Probabilistic XML: Models and Complexity

Benny Kimelfeld and Pierre Senellart

Abstract. Uncertainty in data naturally arises in various applications, such as data integration and Web information extraction. Probabilistic XML is one of the concepts that have been proposed to model and manage various kinds of uncertain data. In essence, a probabilistic XML document is a compact representation of a probability distribution over ordinary XML documents. Various models of probabilistic XML provide different languages, with various degrees of expressiveness, for such compact representations. Beyond representation, probabilistic XML systems are expected to support data management in a way that properly reflects the uncertainty. For instance, query evaluation entails probabilistic inference, and update operations need to properly change the entire probability space. Efficiently and effectively accomplishing data-management tasks in that manner is a major technical challenge. This chapter reviews the literature on probabilistic XML. Specifically, this chapter discusses the probabilistic XML models that have been proposed, and the complexity of query evaluation therein. Also discussed are other data-management tasks like updates and compression, as well as systemic and implementation aspects.

1 Introduction

Data managed by modern database applications are often *uncertain*. A few examples are the following. When information from different sources is conflicting,

Benny Kimelfeld
IBM Research–Almaden
San Jose, CA 95120, USA
e-mail: kimelfeld@us.ibm.com

Pierre Senellart
Institut Télécom; Télécom ParisTech; CNRS LTCI
Paris, France
e-mail: pierre.senellart@telecom-paristech.fr

Z. Ma & L. Yan (Eds.): *Advances in Probabilistic Databases*, STUDFUZZ 304, pp. 39–66.
DOI: 10.1007/978-3-642-37509-5_3

inconsistent, or simply presented in incompatible forms, the result of integrating these sources necessarily involves uncertainty as to which fact is correct or which is the best mapping to a global schema. When data result from automatic and imprecise tasks, such as information extraction, data mining, or computer vision, it is commonly annotated by a score representing the *confidence* of the system in the correctness of the data. When data are gathered from sensor networks, they come with the inherent imprecision in the measurement of sensors. Even when data is generated by humans, they are not necessarily certain: diagnostics of diseases stored in a hospital database are affected by the imprecision of human judgment (in addition to that of the diagnostics themselves). This ubiquity of uncertain data is all the truer when one deals with the World Wide Web, which is a heterogeneous collection of data that is constantly updated by individuals and automated processes.

Data uncertainty is often ignored, or modeled in a specific, per-application manner. This may be an unsatisfying solution in the long run, especially when the uncertainty needs to be retained throughout complex and potentially imprecise processing of the data. As an example, consider sensor data being gathered in a database, mined to extract interesting patterns, annotated by human experts, then integrated together with the result of other such analyses, independently made. Each of these steps, from the initial collection to the final integration, should be aware of the uncertain character of handled data; furthermore, each of these steps may even introduce further uncertainty. The goal of uncertain data management is to provide a unifying framework and a unifying system to handle the semantics of uncertainty, in the database itself. This goal is in line with the motivation behind DBMSs themselves, which were proposed in the 1970s as a uniform solution to the problem of managing data, while replacing previous systems that were tied to particular applications (e.g., accounting, cataloging, etc.).

Naturally, there are various ways to model uncertainty. Examples include representation of missing information (from SQL NULLs to more elaborate models of *incomplete data* [34]), fuzzy logic and fuzzy sets [29], and the Dempster-Shafer theory [70]. In this chapter, we consider *probabilistic* models that represent probability distributions over ordinary databases, and are based on the rich mathematical formalism of probability theory. Of course, quite a few real-life applications provide data that are probabilistic in nature. Examples include conditional random fields [46] (used in information extraction) and statistics-based tasks involved in natural-language processing [50]. But even when applications do not provide obvious probabilistic data, they often provide confidence scores that can be mapped to probability values.

Representing and Querying Probabilistic Databases

A *probabilistic database* is, conceptually, a probability space over ordinary databases, each of which is called a *possible world* [22]. In practice, such a probability space is obtained by introducing uncertainty about the value (or existence) of individual

data items. If there are many such uncertain data items, then the number of possible worlds may be too large to manage or even to store. However, applications usually make assumptions on the correlation among the uncertain items (e.g., independence), and such assumptions typically allow for a substantially smaller (e.g., logarithmic-size) representation of the probability space. Hence, from a database point of view, the goal is to provide a proper language and underlying data model to specify and represent probabilistic databases in a *compact* manner (e.g., by building in the model assumptions of independence).

But a database is not just about storage. A central role of a database is to provide a clean, easy, and general way of accessing its data (while abstracting away from the actual implementation). In particular, a database supports a high-level language like SQL or XQuery. In the context of probabilistic databases, the correspondent of querying is that of finding *events* and *inferring* their probabilities. Hence, we would like the database not just to "store probabilities," but to actually understand their semantics and support inference tasks. The common realization of that [22, 31, 45, 53] is to allow the user to phrase ordinary queries (of the kind she would pose to an ordinary database), while the database associates each query answer with its computed probability (i.e., the probability that the answer holds true in a random possible world).

Finally, another central task of a database is to provide high performance for the operations it supports (e.g., query evaluation). This aspect is particularly challenging in the case of a probabilistic database, due to the magnitude of the actual probability space that such a database can (compactly) represent. As an example, for query evaluation (under the semantics mentioned in the previous paragraph), the baseline way of computing the probabilities is through the enumeration of all possible worlds, which is prohibitively intractable. Hence, we would like the operations to be performed on the compact representation itself rather than on the possible worlds. From the theoretical-complexity point of view, we require efficiency to be under the assumption that the input consists of the database in its compact form; in particular, "polynomial-time" is in the size of the compact representation, and not in that of the implied probability space. Not surprisingly, this requirement leads very quickly to computational hardness [22] (and sometimes even hardness of approximation [26, 45]). However, as we discuss throughout the chapter, in the case of probabilistic XML there are a few important settings where querying is tractable.

There is a rich literature on probabilistic relational databases [17, 21, 32, 35, 56, 61, 67]. In contrast, here we discuss *probabilistic XML* models, which represent probabilistic spaces over labeled trees. XML is naturally adapted to a number of applications where data is tree-like, including Web data or natural language parsing, and we give next specific examples of applications of probabilistic XML models. Later in this chapter (Section 6.3), we discuss the connection between probabilistic relational models and probabilistic XML models.

Probabilistic XML Applications

Following are concrete examples of applications or application areas where probabilistic XML is a natural data model and, moreover, the need to *query* probabilistic XML arises.

- **XML data integration.** Assume that a number of sources on the Web export XML information in potentially different schemas. To represent the result of the integration, we need a way to capture the uncertainty in the schema mappings, in deduplication, or in resolving conflicting information. This uncertainty can be characterized by probabilistic mappings [26] and probabilistic data integration rules [38, 39]. The outcome of the integration process can naturally be viewed as probabilistic XML (which is useful to query, update, and so on).
- **Web information extraction.** Extracting information from Web data means detecting, in a Web page, instances of concepts, or relations between these instances, based on the content or structure of these Web pages. A typical output is therefore a tree-like document, with local annotations about extracted information. Current extraction techniques, whether they are unsupervised or rely on training examples, are by nature imprecise, and several possible annotations might be produced for the same part of the Web page, with confidence scores. This is for instance the case with conditional random fields for XML [36] that produce probabilistic labels for part of the original HTML document; probabilistic XML is a natural way to represent that.
- **Natural language parsing.** Parsing natural language consists in building syntax trees out of sentences. This is an uncertain operation, because of the complexity of the natural language, and its inherent ambiguity. Indeed, some sentences like "I saw her duck" have several possible syntax trees. A parser will typically rely on statistics gathered from corpora to assign probabilities to the different possible parse trees of a sentence [50]. This probability space of parse trees can then be seen as probabilistic XML data [18].
- **Uncertainty in collaborative editing.** Consider users collaborating to edit documentation structured in sections, subsections, paragraphs and so on, as in the online encyclopedia Wikipedia. In an open environment, some of these contributions may be incorrect, or even spam and vandalism. If we have some way to estimate the trustworthiness of a contributor, we can represent each individual edit as an uncertain operation on a probabilistic XML document that represents the integration of all previous edits [1].
- **Probabilistic summaries of XML corpora.** Querying and mining a large corpus of XML documents (e.g., the content of the DBLP bibliography) can be time-consuming. If we are able to summarize this corpus as a compact probabilistic model [6], namely probabilistic XML, we can then use this model to get (approximations of) the result of querying or mining operations on the original corpus.

Organization

The remaining of this chapter is organized as follows. We first introduce the basic concepts, mainly XML, probabilistic XML, p-documents, and ProTDB as our main example of a concrete p-document model (Section 2). Next, we talk about querying probabilistic documents in general, and within ProTDB in particular (Section 3). We then review and discuss additional models (Section 4) and additional problems of interest (Section 5). Finally, we discuss practical aspects of probabilistic XML systems (Section 6) and conclude (Section 7).

As a complement to this chapter, we maintain an updated list of resources (especially, a hyperlinked bibliography) pertaining to probabilistic XML online at `http://www.probabilistic-xml.org/`.

2 Probabilistic XML

In this section, we describe the formal setting of this chapter, and in particular give the formal definitions of our basic concepts: an (ordinary) XML document, a probabilistic XML space, and the p-document representation of probabilistic XML.

2.1 XML Documents

We assume an infinite set Σ of *labels*, where a label in Σ can represent an XML tag, an XML attribute, a textual value embedded within an XML element, or the value of an attribute. The assumption that Σ is infinite is done for the sake of complexity analysis. An *XML document* (or just *document* for short) is a (finite) directed and ordered tree, where each node has a label from Σ. The label of a document node v is denoted by $\text{label}(v)$. We denote by $\mathbf{D}_\Sigma$ the (infinite) set of all documents.

As an example, the bottom part of Figure 1 shows a document d. In this figure, as well as in other figures, labels that represent textual values (e.g., "*car financing*") are written in italic font, as opposed to labels that represent tags (e.g., "title"), which are written in normal font. Note that the direction of edges is not explicitly specified, and is assumed to be downward. Similarly, order among siblings is assumed to be left-to-right.

2.2 px-Spaces

A *probabilistic XML space*, abbreviated *px-space*, is a probability space over documents. Although we will briefly discuss *continuous* px-spaces (Section 4.5), our focus is mainly on *discrete* px-spaces. So, unless stated otherwise, we will implicitly assume that a px-space is discrete. Specifically, we view a px-space as a pair $\mathscr{X} = (D, p)$, where D is a finite or countably infinite set of documents, and $p : D \to [0,1]$ is a *probability function* satisfying $\sum_{d \in D} p(d) = 1$. The *support* of a

px-space $\mathscr{X} = (D, p)$ is the set of documents $d \in D$, such that $p(d) > 0$. We say that the px-space $\mathscr{X}$ is *finite* if $\mathscr{X}$ has a finite support; otherwise, $\mathscr{X}$ is *infinite*.

When there is no risk of ambiguity, we may abuse our notation and identify a px-space $\mathscr{X}$ by the random variable that gets a document chosen according to the distribution of $\mathscr{X}$. So, for example, if $\mathscr{X} = (D, p)$ and d is a document, then $\Pr(\mathscr{X} = d)$ (in words, *the probability that* $\mathscr{X}$ *is equal to* d) is $p(d)$ if $d \in D$, and 0 otherwise.

2.3 p-Documents

A px-space is encoded by means of a *compact representation*. Later in this chapter, we will discuss the plethora of representation models proposed and studied in the literature. The basic notion underlying most of those models is that of a *p-document* [4, 43].

Formally, a p-document is a tree $\mathscr{P}$ that is similar to an XML document, except that $\mathscr{P}$ has a distinguished set of *distributional nodes* in addition to the *ordinary nodes* (that have labels from Σ). The ordinary nodes of $\mathscr{P}$ may belong to documents in the encoded px-space. Distributional nodes, on the other hand, are used only for defining the probabilistic process that generates random documents (but they do not actually occur in those documents). As an example, Figure 1 shows a p-document $\mathscr{P}$, where the distributional nodes are the ones represented by boxes with rounded corners (and denoted by v_1, v_2, and so on). The words ind and mux inside those boxes will be discussed later. Each distributional node specifies a probability distribution over subsets of its children; later on, we will define several *types* of distributional nodes (like ind and mux), where each type defines the way these distributions are encoded. In the probabilistic process that generates a random document, a distributional node randomly chooses a subset of its children according to the distribution specified for that node. The root and leaves of a p-document are required to be ordinary nodes.

Next, we describe the px-space (D, p) defined by a p-document $\mathscr{P}$ by specifying a sampling process that generates a random document. Note that such a process well defines the px-space (D, p) as follows: D consists of all the documents that can be produced in this process, and $p(d)$ (where $d \in D$) is the probability that d is obtained in this process.

The random document is generated by the p-document $\mathscr{P}$ in two steps. First, each distributional node of $\mathscr{P}$ randomly chooses a subset of its children. Note that the choices of different nodes are not necessarily probabilistically independent. All the unchosen children and their descendants (even descendants that have been chosen by their own parents) are deleted. The second step removes all the distributional nodes. If an ordinary node u remains, but its parent is removed, then the new parent of u is the lowest ordinary node v of $\mathscr{P}$, such that v is a proper ancestor of u. Note that two different applications of the first step may result in the same random document generated (and for further discussion on that, see [43]).

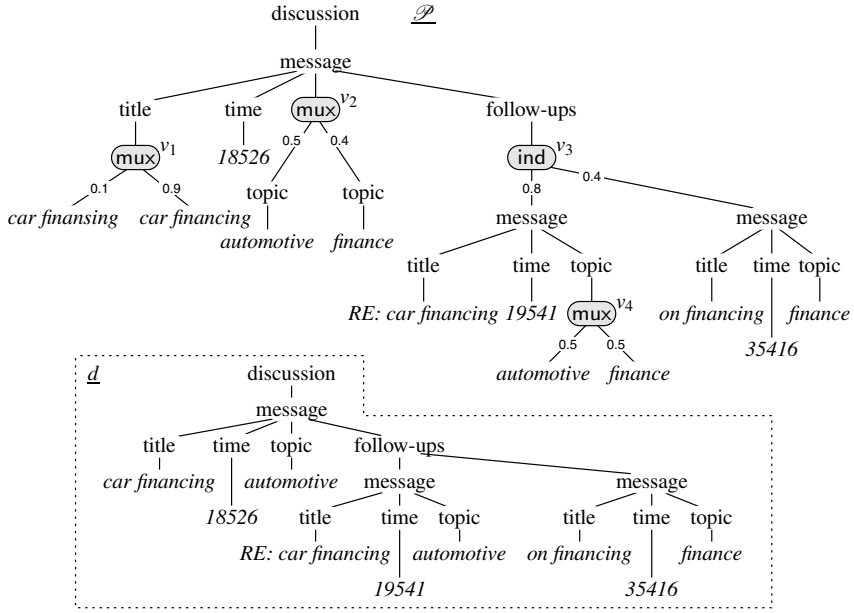

Fig. 1 A p-document $\mathscr{P}$ in $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ (top) and a sample document d of $\mathscr{P}$ (bottom)

A Concrete Model: ProTDB

We now construct a concrete model of p-documents, namely, the ProTDB model [53]. For that, we define two types of distributional nodes. Recall that when defining a type of distributional nodes, we need to specify the encoding and meaning of the random process in which a distributional node of that type selects children. In Section 4, we will define additional types of distributional nodes (hence, additional concrete models).

A ProTDB document has two types of distributional nodes.

- ind: A distributional node v of type ind specifies for each child w, the probability of choosing w. This choice is independent of the other choices of children, of either v or other distributional nodes in the p-document.
- mux: A distributional node v of type mux chooses at most one child w (that is, different children are mutually exclusive, hence the name mux) with a specified probability for w. We require the sum of probabilities along the children of v to be at most 1; the complement of this sum of probabilities is the probability that v chooses none of its children.

Example 1. The top part of Figure 1 shows a ProTDB p-document $\mathscr{P}$. The type of each distributional node is written in the corresponding box. For instance, node v_1 is a distributional node of type mux; as shown by the numbers on its outgoing edges, v_1 chooses its left child and right child with probability 0.1 and 0.9, respectively. Note that the mux node v_2 chooses none of its children with probability 0.1

$(= 1 - 0.4 - 0.5)$. Finally, observe that the ind node v_3 makes independent choices about its two children; for example, it chooses *just* the left child with probability $0.8 \times (1 - 0.4)$, both children with probability 0.8×0.4, and no children at all with probability $(1 - 0.8) \times (1 - 0.4)$.

In the bottom, Figure 1 shows a sample document d of $\mathscr{P}$. Let us now compute the probability of d. For d to be produced, the following independent events should take place:

- v_1 chooses its right child. This event occurs with probability 0.9.
- v_2 chooses its left child. This event occurs with probability 0.5.
- v_3 chooses both of its children. This event occurs with probability $0.8 \times 0.4 = 0.32$.
- v_4 chooses its right child. This event occurs with probability 0.5.

Hence, the probability of d is given by

$$\Pr(\mathscr{P} = d) = 0.9 \times 0.5 \times 0.32 \times 0.5 = 0.072\,. \qquad \square$$

We follow the conventional notation [4] that, given k types type_1, type_2, …, type_k of distributional nodes (such as ind, mux, and the types that we define later), $\mathsf{PrXML}^{\{\text{type}_1,\text{type}_2,\ldots,\text{type}_k\}}$ denotes the model of p-documents that use distributional nodes only among type_1, type_2, …, type_k. Hence, under this notation ProTDB is the model $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ (and for the p-document $\mathscr{P}$ of Figure 1 we have $\mathscr{P} \in \mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$). Observe that $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ strictly contains $\mathsf{PrXML}^{\{\mathsf{ind}\}}$, $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ and $\mathsf{PrXML}^{\{\}}$ (which is the set $\mathbf{D}_\Sigma$ of ordinary documents).

3 Query Evaluation

In this section, we discuss a central aspect in the management of probabilistic XML—query evaluation. In general, a query Q maps a document d to a value $Q(d)$ in some domain dom_Q; that is, a query is a function $Q : \mathbf{D}_\Sigma \to dom_Q$. As an example, in the case of a *Boolean* query, dom_Q is the set $\{\textbf{true}, \textbf{false}\}$; in that case we may write $d \models Q$ instead of $Q(d) = \textbf{true}$ (and $d \not\models Q$ instead of $Q(d) = \textbf{false}$). In the case of an *aggregate* query, dom_Q is usually the set $\mathbb{Q}$ of rational numbers. Later on, we discuss additional types of queries.

A px-space $\mathscr{X}$ and a query Q naturally define a probability distribution over dom_Q, where the probability of a value $a \in dom_Q$ is given by $\Pr(Q(\mathscr{X}) = a)$. We usually follow the conventional semantics [22] that, when evaluating Q over $\mathscr{X}$, the output represents that distribution. For example, if Q is a Boolean query, then the goal is to compute the number $\Pr(\mathscr{X} \models Q)$.

3.1 Query Languages

We now describe the languages of queries that capture the focus of this chapter: tree-pattern queries, monadic second-order queries, and aggregate queries.

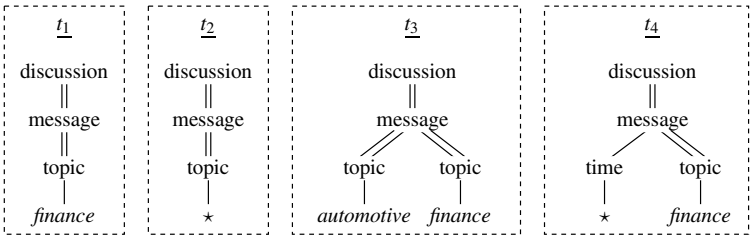

Fig. 2 Tree patterns

3.1.1 Tree-Pattern Queries

Tree-pattern queries (a.k.a. *twig queries* [9, 12]), or just *tree patterns* for short, correspond to the navigational fragment of XPath restricted to child and descendant edges. Specifically, a tree pattern is a Boolean query that is represented by a tree t with *child* and *descendant* edges. In our figures, child and descendant edges are depicted by single and double lines, respectively. Each node of the tree t is labeled with either a label of Σ or with the special *wildcard* symbol $\star$ (and we assume that $\star \notin \Sigma$). A *match* of a tree pattern t in a document d is a mapping μ from the nodes of t to those of d, such that μ maps root to root, child edges to edges, and descendant edges to paths (with at least one edge); furthermore, μ preserves labels of Σ, that is, for a node v of t, if $\text{label}(v) \neq \star$ then $\text{label}(v) = \text{label}(\mu(v))$. Note that a tree pattern ignores the order among siblings in a document. (Queries that take sibling order into account will be discussed in the next section.)

Example 2. Four tree patterns, $t_1, \ldots, t_4$, are shown in Figure 2. Child and descendant edges are represented by single and double lines, respectively. As in documents (and p-documents), edges are implicitly directed top down. As specific examples, let us consider the patterns t_1 and t_4. The pattern t_1 says that some message in the document has a topic descendant (where this topic can be that of the message or of a descendant message) with a child *finance*. The pattern t_4 is the same, except that it also requires the message to have a time child, and the time child to have a child (any child, as indicated by $\star$) of its own. □

Tree patterns are often used not just as Boolean queries, but also as queries that produce tuples of nodes (or tuples of labels). Informally speaking, these tuples are obtained by projecting the matches to a selected sequence of nodes of the tree pattern. For the sake of simplicity, here we restrict the discussion to the Boolean case. It is important to note, though, that under the standard notion of query evaluation for such queries[1] [22], evaluating a non-Boolean tree pattern reduces in polynomial time to evaluating a Boolean one [45].

[1] Under this notion, the output consists of every possible result tuple $\mathbf{a}$ and its marginal probability $\Pr(\mathbf{a} \in Q(\mathscr{X}))$.

3.1.2 Monadic Second-Order Tree Logic (MSO)

A language that is far more expressive than tree patterns is that of *Monadic Second-Order tree logic* (MSO). A query in MSO is a Boolean formula over the document nodes. The vocabulary includes two binary relations over nodes x and y: "x is the parent of y," denoted $E(x,y)$, and "x is a following sibling of y," denoted $y < x$. For each label $\lambda \in \Sigma$, the vocabulary includes also the unary relation "λ is the label of x," denoted $\lambda(x)$. The formula is in first-order logic, extended with quantification over set variables. This quantification allows, among other things, to express conditions on the set of all ancestors or descendants of a node, or on that of all nodes following a given node in document order. For a more formal definition the reader is referred to the vast literature on MSO for trees (e.g., Neven and Schwentick [52]).

Example 3. For illustration, the following MSO query says that there is a message with two descendants that are consecutive sibling messages on the topic *finance*.

$$\begin{aligned}\exists x, y_1, y_2 [&\text{message}(x) \wedge \text{message}(y_1) \wedge \text{message}(y_2) \\ &\wedge \underline{\textit{descendant}}(x, y_1) \wedge \underline{\textit{descendant}}(x, y_2) \wedge \underline{\textit{next-sibling}}(y_1, y_2) \\ &\wedge \underline{\textit{finance-topic}}(y_1) \wedge \underline{\textit{finance-topic}}(y_2)]\end{aligned}$$

In the formula above, $\underline{\textit{descendant}}(x,y)$ is phrased in MSO as follows.

$$\forall S[S(x) \wedge \forall z_1, z_2 (S(z_1) \wedge E(z_1, z_2) \rightarrow S(z_2)) \rightarrow S(y)]$$

Similarly, $\underline{\textit{next-sibling}}(y_1, y_2)$ is given by $y_1 < y_2 \wedge \neg \exists z [y_1 < z < y_2]$. Finally, $\underline{\textit{finance-topic}}(y)$ is phrased in MSO as follows.

$$\exists z, w [E(y,z) \wedge E(z,w) \wedge \text{topic}(z) \wedge \textit{finance}(w)] \qquad \square$$

MSO queries are closely related to the notion of a *(bottom-up) nondeterministic tree automaton* (NTA). Specifically, every MSO query can be translated into an NTA, such that the documents that satisfy the MSO query are precisely those that are accepted by the NTA; conversely, every NTA can be similarly translated into an MSO query [23, 52, 62].

3.1.3 Join Queries

Both tree patterns and MSO queries can be extended by adding *value joins* that test whether two nodes (for tree patterns), or two first-order variables (for MSO), have the same label. Value joins are fairly commonly used in XPath[2]; for instance, they allow us to dereference identifiers used as foreign keys.

Example 4. The following query in MSO extended with the *same-label* predicate tests whether two messages that are descendant of each other have the same topic:

[2] The first version of the XPath language only supports a limited form of value joins, but this restriction is lifted in the latest version.

$$\exists x_1,x_2,x_3,y_1,y_2,y_3[\text{message}(x_1)\wedge\text{message}(y_1)\wedge\underline{\textit{descendant}}(x_1,y_1)$$
$$\wedge E(x_1,x_2)\wedge\text{topic}(x_2)\wedge E(x_2,x_3)$$
$$\wedge E(y_1,y_2)\wedge\text{topic}(y_2)\wedge E(y_2,y_3)$$
$$\wedge \textit{same-label}(x_3,y_3)] \qquad \square$$

3.1.4 Aggregate Queries

In this chapter, an *aggregate function* is a function α that takes as input a set V of document nodes, and returns as output a numerical (rational) number $\alpha(V)\in\mathbb{Q}$. Some of the aggregate functions we consider, like sum, need to assume that the label of a node is a number; to accommodate that, we fix a function *num* over the document nodes, such that $num(v)=\text{label}(v)$ if $\text{label}(v)$ is a number, and otherwise, we arbitrarily determine $num(v)=0$. Specifically, we will discuss the following aggregate functions.

- Count: $\mathsf{count}(V)\stackrel{\text{def}}{=}|V|$.
- Count distinct: $\mathsf{countd}(V)\stackrel{\text{def}}{=}|\{\text{label}(v)\mid v\in V\}|$; that is, $\mathsf{countd}(V)$ is the number of distinct labels that occur in V, regardless of the multiplicity of these labels.
- Sum: $\mathsf{sum}(V)\stackrel{\text{def}}{=}\sum_{v\in V} num(v)$.
- Average: $\mathsf{avg}(V)\stackrel{\text{def}}{=}\mathsf{sum}(V)/|V|$; if V is empty, then $\mathsf{avg}(V)$ is *undefined*.
- Min/max: $\mathsf{min}(V)\stackrel{\text{def}}{=}\min_{v\in V} num(v)$, $\mathsf{max}(V)\stackrel{\text{def}}{=}\max_{v\in V} num(v)$.

An *aggregate query* applies an aggregate function to the set of nodes that is selected by another query (of a different type). Specifically, here we consider aggregate queries that we write as $\alpha\circ t[w]$, where α is an aggregate function, t is a tree pattern, and w is a node of t. The evaluation of $\alpha\circ t[w]$ over a document d results in the number $\alpha(V)$, where V is the set of nodes v of d, such that there exists a match μ of t in d with $\mu(w)=v$; that is:

$$\alpha\circ t[w](d)\stackrel{\text{def}}{=}\alpha(\{v\mid \mu(w)=v \text{ for some match } \mu \text{ of } t \text{ in } d\})$$

Example 5. Consider the tree pattern t_2 of Figure 2, and let w be the wildcard (denoted $\star$) node. When applied to the document d of Figure 1, the query $\mathsf{count}\circ t_2[w]$ returns 3, which is the number of nodes with a "topic" parent. In contrast, $\mathsf{countd}\circ t_2[w](d)$ is 2, which is the number of *distinct* topics (i.e., distinct labels of nodes with a "topic" parent) in d.

As another example, consider the tree pattern t_4 of Figure 2 and, again, let w be the wildcard node. The query $\mathsf{min}\circ t_4[w]$ returns the earliest time of a message that has a descendant message on the topic *finance*; hence, $\mathsf{min}\circ t_4[w](d)=18526$. $\square$

3.2 Complexity for ProTDB

Nierman and Jagadish [53] studied the evaluation of (non-Boolean) tree patterns without projection, and showed computability in polynomial time. Although projection leads to hardness in the relational probabilistic model [22], Kimelfeld et

al. [45] showed that tree patterns with projection, and in particular Boolean tree patterns, can be evaluated in polynomial time in ProTDB [45]. Cohen et al. [20] extended this result to MSO queries. The main reason behind this tractability is that it is possible to evaluate queries directly over a ProTDB tree in a bottom-up manner, making use of the locality of both the p-document and the query. This can be done using dynamic programming for tree patterns [43], and through the computation of a product automaton of the query and the p-document in the MSO case [10].

Theorem 1. [20] *Let Q be an MSO query (e.g., a tree pattern). The problem "compute $\Pr(\mathscr{P} \models Q)$ given $\mathscr{P} \in \mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$" is in polynomial time.*

Observe that Theorem 1 is phrased in terms of *data complexity* [65], which means that the query is held fixed. As mentioned by Kimelfeld et al. [45], the evaluation of tree patterns becomes intractable if the query is given as part of the input. Actually, it was shown [20, 45] that over ProTDB the evaluation of tree patterns, and even MSO queries, is *fixed-parameter tractable* (abbr. FPT) [24], which means that only the coefficients (rather than the degree) of the polynomial depend on the query[3] (hence, FPT is stronger than "polynomial data complexity"). Nevertheless, while for tree patterns this dependence is "merely" exponential, for general MSO queries this dependence is not any elementary function (unless $\mathrm{P} \neq \mathrm{NP}$), since that is already the case when the p-document is ordinary (deterministic) [28, 51].

Tractability (in terms of data complexity) is lost when tree patterns are extended with (value) joins [5]. This is not surprising, for the following reason. Tree patterns with joins over trees can simulate Conjunctive Queries (CQs) over relations. Moreover, tree patterns with joins over $\mathsf{PrXML}^{\{\mathsf{ind}\}}$ can simulate CQs over "tuple-independent" probabilistic relations [22]. But the evaluation of CQs over tuple-independent probabilistic databases can be intractable even for very simple (and small) CQs [22]. Interestingly, it has been shown that adding *any* (single) join to *any* tree pattern results in a query that is intractable, unless that query is equivalent to a join-free pattern [42].

Theorem 2. [42] *If Q is a tree pattern with a single join predicate, then one of the following holds.*

1. *Q is equivalent to a tree pattern (hence, can be evaluated in polynomial time).*
2. *The problem "compute $\Pr(\mathscr{P} \models Q)$ given $\mathscr{P} \in \mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$" is #P-hard.*

Recall that #P is the class of functions that count the number of accepting paths of the input of an NP machine [64]; this class is highly intractable, since using an oracle to a #P-hard function one can solve in polynomial time every problem in the polynomial hierarchy [63].

Next, we discuss aggregate queries. Cohen et al. [19] showed that for the aggregate functions count, min, and max, the evaluation of the corresponding aggregate queries is in polynomial time for $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$. (That result of Cohen et al. [19] is actually for a significantly broader class of queries, which they refer to as "constraints.") Note that, for these specific functions, the number of possible results

[3] For a formal definition of FPT the reader is referred to Flum and Grohe's book [27].

(numbers) q is polynomial in the size of the input p-documents; hence the evaluation of an aggregate query Q reduces (in polynomial time) to the evaluation of $\Pr(Q(\mathscr{P}) = q)$.

Theorem 3. [19] *Let Q be the aggregate query $\alpha \circ t[w]$. If α is either* count, min *or* max*, then the problem "compute $\Pr(Q(\mathscr{P}) = q)$ given $\mathscr{P} \in \mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ and $q \in \mathbb{Q}$" is in polynomial time.*

Note that an immediate consequence of Theorem 3 is that we can evaluate, in polynomial time, Boolean queries like $\mathsf{count} \circ t[w] > q$ (i.e., where equality is replaced with a different comparison operator). Unfortunately, this result does not extend to the aggregate functions countd, sum and avg.

Theorem 4. [5, 19] *For each α among* countd, sum *and* avg *there is an aggregate query $Q = \alpha \circ t[w]$, such that the problem "determine whether $\Pr(Q(\mathscr{P}) = q) > 0$ given $\mathscr{P} \in \mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ and $q \in \mathbb{Q}$" is NP-complete.*

A particularly interesting fact that is shown by Theorems 3 and 4 is that there is an inherent difference between the complexity of count and countd when it comes to query evaluation over $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$.

4 Additional p-Documents and Extensions

We now discuss additional representation systems for probabilistic XML. Some of these systems are p-document models with additional kinds of distributional nodes, and other systems are extensions of the p-document concept. We discuss the expressive power of these representation systems, and the complexity of query answering.

4.1 Long-Distance Dependencies

The mux and ind distributional nodes encode *local* dependencies between nodes, in the sense that the presence of a node in the document depends just on the presence of its parent and (in the case of mux) its siblings. However, it is often desired to represent *long-distance* dependencies to capture correlations among nodes of arbitrary locations in the document tree. Towards that, we introduce new kinds of distributional nodes. Assume a finite set $\{e_1 \dots e_n\}$ of independent Boolean random variables (called *Boolean events*), and a probability $\Pr(e_i)$ for each of these e_i. We define two new kinds of distributional nodes:

- cie [2, 4]: A distributional node v of type cie specifies for each child w of v a *conjunction of independent events* e_k or their negation $\neg e_k$ (e.g., $e_2 \wedge \neg e_5 \wedge e_6$).
- fie [41]: A distributional node v of type fie specifies for each child w of v an arbitrary propositional formula on the e_is (e.g., $e_2 \vee (e_3 \wedge \neg e_7)$).

Recall from Section 2 that, to define the semantics of a type of distributional node, we need to specify how a random subset of children is chosen by a node of that type.

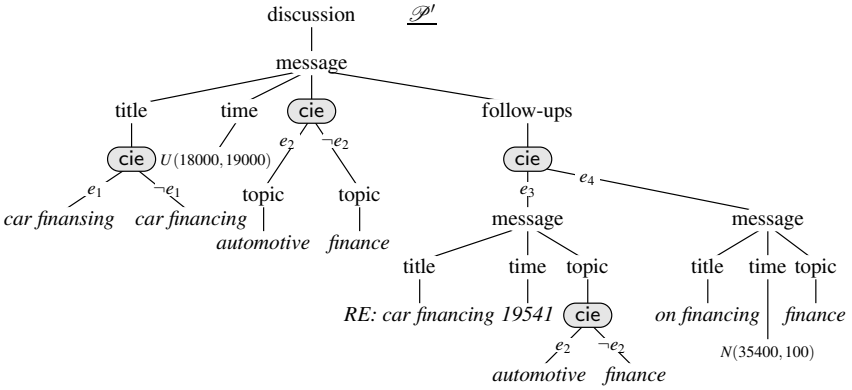

Fig. 3 A continuous p-document $\mathscr{P}'$ in $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ with $\Pr(e_1) = 0.1$, $\Pr(e_2) = 0.5$, $\Pr(e_3) = 0.8$, $\Pr(e_4) = 0.4$

For cie and fie, the specification is as follows. At the beginning of the process, we draw a random truth assignment τ to the events $e_1, \ldots, e_n$, independently of one another and according to the probabilities $\Pr(e_1), \ldots, \Pr(e_n)$. Then, each distributional node selects the children that are annotated by a formula that evaluates to true under τ. (We then proceed to the second step, as described in Section 2.)

Example 6. An example p-document $\mathscr{P}'$ of $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ is shown in Figure 3. Disregard for now the leaf nodes under "time" nodes (these nodes contain continuous distributions that will be discussed in Section 4.5). The p-document $\mathscr{P}'$ is somewhat similar to $\mathscr{P}$ of Figure 1: there is uncertainty in the title of the first message, in its topic, and in the existence of the two follow-up messages, which are independent of each other. However, there is also a fundamental difference. The topic of the first follow-up is correlated with that of the original message: either both are set to "*automotive*" or both are set to "*finance*." This reflects what a topic extraction system might do, if it has a global view of the whole discussion. □

We now look at the relative expressiveness and succinctness of p-documents defined with ind, mux, cie, and fie distributional nodes. In terms of expressiveness, $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$, $\mathsf{PrXML}^{\{\mathsf{cie}\}}$, and $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ are all able to represent all finite probability distributions over documents and are therefore equivalent [4] (as already noted, this is not the case for $\mathsf{PrXML}^{\{\mathsf{ind}\}}$, $\mathsf{PrXML}^{\{\mathsf{mux}\}}$ and, obviously, $\mathsf{PrXML}^{\{\}}$). However, in terms of succinctness the picture is different: while there is a polynomial-time transformation of a $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$ p-document into an equivalent $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ p-document, the converse is not true [44]. Similarly, $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ is a subset of $\mathsf{PrXML}^{\{\mathsf{fie}\}}$, but a transformation from $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ into $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ entails an inevitable exponential blowup [41].

The families $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ and (a fortiori) $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ are thus exponentially more succinct than ProTDB. However, this succinctness comes at a cost: query evaluation is now intractable. More precisely, *every* (Boolean) tree-pattern query is

#*P*-hard over $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ (and $\mathsf{PrXML}^{\{\mathsf{fie}\}}$), except for some trivial cases [44, 45]. The situation in $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ is essentially the same as that in $\mathsf{PrXML}^{\{\mathsf{fie}\}}$, although a few specific types of queries are tractable over $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ and yet intractable over $\mathsf{PrXML}^{\{\mathsf{fie}\}}$: projection-free tree patterns with joins [58], and expected values for some types of aggregate queries [5, 7].

The intractability of querying p-documents with long-distance dependencies discussed above concerns the computation of the *exact* probability of a query. It makes sense to look also at approximation algorithms [45]. The simplest way to approximate query probability is by Monte-Carlo sampling: pick a random document, evaluate the query, and iterate. The approximated probability will then be the ratio of draws for which the probability evaluated to true. This approach yields a polynomial-time algorithm for obtaining an *additive* approximation of the query probability; that is, a number that is guaranteed, with high confidence, to be in the interval $[p-\varepsilon; p+\varepsilon]$ around the exact probability p. Using other means [37], in the case of tree patterns over $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ it is also possible to obtain a (polynomial-time) *multiplicative* approximation (i.e., a number in the interval $[(1-\varepsilon)p, (1+\varepsilon)p]$) [45].

4.2 Conditional Models

As mentioned earlier, a central drawback in the ProTDB model (i.e., $\mathsf{PrXML}^{\{\mathsf{ind},\mathsf{mux}\}}$) and some other models proposed in the literature (e.g., [33]) is the assumption of probabilistic independence among probabilistic choices; in turn, this assumption is the key reason for the tractability of query evaluation [43]. However, even simple additional information about the database may give rise to intricate correlations. As a simple example, consider again the p-document in Figure 1. Even if we do not know the exact structure of the messages (hence, we use probabilistic rather than deterministic XML), it is likely that we know the total number of messages, and precisely (with no uncertainty involved). This new detail introduces dependency among the children of v_3, since now a random world cannot have too many (or too few) messages altogether. A more intricate statement can be the fact that at least 90% of the messages with the topic *automotive* have one or more *automotive* follow-ups; note that this statement implies correlation between the distributional nodes v_2 and v_4.

To incorporate such additional information, Cohen et al. [19] suggested to specify *constraints* in addition to the p-document. They presented a language for specifying constrains that may involve aggregate functions (e.g., "the total number of messages is 392," and "at least 80% of the messages have follow-ups").[4] Formally, a *Probabilistic XML Database* (PXDB) is a pair $(\mathscr{P},\mathscr{C})$, where $\mathscr{P}$ is a p-document and $\mathscr{C}$ is a set of *constraints*. The px-space that is defined by a PXDB $(\mathscr{P},\mathscr{C})$ is the sub-space of $\mathscr{P}$ conditioned on the satisfaction of each constraint of $\mathscr{C}$ (in other words, we restrict the px-space to the possible worlds that satisfy $\mathscr{C}$, and normalize the probabilities). Cohen et al. gave polynomial-time algorithms for various central tasks, such as *sampling* and *querying*, where their queries are tree patterns with

[4] For the precise specification of this language, see [19].

some aggregate functions (that include count, and min/max).[5] Similar tractability results have been shown for the case where both constraints and queries are phrased in MSO [20].

4.3 Recursive Markov Chains

In principle, p-documents provide means of representing arbitrary *finite* px-spaces. Some applications, however, require the ability to represent infinite sets of possible worlds. Consider again the example document of Figure 1; all such documents describing email discussions conform to the following schema, given as a DTD:

```
discussion: (message*)
   message: (title, time, topic?, follow-ups?)
follow-ups: (message*)
```

There are infinitely many documents conforming to this DTD, of arbitrarily large depth and width. In order to represent a discussion in which the number of messages and the structure of the discussion itself is fully uncertain, we need to be able to model, in a concise manner, infinite px-spaces.

The formalism of *recursive Markov chains* [25] is used for describing recursive probabilistic processes. Alternatives are described using a Markov chain, where each node in the chain can be a call to another (or the same) chain. This formalism naturally lends itself to the representation of potentially infinite px-spaces, as shown by Benedikt et al. [10]. In that work, Benedikt et al. study the tractability of MSO queries over px-spaces represented by recursive Markov chains (and restrictions thereof). In particular, recursive Markov chains that are *hierarchical* (i.e., when there are no cycles in the call graph) are tractable if we assume that all arithmetic operations have unit cost.[6] Hierarchical Markov chains can be seen as a generalization of p-documents defined with directed acyclic graphs instead of trees, a model introduced in [10, 20]. If we further restrict recursive Markov chains so that no Markov chain is called at two different positions (they are thus *tree-like*), we obtain a fully tractable model that generalizes $\mathsf{PrXML}^{\{\mathsf{mux},\mathsf{ind}\}}$ (and even more succinct models, e.g., $\mathsf{PrXML}^{\{\mathsf{exp}\}}$ [44]).

4.4 SCFGs

A *Context-Free Grammar* (CFG) specifies a process of producing parse trees for strings in a nondeterministic manner; indeed, a specific string may have multiple (even infinitely many) parse trees, since multiple production rules can be specified for a nonterminal. A *stochastic* (or *probabilistic*) *Context-Free Grammar* (SCFG) is similar to a CFG, except that the rules are augmented with probabilities; that is, the

[5] This language allows for nesting of queries, and the reader is referred to [19] for the exact details.

[6] Without this assumption, we lose tractability because the exact probability of a query may require exponentially many bits in the size of the representation.

production of a nonterminal becomes a probabilistic, rather than a nondeterministic, process.

When given a string, an SCFG implies a probability space over the possible parse trees of the string (where the probability of a parse tree corresponds to the confidence of the SCFG in that tree). Since this space comprises of labeled trees, we can view it as a (possibly infinite) px-space, on which we can evaluate XML queries (e.g., "find each noun phrase that forms an object for the verb *likes*"). Cohen and Kimelfeld [18] studied the problem of evaluating a tree-pattern query over the px-space that is represented by an SCFG and a string. In particular, they showed that this task is tractable for the class of *weakly linear SCFGs* (that generalizes popular normal forms like linear SCFGs, and Chomsky or Greibach normal forms). It follows from known results [25] that, in the general case, query probabilities do not have a polynomial-size bit representation, and can even be irrational.

4.5 Continuous Distributions

So far, all probabilistic XML models we have considered represent *discrete* probability distributions, where the uncertainty is either in the structure of the document or in the choice of a value from a finite collection of options. But some sources of uncertainty, such as the imprecision in sensor measurements, are essentially *continuous*. So, following [5, 7] we introduce the possibility of labeling leaves of p-documents with not only constant values, but *continuous probability distributions* of values (as usual, represented in some compact manner). For example, we might say that a given leaf represents a uniform distribution between two constants.

Example 7. Consider again the p-document $\mathscr{P}'$ of Figure 3. Two of the "time" nodes have for leaf a continuous distribution. The first one, $U(18000, 19000)$ represents a uniform distribution in the interval $[18000; 19000]$, which is adapted to the case when nothing else is known about the timestamp, perhaps because of a coarse granularity in the way the message metadata was displayed. The second distribution, $N(35400, 100)$ is a Gaussian centered around 35400 and with a standard deviation of 100. Such a timestamp might arise from a known imprecision in the date of the computer system that produced the timestamp. One can check that the document d of Figure 1 is one of the possible worlds represented by $\mathscr{P}'$ (but of course, it has a zero probability due to the continuous distributions). □

Observe that we cannot use our current formalism of a px-space to define the semantics of a p-document with continuous values, since our px-space is discrete, and in particular, is defined by means of a probability of each possible world. Nevertheless, px-spaces can be properly extended to a continuous version by constructing a σ-algebra of sets of possible worlds, and define a probability measure over this σ-algebra, as done by Abiteboul et al. [7]. When this is done, we can investigate the complexity of query evaluation, as usual. Tree patterns are not of much interest in this case, because if a query node is matched against a node with continuous distribution, the probability of this match is usually zero. But of course, aggregate queries make sense. As shown by Abiteboul et al. [7], the tractability of aggregate

queries with functions such as count, min, or max extends from (discrete) ProTDB to the continuous case, as long as the class of probability distributions present in the p-document can be efficiently convoluted, summed, integrated, and multiplied. This is for instance the case of distributions defined by piecewise polynomials, a generalization of uniform distributions.

5 Other Problems of Interest

In the previous sections, we discussed the task of query evaluation over different models of probabilistic XML. Here, we discuss additional tasks. Specifically, we address *updating* and *typing*, which are classical XML operations. We also discuss *compression*—the problem of finding a representation of a smaller size, and *top-k querying*—retrieving the most probable answers to a tree-pattern or a keyword-search query. Finally, we list additional tasks that are mostly left as open problems.

5.1 Updates

In update languages like *XUpdate* or the *XQuery Update Facility*, the specification of update operations entail *locator queries* that indicate, as XPath or XQuery expressions, the locations where data are to be inserted, modified, or deleted. An elementary probabilistic update operation can thus be defined as consisting of a locator query, a specification of the operation to be performed at matched locations (e.g., a tree to be inserted), and a probability that the update should be performed (provided that the locator query matches); such an operation has been studied by Abiteboul et al. [4]. The semantics of updates is defined as for queries: the result of an update on a probabilistic database should be a representation of a probabilistic space obtained from the original probabilistic space by applying the update on every possible world. Again, we want to avoid the exponential enumeration of possible worlds and perform the update directly on the original probabilistic document. Updates are of particular interest since they can be seen as a fundamental mechanism for constructing a probabilistic XML document: a sequence of uncertain update operations applied to a deterministic XML document [1].

Limiting our study to ProTDB and models with long-distance dependencies, we observe the following tradeoff on update tractability [41], in terms of data complexity:

- The result of an update operation is computable in polynomial time over ProTDB for a restricted set of non-branching tree pattern queries (specifically, those without descendant edges or those whose locator query returns the node at the bottom of the tree pattern).
- In general, computing the result of an update operation over ProTDB is intractable.
- The result of an update operation is computable in polynomial time over the family $\mathsf{PrXML}^{\{\mathsf{fie}\}}$, for updates defined by tree-pattern queries with joins.

The reason for the tractability of updates in $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ (while querying operations are hard) is that updates do not entail computation of probabilities; we just manipulate event formulas without computing their probabilities.

Updating probabilistic XML documents highlights the following issue in models different from ProTDB and $\mathsf{PrXML}^{\{\mathsf{fie}\}}$: the model may lack the property of being a *strong representation system* [3] for the query language used in locator queries; this means that it is impossible to represent the output of a query (or the result of an update based on this query language) in the model. This is the case for ProTDB extended with continuous value distributions, and the language of aggregate tree-pattern queries (or even tree-pattern queries with inequalities). To be able to apply updates on such probabilistic models, the challenge is to define generalizations of these models (and of the corresponding querying techniques) that are strong representation systems.

5.2 Typing

Typing an XML document, that is, testing whether the document is valid against a schema defined in some schema language (e.g., DTD), is another fundamental data-management problem in XML. Similarly to Boolean querying, typing a probabilistic XML document should return a probability, namely, the probability that a random document is valid. As shown by Cohen et al. [20], when the schema can be defined by a deterministic bottom-up tree automaton (which is the case for DTDs, disregarding for now keys and foreign keys), computing the probability that a ProTDB p-document is valid is in polynomial time in the size of both the p-document and the schema. Essentially, this computation is done by running the automaton over the p-document, maintaining on the way some data structures that allow us to compute the probability that a node has type q given the corresponding probabilities of its children. This result can be generalized in a number of ways. First, tractability extends to computing the probability of a fixed query (say, a tree pattern) in the probabilistic space that is restricted to only those worlds that are valid against a schema [20]. Second, the data model can be generalized to recursive Markov chains, and we basically have tractability in the same classes of recursive Markov chains where MSO query answering is tractable [10]. Third, adding constraints (such as keys and foreign keys) renders typing intractable, though it is still tractable to test whether the probability of being valid against a schema with constraints is exactly one [20].

5.3 Compression

A fundamental advantage of using probabilistic XML models, such as ProTDB, is their potential compactness in representing probabilistic spaces. Depending on the application, obtaining such a compact model might not be straightforward. The direct translation of a set of possible worlds with probabilities into a $\mathsf{PrXML}^{\{\mathsf{mux},\mathsf{ind}\}}$ document, for instance, simply enumerates all possible worlds as children of a mux node and has the same size as the original space. The *compression* or *simplification*

problem [39] is to obtain, given a probabilistic XML document, another more compact document that defines the same px-space.

In ProTDB, a basic operation that can be used to simplify a p-document is to push distributional nodes down the tree whenever possible, merging ordinary nodes in the process [66]. Another direction is to apply regular XML compression techniques [13] to compress the probabilistic tree into a probabilistic DAG while retaining querying tractability (assuming unit-cost arithmetics), as discussed in Section 4.3. Veldman et al. [66] explored the combination of probabilistic XML simplification techniques with ordinary XML compression, demonstrating gain in the size of the representation.

5.4 Top-k Queries

Chang et al. [16] studied the problem of finding, in a probabilistic XML document, the *top-k* query answers, that is, the k answers with the highest probabilities (where k is a specified natural number). Their model of probabilistic XML is ProTDB, and as queries they considered projection-free path patterns. Another type of a top-k query arises in *keyword search*. Information retrieval by keyword search on probabilistic XML has been studied by Li et al. [47]. Specifically, they perform keyword search in the ProTDB model by adopting the notion of *Smallest Lower Common Ancestor* (SLCA) [69], which defines when an XML node constitutes an answer for a keyword-search query. More particularly, the problem they explore is that of finding the k nodes with the highest probabilities of being SLCAs in a random world.

5.5 Open Problems

We now discuss important open problems around management operations on probabilistic XML. Despite the existence of techniques for compressing ProTDB documents [66], we lack a good understanding on when compression is possible and whether it is possible to obtain an *optimal* representation (with respect to compactness) of a px-space, in ProTDB and other models. A fundamental problem related to this one concerns *equivalence* of probabilistic XML documents: decide whether two representations define the same px-space [39]. As shown in [57], this problem admits a randomized polynomial-time decision procedure for $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ when p-documents are shallow, giving some hope of obtaining a more systematic procedure for minimizing the size of a p-document. Nevertheless, the exact complexities of the equivalence problem, of testing optimality, and of minimization itself, remain open problems.

Compressing a discrete px-space into a compact p-document is somewhat akin to the problem of XML schema inference from XML data [11]: in both cases, the goal is to obtain a compact model of a set of documents. There are two differences, however. First, an XML schema represents a set of XML documents, while a p-document represents a probabilistic distribution thereof. Second, it is assumed that XML schema inference generalizes the observation of the example documents and

that some documents valid against the schema are not present in the original collection, while compression preserves the px-space. Relaxing this last assumption leads to the problem of *probabilistic schema inference*, that is, learning a probabilistic model, with potential generalization, for a corpus of XML documents. A first work in this direction is by Abiteboul et al. [6], where the skeleton of the schema is given, and probabilities are learned to optimize the likelihood of the corpus. Adapting XML schema inference techniques to directly generate probabilistic models would allow us to generalize any collection of XML documents as a probabilistic XML document.

The focus of most of the literature on probabilistic XML is on modeling and querying, while only little exploration has been done on other aspects of probabilistic XML management. One of the important aspects that deserve further exploration is that of *mining*, namely, discovering important patterns and trends (e.g., frequent items, correlations, summaries of data values, etc.) in probabilistic XML documents. Kharlamov and Senellart [40] discuss how some mining tasks can be answered using techniques of probabilistic XML querying. Nevertheless, it is to be explored whether other techniques (e.g., based on ordinary frequent itemset discovery) can provide more effective mining.

6 Practical Aspects

In this section, we discuss some practical aspects of probabilistic XML management. We first consider system architecture and indexing, and then elaborate on the practical challenges that remain to be tackled towards a full-fledged database-management system for probabilistic XML. (To the best of our knowledge, up to now only prototypical systems have been developed.)

6.1 System Architecture

The first question is that of the general architecture of a probabilistic XML system: should it be *(a)* built on top of a probabilistic relational database system, *(b)* based on a query-evaluation engine for ordinary XML, *or (c)* engineered from scratch to easily accommodate the existing algorithmic approaches for probabilistic XML? We overview these three approaches, pointing to preliminary work, and noting advantages and shortcomings of each.

Over a Probabilistic Relational Engine

Much effort has been put on building efficient systems for managing probabilistic relational data. These systems include Trio [67], MayBMS [32] and its query evaluator SPROUT [54] (in turn, these systems are usually built on top of an ordinary relational database engine). Leveraging these efforts to the probabilistic XML case makes sense, and basically amounts to encoding probabilistic XML data into probabilistic tables, and tree-pattern queries into conjunctive queries. This direction is explored by Hollander and van Keulen [31] with Trio, where feasibility is

demonstrated for different kinds of XML-to-relation encodings. However, the relational queries that result from those encodings are of a specific form (e.g., inequalities are used to encode descendant queries) for which optimizations are not always available to the probabilistic relational engine.

Over of an XML Query Engine

Alternatively, it is possible to rely on native XML database systems (such as eXist[7] or MonetDB/XQuery[8]) to evaluate queries over probabilistic XML documents, delegating components such as indexing of document structure and query optimization to the underlying XML database engine. It requires either to modify the internals of the XML query evaluation engine to deal with probabilities, or to be able to rewrite queries over probabilistic XML documents as queries over ordinary documents. The latter approach is demonstrated by Senellart and Souihli [59]; there, tree-pattern queries with joins over p-documents of $\mathsf{PrXML}^{\{\mathsf{cie}\}}$ are rewritten into XQuery queries that retrieve each query match, along with a propositional formula that represents the probability of the match. All XML processing is therefore handed out to the XQuery query engine, and the problem is reduced to probability evaluation of propositional formulas.

Independent Implementation

The previous two architectures do not make use of the specificities of probabilistic XML, and in particular, of the techniques that have been developed for querying probabilistic XML. An alternative is thus to design a probabilistic XML system around one or more of these techniques (e.g., bottom-up dynamic programming [43]), and thereby utilize the known algorithms at query time [44]. The downside of this approach is that existing algorithms are main-memory intensive. Furthermore, the implemented system is typically applicable to only a limited probabilistic model (e.g., [44] supports just ProTDB documents, though it should be possible to use a similar bottom-up approach for hierarchical Markov chains [10] and to support continuous distributions [7]), and to a limited class of queries (e.g., [44] supports just tree patterns, but it should also be possible to extend it to MSO by combining the algorithm of [20] and a toolkit such as Mona [30] for converting queries into tree automata).

6.2 Indexing

We now consider *indexing* as a mean of enhancing the efficiency of query evaluation over probabilistic XML. When a probabilistic XML system is implemented on top of an XML database system, we can rely on this system to properly index the tree structure and content. Still remaining is the question of how to provide efficient access to the probabilistic annotations.

[7] `http://exist.sourceforge.net/`

[8] `http://monetdb.project.cwi.nl/monetdb/XQuery/`

The PEPX system [48] proposes to index ProTDB documents in the following manner: instead of storing with each child of a mux or ind node the probability of being selected by its parent, store the marginal probability that the child exists. Coupled with indexing of the tree structure, it allows much more efficient processing of simple queries, since a single access suffices to retrieve the probability of a node, and accessing all ancestors of this node is not required. This approach has also been taken by Li et al. [49] who adapted the TwigStack algorithm [12] to the evaluation of projection-free patterns in a ProTDB document.

This is certainly not the last word on probabilistic XML indexing, though. An interesting direction would be to combine structure-based indexing with probability-based indexing. Conceivably, such an approach has the potential of enhancing the efficiency of finding the most probable answers [16] or answers with a probability above a specified threshold [43].

6.3 Remaining Challenges

We now highlight some of the challenges that remain on the way to implementing a full-fledged system for managing probabilistic XML.

We first discuss the choice of method for query evaluation. Depending on the data model in use, and depending on the query language, we have a variety of techniques, exact or approximate: bottom-up algorithm in the absence of long-distance correlations [43], naïve enumeration of all possible worlds, Monte-Carlo sampling, relative approximation [44], and so on. Each of these has specific particularities in terms of the range of query and data it can be applied to, its evaluation cost, and its approximation guarantee. Hence, it is likely that some methods are suitable in some cases and other methods are suitable in other cases. A system should have a wealth of evaluation techniques and algorithms, and should be able to make proper decisions on which technique to use for providing a quick and accurate result. For example, the system may be given precision boundaries, and it should then select the most efficient approximation that guarantees these boundaries. Alternatively, given a time budget, a system should be able to select an exact or approximation technique (as precise as possible) for performing query evaluation within that budget. This process can be carried out at the level of the whole query, or at the level of each sub-query. For instance, in some cases it may be beneficial to combine probabilities that are computed (for different parts of the query and/or the document) by deploying different techniques. This suggests relying on cost-based, optimizer-like, query planning where each implementation of a (sub-)query evaluation is associated with an estimated cost, of both time and approximation. First steps are highlighted in [60].

There is also a need for a deeper understanding of the connection between probabilistic XML and probabilistic relational data. This is obviously critical if one is to implement a probabilistic XML system on top of a probabilistic relational database; but, it is important in other architectures as well, for identifying techniques in the relational setting that carry over to the XML setting. The connection is not so straightforward. It is of course easy to encode trees into relations or relations into trees, but

in both cases the encoding has special shapes: relations encoding trees are tree-like (with treewidth [55] one) and relations encoded as trees are shallow and have repetitive structure. Typical query languages are different, too: tree-pattern queries or MSO on one side, conjunctive queries or the relational algebra on the other. When trees are encoded into relations, tree-pattern queries become a particular kind of conjunctive queries, involving hierarchically structured self joins, a class for which it is not always possible to obtain efficient query plans over arbitrary databases [61]. Some results from the probabilistic XML setting (such as the bottom-up evaluation algorithm for ProTDB) have no clear counterpart in the relational world, and vice versa. A unifying view of both models would help in building systems for managing both probabilistic relational and XML data.

The last challenge we highlight is that of optimizing query evaluation by reusing computed answers of previous queries. This can be seen as a case of *query answering using views*, a problem that has been extensively studied in the deterministic XML setting [8, 15, 68]. There is little known on whether and how (materialized) views can be used for query answering in the probabilistic XML setting, though Cautis and Kharlamov [14] have made a preliminary study of the problem in the setting of ProTDB, where they show that the major challenge is not retrieving query answers, but computing their probabilities.

7 Conclusions

We reviewed the literature on probabilistic XML models, which are essentially representation systems for compactly encoding probability distributions over labeled trees. A variety of such representation systems have been proposed, and each provides a different trade-off between expressiveness and compactness on the one hand, and management complexity on the other hand. Specifically, ProTDB [53] and some of its extensions (e.g., ProTDB augmented with constraints or continuous distributions, and tree-like Markov chains) feature polynomial-time querying for a rich query language (MSO, or aggregate queries defined by tree-patterns). In contrast, query evaluation is intractable in other models such as $\mathsf{PrXML}^{\{\mathsf{fie}\}}$ (that allows for correlation among arbitrary sets of nodes) or arbitrary recursive Markov chains (that can represent spaces of unbounded tree height or tree width).

We mentioned various open problems throughout this chapter. Two of these deserve particular emphasis. First, the connection to probabilistic relational models needs better understanding, from both the theoretical viewpoint (e.g., what makes tree-pattern queries over ProTDB tractable, when they are encoded into relations?) and the practical viewpoint (e.g., can we build on a system such as Trio [67] or MayBMS [32] to effectively manage probabilistic XML data?). Second, further effort should be made to realize and demonstrate the ideal of using probabilistic XML databases, or probabilistic databases in general, to answer data needs of applications (rather than devising per-application solutions); we discussed some of the wide range of candidate applications in the introduction. We believe that the research of recent years, which is highly driven by the notable popularity of probabilistic

databases in the database-research community, constitutes a significant progress towards this ideal, by significantly improving our understanding of probabilistic (XML) databases, by developing a plethora of algorithmic techniques, and by building prototype implementations thereof.

Acknowledgements. We are grateful to Evgeny Kharlamov for his helpful comments. This work was partially supported by the European Research Council grant Webdam (under FP7), grant agreement 226513.

References

[1] Abdessalem, T., Ba, M.L., Senellart, P.: A probabilistic XML merging tool. In: EDBT (2011)
[2] Abiteboul, S., Senellart, P.: Querying and updating probabilistic information in XML. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 1059–1068. Springer, Heidelberg (2006)
[3] Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
[4] Abiteboul, S., Kimelfeld, B., Sagiv, Y., Senellart, P.: On the expressiveness of probabilistic XML models. VLDB Journal 18(5) (2009)
[5] Abiteboul, S., Chan, T.H.H., Kharlamov, E., Nutt, W., Senellart, P.: Aggregate queries for discrete and continuous probabilistic XML. In: ICDT (2010)
[6] Abiteboul, S., Amsterdamer, Y., Deutch, D., Milo, T., Senellart, P.:) Optimal probabilistic generators for xml corpora. In: BDA (2011a)
[7] Abiteboul, S., Chan, T.H.H., Kharlamov, E., Nutt, W., Senellart, P.: Capturing continuous data and answering aggregate queries in probabilistic XML. ACM Transactions on Database Systems (2011b)
[8] Afrati, F.N., Chirkova, R., Gergatsoulis, M., Kimelfeld, B., Pavlaki, V., Sagiv, Y.: On rewriting XPath queries using views. In: EDBT. ACM International Conference Proceeding Series, vol. 360, pp. 168–179. ACM (2009)
[9] Amer-Yahia, S., Cho, S., Lakshmanan, L.V.S., Srivastava, D.: Minimization of tree pattern queries. In: SIGMOD (2001)
[10] Benedikt, M., Kharlamov, E., Olteanu, D., Senellart, P.: Probabilistic XML via Markov chains. Proceedings of the VLDB Endowement 3(1) (2010)
[11] Bex, G.J., Neven, F., Vansummeren, S.: Inferring XML schema definitions from XML data. In: VLDB (2007)
[12] Bruno, N., Koudas, N., Srivastava, D.: Holistic twig joins: optimal XML pattern matching. In: SIGMOD (2002)
[13] Buneman, P., Grohe, M., Koch, C.: Path queries on compressed XML. In: VLDB (2003)
[14] Cautis, B., Kharlamov, E.: Challenges for view-based query answering over probabilistic XML. In: AMW (2011)
[15] Cautis, B., Deutsch, A., Onose, N.: XPath rewriting using multiple views: Achieving completeness and efficiency. In: WebDB (2008)
[16] Chang, L., Yu, J.X., Qin, L.: Query ranking in probabilistic XML data. In: EDBT (2009)
[17] Cheng, R., Singh, S., Prabhakar, S.: U-DBMS: A database system for managing constantly-evolving data. In: VLDB (2005)

[18] Cohen, S., Kimelfeld, B.: Querying parse trees of stochastic context-free grammars. In: ICDT (2010)
[19] Cohen, S., Kimelfeld, B., Sagiv, Y.: Incorporating constraints in probabilistic XML. ACM Transactions on Database Systems 34(3) (2009)
[20] Cohen, S., Kimelfeld, B., Sagiv, Y.: Running tree automata on probabilistic XML. In: PODS (2009)
[21] Dalvi, N., Ré, C., Suciu, D.: Probabilistic databases: Diamonds in the dirt. Communications of the ACM 52(7) (2009)
[22] Dalvi, N.N., Suciu, D.: Efficient query evaluation on probabilistic databases. VLDB Journal 16(4) (2007)
[23] Doner, J.: Tree acceptors and some of their applications. Journal of Computer Systems and Science 4(5) (1970)
[24] Downey, R.G., Fellows, M.R.: Parameterized Complexity. Monographs in Computer Science. Springer (1999)
[25] Etessami, K., Yannakakis, M.: Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. Journal of the ACM 56(1) (2009)
[26] Fagin, R., Kimelfeld, B., Kolaitis, P.: Probabilistic data exchange. In: ICDT (2010)
[27] Flum, J., Grohe, M.: Parameterized Complexity Theory. In: Texts in Theoretical Computer Science. An EATCS Series. Springer (2006)
[28] Frick, M., Grohe, M.: The complexity of first-order and monadic second-order logic revisited. Annals of Pure and Applied Logic 130(1-3) (2004)
[29] Galindo, J., Urrutia, A., Piattini, M.: Fuzzy Databases: Modeling, Design And Implementation. IGI Global (2005)
[30] Henriksen, J., Jensen, J., Jørgensen, M., Klarlund, N., Paige, B., Rauhe, T., Sandholm, A.: Mona: Monadic second-order logic in practice. In: Brinksma, E., Steffen, B., Cleaveland, W.R., Larsen, K.G., Margaria, T. (eds.) TACAS 1995. LNCS, vol. 1019, pp. 89–110. Springer, Heidelberg (1995)
[31] Hollander, E., van Keulen, M.: Storing and querying probabilistic XML using a probabilistic relational DBMS. In: MUD (2010)
[32] Huang, J., Antova, L., Koch, C., Olteanu, D.: MayBMS: a probabilistic database management system. In: SIGMOD (2009)
[33] Hung, E., Getoor, L., Subrahmanian, V.S.: PXML: A probabilistic semistructured data model and algebra. In: ICDE (2003)
[34] Imieliński, T., Lipski Jr., W.: Incomplete information in relational databases. Journal of the ACM 31(4) (1984)
[35] Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C.M., Haas, P.J.: MCDB: a Monte Carlo approach to managing uncertain data. In: SIGMOD (2008)
[36] Jousse, F., Gilleron, R., Tellier, I., Tommasi, M.: Conditional random fields for XML trees. In: ECML Workshop on Mining and Learning in Graphs (2006)
[37] Karp, R.M., Luby, M., Madras, N.: Monte-Carlo approximation algorithms for enumeration problems. Journal of Algorithms 10(3) (1989)
[38] van Keulen, M., de Keijzer, A.: Qualitative effects of knowledge rules and user feedback in probabilistic data integration. VLDB Journal 18(5) (2009)
[39] van Keulen, M., de Keijzer, A., Alink, W.: A probabilistic XML approach to data integration. In: ICDE (2005)
[40] Kharlamov, E., Senellart, P.: Modeling, querying, and mining uncertain XML data. In: Tagarelli, A. (ed.) XML Data Mining: Models, Methods, and Applications. IGI Global (2011)

[41] Kharlamov, E., Nutt, W., Senellart, P.: Updating probabilistic XML. In: EDBT/ICDT Workshop on Updates in XML (2010)
[42] Kharlamov, E., Nutt, W., Senellart, P.: Value joins are expensive over (probabilistic) XML. In: LID (2011)
[43] Kimelfeld, B., Sagiv, Y.: Matching twigs in probabilistic XML. In: VLDB (2007)
[44] Kimelfeld, B., Kosharovski, Y., Sagiv, Y.: Query efficiency in probabilistic XML models. In: SIGMOD (2008)
[45] Kimelfeld, B., Kosharovsky, Y., Sagiv, Y.: Query evaluation over probabilistic XML. VLDB Journal 18(5) (2009)
[46] Lafferty, J., McCallum, A., Pereira, F.: Conditional Random Fields: Probabilistic models for segmenting and labeling sequence data. In: ICML (2001)
[47] Li, J., Liu, C., Zhou, R., Wang, W.: Top-k keyword search over probabilistic XML data. In: ICDE (2011)
[48] Li, T., Shao, Q., Chen, Y.: PEPX: a query-friendly probabilistic XML database. In: MUD (2006)
[49] Li, Y., Wang, G., Xin, J., Zhang, E., Qiu, Z.: Holistically twig matching in probabilistic XML. In: ICDE (2009)
[50] Manning, C.D., Schutze, H.: Foundations of Statistical NLP. MIT Press (1999)
[51] Meyer, A.R.: Weak monadic second-order theory of successor is not elementary recursive. Logic Colloquium 453 (1975)
[52] Neven, F., Schwentick, T.: Query automata over finite trees. Theoretical Computer Science 275(1-2) (2002)
[53] Nierman, A., Jagadish, H.V.: ProTDB: Probabilistic data in XML. In: VLDB (2002)
[54] Olteanu, D., Huang, J., Koch, C.: SPROUT: Lazy vs. eager query plans for tuple-independent probabilistic databases. In: ICDE (2009)
[55] Robertson, N., Seymour, P.D.: Graph minors. III. Planar tree-width. Journal of Combinatorial Theory, Series B 36(1) (1984)
[56] Sen, P., Deshpande, A., Getoor, L.: PrDB: managing and exploiting rich correlations in probabilistic databases. VLDB Journal 18(5) (2009)
[57] Senellart, P.: Comprendre le Web caché. Understanding the hidden Web. PhD thesis, Université Paris XI (2007)
[58] Senellart, P., Abiteboul, S.: On the complexity of managing probabilistic XML data. In: PODS (2007)
[59] Senellart, P., Souihli, A.: ProApproX: A lightweight approximation query processor over probabilistic trees. In: SIGMOD (2011)
[60] Souihli, A.: Efficient query evaluation over probabilistic XML with long-distance dependencies. In: EDBT/ICDT PhD Workshop (2011)
[61] Suciu, D., Olteanu, D., Ré, C., Koch, C.: Probabilistic Databases. Morgan & Claypool (2011)
[62] Thatcher, J.W., Wright, J.B.: Generalized finite automata theory with an application to a decision problem of second-order logic. Mathematical Systems Theory 2(1) (1968)
[63] Toda, S., Ogiwara, M.: Counting classes are at least as hard as the polynomial-time hierarchy. SIAM Journal on Computing 21(2), 316–328 (1992)
[64] Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)
[65] Vardi, M.Y.: The complexity of relational query languages (extended abstract). In: STOC (1982)

[66] Veldman, I., de Keijzer, A., van Keulen, M.: Compression of probabilistic XML documents. In: Godo, L., Pugliese, A. (eds.) SUM 2009. LNCS, vol. 5785, pp. 255–267. Springer, Heidelberg (2009)
[67] Widom, J.: Trio: A system for integrated management of data, accuracy, and lineage. In: CIDR (2005)
[68] Xu, W., Özsoyoglu, Z.: Rewriting XPath queries using materialized views. In: VLDB (2005)
[69] Xu, Y., Papakonstantinou, Y.: Efficient keyword search for smallest LCAs in XML databases. In: SIGMOD (2005)
[70] Zadeh, L.A.: A simple view of the Dempster-Shafer theory of evidence and its implication for the rule of combination. AI Magazine 7(2) (1986)