Aggregate Queries for Discrete and Continuous Probabilistic XML*

Serge Abiteboul
INRIA Saclay
4 rue J.Monod
91893 Orsay Cedex, France
serge.abiteboul@inria.fr

T.-H. Hubert Chan
Dept. of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong
hubert@cs.hku.hk

Evgeny Kharlamov†
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
kharlamov@inf.unibz.it

Werner Nutt
Free University of Bozen-Bolzano
Dominikanerplatz 3
39100 Bozen, Italy
nutt@inf.unibz.it

Pierre Senellart
Institut Télécom; Télécom ParisTech
CNRS LTCI, 46 rue Barrault
75634 Paris, France
pierre@senellart.com

ABSTRACT

Sources of data uncertainty and imprecision are numerous. A way to handle this uncertainty is to associate probabilistic annotations to data. Many such probabilistic database models have been proposed, both in the relational and in the semi-structured setting. The latter is particularly well adapted to the management of uncertain data coming from a variety of automatic processes. An important problem, in the context of probabilistic XML databases, is that of answering aggregate queries (count, sum, avg, etc.), which has received limited attention so far. In a model unifying the various (discrete) semi-structured probabilistic models studied up to now, we present algorithms to compute the distribution of the aggregation values (exploiting some regularity properties of the aggregate functions) and probabilistic moments (especially, expectation and variance) of this distribution. We also prove the intractability of some of these problems and investigate approximation techniques. We finally extend the discrete model to a continuous one, in order to take into account continuous data values, such as measurements from sensor networks, and present algorithms to compute distribution functions and moments for various classes of continuous distributions of data values.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: Logical Design, Languages—*data models, query languages*; F.2.0 [**Analysis of Algorithms and Problem Complexity**]: General

General Terms

Algorithms, Theory

Keywords

XML, probabilistic databases, aggregation, complexity, algorithms

*This research was funded by the European Research Council grant Webdam (under FP7), grant agreement 226513, and by the Dataring project of the French ANR.

†The author is co-affiliated with INRIA Saclay.

ICDT 2010, March 22–25, 2010, Lausanne, Switzerland.

1. INTRODUCTION

The (HTML or XML) Web is an important source of uncertain data, for instance generated by imprecise automatic tasks such as information extraction. A natural way to model this uncertainty is to annotate semistructured data with probabilities. This is the basis of recent works that consider queries over such imprecise hierarchical information [3, 15, 16, 19, 21, 23, 27, 28]. An essential aspect of query processing has been ignored in all these works, namely, aggregate queries. This is the problem we consider here. We provide a comprehensive study of query processing for a very general model of imprecise data and a very large class of aggregate queries.

We consider *probabilistic XML documents* and the unifying representation model of *p-documents* [2, 19]. A p-document can be viewed as a probabilistic process that randomly generates XML documents. Some nodes, namely distributional nodes, specify how to perform this random generation. We consider three kinds of distributional operators: *cie*, *mux*, *det*, respectively for *conjunction of independent events* (a node is selected if a conjunction of some probabilistic conditional events holds), *mutually exclusive* (at most one node selected from a set of a nodes), and *deterministic* (all nodes selected). This model, introduced in [2, 19], captures a very large class of models for probabilistic trees that had been previously studied. For queries, we consider tree-pattern queries possibly with value joins and the restricted case of single-path queries. For aggregate functions, we consider the standard ones, namely, sum, count, min, max, countd (count distinct) and avg (average).

A p-document is a (possibly very compact) representation of a probabilistic space of (ordinary) documents, i.e., a finite

set of possible documents, each with a particular probability. In the absence of a grouping operation à la SQL (`GROUP BY`), the result of an aggregate query is a single value for each possible document. Therefore, an aggregate query over a p-document is a random variable and the result is a distribution, that is, the set of possible values, each with its probability. It is also interesting to consider summaries of the distribution of the result random variable (that is possibly very large), and in particular, its expected value and other probabilistic moments. When grouping is considered, a single value (again a random variable) is obtained for each match of the grouping part of the query. We investigate the computation of the distributions of random variables (in presence of grouping or not) and of their 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 an increase in complexity. For documents with *cie* nodes, we show the problems are hard (typically NP- or $\mathsf{FP}^{\#\mathsf{P}}$-complete). For count and sum, in the restricted setting of single-path queries, we show how to obtain moments in P. We also present Monte-Carlo methods that allow tractable approximations of probabilities and moments. On the other hand, with the milder forms of imprecision, namely *mux* and *det*, the complexity is lower. Computing the distribution for tree-pattern queries involving 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. On the positive side, we can compute expected values (and moments) for most aggregate tree-pattern queries in P. When we move to queries involving joins, the complexity of moment and distribution computation becomes $\mathsf{FP}^{\#\mathsf{P}}$-complete.

A main novelty of this work is that we also consider probabilistic XML documents involving continuous probability distributions, which captures a very frequent situation occurring in practice. We formally extend the probabilistic XML model by introducing leaves representing continuous value distributions. We explain how the techniques for the discrete case can be adapted to the continuous case and illustrate the approach by results that can be obtained.

The paper is organized as follows. After presenting preliminaries and introducing the problems in Sections 2 and 3, we consider *cie* nodes in Section 4. In Section 5, we consider monoid aggregate functions in the context of *mux* and *det* nodes. Continuing with this model, we study complexity of distributions and moments in Section 6. We briefly discuss approximation algorithms in Section 7. Continuous probability distributions are considered in Section 8. Finally, we present related work and conclude in Section 9.

A preliminary version of some of this work appeared in [1] (a national conference without proceedings).

2. DETERMINISTIC DATA AND QUERIES

We recall the data model and query languages we use.

We assume a countable set of *identifiers* $\mathcal{I}$ and one of *labels* $\mathcal{L}$, such that $\mathcal{I} \cap \mathcal{L} = \varnothing$. The set of labels includes a set of data values (e.g., the integers, on which the aggregate functions will be defined). A *document* is a pair $d = (t, \theta)$, consisting of a finite, unordered[1] tree t, where each node has a unique identifier v and a label $\theta(v)$. We use the standard notions *child* and *parent*, *descendant* and *ancestor*, *root* and *leaf* in the usual way. To simplify the presentation, we assume that the leaves of documents are labeled with data values and the other nodes by non-data labels, that are called tags. The sets of *nodes* and *edges* of d are denoted, respectively, by $\mathcal{I}(d)$ and $\mathcal{E}(d)$, where $\mathcal{E}(d) \subseteq \mathcal{I}(d) \times \mathcal{I}(d)$. We denote the root of d by $\mathsf{root}(d)$ and the empty tree, that is, the tree with no nodes, by ε.

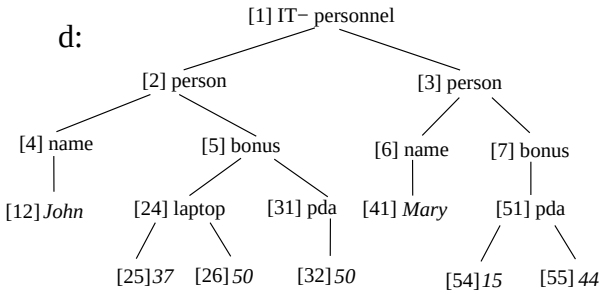

Figure 1: Document d: personnel in IT department.

EXAMPLE 1. *Consider the document d in Figure 1. Identifiers appear inside square brackets before labels. The document describes the personnel of an IT department and the bonuses distributed for different projects. The document d indicates* `John` *worked under two projects (*`laptop` *and* `pda`*) and got bonuses of* `37` *and* `50` *in the former project and* `50` *in the latter one.*

An *aggregate function* maps a finite bag of values (e.g., rationals) into some domain (possibly the same or different). In particular, we assume that any aggregate function is defined on the empty bag. In the paper we study the common functions: sum, count, min, countd (count distinct), and avg (average) under the usual semantics. Our results easily extend to max and top$K$.

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. Again to simplify, we ignore this issue here and assume they all have the proper type. It is straightforward to extend our models and results with a more refined treatment of typing.

As we will see some particular aggregate functions, the so-called *monoid* ones [10], play a particular role in our investigation, because they can be handled by a divide-and-conquer strategy. Formally, a structure $(M, \oplus, \perp)$ is called an *abelian monoid* if $\oplus$ is an associative and commutative binary operation with $\perp$ as identity element. If no confusion arises, we speak of the monoid M. An aggregate function is a *monoid* one if for some monoid M and any $a_1, \ldots, a_n \in M$:

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

It turns out that sum, count, min, max and top$K$ 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 $+$. On the other hand, it is easy to check that neither avg nor countd are monoid aggregate functions.

[1]Ignoring the ordering of the children of nodes is a common simplification over the XML model that does not significantly change the results of this paper.

Finally, we introduce tree pattern queries with joins, with join-free queries and single-path queries as special cases. We then extend them to aggregate queries.

We assume a countable set of variables *Var*. A *tree pattern* (with joins), denoted Q, is a tree with two types of edges: child-edges, denoted $E_/$, and descendent edges, denoted $E_{//}$. The nodes of the tree are labeled by a labeling function[2] λ with either labels from $\mathcal{L}$ or with variables from *Var*. Variables that occur more than once are called *join variables*. We refer to nodes of Q as n, m in order to distinguish them from the nodes of documents.

A *tree pattern query with joins* has the form $Q[\bar{n}]$, where Q is a tree pattern with joins and $\bar{n}$ is a tuple of nodes of Q (defining its output). We sometimes identify the query with the pattern and write Q instead of $Q[\bar{n}]$ if $\bar{n}$ is not important or clear from the context. If $\bar{n}$ is the empty tuple, we say that the query is Boolean. A query is *join-free* if every variable in its pattern occurs only once. If the set of edges $E_/ \cup E_{//}$ of a query is a linear order, the query is a *single-path query*. We denote the set of all tree pattern queries, which may have joins, as TPJ. The subclasses of join-free and single path queries are denoted as TP and SP, respectively.

A *valuation* ν maps query nodes to document nodes. A document *satisfies* a query if there exists a *satisfying* valuation, which maps query nodes to the document nodes in a way that is consistent with the edge types, the labeling, and the variable occurrences. That is, (1) nodes connected by child/descendant edges are mapped to nodes that are children/descendants of each other; (2) query nodes with label a are mapped to document nodes with label a; and (3) two query nodes with the same variable are mapped to document nodes with the same label.

Slightly differently from other work, we define that applying a query $Q[\bar{n}]$ to a document d returns a set of tuples of nodes: $Q(d) := \{\nu(\bar{n}) \mid \nu \text{ satisfies } Q\}$. One obtains the more common semantics, according to which a query returns a set of tuples of labels, by applying the labeling function of d to the tuples in $Q(d)$.

An *aggregate* TPJ-query has the form $Q[\alpha(n)]$, where Q is a tree pattern, n is a node of Q and α is an aggregate function. We evaluate such a $Q[\alpha(n)]$ in three steps: First, the non-aggregate query $Q' := Q[n]$ over d, obtaining a set of nodes $Q'(d)$. We then compute the *bag* B of labels of $Q'(d)$, that is $B := \{\!\{\theta(n) \mid n \in Q'(d)\}\!\}$. Finally we apply α to B. Identifying the aggregate query with its pattern, we denote the value resulting from evaluating Q over d as $Q(d)$.

If $Q[n]$ is a non-aggregate query and α an aggregate function, we use the shorthand Q^α to denote the aggregate query $Q[\alpha(n)]$. More generally, we denote the set of aggregate queries obtained from queries in TPJ, SP, TP and some function α, as $\mathsf{TPJ}^\alpha, \mathsf{SP}^\alpha, \mathsf{TP}^\alpha$, respectively.

The syntax and semantics above can be generalized in a straightforward fashion to aggregate queries with SQL-like `GROUP BY`. Such queries are written $Q[\bar{n}, \alpha(n)]$ and return an aggregate value for every binding of $\bar{n}$ to a tuple of document nodes. Since we can reduce the evaluation of such queries to the evaluation of several simpler queries of the kind defined before, while increasing the data complexity by no more than a polynomial factor, we restrict ourselves to that simpler case.

[2]We denote the labeling function for queries as λ in order to distinguish it from the labeling function θ for documents.

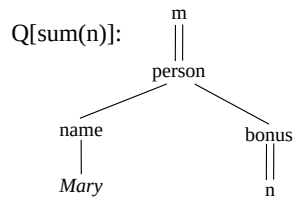

Figure 2: Query: sum of bonuses for Mary.

EXAMPLE 2. *Continuing with Example 1, one may want to compute the* sum *of bonuses for* each *person in the department. A* $\mathsf{TP}^{\mathsf{sum}}$ *query Q that computes bonuses for Mary is in Figure 2. The query result $Q(d)$ is 59.*

3. DISCRETE PROBABILISTIC DATA

We next present discrete probability spaces over data trees (see [2] for a more detailed presentation) and formalize the problems we will study in the following sections.

3.1 px-Spaces and p-Documents

A *finite probability space* over documents, px-*space* for short, is a pair $\mathcal{S} = (\mathcal{D}, \mathsf{Pr})$, where $\mathcal{D}$ is a finite set of documents and Pr maps each document to a probability $\mathsf{Pr}(d)$ such that $\sum\{\mathsf{Pr}(d) \mid d \in \mathcal{D}\} = 1$.

p-Documents: Syntax. Following [2], we now introduce a very general syntax for representing compactly px-spaces, called *p-documents*. A p-document is similar to a document, with the difference that it has two types of nodes: *ordinary* and *distributional*. Distributional nodes are used for defining the probabilistic process that generates random documents but they do not actually occur in these. Ordinary nodes have labels and they may appear in random documents. We require the leaves to be ordinary nodes[3].

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

EXAMPLE 3. *Two p-documents are shown in Figures 3 and 4. The first one has only cie distributional nodes. For example, node n_{21} has label $cie(E)$ and two children n_{22} and n_{24}, such that $E(n_{22}) = \neg x$ and $E(n_{24}) = x$. The second p-document has only mux and det distributional nodes. Node n_{52} has label $mux(\mathsf{Pr})$ and two children n_{53} and n_{56}, such that $\mathsf{Pr}(n_{53}) = 0.7$ and $\mathsf{Pr}(n_{56}) = 0.3$.*

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

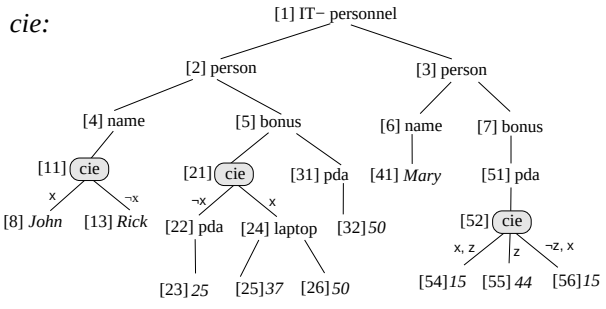

Figure 3: PrXML^{cie} p-document: IT department.

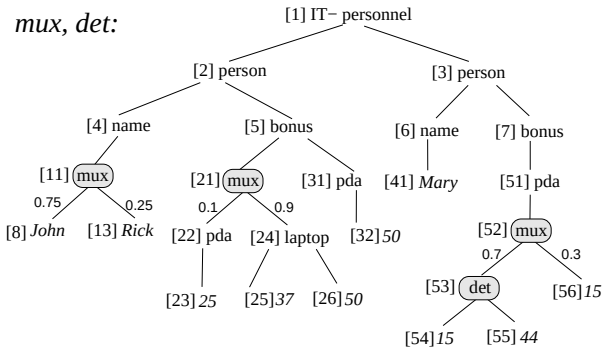

Figure 4: PrXML^{mux,det} p-document: IT department.

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, PrXML^{mux,det} is the class of p-documents with only *mux* and *det* distributional nodes, like $\widehat{\mathcal{P}}$ on Figure 4.

p-Documents: Semantics. The *semantics of a p-document* $\widehat{\mathcal{P}}$, denoted by $[\![\widehat{\mathcal{P}}]\!]$, is a px-space over *random documents*, where the documents 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 where $\nu(E(v))$ 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.[4]

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 document $\mathcal{P}$. 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. *One can obtain the document d in Figure 1 by applying the randomized process to the p-document in Figure 4. Then the probability of d is* $\mathsf{Pr}(d) = .75 \times .9 \times .7 = .4725$. *One can also obtain d from the p-document in Figure 3, assuming that* $\mathsf{Pr}(x) = .85$ *and* $\mathsf{Pr}(z) = .055$, *by assigning* $\{x/1, z/1\}$. *In this case the probability of d is* $\mathsf{Pr}(d) = .85 \times .055 = .04675$.

Remark. In our analysis, we only consider distributional nodes of the types *cie*, *mux*, and *det*. In [2] two more types of distributional nodes (*ind* and *exp*) are considered. As shown there, the first kind can be captured by *mux* and *det*, while the second is a generalization of *mux* and *det* and most results of PrXML^{mux,det} can be extended to PrXML^{exp}. As proved in [2], PrXML^{cie} is strictly more expressive than PrXML^{mux,det}. It was shown in [19, 20] that data complexity of answering TP-queries is intractable for PrXML^{cie} (FP$^{\#\mathsf{P}}$-complete) whereas it is polynomial for PrXML^{mux,det}.

[4]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 [2]: *mux* alone can express that subtrees are mutually exclusive, but in combination with *det* it can also express this on *subforests*.

3.2 Aggregating Discrete Probabilistic Data

Let Q^α be an aggregate query and $\mathcal{S} = (\mathcal{D}, \mathsf{Pr})$ be a px-space of documents. Since Q^α maps elements of the probability space $\mathcal{S}$ to values in the range of α, we can see Q^α as a random variable.

We therefore define the result of applying Q^α to $\mathcal{S}$ as the distribution of this random variable, that is

$$(Q^\alpha(\mathcal{S}))(c) = \sum \Big\{ \mathsf{Pr}(d) \;\Big|\; d \in \mathcal{D},\ Q^\alpha(d) = c \Big\},$$

for c in the range of α.

Since in applications px-spaces are given under the form of p-documents, we further extend the definition to p-documents by defining $Q^\alpha(\widehat{\mathcal{P}}) := Q^\alpha([\![\widehat{\mathcal{P}}]\!])$. We denote the random variable over the p-document $\widehat{\mathcal{P}}$ corresponding to Q as $Q(\mathcal{P})$.

EXAMPLE 5. *Evaluation of the query* $Q[\mathsf{sum}(n)]$ *from Example 2 over the cie-document in Figure 3 gives the distribution* $\{(0, 0.14175), (15, 0.80325), (44, 0.00825)\,(59, 0.04675)\}$, *while evaluation over the mux-det-document in Figure 4 gives the distribution* $\{(15, 0.3), (59, 0.7)\}$.

Computational Problems. For an aggregate query Q, we are interested in the following three problems, where the input parameters are a p-document $\widehat{\mathcal{P}}$ with corresponding random document $\mathcal{P}$ and possibly a number c:

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

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

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

Membership and *probability computation* can be used to return to a user the distribution $Q(\widehat{\mathcal{P}})$ of an aggregate query. Computing the entire distributions may be too costly or the user may prefer a *summary* of the distributions. For example, a user may want to know its expected value $\mathbb{E}(Q(\mathcal{P}))$ and the variance $\mathrm{Var}(Q(\mathcal{P}))$. In general the summary can

be an arbitrary k-th moment $\mathbb{E}(Q(\mathcal{P})^k)$ and the *moment computation* problem addresses this issue.[5]

In the following, we investigate these problems for the classes of *cie* documents and *mux-det* documents. For each class, we further distinguish between aggregate queries of the types SP, TP, and TPJ with the functions min, count, sum, countd and avg. We do not discuss max and top$K$ since they behave similarly as min. In the paper we mainly speak about *data-complexity*, when the input is a p-document and the query is fixed. Occasionally we also consider *combined complexity*, when both the p-document and the query are inputs of the problem.

4. AGGREGATING PrXML^{cie}

We now study the problems introduced in Section 3 for the most general class of p-documents, PrXML^{cie}. By definition, one approach is to first construct the entire px-space of a p-document $\widehat{\mathcal{P}}$, then to apply the aggregate query Q to each document in $[\![\widehat{\mathcal{P}}]\!]$ separately, and finally combine the results to obtain the distribution $Q(\widehat{\mathcal{P}})$. 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. All the decision problems are NP-complete while computational problems are NP-hard or FP$^{\#\mathsf{P}}$-complete. The only exception is the computation of moments for aggregate single-path queries with sum and count. The intractability is due to dependencies between nodes of p-documents expressed using variables.

4.1 Principles

We now show several general principles for p-documents that are used later on to support the results.

Functions in #P and FP$^{\#\mathsf{P}}$. We recall here the definitions of some classical complexity classes (see, e.g., [24]) that characterize the complexity of aggregate 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 [9], 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 [25], 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 [13].

[5]The variance is the *central* moment of order 2; it is known that the central moment of order k can be tractably computed from the regular moments of order $\leqslant k$.

Proposition 6. *Let α be an aggregate function that is computable in polynomial time and Q be an aggregate* TPJ*-query using α. If α is scalable, then the following functions mapping p-documents to rational numbers are in* FP$^{\#\mathsf{P}}$*:*

1. *for every $c \in \mathbb{Q}$ the function $\widehat{\mathcal{P}} \mapsto \mathsf{Pr}(Q(\mathcal{P}) = c)$;*
2. *for every $k \geqslant 1$, the function $\widehat{\mathcal{P}} \mapsto \mathbb{E}(Q(\mathcal{P})^k)$.*

The proposition above shows membership in FP$^{\#\mathsf{P}}$ of both probability and moment computation for aggregate queries in TPJ with all aggregate functions mentioned in the paper.

Reducing Query Evaluation to Aggregation. Now we show that for answering aggregate SP-queries it is possible to isolate aggregation from query processing.

Let $\widehat{\mathcal{P}}$ be in PrXML^{cie}. If Q is an SP-query, we can apply it naively to $\widehat{\mathcal{P}}$, ignoring the distributional nodes. The result $\widehat{\mathcal{P}}_Q$ is the subtree of $\widehat{\mathcal{P}}$ containing the original root and as leaves the nodes satisfying Q (i.e., the nodes matched by the free variable of Q). Interestingly, it turns out that for all aggregate functions α, evaluating Q^α over $\widehat{\mathcal{P}}$ is the same as applying α to $\widehat{\mathcal{P}}_Q$. If $\widehat{\mathcal{P}}$ is in PrXML^{mux,det}, then $\widehat{\mathcal{P}}_Q$ can be obtained analogously and, again, evaluating Q^α over $\widehat{\mathcal{P}}_Q$ can be reduced to evaluating α over $\widehat{\mathcal{P}}_Q$. Therefore, answering an aggregate SP-query Q^α over $\widehat{\mathcal{P}}$ in PrXML^{cie,mux,det} can be done in two steps, first one queries $\widehat{\mathcal{P}}$ with the non-aggregate part Q, which results in a p-document $\widehat{\mathcal{P}}_Q$, and then one aggregates *all* the leaves of $\widehat{\mathcal{P}}_Q$. The previous discussion leads to the following result.

Proposition 7. *Let $Q[\bar{n}]$ be a non-aggregate* SP*-query. Then for every p-document $\widehat{\mathcal{P}} \in$ PrXML^{cie,mux,det} we can compute in time quadratic in $|Q| + |\widehat{\mathcal{P}}|$ a p-subdocument $\widehat{\mathcal{P}}_Q$ of $\widehat{\mathcal{P}}$ such that for every aggregate function α we have:*

$$Q^\alpha(\widehat{\mathcal{P}}) = \alpha(\widehat{\mathcal{P}}_Q).$$

Hardness Results for Branching Queries. With the next lemma, we can translate data complexity results for non-aggregate queries to lower bounds of the complexity of computing probabilities of aggregate values and moments of distributions. An aggregate function α is *faithful* if $\alpha(\{\!\{1\}\!\}) = 1$.

Lemma 8. *Let Q be* TPJ*-query, $\widehat{\mathcal{P}}$ a p-document and α a faithful aggregate function. Then one can construct in linear time an aggregate* TPJ*-query Q'_α with the function α and a p-document $\widehat{\mathcal{P}}'$ such that for any $k \geqslant 1$,*

$$\mathsf{Pr}(\mathcal{P} \models Q) \;=\; \mathsf{Pr}(Q'_\alpha(\mathcal{P}') = 1) \;=\; \mathbb{E}(Q'_\alpha(\mathcal{P}')^k).$$

Moreover,

1. *if $Q \in$ TP, then $Q'_\alpha \in$ TP$^\alpha$;*
2. *if $\widehat{\mathcal{P}} \in$ PrXML^{cie}, then $\widehat{\mathcal{P}}' \in$ PrXML^{cie};*
3. *if $\widehat{\mathcal{P}} \in$ PrXML^{mux,det}, then $\widehat{\mathcal{P}}' \in$ PrXML^{mux,det}.*

In [19] it has been proved that for every non-trivial Boolean tree pattern query, computing the probability to match *cie*-documents is #P-hard. By reducing #2DNF, we can show

that for the more restricted case of *mux-det* documents, evaluation of tree pattern queries with joins can be #P-hard.

LEMMA 9. *There is a Boolean* TPJ*-query with* #P*-hard data complexity over* $\mathsf{PrXML}^{mux,det}$.

The result in [19] and the previous lemma yield immediately the following complexity lower bounds for probability and moment computation for TP and TPJ.

COROLLARY 10. *For any faithful aggregate function* α*, there exist an aggregate* TP*-query* Q_1 *and an aggregate* TPJ*-query* Q_2*, both with function* α*, such that each of the following computation problems is* #P*-hard:*

1. *probability computation for* Q_1 *over* PrXML^{cie}*;*
2. *k-th moments of* Q_1 *over* PrXML^{cie}*, for any* $k \geqslant 1$*;*
3. *probability computation for* Q_2 *over* $\mathsf{PrXML}^{mux,det}$*;*
4. *k-th moments of* Q_2 *over* $\mathsf{PrXML}^{mux,det}$*, for any* $k \geqslant 1$.

We are ready to present aggregation of PrXML^{cie}.

4.2 Computational Problems

We first show how to check for membership over PrXML^{cie}.

THEOREM 11 (MEMBERSHIP). *Let* α *be one of* sum, min, count, avg*, and* countd*. Then membership over* PrXML^{cie} *is in* NP *for the class* TPJ^{α}*. Moreover, the problem is* NP*-hard for any aggregate query in* SP^{α}.

The upper bound holds because, given a query, guessing a world and evaluating the query takes no more than polynomial time. The lower bound follows from the next lemma.

LEMMA 12. *Let* Q *be an* SP*-query with one free variable and 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* $\widehat{\mathcal{P}}_\varphi \in \mathsf{PrXML}^{cie}$ *such that the following are equivalent: (1)* φ *is falsifiable, (2)* $\mathsf{Pr}(Q^\alpha(\mathcal{P}) = 1) > 0$ *over* $\widehat{\mathcal{P}}_\varphi$ *for* some $\alpha \in \boldsymbol{AGG}$*, (3)* $\mathsf{Pr}(Q^\alpha(\mathcal{P}) = 1) > 0$ *over* $\widehat{\mathcal{P}}_\varphi$ *for* all $\alpha \in \boldsymbol{AGG}$.

We next show how to compute probability over PrXML^{cie}.

THEOREM 13 (PROBABILITY). *Let* α *be one of* sum, count, min, avg*, and* countd*. Then probability computation over* PrXML^{cie} *is in* $\mathsf{FP}^{\#\mathsf{P}}$ *for the class* TPJ^{α}*. Moreover, the problem is* #P*-hard for every query in* SP^{α}.

PROOF. (Sketch) The $\mathsf{FP}^{\#\mathsf{P}}$ upper bound follows from Proposition 6 and #P-hardness can be shown by a reduction of probability computation for DNF propositional formulas (that is known to be #P-hard), see the following lemma. □

The following lemma supports Theorems 13 and 15.

LEMMA 14. *Let* α *be one of* sum, count, min, avg, countd*, and* β *be one of* min, avg, countd*. Let* Q^α *and* Q^β *be* SP*-queries. Then for every propositional DNF formula* φ*, one can compute in polynomial time a p-document* $\widehat{\mathcal{P}}_\varphi \in \mathsf{PrXML}^{cie}$ *such that the following are equivalent:*

1. $\mathsf{Pr}(Q^\alpha(\mathcal{P}_\varphi) = 0) = 1 - \mathsf{Pr}(\varphi)$*;*
2. $\mathbb{E}(Q^\beta(\mathcal{P}_\varphi)^k) = 1 - \mathsf{Pr}(\varphi)$ *for any* $k \geqslant 1$.

PrXML^{cie}	Aggregate query language			
	SP		TP	TPJ
Membership	NP-c		NP-c	NP-c
Probability	$\mathsf{FP}^{\#\mathsf{P}}$-c		$\mathsf{FP}^{\#\mathsf{P}}$-c	$\mathsf{FP}^{\#\mathsf{P}}$-c
Moments	count, sum	P	$\mathsf{FP}^{\#\mathsf{P}}$-c	$\mathsf{FP}^{\#\mathsf{P}}$-c
	others	$\mathsf{FP}^{\#\mathsf{P}}$-c		

Table 1: Data complexity of query evaluation over PrXML^{cie}. NP-c means NP-complete.

We finally show how to compute moments over PrXML^{cie}.

THEOREM 15 (MOMENTS). *Let* α *be one of* sum, count, min, avg*, and* countd*. Then computation of moments of any degree over* PrXML^{cie} *is in* $\mathsf{FP}^{\#\mathsf{P}}$ *for the class* TPJ^{α}*. Moreover, the problem is*

1. *of polynomial combined complexity for the classes* $\mathsf{SP}^{\mathsf{sum}}$ *and* $\mathsf{SP}^{\mathsf{count}}$*;*
2. #P*-hard for any query in the classes* $\mathsf{SP}^{\mathsf{min}}$*,* $\mathsf{SP}^{\mathsf{avg}}$ *and* $\mathsf{SP}^{\mathsf{countd}}$*;*
3. #P*-hard for some query in* $\mathsf{TP}^{\mathsf{sum}}$ *and* $\mathsf{TP}^{\mathsf{count}}$.

PROOF. Again, as for Theorem 13, the $\mathsf{FP}^{\#\mathsf{P}}$ upper bound follows from Proposition 6. Claim 2 follows from Lemma 14 and its analogues for min, countd, and avg. Claim 3 follows from Corollary 10.

To prove Claim 1, we rely on Proposition 7, which reduces answering aggregate SP-queries to evaluating aggregate functions, and the following lemmas. □

The next lemma shows that the 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 16. *Let* $\mathcal{S}$ *be a* px*-space and* V *be the set of all leaves occurring in the documents 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*

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

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

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

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 17. *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*

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

Now we are ready to conclude the proof of the theorem.

PROOF. OF THEOREM 15.1 By Lemma 16, 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 17, the second term can be computed in polynomial time. This shows that for every $k \geqslant 1$, 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. □

Table 1 gives an overview of the data complexity results of this section.

5. MONOID AGGREGATES

The previous section highlighted the inherent difficulty of computing aggregate queries over *cie*-documents. The intuitive reason for this difficulty is that the event variables used in a p-document can impose constraints between the structure of subdocuments in very different locations. In contrast, *mux*,*det*-documents only express "local" dependencies. As a consequence, for the special case of single path queries and monoid aggregate functions, *mux*-*det* documents allow for a conceptually simpler computation of distributions, which in a number of cases is also computationally efficient.

The key to developing methods in this setting is Proposition 7, which reduces the evaluation of a single path aggregate query Q^α over $\widehat{\mathcal{P}}$ to the evaluation of the function α over the document $\widehat{\mathcal{P}}_Q$. Note that $\widehat{\mathcal{P}}_Q$ is again a *mux*-*det* document if $\widehat{\mathcal{P}}$ is one. Therefore, we can concentrate on the question of evaluating α over *mux*,*det*-documents.

We are going to show how a *mux*,*det*-document $\widehat{\mathcal{P}}$ can be seen as a recipe for constructing the px-space $[\![\widehat{\mathcal{P}}]\!]$ in a bottom-up fashion, starting from elementary spaces represented by the leaves and using essentially two kinds of operations, convex union and product. Convex union corresponds to *mux*-nodes and product corresponds to *det*-nodes and regular nodes. (To be formally correct, we would need to distinguish between two slightly different versions of product for *det* and regular nodes. However, to simplify our exposition, we only discuss the case of regular nodes and briefly indicate below the changes necessary to deal with *det*-nodes.)

For any α, the distribution over the space described by a leaf of $\widehat{\mathcal{P}}$ is a Dirac distribution, that is, a distribution of the form δ_a, where $\delta_a(b) = 1$ if and only if $a = b$. For monoid functions α, the two operations on spaces, convex union and product, have as counterparts two operations on distributions, convex sum and convolution, by which one can construct the distribution $\alpha(\widehat{\mathcal{P}})$ from the Dirac distributions of the leaves of $\widehat{\mathcal{P}}$. We sketch in the following both the operations on spaces and on distributions, and the way in which they are related.

As the base case, consider a leaf node v with label l. This is the simplest p-document possible, which constitutes an elementary px-space that contains one document, namely node v with label l, and assigns the probability 1 to that document. Over this space, α evaluates with probability 1 to $\alpha(\{\!|l|\!\})$, hence, the probability distribution is $\delta_{\alpha(\{\!|l|\!\})}$. As a special case, if α is a monoid aggregation function over M, the distribution of α over the space containing only the empty document ε is $\delta_\perp$, where $\perp$ is the identity of M.

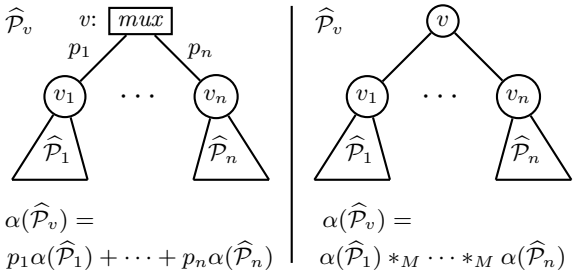

Figure 5: Distribution of monoid functions over composed $\mathsf{PrXML}^{mux,det}$ documents.

Inductively, suppose that v is a *mux*-node in $\widehat{\mathcal{P}}$, the subtrees below v are $\widehat{\mathcal{P}}_1, \ldots, \widehat{\mathcal{P}}_n$, and the probability of the i-th subtree $\widehat{\mathcal{P}}_i$ is p_i (see Figure 5, left). Without loss of generality we can assume that the p_i are *convex coefficients,* that is, $p_1 + \cdots + p_n = 1$, since we admit the empty tree as a special p-document.

Let $\widehat{\mathcal{P}}_v$ denote the subtree rooted at v. Then the semantics of *mux*-nodes implies that the px-space $[\![\widehat{\mathcal{P}}_v]\!] = (\mathcal{D}_v, \mathsf{Pr}_v)$ is the *convex union* of the spaces $[\![\widehat{\mathcal{P}}_i]\!] = (\mathcal{D}_i, \mathsf{Pr}_i)$, which means the following: (1) $\mathcal{D}_v$ is the disjoint union of the $\mathcal{D}_i$ (in other words, for any $d \in \mathcal{D}_v$, there is exactly one $\mathcal{D}_i$ such that $d \in \mathcal{D}_i$); (2) for any document $d \in \mathcal{D}_v$, we have that $\mathsf{Pr}_v(d) = p_i\mathsf{Pr}_i(d)$, where $d \in \mathcal{D}_i$.

As a consequence, $\alpha(\widehat{\mathcal{P}}_v)(c)$, the probability that α has the value c over $\widehat{\mathcal{P}}_v$, equals the weighted sum $p_1\alpha(\widehat{\mathcal{P}}_1)(c) + \cdots + p_n\alpha(\widehat{\mathcal{P}}_n)(c)$ of the probabilities that α has the value c over $\widehat{\mathcal{P}}_1, \ldots, \widehat{\mathcal{P}}_n$. In a more compact notation we can write this as $\alpha(\widehat{\mathcal{P}}_v) = p_1\alpha(\widehat{\mathcal{P}}_1) + \cdots + p_n\alpha(\widehat{\mathcal{P}}_n)$, which means that the distribution $\alpha(\widehat{\mathcal{P}}_v)$ is a *convex sum* of the $\alpha(\widehat{\mathcal{P}}_i)$.

For the second induction step, suppose that v is a regular non-leaf node in $\widehat{\mathcal{P}}$, with the label l. (see Figure 5, right). Similar to the previous case, suppose that the subtrees below v are $\widehat{\mathcal{P}}_1, \ldots, \widehat{\mathcal{P}}_n$, that $[\![\widehat{\mathcal{P}}_v]\!] = (\mathcal{D}_v, \mathsf{Pr}_v)$ and that $[\![\widehat{\mathcal{P}}_i]\!] = (\mathcal{D}_i, \mathsf{Pr}_i)$ for $1 \leqslant i \leqslant n$. Moreover, the $\mathcal{D}_i$ are mutually disjoint.

Every document $d \in \mathcal{D}_v$ has as root the node v, which carries the label l, and subtrees $d_1, \ldots, d_n$, where $d_i \in \mathcal{D}_i$. We denote such a document as $d = v^l(\{d_1, \ldots, d_n\})$. Conversely, according to the semantics of regular nodes in *mux*-*det* documents, every combination $\{d_1, \ldots, d_n\}$ of documents $d_i \in \mathcal{D}_i$ gives rise to an element $v^l(\{d_1, \ldots, d_n\}) \in \mathcal{D}_v$. (Note that, due to the mutual disjointness of the $\mathcal{D}_i$, the elements of $\mathcal{D}_v$ are in bijection with the tuples in the Cartesian product $\mathcal{D}_1 \times \cdots \times \mathcal{D}_n$.)

Consider a collection of documents $d_i \in \mathcal{D}_i$, $1 \leqslant i \leqslant n$, with probabilities $q_i := \mathsf{Pr}_i(d_i)$. Each d_i is the result of dropping some children of *mux*-nodes in $\widehat{\mathcal{P}}_i$ and q_i is the product of the probabilities of the surviving children. Then $d := v^l(d_1, \ldots, d_n)$ is the result of dropping simultaneously the same children of those *mux*-nodes, this time within $\widehat{\mathcal{P}}_v$. The set of surviving children in $\widehat{\mathcal{P}}_v$ is exactly the union of the sets of children having survived in each $\widehat{\mathcal{P}}_i$ and, consequently, for the probability $q := \mathsf{Pr}_v(d)$ we have that $q = q_1 \cdots q_n$. In summary, this shows that the probability space $(\mathcal{D}_v, \mathsf{Pr}_v)$ is structurally the same as the *product* of the spaces $(\mathcal{D}_i, \mathsf{Pr}_i)$.

Suppose now that, in addition, α is a monoid aggregate function taking values in $(M, \oplus, \bot)$. Then for any document $d = v^l(d_1, \ldots, d_n) \in \widehat{\mathcal{P}}_v$ we have that $\alpha(d) = \alpha(d_1) \oplus \cdots \oplus \alpha(d_n)$. Hence, the probability that $\alpha(\mathcal{P}_v) = c$ is the sum of all products $\mathsf{Pr}(\alpha(\mathcal{P}_1) = c_1) \cdots \mathsf{Pr}(\alpha(\mathcal{P}_n) = c_n)$ such that $c = c_1 \oplus \cdots \oplus c_n$. Motivated by this observation, we define the following operation. For any functions f, $g\colon M \to \mathbb{R}$, the *convolution* of f and g with respect to M is the function $f *_M g\colon M \to \mathbb{R}$ such that

$$(f *_M g)(m) = \sum_{m_1, m_2 \in M\colon\, m_1 \oplus m_2 = M} f(m_1) g(m_2). \qquad (1)$$

From our observation above it follows that the distribution $\alpha(\widehat{\mathcal{P}}_v)$ is the convolution of the distributions $\alpha(\widehat{\mathcal{P}}_i)$ with respect to M, that is,

$$\alpha(\widehat{\mathcal{P}}_v) = \alpha(\widehat{\mathcal{P}}_1) *_M \cdots *_M \alpha(\widehat{\mathcal{P}}_n). \qquad (2)$$

For *det*-nodes v, the same equation applies, although the supporting arguments are a bit more complicated. The crucial difference is that for *det*-nodes, $[\![\widehat{\mathcal{P}}_v]\!]$ is a space of forests, not trees, since the trees (or forests) in the $[\![\widehat{\mathcal{P}}_i]\!]$ are combined without attaching them to a new root.

We summarize how one can use the operations introduced to obtain the distribution of a monoid aggregate function over a *mux-det* document.

THEOREM 18. *Let α be a monoid aggregation function taking values in M and $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{mux,det}$. Then $\alpha(\widehat{\mathcal{P}})$ can be obtained in a bottom-up fashion by*

1. *attaching a Dirac distribution to every leaf and for every occurrence of the empty document;*
2. *taking convex sums at every mux-node; and*
3. *taking convolutions with respect to $\mathbb{R}$ at each det and each regular non-leaf node.*

Essentially the same relationship between distributions as spelled out in Theorem 18 exists also if we allow continuous distributions at the leaves of documents. An evaluation algorithm then has to compute convex sums and convolutions, starting from continuous instead of Dirac distributions (we will discuss this in detail in Section 8).

The *carrier* of a function $f\colon M \to \mathbb{R}$ is the set of elements $m \in M$ such that $f(m) \neq 0$. Since for any $\widehat{\mathcal{P}}$ the carrier of $\mathsf{min}(\widehat{\mathcal{P}})$ and of $\mathsf{count}(\widehat{\mathcal{P}})$ has at most as many elements as there are leaves in $\widehat{\mathcal{P}}$, we can draw some immediate conclusions from Theorem 18.

COROLLARY 19. *For any mux,det-document $\widehat{\mathcal{P}}$,*

1. *the distributions $\mathsf{count}(\widehat{\mathcal{P}})$ and $\mathsf{min}(\widehat{\mathcal{P}})$ can be computed in time polynomial in $|\widehat{\mathcal{P}}|$;*
2. *the distribution $\mathsf{sum}(\widehat{\mathcal{P}})$ can be computed in time polynomial in $|\widehat{\mathcal{P}}| + |\mathsf{sum}(\widehat{\mathcal{P}})|$.*

PROOF. (Sketch) Claim 1 holds because computing a convex sum and convolutions with respect to "+" and min of two distributions is polynomial and all distributions involved in computing $\mathsf{count}(\widehat{\mathcal{P}})$ and $\mathsf{min}(\widehat{\mathcal{P}})$ have size $O(|\widehat{\mathcal{P}}|)$. Claim 2 holds because, in addition, a convex sum and the convolution with respect to "+" of two distributions have at least the size of the largest of the two arguments. □

$\mathsf{PrXML}^{mux,det}$	Aggregate query language	
	SP, TP	TPJ
Membership	sum, avg, countd NP-c count, min P	NP-c count, min NP
Probability	avg, countd $\mathsf{FP}^{\#\mathsf{P}}$-c count, min P	$\mathsf{FP}^{\#\mathsf{P}}$-c
Probability (for sum)	SP: P^*; TP: $\mathsf{FP}^{\#\mathsf{P}}$	$\mathsf{FP}^{\#\mathsf{P}}$-c
Moments	SP: P; TP: avg $\mathsf{FP}^{\#\mathsf{P}}$, others P	$\mathsf{FP}^{\#\mathsf{P}}$-c

Table 2: Data complexity of query evaluation over $\mathsf{PrXML}^{mux,det}$. NP-c means NP-complete, NP means membership in NP and * means in the size of |input|+|distribution|.

Remark. For the monoid of integers with addition, (1) is the same as the well-known discrete convolution. (2) is in fact a special case of a general principle: If X and Y are two M-valued random variables on the probability spaces $\mathcal{X}$, $\mathcal{Y}$, with distributions f, g, respectively, then the distribution of $X \oplus Y\colon \mathcal{X} \times \mathcal{Y} \to M$ is the convolution $f *_M g$ of f and g. This principle has also been applied in [26] in the context of queries with aggregation constraints over probabilistic relational databases.

6. AGGREGATING $\mathsf{PrXML}^{mux,det}$

We investigate the three computational problems for aggregate queries for the restricted class of $\mathsf{PrXML}^{mux,det}$, drawing upon the principles developed in the preceding section.

THEOREM 20 (MEMBERSHIP). *Let α be one of* sum, count, min, avg, *and* countd. *Then membership over $\mathsf{PrXML}^{mux,det}$ is in NP for the class TPJ^α. Moreover, the problem is*

1. *NP-hard for every query in $\mathsf{SP}^{\mathsf{sum}}$, $\mathsf{SP}^{\mathsf{avg}}$ and $\mathsf{SP}^{\mathsf{countd}}$;*
2. *of polynomial combined complexity for the classes $\mathsf{SP}^{\mathsf{min}}$ and $\mathsf{SP}^{\mathsf{count}}$;*
3. *of polynomial data complexity for any query in $\mathsf{TP}^{\mathsf{min}}$ and $\mathsf{TP}^{\mathsf{count}}$.*

PROOF. (Sketch) The NP upper bound is inherited from the *cie*-case (Theorem 11). Claim 1 can be shown by a reduction of subset-sum and exact cover by 3-sets. Claims 2 and 3 follow from their counterparts (Claims 1 and 2, respectively) in Theorem 21. □

We next consider probability computation.

THEOREM 21 (PROBABILITY). *Let α be one of* sum, count, min, avg, *and* countd. *Then probability computation over $\mathsf{PrXML}^{mux,det}$ is in $\mathsf{FP}^{\#\mathsf{P}}$ for the class TPJ^α. Moreover, the problem is*

1. *of polynomial combined complexity for the classes $\mathsf{SP}^{\mathsf{min}}$ and $\mathsf{SP}^{\mathsf{count}}$;*
2. *of polynomial data complexity for any query in $\mathsf{TP}^{\mathsf{min}}$ and $\mathsf{TP}^{\mathsf{count}}$;*
3. *#P-hard for any query in $\mathsf{SP}^{\mathsf{avg}}$ and $\mathsf{SP}^{\mathsf{countd}}$;*
4. *#P-hard for some query in $\mathsf{TPJ}^{\mathsf{sum}}$, $\mathsf{TPJ}^{\mathsf{count}}$ and $\mathsf{TPJ}^{\mathsf{min}}$.*

PROOF. (Sketch) The $\mathsf{FP}^{\#\mathsf{P}}$ upper bound is inherited from the *cie*-case (Theorem 13). Claim 1 follows from Corollary 19, since, due to Proposition 7, for an aggregate SP-query Q^α we have that $Q^\alpha(\widehat{\mathcal{P}}) = \alpha(\widehat{\mathcal{P}}_Q)$.

Regarding Claim 2, algorithms for count and min can be developed in a straightforward way, applying the techniques in [8] to evaluate TP-queries with aggregate constraints. For a given p-document, there are only linearly many possible values for min and count, the probability of which can be computed in polynomial time by incorporating them in constraints. Consequently, the entire distribution of min or count can be computed in polynomial time.

Claim 3 can be shown by a reduction of the $\#K$-cover problem for countd and the #Non-Negative-Subset-Average problem for avg.[6] Claim 5 follows from Corollary 10. □

Finally, we consider moments over $\mathsf{PrXML}^{mux,det}$.

THEOREM 22 (MOMENTS). *Let α be one of* sum, count, min, avg, *and* countd. *Then computation of moments of any degree over $\mathsf{PrXML}^{mux,det}$ is in $\mathsf{FP}^{\#\mathsf{P}}$ for the class TPJ^α. Moreover, the problem is*

1. *of polynomial combined complexity for the class SP^α;*
2. *of polynomial data complexity for the class TP^α, if $\alpha \neq$ avg;*
3. *$\#\mathsf{P}$-hard for some query in TPJ^α.*

PROOF. The $\mathsf{FP}^{\#\mathsf{P}}$ upper bound is inherited from the *cie*-case (Theorem 15). Claim 3 follows from Corollary 10.

Regarding Claim 1, all our algorithms first reduce aggregate query answering to function evaluation (see Proposition 7). The algorithm for count and sum is a refinement for the one for the *cie*-case (Theorem 15). The algorithm for min works on the entire distribution, which can be computed in polynomial time (Corollary 19).

For countd we apply similar techniques of regrouping sums to those that we used for sum in Lemma 16. In doing so, we exploit the fact that the probability for a value (or sets of values of fixed cardinality) to occur in a query result over a *mux*,*det*-document can be computed in polynomial time, which follows from work in [19].

The algorithm for avg traverses p-documents in a bottom-up fashion. It maintains conditional moments of sum for each possible value of count and combines them in two possible ways, according to the node types.[7]

Regarding Claim 2, moments for count and min can be computed directly from the distributions, which can be constructed in polynomial time as sketched in the proof of Theorem 21.2.

Algorithms for sum and countd can be based on a generalisation of the principle of regrouping sums (see Lemma 16) for tree pattern queries. Analogously as for the case of single-path queries, the crucial element for the complexity of the sum-algorithm is the difficulty of computing the probability that a node (or sets of nodes of fixed cardinality) occur in a query result. For tree pattern queries without joins, these probabilities can be computed in polynomial time adapting the techniques in [19]. A variation of this principle, where the probabilities of a given set of values to occur in a query result is computed, gives an algorithm for countd. □

[6]The same problems has been used earlier in [26] to show #P-hardness of evaluating relational queries with countd and avg-constraints.

[7]A technique that is similar in spirit has been presented in [18] for probabilistic streams.

Table 2 gives an overview of data complexity results of this section.

7. APPROXIMATIONS AND SAMPLING

Without loss of generality, we only discuss how to estimate cumulative distributions $\mathsf{Pr}(Q^\alpha(\mathcal{P}) \leqslant c)$ and moments $\mathbb{E}(Q^\alpha(\mathcal{P})^k)$ for aggregate TPJ-queries. Notice that by using cumulative distributions one can also approximate the probability of individual values: to estimate $\mathsf{Pr}(Q^\alpha(\mathcal{P}) = c)$, we estimate $\mathsf{Pr}(Q^\alpha(\mathcal{P}) \leqslant c + \gamma)$ and $\mathsf{Pr}(Q^\alpha(\mathcal{P}) \leqslant c - \gamma)$ for a small γ (that depends on α and $\widehat{\mathcal{P}}$) and subtract the second from the first.

For instance, in order to approximate the cumulative probability $\mathsf{Pr}(Q^{\mathsf{countd}}(\mathcal{P}) \leqslant 100)$, one evaluates the query on independent random samples of worlds of $\widehat{\mathcal{P}}$, and then use the ratio of resulting samples where countd is at most 100 as an estimator. Similarly, for approximating $\mathbb{E}(Q^{\mathsf{countd}}(\mathcal{P}))$, one returns the average of countd over the results.

Using Hoeffding's Bound [14] we obtain the following two propositions for approximating a point for the cumulative distribution of an aggregate query and moments of any degree, respectively.

PROPOSITION 23. *Let Q be an aggregate TPJ-query, $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{cie,mux,det}$ a p-document and $x \in \mathbb{Q}$. Then for any rationals ε, $\delta > 0$, it is sufficient to have $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$ samples so that with probability at least $1-\delta$, the quantity $\mathsf{Pr}(Q(\mathcal{P}) \leqslant x)$ can be estimated with an additive error of ε.*

Observe that the number of samples in Proposition 23 is independent of the size of $\widehat{\mathcal{P}}$. A problem may arise if $\mathsf{Pr}(Q(\mathcal{P}) \leqslant x) \leqslant \varepsilon$, since then an additive error of ε makes the estimate useless. However, for probabilities above a threshold p_0, it is enough to have the number of samples proportional to $1/p_0^2$ (with additive error, say $p_0/10$).

PROPOSITION 24. *Let Q be an aggregate TPJ-query, f a function mapping $\mathbb{Q}$ to $\mathbb{Q}$, such that $f(Q(\mathcal{P}))$ ranges over an interval of width R and $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{cie}$ a p-document. Then, for any rationals ε, $\delta > 0$, it is sufficient to have $O(\frac{R^2}{\varepsilon^2} \log \frac{1}{\delta})$ samples so that, with probability at least $1-\delta$, the quantity $\mathbb{E}(f(Q(\mathcal{P})))$ can be estimated with additive error of ε.*

As a consequence, if Q takes values in $[0, R]$, choosing $f(x) := x^k$ yields that the k-th moment of $Q(\mathcal{P})$ around zero can be estimated with $O(\frac{R^{2k}}{\varepsilon^2} \log \frac{1}{\delta})$ samples.

Observe that if the range R has magnitude polynomial in size of $\widehat{\mathcal{P}}$, then we have a polynomial-time estimation algorithm. For example, to approximate $\mathbb{E}(Q^{\mathsf{countd}}(\widehat{\mathcal{P}}))$ it is enough to draw a quadratic number of samples, since the range R is at most the number of the leaves in $\widehat{\mathcal{P}}$.

8. CONTINUOUS PROBABILISTIC DATA

We generalize p-documents to documents whose leaves are labeled with (representations of) probability distributions over the reals, instead of single values. We give semantics to such documents in terms of continuous distributions over documents with real numbers on their leaves.

***Continuous* px-*Spaces*.** In the discrete case, a p-document defines a finite set of trees and probabilities assigned to them.

In the continuous case, a p-document defines an uncountably infinite set of trees with a continuous distribution, which assigns probabilities to (typically infinite) sets of trees, the possible events, which form a σ-algebra. We refer to a textbook on measure and probability theory such as [5] for the definitions of the concepts used in this section.

From now on, we consider only documents whose leaves are labeled with real numbers. We say that two documents $d = (t, \theta)$ and $d' = (t', \theta')$ are *structurally equivalent,* denoted $d \sim_{st} d'$, if $t = t'$ and $\theta(v) = \theta(v')$ for every v that is not a leaf of t. That is, d and d' differ only in the labels of the leaves. Obviously, $\sim_{st}$ is an equivalence relation on the set of all documents. Intuitively, the structure and the labels of inner nodes fix the structure of a document while the leaves contain values.

A set of documents $\mathcal{D}$ is *structurally finite* (or sf for short) if (1) for any document $d \in \mathcal{D}$ and any d' that is structurally equivalent to d, we have $d' \in \mathcal{D}$; (2) $\mathcal{D}$ consists only of finitely many $\sim_{st}$-equivalence classes. That is, intuitively, if it contains a document d, then it contains also all documents that have the same structure, but different values, and it contains only finitely many structurally distinct documents.

Let $\mathcal{D}$ be an sf set of documents. We define a σ-algebra $\mathcal{A}_\mathcal{D}$ on $\mathcal{D}$ and then probabilities on $\mathcal{A}_\mathcal{D}$ by doing so first for each $\sim_{st}$-class and then for $\mathcal{D}$ as a whole.

Let $d_0 = (t_0, \theta_0)$ be a document, $\bar{l} := (l_1, \ldots, l_k)$ a tuple consisting of the leaf nodes of d_0, and $[d_0]_{\sim_{st}}$ the equivalence class of d_0 under $\sim_{st}$. For every document $d = (t, \theta)$ with $d \in [d_0]_{\sim_{st}}$ we define $\theta(\bar{l}) := (\theta(l_1), \ldots, \theta(l_k))$ a k-tuple of real numbers. In fact, this mapping of tuples of leaf values to tuples of numbers is a bijection between $[d_0]_{\sim_{st}}$ and $\mathbb{R}^k$, which we denote as β. The standard σ-algebra on $\mathbb{R}^k$ is the algebra of Borel sets. We use β to introduce a σ-algebra $\mathcal{A}_0$ on $[d_0]_{\sim_{st}}$. We say that $\mathcal{D}_0 \in \mathcal{A}_0$ for a set $\mathcal{D}_0 \subseteq [d_0]_{\sim_{st}}$ if and only if β maps $\mathcal{D}_0$ to a Borel set of $\mathbb{R}^k$. In the same vein, we can identify probability distributions over $\mathbb{R}^k$ with distributions over $[d_0]_{\sim_{st}}$. Note that, due to symmetry, the definition of $\mathcal{A}_0$ does not depend on the specific order of the leaves that is used by β.

Now, suppose that $\mathcal{D} = \bigcup_{i=1}^{n} [d_i]_{\sim_{st}}$ and that $\mathcal{A}_i$ is the σ-algebra on $[d_i]_{\sim_{st}}$ defined above. Then we define

$$\mathcal{A}_\mathcal{D} := \{\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_n \mid \mathcal{D}_i \in \mathcal{A}_i\}.$$

Clearly, since all the $\mathcal{A}_i$ are σ-algebras, $\mathcal{A}_\mathcal{D}$ is a σ-algebra. Moreover, suppose that for each equivalence class $[d_i]_{\sim_{st}}$ we have a probability distribution Pr_i and that $p_1, \ldots, p_n$ are convex coefficients (that is, $p_i \geqslant 0$ and $p_1 + \cdots + p_n = 1$). Then we define for every $\mathcal{D}' \in \mathcal{A}_\mathcal{D}$

$$\mathsf{Pr}(\mathcal{D}') := \sum_{i=1}^{n} p_i \cdot \mathsf{Pr}_i(\mathcal{D}' \cap [d_i]_{\sim_{st}}).$$

Clearly, Pr is a probability on $\mathcal{A}_\mathcal{D}$. Conversely, every probability Pr over $(\mathcal{D}, \mathcal{A}_\mathcal{D})$ can be uniquely decomposed into probabilities Pr_i over the $\sim_{st}$-classes of $\mathcal{D}$ such that Pr can be obtained from the Pr_i as described above. Moreover, each Pr_i is *essentially* a probability over some $\mathbb{R}^k$.

p-documents. To support (possibly continuous) distributions on leaves, we extend the syntax of p-documents by an additional type of distributional nodes, the *cont* nodes. A *cont* node has the form $cont(D)$, where D is a representation of a probability distribution over the real numbers. In contrast to the distribution nodes introduced earlier, a *cont* node can only appear as a leaf.

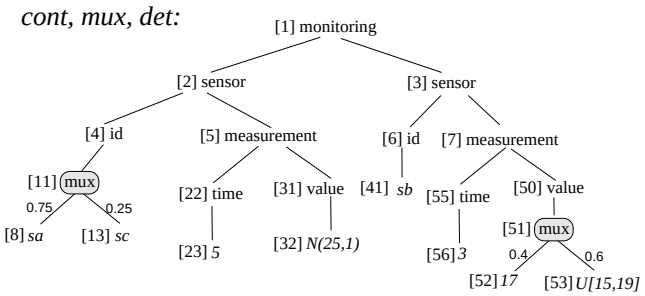

Figure 6: $\mathsf{PrXML}^{cont,mux,det}$ p-document: monitoring.

EXAMPLE 25. *Consider the $\mathsf{PrXML}^{cont,mux,det}$ p-document in Figure 6. The document collects results of (e.g., temperature) monitoring by sensors* `sa`*,* `sb` *and* `sc`*. The data in the document are measurements at time* 3 *by* `sb` *and at time* 5 *by either* `sa` *or* `sc`*. At time* 3 *the measurement is either* 17*, or a value in the interval from* 15 *to* 19*. The fact the latter value is unknown and can be anywhere between* 15 *and* 19 *is represented by a continuous node cont*$(U([15; 19])$*, where U stands for the uniform distribution. We know that both sensors* `sa` *and* `sc` *have an inherent imprecision and the real measurement is normally distributed around the one they sent. We model it by a continuous node with a normal distribution cont*$(N(25; 1))$ *with mean* 25 *and variance* 1.

Any finitely representable distribution can appear in a *cont* node. As an example, we consider in the following piecewise polynomial distributions. A function $f\colon \mathbb{R} \to \mathbb{R}$ is piecewise polynomial if there are points $-\infty = x_0 < x_1 < \ldots < x_m = \infty$ such that for each interval $I_i :=]x_{i-1}, x_i[$, $1 \leqslant i \leqslant m$, the restriction $f_{|I_i}$ of f to I_i is a polynomial. (The points $x_1, \ldots, x_{n-1}$ are the partition points and the intervals $I_1, \ldots, I_m$ are the partition intervals of f.) Every piecewise polynomial function $f \geqslant 1$ with $\int_{-\infty}^{\infty} f = 1$ is the density function of a probability. Clearly, in this case $f_{|I_1}$ and $f_{|I_m}$ are identical to 0. Note that distributions defined by piecewise polynomial densities are a generalization of uniform distributions. Piecewise polynomials are an example of a class of functions stable under convex sum, (classical) convolution, product, and integration. We shall use this stability property to compute the distribution of aggregate query answers.

When the symbol *cont* appears as a superscript of PrXML, possibly in combination with other symbols, it indicates a class of p-documents that have distributions on their leaves. The symbol *cont* can be used with class symbols like the three above as arguments to specify the kind of distributions that can appear.

We define the semantics $[\![\widehat{\mathcal{P}}]\!]$ of continuous p-documents of $\mathsf{PrXML}^{cont,cie,mux,det}$ as a continuous px-space as defined earlier. More precisely, let $\widehat{\mathcal{P}} \in \mathsf{PrXML}^{cont,cie,mux,det}$ and $\widehat{\mathcal{P}}' \in \mathsf{PrXML}^{cie,mux,det}$ be the p-document obtained from $\widehat{\mathcal{P}}$ by replacing every continuous node with an arbitrary value, say, 0. $[\![\widehat{\mathcal{P}}']\!]$ is a (discrete) px-space $(\{d_1 \ldots d_n\}, \{p_1 \ldots p_n\})$ with $\sum p_i = 1$. For a given $1 \leqslant i \leqslant n$, we consider the document $\widehat{\mathcal{P}}_i$ of PrXML^{cont} obtained by putting back in d_i the continuous nodes of $\widehat{\mathcal{P}}$, where the corresponding leaves

still exist. Let $D_{i1} \dots D_{ik}$ be the k probability distributions over the real numbers represented in the *cont* nodes of $\widehat{\mathcal{P}}_i$. We define then a continuous probability distribution Pr_i over $\mathbb{R}^k$ as the product distribution [5] of the D_{ij}'s, i.e., the unique distribution such that $\mathsf{Pr}_i(X_1 \times \cdots \times X_k) = D_{i1}(X_1) \times \cdots \times D_{ik}(X_k)$. Using the inverse of the bijection β discussed earlier, Pr_i can be translated into a probability distribution over $[d_i]_{\sim_{st}}$, the equivalence class of d_i under $\sim_{st}$. Let $\mathcal{D} = \cup_{i=1}^{n}[d_i]_{\sim_{st}}$. We then define as already discussed the probability distribution Pr of $[\![\widehat{\mathcal{P}}]\!]$ on the σ-algebra $\mathcal{A}_D$ as:

$$\mathsf{Pr}(\mathcal{D}') := \sum_{i=1}^{n} p_i \cdot \mathsf{Pr}_i(\mathcal{D}' \cap [d_i]_{\sim_{st}}).$$

Aggregating Continuous Probabilistic Data . Having defined the semantics of continuous p-documents, we now show how the results for aggregate queries obtained in the discrete case can be lifted to the continuous case. Our purpose here is not to give a comprehensive picture of the complexity, as in the discrete case, but to see what kind of tractability results can be obtained. Let us restrict ourself to monoid aggregate functions, and p-documents of $\mathsf{PrXML}^{cont,mux,det}$, which is our main case of tractability in the discrete case. For simplicity, we only deal with single-path queries.

The following result is at the basis of the tractability of monoid aggregate query evaluation in $\mathsf{PrXML}^{cont,mux,det}$.

PROPOSITION 26. *Let X, Y be independent real-valued random variables with probability density functions f, g and cumulative distribution functions F, G (i.e., $F = \int f$, $G = \int g$). We have:*

1. *The density function of $X + Y$ is $f * g$, the convolution of f and g.*
2. *The cumulative distribution function of $\mathsf{max}(X, Y)$ is $F \times G$.*
3. *The cumulative distribution function of $\mathsf{min}(X, Y)$ is $F + G - F \times G$.*

Obviously, there is no hope of computing probabilities of aggregate query answers if it is not possible to somehow combine (either symbolically or numerically) the probability distributions of the leaves. The preceding result hints that if we are able to efficiently apply a number of basic operations on our probability distribution functions, we are able to compute the distribution of the min, max or sum. The following operations are required: convex sums (for *mux* nodes); convolution (for sum, in conjunction with *det* nodes); integration and multiplication (for min and max, in conjunction with *det* nodes). One simple case where we can perform these operations efficiently is when *cont* leaves are piece-wise polynomials of a bounded degree. For a fixed $K > 0$ let $\mathsf{PP}(K)$ be the set of all piecewise polynomial probability distributions whose polynomials have degree $\leqslant K$. It is reasonable to assume that such a bound K exists for every application. This bound ensures that the piecewise polynomial representing the distribution of the query answer has degree polynomial in the size of the document. Hence:

THEOREM 27. *For p-documents in $\mathsf{PrXML}^{cont,mux,det}$ that are labeled with distributions in $\mathsf{PP}(K)$ we have:*

1. *The distribution of results of queries in $\mathsf{SP}^{\mathsf{sum}}$ can be computed in polynomial time in the combined size of the input and the output.*
2. *The distribution of results of queries in $\mathsf{SP}^{\mathsf{max}}$ and $\mathsf{SP}^{\mathsf{min}}$ can be computed in polynomial time.*
3. *All moments of results of queries in $\mathsf{SP}^{\mathsf{sum}}$, $\mathsf{SP}^{\mathsf{max}}$, and $\mathsf{SP}^{\mathsf{min}}$ can be computed in polynomial time.*

Other results from the discrete case can be generalized to the continuous case. For example, it can be shown that moments of queries in $\mathsf{TP}^{\mathsf{sum}}$ can be computed in polynomial time over $\mathsf{PrXML}^{cont,mux,det}$ (and similarly for $\mathsf{SP}^{\mathsf{sum}}$ and $\mathsf{PrXML}^{cont,cie}$), by replacing the *cont* nodes by the expected value of the represented distribution.

9. RELATED WORK AND CONCLUSION

Related Work. The probabilistic XML models that have been proposed in the literature can be grouped in two main categories, depending on the kind of supported probabilistic dependencies: $\mathsf{PrXML}^{mux,det}$-like *local* dependencies [15, 16, 23, 28], or PrXML^{cie}-like *global* dependencies [3, 27], in the spirit of c-tables [17]. We used here the unifying framework of [2, 19].

The complexity of non-aggregate query answering over $\mathsf{PrXML}^{mux,det}$ and PrXML^{cie} has been investigated in [19–21, 27]. Several results presented here either extend or use these works. The dynamic-programming algorithm for computing the probability of a Boolean tree-pattern query from [19–21] is in particular used for Claim 2 of Theorem 22. The same authors have also studied in [8] the problem of tree-pattern query answering over $\mathsf{PrXML}^{mux,det}$ documents with constraints expressed using aggregate functions, i.e., something similar to the `HAVING` queries of SQL. We use their results for proving Claim 2 of Theorem 21.

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 [22], for data exchange [4] and for ontologies [6]. In probabilistic settings, to the best of our knowledge, in addition to the aforementioned [8], only [26] studies aggregate queries. Ré and Suciu consider the problem of evaluating `HAVING` queries (using aggregate functions) in "block-independent databases", which are roughly $\mathsf{PrXML}^{mux,det}$ restricted to relations (limited-depth trees). The complexity bounds of Claim 3 of Theorem 21 use similar arguments than the corresponding results for block-independent databases presented in [26]. In both [8] and [26], the authors discuss the filtering of possible words that do not satisfy a condition expressed using aggregate functions, and do not consider the problem of computing the distribution of the aggregation, or moments thereof. Computation of the expected value of aggregate functions over a data stream of probabilistically independent data items is considered in [18]. This is a simpler setting than ours, but we use similar techniques in the proof of Theorem 22.

There is very little earlier work on querying continuous probability distributions. The authors of [12] build a (continuous) probabilistic model of a sensor network to run subsequent queries on the model instead of the original data. In [7], algorithms are proposed for answering simple classes of queries over uncertain information, typically given by a sensor network. As noted in a recent survey on probabilistic relational databases [11], "although probabilistic databases with continuous attributes are needed in some applications, no formal semantics in terms of possible worlds has been

proposed so far". We proposed in Section 8 such a formal semantics.

Conclusion. We provided algorithms for, and a characterization of the complexity of, computing aggregate queries for both $\mathsf{PrXML}^{mux,det}$ and PrXML^{cie} models, i.e., very general and most interesting probabilistic XML models. We also considered the expected value and other moments, i.e., summaries of the probability distribution of the results of aggregate functions. In the case of $\mathsf{PrXML}^{mux,det}$, we have identified a fundamental property of aggregate functions, that of being *monoid*, that entails tractability. The complexity of aggregate computations in many cases has led us to introduce polynomial-time randomized approximation schemes. Finally, a last original contribution has been the definition of a formal continuous extension of probabilistic XML models. We have shown how some of the results of the discrete case can be adapted.

Because our work has many facets, it may be extended in a number of directions. First, we intend to implement a system that manages imprecise data with aggregate functions. In particular, we want the system to handle continuous probabilities, which are quite useful in practice. A main novelty of the present work is the use of continuous probabilities for data values. We are currently developing the theory in this direction. Finally, observe that although a p-document (with continuous probabilities) represents uncountably infinite possible worlds, they only have finitely many possible structural equivalence classes, and in particular, they all are of bounded height and width. It would be interesting to investigate extensions of the model without this restriction.

References

[1] S. Abiteboul, T.-H. H. Chan, E. Kharlamov, W. Nutt, and P. Senellart. Agrégation de documents XML probabilistes. In *Proc. BDA*, Namur, Belgium, Oct. 2009. Conference without formal proceedings.

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

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

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

[5] R. B. Ash and C. A. Doléans-Dade. *Probability & Measure Theory*. Academic Press, San Diego, CA, USA, 2000.

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

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

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

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

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

[11] N. Dalvi, C. Ré, and D. Suciu. Probabilistic databases: Diamonds in the dirt. *Communications of the ACM*, 52(7), 2009.

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

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

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

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

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

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

[18] T. S. Jayram, S. Kale, and E. Vee. Efficient aggregation algorithms for probabilistic data. In *Proc. SODA*, New Orleans, LA, USA, Jan. 2007.

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

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

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

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

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

[24] C. H. Papadimitriou. *Computational Complexity*. Addison Wesley, Reading, USA, 1994.

[25] 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.

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

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

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