{E}(\alpha(\mathcal{P})^k)$ and the *moment computation* problem addresses this issue[4].

In the following we investigate these four problems for the aggregate functions min, count, sum, countd and avg. We do not discuss max and topK since they behave similarly as min.

3 Aggregating PrXML^{cie} P-Documents

We now study the four problems highlighted in Section 2 for the more general class of p-documents, PrXML^{cie}. Following the definitions, one approach is to first construct the entire px-space of a p-document $\widehat{\mathcal{P}}$, then to apply α to each document in $[\![\widehat{\mathcal{P}}]\!]$ separately, combine the results to obtain the distribution $\alpha(\widehat{\mathcal{P}})$, and finally compute the answers based on $\alpha(\widehat{\mathcal{P}})$. This approach is expensive, since the number of possible documents is exponential in the number of variables occurring in $\widehat{\mathcal{P}}$.

Our complexity results show that for practically all functions and all problems nothing can be done that would be significantly more efficient. Most decision problems are NP-complete while computational problems are FP$^{\#\mathsf{P}}$-complete. The only exception is the computation of moments for sum and count. The intractability is due to dependencies between nodes of p-documents expressed using variables. All intractability results already hold for shallow p-documents with very simple dependencies that connect only events that label siblings. As an polynomial-time alternative, we present an approach to compute approximate solutions by Monte-Carlo methods.

Functions in #P and FP$^{\#\mathsf{P}}$. We recall here the definitions of some classical complexity classes (see, e.g., [23]) that characterize the complexity of aggregation functions on PrXML^{cie}. An $\mathbb{N}$-valued function f is in #P if there is a non-deterministic polynomial time Turing machine T such that for every input w, the number of accepting runs of T is the same as $f(w)$. A function is in FP$^{\#\mathsf{P}}$ if it is computable in polynomial time using an oracle for some function in #P. Following [7], we say that a function is FP$^{\#\mathsf{P}}$-hard if there is a polynomial-time *Turing reduction* (that is, a reduction with access to an oracle to the problem reduced to) from every function in FP$^{\#\mathsf{P}}$ to it. Hardness for #P is defined in a standard way using Karp (many-one) reductions. For example, the function that counts for every propositional 2-DNF formula the number of satisfying assignments is in #P and #P-hard [24], hence #P-complete. We notice that the usage of Turing reductions in the definition of FP$^{\#\mathsf{P}}$-hardness implies that any #P-hard problem is also FP$^{\#\mathsf{P}}$-hard. Therefore, to prove FP$^{\#\mathsf{P}}$-completeness it is enough to show FP$^{\#\mathsf{P}}$ membership and #P-hardness. Note also that #P-hardness clearly implies NP-hardness.

We now consider membership in FP$^{\#\mathsf{P}}$. We say that an aggregate function α is *scalable* if for every p-document $\widehat{\mathcal{P}} \in$ PrXML^{cie} one can compute in polynomial time a natural number M such that for every $d \in [\![\widehat{\mathcal{P}}]\!]$ the product $M \cdot \alpha(d)$ is a natural number. The following result is obtained by adapting proof techniques of [11].

Theorem 1. *Let α be an aggregate function that is computable in polynomial time. If α is scalable, then the following functions mapping p-documents to rational numbers are in* FP$^{\#\mathsf{P}}$*:*
(i) for every $c \in \mathbb{Q}$, the function $\widehat{\mathcal{P}} \mapsto \mathsf{Pr}(\alpha(\mathcal{P}) = c)$;
(ii) for every $k \geq 1$, the function $\widehat{\mathcal{P}} \mapsto \mathbb{E}(\alpha(\mathcal{P})^k)$.

The above theorem shows membership in FP$^{\#\mathsf{P}}$ of both probability and moment computation for all aggregate functions mentioned in the paper, since they are scalable.

[4]The variance is the *central* moment of order 2; it turns out that the central moment of order k can be tractably computed from the regular moments of order $\leq k$.

3.1 Computing sum and count

For sum and count only the computation of moments is tractable.

Theorem 2. *For* PrXML^{cie} *the following complexity results hold for* sum *and* count*:*

1. *Membership is* NP*-complete.*
2. *Probability computation is in* $\mathsf{FP}^{\#\mathsf{P}}$.
3. *Moment computation is in* P *for moments of any degree.*

To prove the theorem we show three lemmas. The first lemma highlights why membership is difficult for PrXML^{cie} and all the aggregate functions of our interest.

Lemma 1 (Reducing Falsifiability to Membership)**.** *Let* $\boldsymbol{AGG} = \{\mathsf{sum}, \mathsf{count}, \mathsf{min}, \mathsf{countd}, \mathsf{avg}\}$*. For every propositional DNF formula* ϕ *one can compute in polynomial time a p-document* $\widehat{\mathcal{P}}_\phi \in \mathsf{PrXML}^{cie}$ *such that the following are equivalent: (1)* ϕ *is falsifiable, (2)* $\mathsf{Pr}(\alpha(\mathcal{P}) = 1) > 0$ *over* $\widehat{\mathcal{P}}_\phi$ *for* some $\alpha \in \boldsymbol{AGG}$*, (3)* $\mathsf{Pr}(\alpha(\mathcal{P}) = 1) > 0$ *over* $\widehat{\mathcal{P}}_\phi$ *for* all $\alpha \in \boldsymbol{AGG}$*.*

Proof. (Sketch) Let $\phi = \phi_1 \vee \cdots \vee \phi_n$, where each ϕ_i is a conjunction of literals. Then $\widehat{\mathcal{P}}_\phi$ has below its root a *cie*-node v with children $v_0, v_1, \ldots, v_n$, where each of $v_1, \ldots, v_n$ is labeled with the number $\frac{1}{2}$ and v_0 with 1. The edge to v_0 is labeled true while the edges to the v_i are labeled with ϕ_i. Clearly, 1 is a possible value for each of sum, count, min, countd and avg if and only if in some world, v_0 is the only child of v, that is, if and only if ϕ is falsifiable. □

The next lemma shows that computation of the expected value for sum over a px-space, regardless whether it can be represented by a p-document, can be polynomially reduced to computation of an auxiliary probability.

Lemma 2 (Regrouping Sums)**.** *Let* $\mathcal{S}$ *be a* px*-space and* V *be the set of all leaves occurring in the forests of* $\mathcal{S}$*. Suppose that the function* θ *labels all leaves in* V *with rational numbers and let* $\mathsf{sum}_\mathcal{S}$ *be the random variable defined by* sum *on* $\mathcal{S}$*. Then*

$$\mathbb{E}(\mathsf{sum}_\mathcal{S}{}^k) = \sum_{(v_1,\ldots,v_k)\in V^k} \Big(\prod_{i=1}^{k} \theta(v_i) \Big) \,\mathsf{Pr}\left(\{F \in \mathcal{S} \mid v_1, \ldots, v_k \text{ occur in } F\}\right),$$

where the last term denotes the probability that a random forest $F \in \mathcal{S}$ *contains all the nodes* $v_1, \ldots, v_k$*.*

Proof. (Sketch) Intuitively, the proof exploits the fact that $\mathbb{E}(\mathsf{sum}_\mathcal{S})$ is a sum over forests of sums over nodes, which can be rearranged as a sum over nodes of sums over forests. □

The auxiliary probability introduced in the previous lemma can be in fact computed in polynomial time for px-spaces represented by $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{cie}$.

Lemma 3 (Polynomial Probability Computation)**.** *There is a polynomial time algorithm that computes, given a p-document* $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{cie}$ *and leaves* $v_1, \ldots, v_k$ *occurring in* $\widehat{\mathcal{P}}$*, the probability*

$$\mathsf{Pr}\Big(\{F \in [\![\widehat{\mathcal{P}}]\!] \mid v_1, \ldots, v_k \text{ occur in } F\}\Big).$$

Proof. (Sketch) Let ϕ_i be the conjunction of all formulas that label the path from the root of $\widehat{\mathcal{P}}$ to v_i for $1 \leq i \leq k$. Obviously, $\mathsf{Pr}\Big(\{F \in [\![\widehat{\mathcal{P}}]\!] \mid v_1, \ldots, v_k \text{ occur in } F\}\Big) = \mathsf{Pr}(\phi_1 \wedge \cdots \wedge \phi_k)$. Note that each ϕ_i is a conjunction of literals and so is their conjunction ϕ. If some variable occurs both positively an negatively in ϕ, then ϕ is unsatisfiable and $\mathsf{Pr}(\phi) = 0$. Otherwise, we collect all variables that occur positively in a set X^+ and all variables that occur negatively in a set X^-. Then $\mathsf{Pr}(\phi) = \prod_{x \in X^+} \Delta(x) \times \prod_{x \in X^-} (1 - \Delta(x))$, where Δ is the probability distribution over the event variables of $\widehat{\mathcal{P}}$.

Clearly, the variable sets X^+ and X^- can be computed by traversing the forest underlying $\widehat{\mathcal{P}}$. By marking nodes visited, the traversal will look at every edge of the forest at most once, which yields the claim. □

Now we are ready to prove the theorem.

Proof of Theorem 2. 1. $\mathsf{Pr}(\mathsf{sum}(\mathcal{P}) = 1) > 0$ or $\mathsf{Pr}(\mathsf{count}(\mathcal{P}) = 1) > 0$ for a given $\widehat{\mathcal{P}}$ can be checked by guessing an assignment for the event variables of $\widehat{\mathcal{P}}$ and performing the aggregation over the resulting document. Hardness follows from Lemma 1, since falsifiability of DNF-formulas is NP-complete [10].

2. Follows from Theorem 1 and the fact that sum and count is scalable.

3. By Lemma 2, the k-th moment of sum over $\widehat{\mathcal{P}}$ is the sum of $|V|^k$ products, where V is the set of leaves of $\widehat{\mathcal{P}}$. The first term of each product, $\prod_{i=1}^{k} \theta(v_i)$, can be computed in time at most $|\widehat{\mathcal{P}}|^k$. By Lemma 3, the second term can be computed in polynomial time. This shows that for every $k \geq 1$, the k-th moment of sum can be computed in polynomial time. The claim for count follows as a special case, where all leaves carry the label 1. □

3.2 Computing min, avg, and countd

The following theorem shows that nothing is tractable for min, avg and countd.

Theorem 3. *For* PrXML^{cie} *the following complexity results hold for* min, avg *and* countd*:*

1. *Membership is* NP*-complete.*
2. *Probability computation is* $\mathsf{FP}^{\#\mathsf{P}}$*-complete.*
3. *Moment computation is* $\mathsf{FP}^{\#\mathsf{P}}$*-complete for moments of any degree.*

To prove the theorem we use the next lemma that highlights why membership is difficult. Recall that #DNF is the problem of counting all satisfying assignments of a propositional DNF-formula.

Lemma 4 (Reducing #DNF to Probability and Expected Value Computation)**.** *For every propositional DNF formula ϕ with n variables one can compute in polynomial time p-documents $\widehat{\mathcal{P}}_\phi \in \mathsf{PrXML}^{cie}$ and $\widehat{\mathcal{Q}}_\phi \in \mathsf{PrXML}^{cie}$ such that the following are equivalent: (1) ϕ has m satisfying assignments. (2) $\mathsf{Pr}(\mathsf{min}(\mathcal{P}_\phi) = 1)$ is $m/2^n$. (3) $\mathbb{E}(\mathsf{min}(\mathcal{Q}_\phi))$ is $1 - m/2^n$.*

Proof. (Sketch) Let $\phi = \phi_1 \vee \cdots \vee \phi_n$ be in DNF with n variables. Consider $\widehat{\mathcal{P}}_\phi$ that has below its root a *cie*-node v with children $v_0, v_1, \ldots, v_n$, where each of $v_1, \ldots, v_n$ is labeled with the number 1 and v_0 with 2. The edge to v_0 is labeled with true while the edges to the v_i are labeled with ϕ_i. To show (1) $\sim$ (2) observe that (by construction) probability of every $\mathcal{P} \in [\![\widehat{\mathcal{P}}]\!]$ is $1/2^n$. Moreover, $\mathsf{min}(\mathcal{P}) = 1$ holds if an only if $\mathcal{P}$ has at least two leaves: v_0 and any other from v_is. Hence, $\mathcal{P}$ is obtained by an assignment of the event variables that satisfies ϕ and $\mathsf{Pr}(\mathsf{min}(\mathcal{P}) = 1)$ is the sum over all such assignments multiplied by the probability of the resulting $\mathcal{P}$, that is $1/2^n$. To show (1) $\sim$ (3) consider $\widehat{\mathcal{Q}}_\phi$, a modification of $\widehat{\mathcal{P}}_\phi$ where the node v_0 is labeled with 1 and all the other v_is with 0, and apply similar reasoning. □

Remark 1. Results of Lemma 4 can be extended to reduction of #DNF to probability and expected value computation for countd and avg.

Now we are ready to prove the theorem.

Proof of Theorem 3. 1. The proof is exactly the same as of the first claim of Theorem 2.

2. Hardness follows from Lemma 4 and Remark 1, since #DNF is #P-complete [10]. Membership follows from Theorem 1 and the fact that min, avg and countd are scalable.

3. Hardness for the first moment, as in the previous part of the theorem, follows from Lemma 4 and this implies hardness for moments of any degree higher than 1. Membership can be shown as in the previous case of theorem. □

3.3 Approximate Computation

As we see earlier in this section, for several aggregate functions on PrXML^{cie} p-documents, probability computation and moment computation are hard. Fortunately, there are general sampling techniques which give randomized approximation algorithms for tackling intractability of computing the above quantities. In the following we present how to estimate cumulative distributions $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x)$ and moments $\mathbb{E}(\alpha(\mathcal{P})^k)$. We notice that from the cumulative distribution $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x)$ one can estimate the distribution $\mathsf{Pr}(\alpha(\mathcal{P}) = x)$ by computing $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x - \gamma)$ and $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x + \gamma)$ for some small γ (that depends on α) and taking the difference of them.

For instance, suppose we wish to consider the aggregate function countd on a p-document $\widehat{\mathcal{P}}$. In particular, say we are interested in approximating the probability $\mathsf{Pr}(\mathsf{countd}(\mathcal{P}) \leq 100)$. This probability can be estimated by drawing independent random samples of the document, and using the ratio of samples for which countd is at most 100 as an estimator. Similarly, if we wish to approximate $\mathbb{E}(\mathsf{countd}(\mathcal{P}))$, we can draw independent random samples and return the the average of countd on the drawn samples.

The first important question is: is it possible at all to have a reasonably small number of samples to get a good estimation? It would not be helpful if an enormous number of samples is necessary. The good news is that the answer to the above question is “yes”. The second question is: how many samples do we need? The following classical result from the probability literature helps us to answer both questions.

Proposition 1 (Hoeffding Bound [13]). *Suppose $\{U_1, U_2, \ldots, U_T\}$ are T independent identically distributed random variables , each of which takes values in an interval of width R and has mean μ. Let $\bar{U} := \frac{1}{T}\sum_{i=1}^{T} U_i$ be the empirical average. Then, for each $\epsilon > 0$,*

$$\mathsf{Pr}(|\bar{U} - \mu| \geq \epsilon) \leq 2 \times \exp\left(-\frac{2\epsilon^2 T}{R^2}\right).$$

Approximating Distribution Points. Suppose α is an aggregation function on some PrXML^{cie} p-document $\widehat{\mathcal{P}}$, and we wish to approximate the probability $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x)$ for some value x. We sample instances of the p-document, and for each sample, let X_i be the corresponding value of the aggregation function. We let U_i to be the Bernoulli variable that takes value 1 when $X_i \leq x$, and 0 otherwise. Then, it follows that $\mathbb{E}(U_i) = \mathsf{Pr}(\alpha(\mathcal{P}) \leq x)$, and $\bar{U} := \frac{1}{T}\sum_{i=1}^{T} U_i$ is an estimate of $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x)$. Hence, we immediately obtain the following result for approximating a point for the cumulative distribution of an aggregation function.

Corollary 1. *For any aggregation function α, p-document $\widehat{\mathcal{P}} \in$ PrXML^{cie}, any value x, and for any $\epsilon, \delta > 0$, it is sufficient to have $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ samples so that with probability at least $1 - \delta$, the quantity $\mathsf{Pr}(\alpha(\mathcal{P}) \leq x)$ can be estimated with an additive error of ϵ.*

Observe that the number of samples in Corollary 1 is independent of the instance size. The only catch is that if the probability that we are trying to estimate is less than ϵ, then an additive error of ϵ would render the estimate useless. Hence, if we only care about probabilities above some threshold p_0, then it is enough to have the number of samples proportional to $1/p_0^2$ (with additive error, say $p_0/10$).

Approximating Moments. Suppose f is some function on an aggregation function α, and we are interested in computing $\mathbb{E}(f(\alpha(\mathcal{P})))$. For each sample, we let $U_i := f(X_i)$, and compute the estimator $\bar{U} := \frac{1}{T}\sum_{i=1}^{T} U_i$.

Corollary 2. *Let $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{cie}$ be a p-document and f be a function on an aggregation function α such that $f(\alpha(\mathcal{P}))$ takes value in an interval of width R. Then, for any $\epsilon, \delta > 0$, it is sufficient to have $O(\frac{R^2}{\epsilon^2}\log\frac{1}{\delta})$ samples so that, with probability at least $1-\delta$, the quantity $\mathbb{E}(f(\alpha(\mathcal{P})))$ can be estimated with additive error of ϵ. In particular, if α takes value in $[0,R]$ and $f(x) := \alpha(\mathcal{P})^k$, then the k-th moment of $\alpha(\mathcal{P})$ around zero can be estimated with $O(\frac{R^{2k}}{\epsilon^2}\log\frac{1}{\delta})$ samples.*

Observe that if the range R has magnitude polynomial in the problem size, then we have a polynomial-time algorithm. In our example of approximation $\mathbb{E}(\mathsf{countd}(\widehat{\mathcal{P}}))$, the range R can be at most the size of the problem instance. Hence, to estimate the expectation, it is enough to draw a quadratic number of random samples, that gives a polynomial time approximation algorithm.

Remark 2. Note that the results we presented are about the *additive error* of an ϵ-approximation, say $\widehat{Q}$, of a value, say Q, that is, $|Q-\widehat{Q}| \le \epsilon$. In [11] the *multiplicative error*, defined as $|\frac{Q-\widehat{Q}}{Q}| \le \epsilon$, is studied, and fully polynomial-time randomized approximation schemes (FPTRAS) give means for computing approximation based on this notion. Both errors are related, since $|Q-\widehat{Q}| \le \epsilon$ implies $Q \ge \widehat{Q}-\epsilon$ we have that the multiplicative error is at most $|\frac{Q-\widehat{Q}}{Q}| \le \frac{\epsilon}{\widehat{Q}-\epsilon}$. Hence, to obtain a certain multiplicative error, one simply needs to get an estimate with some initial additive error (say 1). If the resulting multiplicative error is too large, then one decreases the additive error (say by half), and obtain another estimate, until the multiplicative error is small enough.

4 Distributions of Monoid Aggregates Over $\mathsf{PrXML}^{mux,det}$

We show that any px-space $\mathcal{S}$ that admits a description by a p-document $\widehat{\mathcal{P}}$ in $\mathsf{PrXML}^{mux,det}$ can be constructed in a bottom-up fashion from elementary spaces $\mathcal{S}_e$ by means of three operations: *rooting*, *convex union*, and *product*. This construction reflects the structure of $\widehat{\mathcal{P}}$. Then we show that the distribution $\alpha(\mathcal{S})$ of a monoid aggregate function α on such a space $\mathcal{S}$ can be computed from $\alpha(\mathcal{S}_e)$'s, the distributions over elementary spaces, by means of the three operations *identity*, *convex sum* and *convolution*, which correspond to rooting, convex union and product, respectively. This construction gives a bottom-up algorithm to compute distributions $\alpha(\widehat{\mathcal{P}})$ on p-documents $\widehat{\mathcal{P}}$.

4.1 Algebra of px-Spaces

Operations on Sets of Forests. We first introduce three operations on sets of forests: union, product, and rooting. Let $\mathcal{F}_1, \dots, \mathcal{F}_n$ be sets of forests. All three operations are only applied to sets of forests that are mutually *node-disjoint*, that is, where $\mathcal{V}(\mathcal{F}_i) \cap \mathcal{V}(\mathcal{F}_j) = \emptyset$ for $i \ne j$.

The *union* of such set of forests, denoted as $\mathcal{F}_1 \uplus \cdots \uplus \mathcal{F}_n$, is the union of disjoint sets. Since the arguments are node-disjoint, they are mutually disjoint and for every forest F in the union there is a unique $\mathcal{F}_i$ such that $F \in \mathcal{F}_i$.

The *product*, denoted with the operator "$\otimes$", is defined as

$$\mathcal{F}_1 \otimes \cdots \otimes \mathcal{F}_n := \{F_1 \cup \cdots \cup F_n \mid F_i \in \mathcal{F}_i\}.$$

Similar to a Cartesian product, this product consists of all possible combinations of elements of the arguments. Since the arguments are node disjoint, each forest in the product can be uniquely decomposed into its components $F_1, \ldots, F_n$.

Rooting is a unary operator. We define it first for individual forests. Let v be a node, l a label and F a forest without v. The *rooting of F under v with l*, denoted as $\mathsf{rt}_v^l(F)$, is the document d obtained by combining all documents d' in F under the new root v. Formally,

$$\mathsf{root}(d) := v,\ \theta(\mathsf{root}(d)) := l,\ \mathcal{V}(d) := \mathcal{V}(F) \cup \{v\},\ \mathcal{E}(d) := \mathcal{E}(F) \cup \{(v, \mathsf{root}(d')) \mid d' \in F\}.$$

Rooting is lifted to sets of forests by defining $\mathsf{rt}_v^l(\mathcal{F}) := \{\mathsf{rt}_v^l(F) \mid F \in \mathcal{F}\}$ where $\mathcal{F}$ is such that v does not occur in any $F \in \mathcal{F}$.

Operations on px-spaces. Next we extend the above definitions to px-spaces $\mathcal{S}_1, \ldots, \mathcal{S}_n$, where $\mathcal{S}_i = (\mathcal{F}_i, \mathsf{Pr}_i)$. Our operations are only defined for px-spaces over node-disjoint sets of forests.

Let $p_1, \ldots, p_n$ be nonnegative rational numbers such that $\sum_{i=1}^n p_i = 1$. We call such numbers *convex coefficients*. The *convex union of the $\mathcal{S}_i$ wrt the p_i* is the px-space $(\mathcal{F}, \mathsf{Pr})$ where $\mathcal{F} = \mathcal{F}_1 \uplus \cdots \uplus \mathcal{F}_n$ and for every $F \in \mathcal{F}$ we define $\mathsf{Pr}(F) := p_i \mathsf{Pr}_i(F)$ where i is the unique index such that $F \in \mathcal{F}_i$. Intuitively, this means that the probability of an elementary event in the convex union is its original probability multiplied by the corresponding coefficient. The convex union of the $\mathcal{S}_i$ wrt the p_i is denoted as $p_1\mathcal{S}_1 \uplus \cdots \uplus p_n\mathcal{S}_n$.

The *product of the $\mathcal{S}_i$* is the px-space $(\mathcal{F}, \mathsf{Pr})$ where $\mathcal{F} = \mathcal{F}_1 \otimes \cdots \otimes \mathcal{F}_n$ and for every $F \in \mathcal{F}$ we define $\mathsf{Pr}(F) := \mathsf{Pr}_1(F_1) \times \cdots \times \mathsf{Pr}_n(F_n)$ where $F_i \in \mathcal{F}_i$ are the unique forests such that $F = F_1 \cup \cdots \cup F_n$. Intuitively, the elementary events of the product are combinations of elementary events in the original spaces and the probability of such a combination is the product of the probabilities of its components. The product of the $\mathcal{S}_i$ is denoted as $\mathcal{S}_1 \otimes \cdots \otimes \mathcal{S}_n$.

Let $\mathcal{S}_0 = (\mathcal{F}_0, \mathsf{Pr}_0)$ be a px-space and v be a node with label l such that v does not occur in any forest in $\mathcal{F}_0$. The rooting of $\mathcal{S}_0$ under v with l is the px-space $(\mathcal{F}, \mathrm{Pr})$ where $\mathcal{F} = \mathsf{rt}_v(\mathcal{F}_0)$ and for every element $\mathsf{rt}_v F \in \mathcal{F}$ we define $\mathsf{Pr}(\mathsf{rt}_v(F)) := \mathsf{Pr}_0(F)$. That is, rooting does not change probabilities. The rooting of a space $\mathcal{S}$ under a node v with label l is denoted as $\mathsf{rt}_v^l(\mathcal{S})$.

The Algebra of px-Expressions. Whenever we are given a collection of px-spaces, we can construct more complex ones using the three operations above.

A document is *elementary* if it consists of a single node. The elementary document whose only node is v, carrying the label l, is denoted as d_v^l. A px-space $(\mathcal{F}, \mathsf{Pr})$ is *elementary* if $\mathcal{F}$ contains exactly one elementary document, say d. Since Pr is a probability, it follows that $\mathsf{Pr}(d) = 1$. By abuse of notation, we denote also such a space as d_v^l if no confusion can arise.

To describe the spaces that can be constructed in this way, we introduce expressions that are composed according to the rule

$$E, E_1, \ldots, E_n \quad \to \quad d_v^l \mid (p_1E_1 \uplus \cdots \uplus p_nE_n) \mid (E_1 \otimes \cdots \otimes E_n) \mid \mathsf{rt}_v^l(E),$$

where v ranges over all node identifiers, l over all labels and $p_1, \ldots, p_n$ over all sequences of convex coefficients. An expression is a *px-expression* if any node-identifier is unique.

Every px-expression can be evaluated to yield a unique px-space. This is clearly the case for elementary expressions d_v^l. Moreover, due to the condition on node-indentifiers, convex union and product are always applied to arguments that are node disjoint. A similar argument applies to rooting. We refer to the px-space denoted by E as $[\![E]\!]$.

Example 5. The px-expression

$$E = \mathsf{rt}^{\mathrm{pda}}_{n_{51}}(0.7\,(d^{15}_{n_{54}} \otimes d^{44}_{n_{55}}) \uplus 0.3\, d^{15}_{n_{56}})$$

denotes a px-space containing two documents d_1 and d_2, where $\mathsf{Pr}(d_1) = 0.7$ and $\mathsf{Pr}(d_2) = 0.3$. The root of both is node n_{51}, which has two children in d_1, namely nodes n_{54} and n_{55}, and a single child in d_2, namely n_{56}.

Expressivity of the px-Algebra. In the following we show that the px-spaces representable by px-expressions are exactly those that are definable by p-documents of the class $\mathsf{PrXML}^{mux,det}$.

Since rooting is a unary operation in our algebra, we are only able to map px-expressions to a slightly restricted subclass of $\mathsf{PrXML}^{mux,det}$, which has the same expressivity as the full class. A p-document is *normalized* if each ordinary node has at most one child. We can transform any p-document into an equivalent normalized one if we place below each ordinary node v that is not a leaf a new *det*-node, say v', and turn all children of v into children of v'.

Proposition 2. *Let E be a px-expression and $\widehat{\mathcal{P}}$ be a normalized p-document.*

1. *There exists a normalized p-document $\widehat{\mathcal{P}}_E$ such that $[\![\widehat{\mathcal{P}}_E]\!] = [\![E]\!]$.*
2. *There exists a px-expression $E_{\widehat{\mathcal{P}}}$ such that $[\![E_{\widehat{\mathcal{P}}}]\!] = [\![\widehat{\mathcal{P}}]\!]$.*

Proof. (Sketch) From E, a p-document $\widehat{\mathcal{P}}_E$ can be recursively constructed by creating for every convex union expression in E a *mux*-node, for every product expression a *det*-node and for every rooting expression an ordinary node. The resulting document is normalized because rooting is a unary operation.

For the construction of $E_{\widehat{\mathcal{P}}}$ from $\widehat{\mathcal{P}}$ one proceeds analogously. Since $\widehat{\mathcal{P}}$ is normalized, ordinary nodes can be mapped to the rooting operator.

A straightforward comparison of the semantics of p-documents in Section 2 and the semantics of px-expressions shows that $[\![\widehat{\mathcal{P}}_E]\!] = [\![E]\!]$ and $[\![E_{\widehat{\mathcal{P}}}]\!] = [\![\widehat{\mathcal{P}}]\!]$. □

Example 6. A p-document $\widehat{\mathcal{P}}_E$ corresponding to the expression E in Example 5 is the subtree with root n_{51} of $\widehat{\mathcal{P}}_2$ in Figure 3.

4.2 Distributions of Monoid Aggregate Functions

Let α be an aggregate function that maps bags of elements of X to values in Y and let $\mathcal{S} = (\mathcal{F}, \mathsf{Pr})$ be a px-space. The application of α to $\mathcal{S}$ yields as a result the probability distribution $\alpha(\mathcal{S})$ over Y that satisfies $\alpha(\mathcal{S})(y) = \mathsf{Pr}(\{F \in \mathcal{F} \mid \alpha(F) = y\})$. We will study how $\alpha(\mathcal{S})$ depends on the structure of $\mathcal{S}$ if the latter is defined by a px-expression. In particular, we will show that we can evaluate monoid aggregate functions α on $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{mux,det}$ by, first, computing α on the elementary px-spaces corresponding to the leaves of $\widehat{\mathcal{P}}$ and then combining the resulting distributions iteratively according to the structure of $\widehat{\mathcal{P}}$ in a bottom-up fashion.

We use the letter π as the generic notation for probability distributions. All the distributions we consider in this paper are finite, that is, if π is a distribution over Y, then there are finitely many elements $y_1, \ldots, y_n \in Y$ such that $p_i := \pi(\{y_i\}) \neq 0$ and $\sum_{i=1}^n p_i = 1$.

For any $y \in Y$ there is a probability distribution δ_y such that $\delta_y(z) = 1$ if $y = z$ and 0 otherwise. δ_y is the distribution of a Y-valued random variable if and only if that variable returns with probability 1 the value y.

If $p_1, \ldots, p_n$ are convex coefficients and $\pi_1, \ldots, \pi_n$ are finite probability distributions over Y, then the *convex sum* of $\pi_1, \ldots, \pi_n$ wrt $p_1, \ldots, p_n$, written as $p_1\pi_1 + \cdots + p_n\pi_n$, maps every $y \in Y$ to $p_1\pi_1(y) + \cdots + p_n\pi_n(y)$. The convex sum is a finite probability distribution over Y.

Proposition 3. *Let α be an aggregate function, $\mathcal{S}, \mathcal{S}_1, \dots, \mathcal{S}_n$ node-disjoint* px*-spaces, $p_1, \dots, p_n$ convex coefficients, v a node identifier not occurring in $\mathcal{S}$ and l a label. Then*

1. $\alpha(d_v^l) = \delta_{\alpha(\{\!|l|\!\})}$
2. $\alpha(p_1\mathcal{S}_1 \uplus \cdots \uplus p_n\mathcal{S}_n) = p_1\alpha(\mathcal{S}_1) + \cdots + p_n\alpha(\mathcal{S}_n)$
3. $\alpha(\mathsf{rt}_v^l(\mathcal{S})) = \alpha(\mathcal{S})$.

Proof. The first claim holds because over d_v^l the function α returns with probability 1 the value $\alpha(\{\!|l|\!\})$. To see the second claim, suppose that the probability for α to return y over $\mathcal{S}_i$ is q_i for $i = 1, \dots, n$. Then, by definition, the probability for α to return y over the convex union is $p_1q_1 + \cdots + p_nq_n$. The third claim holds because rooting does not change probabilities. □

With convex union and rooting alone, one can only construct trivial px-spaces. Unfortunately, not much can be said in general about the relationship between α applied to a product space $\mathcal{S}$ and α applied to the arguments $\mathcal{S}_i$ of the product.

However, the situation is much better if α is a monoid aggregate function. An element F of $\mathcal{S}$ is a forest that is the union of node-disjoint forests F_i in $\mathcal{S}_i$ and the value $\alpha(F)$ can be obtained from the components F_i of F, since $\alpha(F) = \alpha(F_1 \uplus \cdots \uplus F_n) = \bigoplus_{i=1}^n \alpha(F_i)$.

Distributions on a monoid can be combined by an additional operation. For any two finite distributions π_1, π_2 over a monoid M with binary operation "$\oplus$", the *convolution* is the distribution over M defined by

$$(\pi_1 * \pi_2)(m) = \sum_{m_1, m_2 \in M\colon\, m = m_1 \oplus m_2} \pi_1(m_1)\pi_2(m_2).$$

Note that the convolution not only depends on the set M, but also on the monoid operation "$\oplus$". It is straightforward to see that convolution is an associative and commutative operation on the probability distributions over M and that the neutral element is the distribution $\delta_\perp$, where $\perp$ is the neutral element of M. Therefore, the notation $\pi_1 * \cdots * \pi_n$ is unambiguous for all $n \geq 0$.

Proposition 4. *Let α be a monoid aggregate function and $\mathcal{S}_1, \dots, \mathcal{S}_n$ be node-disjoint* px*-spaces. Then $\alpha(\mathcal{S}_1 \otimes \cdots \otimes \mathcal{S}_n) = \alpha(\mathcal{S}_1) * \cdots * \alpha(\mathcal{S}_n)$.*

For every monoid aggregate function α with range M, we introduce a mapping "$\alpha(\cdot)$" that maps any px-expression E to a probability distribution $\alpha(E)$ over M. The mapping is defined recursively as (i) $\alpha(d_v^l) = \delta_{\alpha(\{\!|l|\!\})}$, (ii) $\alpha(p_1E_1 \uplus \cdots \uplus p_nE_n) = p_1\alpha(E_1) + \cdots + p_n\alpha(E_n)$, (iii) $\alpha(\mathsf{rt}_v^l(E)) = \alpha(E)$, and (iv) $\alpha(E_1 \otimes \cdots \otimes D_n) = \alpha(E_1) * \cdots * \alpha(E_n)$.

The following theorem says that the computation of distributions from expressions actually reflects the semantics of expressions.

Theorem 4. *Let α be a monoid aggregate function and E be a* px*-expression. Then*

$$\alpha(E) = \alpha([\![E]\!]).$$

Proof. The claim holds clearly for elementary expressions. If follows for arbitrary expressions by an inductive argument using Propositions 3 and 4. □

Complexity Analysis. Let $\mathcal{S}_1$, $\mathcal{S}_2$ be px-spaces and α be a monoid aggregate function. We say that α is *monotone* (with respect to product) if the following holds on the sizes of the distributions:

$$|\alpha(\mathcal{S}_1)| + |\alpha(\mathcal{S}_2)| \leq |\alpha(\mathcal{S}_1 \otimes \mathcal{S}_2)|.$$

It is easy to see that min, count, sum are monotone aggregate functions.

Theorem 5. *Let α be a monotone monoid aggregate function. Then for any $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{mux,det}$ the distribution $\alpha(\widehat{\mathcal{P}})$ can be computed in time polynomial in $|\alpha(\widehat{\mathcal{P}})|$.*

Thus, for monotone aggregate functions the time for computing the answer distribution over a $\mathsf{PrXML}^{mux,det}$ document is polynomial in the size of the *output*.

5 Complexity of Aggregates Over $\mathsf{PrXML}^{mux,det}$

We now study the complexity of the four problems highlighted in Section 2 for the class of p-documents $\mathsf{PrXML}^{mux,det}$. Recall that this class is strictly less expressive than PrXML^{cie} and query answering is essentially easier for it: there is a complexity jump from #P-complete for PrXML^{cie} to P for $\mathsf{PrXML}^{mux,det}$ [18]. We show that for aggregate functions $\mathsf{PrXML}^{mux,det}$ is still easier than PrXML^{cie}, but the complexity gap is not as clear as in the case of queries.

For the monoid aggregate functions count and min, we show that probability distributions can be computed in polynomial time. For sum they can be computed in time polynomial in the size of the output. It means there is the same complexity jump in aggregation as in query answering. For the *non*-monoid aggregate functions countd and avg we show that distribution computation is $\mathsf{FP}^{\#\mathsf{P}}$-complete. It means $\mathsf{PrXML}^{mux,det}$ and PrXML^{cie} are equivalently difficult for aggregation with countd and avg. The results exhibit a complexity *borderline* between monoid and non monoid aggregate functions for $\mathsf{PrXML}^{mux,det}$ p-documents.

5.1 Computing min and count

For min and count all four problems can be solved in polynomial time.

Proposition 5. *For $\mathsf{PrXML}^{mux,det}$, distribution computation is in P for both* min*,* count*.*

Proof. Let V be the set of all the leaves of a given $\widehat{\mathcal{P}}$, and L be the set of V's labels. Since all forests $F \in [\![\widehat{\mathcal{P}}]\!]$ have nodes among $\mathcal{V}(\widehat{\mathcal{P}})$, the values $\mathsf{min}(F)$ are in L. Therefore, the carrier of the distribution $\mathsf{min}(\widehat{\mathcal{P}})$ is a subset of L. Analogously, the carrier of the distribution $\mathsf{count}(\widehat{\mathcal{P}})$ is a subset of $\{1, \ldots, |V|\}$. Therefore, $|\mathsf{min}(\widehat{\mathcal{P}})|$ and $|\mathsf{count}(\widehat{\mathcal{P}})|$ are limited by $|\widehat{\mathcal{P}}|$. From this fact and Theorem 5 the claim of the proposition follows. □

Using the previous and the following propositions, we conclude that all four problems for min and count are polynomial.

Proposition 6. *Given a distribution of an aggregate function, one can decide membership and compute probability and moments in polynomial time in the size of the distribution.*

5.2 Computing sum

We show that computing moments for sum is polynomial, and the other problems are easy only if the size of the distribution $\mathsf{sum}(\widehat{\mathcal{P}})$ is small.

Theorem 6. *For $\mathsf{PrXML}^{mux,det}$ the following complexity results hold for* sum*:*

1. *Membership is* NP*-complete.*
2. *Distribution computation is in polynomial time in the size of the output.*
3. *Distribution computation can be done in exponential time and there is a p-document for which the distribution is exponentially large in the size of the p-document.*
4. *Moment computation is in* P *for moments of any degree.*

The next lemma highlights why membership is difficult. The Subset-Sum problem is, given a finite set A, an $\mathbb{N}$-valued function s on A, and $c \in \mathbb{N}$, is to decide whether there is some $A' \subseteq A$ such that $\sum_{a \in A'} s(a) = c$.

Lemma 5 (Reducing Subset-Sum to Membership)**.** *For every set A, $\mathbb{N}$-valued (weight) function s on A, and $c \in \mathbb{N}$ there is a p-document $\widehat{\mathcal{P}}_{A,s,c} \in \mathsf{PrXML}^{mux,det}$ such that the following are equivalent: (1) there is $A' \subseteq A$: $\sum_{a \in A'} s(a) = c$, (2) $\mathsf{Pr}(\mathsf{sum}(\mathcal{P}_{A,s,c}) = c) > 0$.*

Proof. (Sketch) Let $A := \{a_1, \ldots, a_m\}$. Then $\widehat{\mathcal{P}}_{A,s,c}$ has a root r with m children $v_1, \ldots, v_m$ that are *mux*-nodes. Each v_i has one child a_i labeled with the number $s(a_i)$. The edges to the a_i are labeled with $1/2$. Clearly, c is a possible value for sum if and only if in some world the only children of r are a_i's with labels summing-up to c, that is, if and only if there is $A' \subseteq A$ consisting of these a_i's with the weights summing-up to c. $\square$

Now we are ready to prove the theorem.

Proof of Theorem 6. 1. Membership can be checked by guessing a child for every *mux* node of $\widehat{\mathcal{P}}$ and performing the aggregation over the resulting document. Hardness follows from Lemma 5 because Subset-Sum is NP-complete [10].

2. Follows from monotonicity of sum and Theorem 5.

3. The claim holds since there are at most exponentially many forests in the semantics of p-documents. Consider $\widehat{\mathcal{P}}$ that has a root r with one *det* child that in turn has n *mux*-children each i-th of which has one child v_i labeled with the number 2^i, and the edges to v_i are labeled with $1/2$. Then $|\mathsf{sum}(\widehat{\mathcal{P}})| = 2^n$.

4. Follows from Lemma 2 and the fact that one can compute the probability $\mathsf{Pr}(\{F \in \mathcal{S} \mid v_1, \ldots, v_k \text{ occur in } F\})$ that $\mathcal{P}$ contains all the nodes $v_1, \ldots, v_k$ in polynomial time. This probability is 0 if there is a common *mux* ancestor of any two nodes from $v_1, \ldots, v_k$. Otherwise, one takes a subtree $\widehat{\mathcal{P}}_{v_1,\ldots,v_k}$ of $\widehat{\mathcal{P}}$ with the root r and the leaves $v_1, \ldots, v_k$, and the probability is the product of all probabilities that mark the edges of $\widehat{\mathcal{P}}_{v_1,\ldots,v_k}$. $\square$

5.3 Computing countd and avg

Now now see that for non-monoid aggregates all but moment computation problems are hard.

Theorem 7. *For $\mathsf{PrXML}^{mux,det}$ the following complexity results hold for countd and avg:*
1. *Membership is in NP.*
2. *Probability computation is $\mathsf{FP}^{\#\mathsf{P}}$-complete.*
3. *Moment computation is in P for moments of any degree.*

We present next a lemma that highlights why probability computation is difficult for countd. Recall that the #K-Cover problem is, given a finite set A, a set $\mathcal{A}$ of subsets of A and a constant $k \in \mathbb{N}$, to *count* the number of different combinations $\mathcal{A}' \subseteq \mathcal{A}$ of elements from $\mathcal{A}$ that have the size k and cover A, that is, $A = \bigcup_{S \in \mathcal{A}'} S$.

Lemma 6 (Reducing #K-Cover to Probability Computation)**.** *For every set A, set $\mathcal{A}$ of subsets of A and $k \in \mathbb{N}$ one can compute in polynomial time a p-document $\widehat{\mathcal{P}}_{A,\mathcal{A},k} \in \mathsf{PrXML}^{mux,det}$ and a constant $c_{A,\mathcal{A},k}$ such that the following are equivalent: (1) the number of $\mathcal{A}' \subseteq \mathcal{A}$ s.t. $|\mathcal{A}'| = k$ and $A = \bigcup_{S \in \mathcal{A}'} S$ is m, (2) $\mathsf{Pr}(\mathsf{countd}(\mathcal{P}_{A,B,k}) = k)$ is $m \times c_{A,\mathcal{A},k}$.*

Proof. (Sketch) Let $A = \{a_1, \ldots, a_n\}$, and $\mathcal{A} = \{S_1, \ldots, S_u\}$, let f be a $\mathbb{N}$-valued injective function on A. Let D be a set of $k+1$ natural numbers disjoint from $\{f(S_1), \ldots, f(S_u)\}$. Assume for each i that $S^i_1 \ldots, S^i_{t_i}$ are all the sets containing a_i and t is the maximum among t_i's.

We construct $\mathcal{P}_{A,B,k}$ as a p-document, having below its root n children $a_1, \ldots, a_n$. Each a_i has a *det* child that has t children, namely $S^i_1 \ldots, S^i_{t_i}$ and $D^i_1, \ldots, D^i_{(t-t_i)}$. The edges to S^i_js and D^i_js are all labeled $1/t$. Each S^i_j is a leaf labeled with $f(S^i_j)$ and each D^i_j has $k+1$ children each labeled with a distinct number from D. Nodes D^i_j are needed to guarantee the probability of every world of $\widehat{\mathcal{P}}_{A,B,k}$ to be the same. Intuition behind nodes S^i_j is they indicate that an element a_i in A is "covered" with a set S^i_j in $\mathcal{A}'$.

One can see that $\mathsf{countd}(\mathcal{P}_{A,B,k}) = k$ if and only if $\mathcal{P}_{A,B,k}$ has no nodes D^i_j and k different values $f(S^i_j)$ on its leaves, that is, exactly k elements of $\mathcal{A}$ are chosen that "cover" A. Hence, $\mathsf{Pr}(\mathsf{countd}(\mathcal{P}_{A,B,k}) = k) = m \times p$, where p is the probability of every world of $\widehat{\mathcal{P}}_{A,B,k}$. □

We are ready to prove the theorem.

Proof of Theorem 7. 1. Can be shown as in Theorem 6.

2. Hardness for countd follows from Lemma 6 because #K-Cover is #P-complete [25]. Hardness for avg can be shown by a straightforward reduction from the #P-complete problem #Non-Negative-Subset-Average [25]. Membership follows from the $\mathsf{FP}^{\#\mathsf{P}}$-membership of the corresponding problems for more general class PrXML^{cie}.

3. The proof is more involved, we omit it due to lack of space. □

6 Conclusion

The literature about probabilistic relational databases is quite extensive. One of the early and seminal works is [4] where probabilities are associated with attribute values. Among the number of models and systems that have been proposed for representing and querying probabilistic data, one can distinguish between not fully expressive systems such as the block-independent model [9] (which can be seen as a relational counterpart to $\mathsf{PrXML}^{mux,det}$) and the more complex, lineage-oriented, probabilistic database management systems like Trio [28] and MayBMS [20], that are closer in spirit to PrXML^{cie}. The latter are inspired by Imieliński and Lipski's c-tables [16] (though these are models for incomplete information, they can be applied to probabilistic information in a straightforward way, as noted in [12]).

By contrast, research on probabilistic XML is a newer topic. P-documents have been proposed recently [1, 17] as a generalization of previously developed models [2, 14, 15, 22, 26, 27]. Among these, the one from [22] is essentially $\mathsf{PrXML}^{mux,det}$ while [2, 26] is PrXML^{cie}. Comparison between the expressiveness of all these models is the topic of [1] and [18] investigates query processing.

Only a few works have considered aggregate queries in a setting of incomplete data. In non-probabilistic settings aggregate queries were studied for conditional tables [21], for data exchange [3] and for ontologies [5]. In probabilistic settings, to the best of our knowledge, only two works [6, 25] study aggregate queries. Ré and Suciu [25] consider the problem of evaluating `HAVING` queries (using aggregate functions) in block-independent databases. Cohen, Kimelfeld, and Sagiv [6] deal with query answering when external constraints are imposed over a $\mathsf{PrXML}^{mux,det}$ database, these constraints involving aggregate functions. In both cases, the authors discuss the filtering of possible words that do not satisfy a condition expressed using aggregate functions, and neither on computing the distribution of the aggregation. Some connections exist to these works, however. In particular, the complexity bounds obtained for probability computation of avg in Section 5.3 are a direct translation of the corresponding results for block-independent databases presented in [25].

In addition to providing algorithms for, and a full characterization of, the complexity of computing the aggregation for both $\mathsf{PrXML}^{mux,det}$ and PrXML^{cie} models (which corresponds to the most studied and interesting probabilistic XML models), an originality of our contribution is to consider the expected value or other moments as summaries of the probability distribution of an aggregate function. In the case of $\mathsf{PrXML}^{mux,det}$, we have identified a specific property of aggregate functions (that of being *monoid* aggregate functions) that entails tractability. The intractability of most aggregation problems for PrXML^{cie} has also led us to propose polynomial-time randomized approximation schemes.

As the first work on aggregation queries in probabilistic XML, our research can be extended in a number of ways. First, it is important to investigate more expressive query languages than the single-path queries we considered in Section 2. The locality of $\mathsf{PrXML}^{mux,det}$ should make it possible to evaluate the result of an aggregation defined by an arbitrary tree-pattern query [18], or even other forms of navigational queries defined with a bottom-up tree automaton [7]. Adapting the query evaluation algorithms proposed in [7, 18] to aggregation queries is ongoing work. This may allow us to deal with queries involving arbitrary grouping of the aggregation results too. The work presented here could also be straightforwardly extended to p-documents with *exp* distributional nodes. Finally, aggregation queries that inherently manipulate leaf values would be especially useful in conjunction with *continuous* probability distribution of these leaf values: the result of an experimental measure might be a uniform distribution between two values, or a Gaussian centered around a given value, etc. A preliminary study of this problem seems to indicate that it is not harder to deal with such continuous probability distributions on leaves of a p-document, at least when integration and differentiation of these distributions is tractable, either symbolically or numerically.

References

[1] S. Abiteboul, B. Kimelfeld, Y. Sagiv, and P. Senellart. On the expressiveness of probabilistic XML models. *The VLDB Journal*, 2009. To appear.

[2] S. Abiteboul and P. Senellart. Querying and updating probabilistic information in XML. In *Proc. EDBT*, Munich, Germany, Mar. 2006.

[3] F. N. Afrati and P. G. Kolaitis. Answering aggregate queries in data exchange. In *Proc. PODS*, Vancouver, BC, Canada, June 2008.

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

[5] D. Calvanese, E. Kharlamov, W. Nutt, and C. Thorne. Aggregate queries over ontologies. In *Proc. ONISW*, Napa, CA, USA, Oct. 2008.

[6] S. Cohen, B. Kimelfeld, and Y. Sagiv. Incorporating constraints in probabilstic XML. In *Proc. PODS*, Vancouver, BC, Canada, June 2008.

[7] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *Proc. PODS*, Providence, RI, USA, June 2009.

[8] S. Cohen, Y. Sagiv, and W. Nutt. Rewriting queries with arbitrary aggregation functions using views. *TODS*, 31(2):672–715, 2006.

[9] N. N. Dalvi and D. Suciu. Management of probabilistic data: foundations and challenges. In *Proc. PODS*, Beijing, China, June 2007.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[11] E. Grädel, Y. Gurevich, and C. Hirsch. The complexity of query reliability. In *Proc. PODS*, Seattle, WA, USA, June 1998.

[12] T. J. Green and V. Tannen. Models for incomplete and probabilistic information. In *Proc. EDBT Workshops, IIDB*, Munich, Germany, Mar. 2006.

[13] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):16–30, 1963.

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

[15] E. Hung, L. Getoor, and V. S. Subrahmanian. Probabilistic interval XML. *TOCL*, 8(4), 2007.

[16] T. Imieliński and W. Lipski. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[17] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query efficiency in probabilistic XML models. In *Proc. SIGMOD*, Vancouver, BC, Canada, June 2008.

[18] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query evaluation over probabilistic XML. *The VLDB Journal*, 2009. To appear.

[19] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic XML. In *Proc. VLDB*, Vienna, Austria, Sept. 2007.

[20] C. Koch. MayBMS: A system for managing large uncertain and probabilistic databases. In C. Aggarwal, editor, *Managing and Mining Uncertain Data.* Springer-Verlag, 2009.

[21] J. Lechtenbörger, H. Shu, and G. Vossen. Aggregate queries over conditional tables. *Journal of Intelligent Information Systems*, 19(3):343–362, 2002.

[22] A. Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proc. VLDB*, Hong Kong, China, Aug. 2002.

[23] C. H. Papadimitriou. *Computational Complexity.* Addison Wesley Pub. Co., Reading, USA, 1994.

[24] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal of Computing*, 12(4):777–788, 1983.

[25] C. Ré and D. Suciu. Efficient evaluation of HAVING queries on a probabilistic database. In *Proc. DBPL*, Vienna, Austria, Sept. 2007.

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

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

[28] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. CIDR*, Asilomar, CA, USA, Jan. 2005.